

BAUD Candice

ENSAE 2ème année

*Stage d'application*

Année 2022-23

---

# Utilisation de méthodes de Monte Carlo pour la prédition météorologique

---

Linköping University (LiU),  
Linköping, Suède

*Supervisé par Mr LINDSTEN FREDRIK*  
Du 1er juin au 30 septembre 2023



Novembre 2023

ENSAE PARIS,

TSA 26644

Service des relations entreprises et des stages

5, avenue Henry le Châtelier - 91764 Palaiseau CEDEX - France - Tel : +33 (0)1 70 26 67 39 - courriel :  
[stage@ensae.fr](mailto:stage@ensae.fr)

## **Remerciements**

Je souhaite avant tout remercier l'université de Linköping pour m'avoir accueillie et offert une supervision de très grande qualité durant ces quatre mois. Particulièrement, je remercie Fredrik Lindsten, mon superviseur pour sa précision, sa patience, sa flexibilité et son attention.

Je remercie également l'ensemble des membres de la division IDA-STIMA qui m'ont intégrée à l'équipe, m'ont fait découvrir la culture suédoise et ont toujours été bienveillants et chaleureux.

Je remercie le SMHI pour ce projet très intéressant. Particulièrement, je remercie Tomas Schön pour le temps passé à expliquer les modèles, pour sa clarté et sa disponibilité.

Enfin, et surtout, je remercie Mr Chopin pour m'avoir mise en contact avec Mr Lindsten, rendant ce stage possible, et pour avoir été mon correspondant à l'ENSAE.

## Abréviations

- **SMHI** : Institut Suédois de Météorologie et d'Hydrologie
- **SURFEX** : Système d'Utilisation de Recherche pour la Forêt et l'Environnement
- **PF** : Particle Filter
- **SSM** : State Space Model
- **IS** : Importance Sampling
- **SIS** : Sequential Importance Sampling
- **SMC** : Sequential Monte Carlo
- **BPF** : Bootstrap Particle Filter
- **FAPF** : Fully Adapted Particle Filter
- **APF** : Auxiliary Particle Filter
- **DAC** : Divide And Conquer
- **NSMC** : Nested Sequential Monte Carlo
- **KF** : Kalman Filter
- **EKF** : Extended Kalman Filter
- **ESS** : Effective Sample Size

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Modèle Surfex</b>	<b>2</b>
<b>3</b>	<b>Données disponibles</b>	<b>2</b>
<b>4</b>	<b>Techniques de Monte Carlo</b>	<b>4</b>
4.1	Cadre général . . . . .	4
4.1.1	State-space model . . . . .	4
4.1.2	Filtrage . . . . .	5
4.2	Monte Carlo classique . . . . .	7
4.2.1	Importance sampling . . . . .	7
4.2.2	Self-normalized importance sampling . . . . .	8
4.2.3	Sequential Importance Sampling . . . . .	9
4.3	Monte Carlo séquentiel . . . . .	10
4.3.1	Implémentation d'un algorithme de SMC . . . . .	10
4.3.2	Bootstrap particle filter . . . . .	11
4.3.3	Auxiliary particle filter . . . . .	13
4.3.4	Fully adapted particle filter . . . . .	14
4.4	En grande dimension . . . . .	14
4.4.1	Divide and Conquer . . . . .	14
4.4.2	Nested sequential Monte Carlo . . . . .	16
<b>5</b>	<b>Résultats</b>	<b>17</b>
5.1	Implémentation sur modèles linéaires gaussiens . . . . .	17
5.1.1	Modèle linéaire Gaussien . . . . .	17
5.1.2	Filtre de Kalman . . . . .	17
5.1.3	Résultats des différents algorithmes . . . . .	18
5.2	Application aux modèles météorologiques . . . . .	26
5.2.1	Modèle de Lorenz96 . . . . .	26
5.2.2	Ensemble Kalman filter . . . . .	27
5.2.3	Différents algorithmes considérés et calculs . . . . .	28
5.2.4	Résultats . . . . .	29

<b>6 Conclusion</b>	<b>30</b>
<b>7 Bonus : Effectuer un stage à l'étranger</b>	<b>31</b>
7.1 Géographie . . . . .	31
7.1.1 Quelques photos de villes scandinaves . . . . .	32
7.2 Gastronomie . . . . .	33
7.2.1 Le fika . . . . .	34
7.2.2 Spécialités salées atypiques . . . . .	34
<b>8 Annexes</b>	<b>36</b>
8.1 Algorithmes . . . . .	36
8.2 Figures . . . . .	38
<b>Table des figures</b>	<b>49</b>
<b>Liste des tableaux</b>	<b>50</b>
<b>Références</b>	<b>51</b>

## 1 Introduction

La prévision météorologique repose sur la quête de comprendre et d'anticiper les fluctuations de la nature à travers l'étude minutieuse des processus atmosphériques. En combinant des données scientifiques, des modèles mathématiques sophistiqués et une analyse attentive, la prévision météorologique vise à atténuer les risques liés aux intempéries, à optimiser les opérations liées au transport, à l'agriculture, à l'énergie et à diverses autres industries, tout en contribuant à la sécurité, au bien-être et à la résilience de notre société face aux aléas climatiques. Si nous avons Météo France, les Suédois ont eux le SMHI (Institut Suédois de Météorologie et d'Hydrologie) qui s'occupe de développer de tels modèles, en coopération avec les autres pays Scandinaves.

En collaboration entre l'Université de Linköping et le SMHI, j'ai pendant mon stage cherché à implémenter des méthodes de Monte Carlo pour améliorer les prévisions météorologiques concernant les sols. Ces prévisions sont particulièrement utiles pour l'agriculture et l'étude des éco-systèmes. Les variables d'intérêt comportaient entre autres l'humidité du sol, la quantité de neige, le pourcentage de glace dans le sol, et ce à des profondeurs différentes.

Le rapport qui suit commence par une présentation du problème météorologique et des données, puis explicite les différentes méthodes statistiques utilisées, appartenant aux techniques de Monte Carlo. La dernière partie se concentre sur les résultats d'implémentations théoriques sur différents modèles.

## 2 Modèle Surfex

Le modèle météorologique considéré par le SMHI est le modèle SURFEX (Système d’Utilisation de Recherche pour la Forêt et l’Environnement). Développé initialement dans les années 1990 par le Centre National de Recherches Météorologiques (CNRM) du Centre National de la Recherche Scientifique (CNRS) et Météo-France ; il représente un système de modélisation des processus environnementaux. Il se concentre sur la simulation des interactions entre l’atmosphère, la surface terrestre et les différents composants du système Terre. Les applications de ce modèle sont majoritairement la prédition météorologique, la modélisation climatique, les études hydrologiques et les évaluations des impacts environnementaux.

Dans mon projet, je me concentre sur les sols et la végétation [1], plus particulièrement sur la partie diffusive détaillée dans le chapitre 4, partie 1.3.

## 3 Données disponibles

Les données disponibles proviennent de mesures réalisées au nord de la Finlande durant 10 ans. On se concentre ici sur le sol et la végétation décrits par le modèle Surfex [1], [2]. On dispose de données sur différents *patches* (parcelles) :

- **Premier patch** : Agriculture
- **Deuxième patch** : Arbres

Pour chacun de ces patchs, on dispose de deux couches :

- **Une couche qui correspond à la neige**, contenant 12 niveaux pour lesquels on dispose des variables suivantes :
  - la température de la neige
  - la densité de la neige
  - la quantité d’eau contenue
  - l’âge de la neige
- **Une couche qui correspond au sol**, contenant 14 niveaux pour lesquels on dispose des variables suivantes :
  - la température du sol
  - la quantité d’eau

- la quantité d'eau glacée

On peut résumer les informations précédentes selon ce schéma :

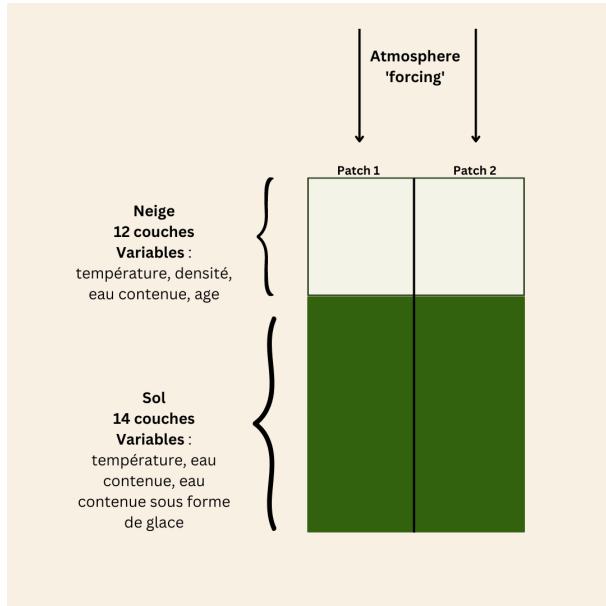


FIGURE 2 – Résumé des données

Etant donné cette structure, on cherche à extraire de l'information des mesures que l'on a. C'est-à-dire, il est possible que l'on ne mesure pas toutes les variables à toutes les profondeurs. Dans ce cas-là, on souhaite inférer des informations sur les états non observés en se basant sur ce que l'on observe. Pour prendre un exemple concret : admettons que l'on puisse mesurer l'épaisseur de la neige mais pas la température du sol en surface. On peut s'imaginer que quelque chose lie ces deux grandeurs (par exemple une relation de proportionnalité). Connaissant l'épaisseur de neige on a alors la température du sol. Notre objectif principal est donc à partir de mesures de pouvoir identifier la loi du phénomène d'intérêt non observé.

Le problème auquel nous faisons face apparaît multidimensionnel : on s'intéresse à de nombreuses variables, dans des couches différentes, et on étudie ceci en fonction du temps. Notre système évolue temporellement au gré des saisons et les couches s'influencent les unes les autres au niveau spatial.

Nous allons mettre en place des filtres à particules pour approximer les lois des phénomènes étudiés dont les principes et algorithmes sont détaillés ci-après.

## 4 Techniques de Monte Carlo

Comme expliqué précédemment, le projet vise à améliorer la prévision météorologique en identifiant la loi du phénomène étudié. Pour cela, nous utilisons des méthodes couramment appelées filtres à particules. Ces dernières représentent des hypothèses sur notre phénomène, dont on évalue la pertinence par rapport aux mesures dont nous disposons. Leur pertinence est évaluée par un poids attribué à chacune. L'approximation finale de la densité est alors une somme pondérée de mesures de Dirac. En ayant une infinité de particules, on retrouve la distribution.

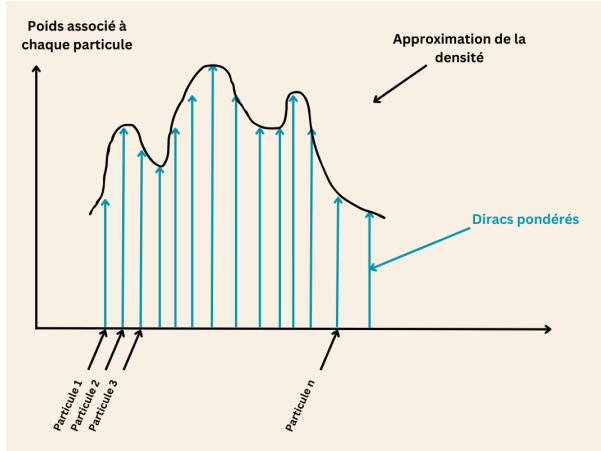


FIGURE 3 – Approximation de la densité par des diracs pondérés

La figure [3] permet de visualiser la méthode d'approximation par des diracs. Les mesures de dirac associées aux particules et à leurs poids sont représentées en bleu. La courbe noire représente la distribution approximée par les particules.

### 4.1 Cadre général

#### 4.1.1 State-space model

On s'intéresse à un modèle dynamique dans lequel on a des mesures temporelles d'une séquence  $y_{1:t} = (y_1, \dots, y_t)$ . On définit un *state-space model* [2] qui consiste en deux processus stochastiques : l'un est observé et mesuré  $Y_t$ , tandis que l'autre est latent  $X_t$ . On peut représenter ce type de modèles par une équation générale :

$$X_{t+1} = f(X_t) + V_t$$

$$Y_t = g(X_t) + E_t$$

où  $\theta$  est un paramètre, et  $V_t$  et  $E_t$  sont des termes d'erreur (*process noise* et *measurement noise*). On peut rééxprimer le SSM (State Space Model) avec les équations suivantes :

$$X_{t+1}|X_t = x_t \sim p(x_{t+1}|x_t)$$

$$Y_t|X_t = x_t \sim p(y_t|x_t)$$

$$X_0 \sim p(x_0)$$

On représente graphiquement sur la figure [4] le modèle.  $X_t$  modélise la dynamique d'évolution spatiale, tandis que  $Y_t$  modélise les mesures et la relation avec  $X$  (non observé).

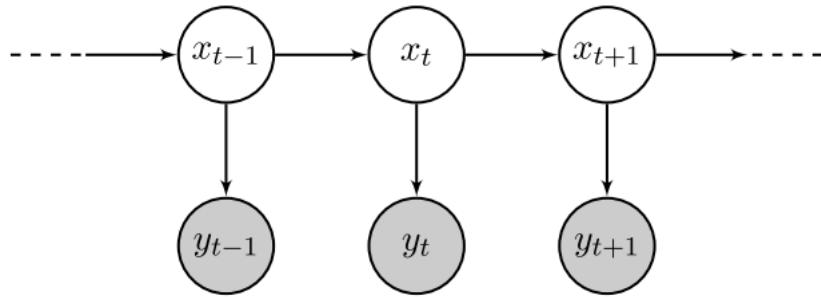


FIGURE 4 – State space model, extrait de [3]

On peut définir deux quantités :

$p(x_t|x_{t-1})$  la *transition density*

$p(y_t|x_t)$  la densité des observations

Notre problème revient donc à déduire des informations sur les valeurs de  $X$  sachant qu'on ne l'observe pas, mais en s'aidant des valeurs observées de  $Y$ .

#### 4.1.2 Filtrage

On se concentre sur l'extraction d'information sur l'état  $x_t$ , sachant l'information que l'on connaît jusqu'au temps  $t$ , contenue dans  $y_{1:t}$ <sup>1</sup>. Cela signifie que l'on souhaite construire de manière séquentielle une estimation de l'état  $x_t$  à partir des observations disponibles jusqu'à ce moment.

1.  $y_{1:t}$  est la convention pour définir les valeurs de  $y$  entre 1 et  $t$

tielle la quantité  $p(x_t|y_{1:t})$ . On peut réécrire la quantité avec la formule de Bayes comme

$$p(x_t|y_{1:t}) = p(x_t|y_t, y_{1:t-1}) = \frac{p(y_t|x_t, y_{1:t-1})p(x_t|y_{1:t-1})}{p(y_t|y_{1:t-1})}$$

Par définition du SSM, on a indépendance conditionnelle et donc on obtient

$$p(x_t|y_{1:t}) = \frac{p(y_t|x_t)p(x_t|y_{1:t-1})}{p(y_t|y_{1:t-1})}$$

De plus,

$$p(x_t|y_{1:t-1}) = \int p(x_t, x_{t-1}|y_{1:t-1})dx_{t-1} = \int p(x_t|x_{t-1})p(x_{t-1}|y_{1:t-1})dx_{t-1}$$

On peut donc réécrire

$$p(x_t|y_{1:t}) = \frac{p(y_t|x_t) \int p(x_t|x_{t-1})p(x_{t-1}|y_{1:t-1})dx_{t-1}}{p(y_t|y_{1:t-1})}$$

Notre but étant d'approximer  $p(x_t|y_{1:t})$ , et l'expression ci dessus ne présentant pas de solution analytique, nous allons utiliser un ensemble de particules, où chacune de ces particules représente un échantillon hypothétique de  $X_t$ . L'algorithme de filtrage fonctionne de manière récursive, mettant à jour l'ensemble de particules à mesure que de nouvelles mesures deviennent disponibles. À chaque instant, les particules sont propagées à travers le modèle de transition d'état et pondérées en fonction de leur vraisemblance par rapport aux mesures : les particules ayant des poids plus élevés ont plus de chances de représenter l'état réel du système. Après la mise à jour des poids des particules, une étape de rééchantillonnage est effectuée pour sélectionner un nouvel ensemble de particules pertinentes pour l'instant suivant.

Dans la section suivante, j'introduis les méthodes de Monte Carlo avant de rentrer plus en détail dans les différents types de filtrage considérés durant mon stage et leurs particularités. Cette partie théorique permet de comprendre les résultats des implémentations sur modèles linéaires gaussiens [5.1] et sur modèles météorologiques [5.2].

Quelques notations pour la suite :

- Filtering density  $p(x_t|y_{1:t})$
- Joint smoothing density  $p(x_{1:T}|y_{1:T})$
- Marginal smoothing density  $p(x_t|y_{1:T})$
- Prediction density  $p(x_{t+1}|y_{1:t})$

## 4.2 Monte Carlo classique

Considérons que nous voulions calculer la valeur :

$$E_{\pi}[\phi(X)] = \int \phi(x)\pi(x)dx$$

Alors, nous pouvons faire appel à des méthodes de Monte Carlo ! L'idée est que si l'on a  $X \sim \pi(x)$ , et que l'on a  $N$  échantillons  $(x^i)_{i=1,\dots,N}$  tirés selon la loi  $\pi(x)$ , on peut approximer empiriquement la densité

$$\hat{\pi}^N(x) = \frac{1}{N} \sum_{i=1}^N \delta_{x^i}(x)$$

En considérant l'approximation de l'espérance sous la loi  $\pi$  de  $\phi(X)$  où  $\phi$  est une fonction et  $X$  une variable aléatoire. Celle-ci peut se réécrire :

$$E_{\pi}[\phi(X)] = \int_X \phi(x)\pi(x)dx \approx \frac{1}{N} \sum_{i=1}^N \phi(x^i) \text{ où } x^i \sim \pi \text{ i.i.d.}$$

Si l'on peut simuler des échantillons suivant la loi  $\pi$  i.i.d., alors on peut effectuer l'approximation de l'intégrale.

De plus, en utilisant la loi forte des grands nombres, on note la convergence presque sûre de notre approximation vers la réelle valeur de l'intégrale. Le théorème de la limite centrale donne également une loi asymptotique normale [2].

Cette idée de pouvoir approximer une intégrale par une moyenne sur des points simulés est la clé des méthodes développées ensuite. Nous allons approximer les distributions comme des moyennes pondérées de diracs [3].

### 4.2.1 Importance sampling

Dans de nombreux cas, on ne peut pas simuler des échantillons suivant la loi  $\pi$  (loi cible). On peut alors utiliser l'*importance sampling* [2] [4], qui permet de construire une approximation en utilisant des échantillons venant d'un *proposal*, et qui assigne des poids (*importance weights*) pour corriger l'erreur par rapport à la vraie distribution. L'avantage de cette méthode est que l'on choisit un *proposal* pour lequel il est facile de simuler des points. En notant  $q$  le *proposal* on a :

$$E_{\pi}[\phi(x)] = \int \phi(x) \frac{\pi(x)}{q(x)} q(x) dx$$

On définit la fonction de poids

$$w(x) = \frac{\pi(x)}{q(x)}$$

Et on peut réécrire notre intégrale

$$E_\pi[\phi(x)] = \int \phi(x) w(x) q(x) dx = E_q[\phi(x') w(x')] \text{ où } x' \sim q(x')$$

L'estimateur de Monte Carlo est alors <sup>2</sup>

$$E_\pi[\phi(x)] \approx \frac{1}{N} \sum_{i=1}^N w^i \phi(x^i)$$

Le *proposal* doit être choisi de manière à ce que l'on puisse facilement échantillonner selon sa loi, et ne doit pas être trop différent de la loi cible. C'est-à-dire, à minima avoir une densité positive partout où c'est le cas pour la loi cible [2]. Autrement, cela signifie que certaines zones de densité non nulle pour notre densité cible seront oubliées lors de l'échantillonage.

L'estimateur est fortement consistant et converge p.s. vers la vraie valeur de l'intégrale quand  $N$  augmente vers l'infini. L'estimateur est également non biaisé et satisfait le théorème de la limite centrale, c'est à dire que sa convergence s'effectue au taux  $\frac{1}{\sqrt{N}}$

A l'issue de l'algorithme d'importance sampling, on récupère donc les poids d'importance et les échantillons de  $x$ .

Un inconvénient de l'importance sampling est qu'il nécessite de connaître la constante de normalisation de la densité.

#### 4.2.2 Self-normalized importance sampling

Lorsque l'on ne connaît pas la constante de normalisation, on peut utiliser du *Self-normalized importance sampling*. On réécrit :

$$E_\pi[\phi(x)] = \frac{\int \phi(x) w(x) q(x) dx}{\int w(x) q(x) dx} \text{ où } w(x) = \frac{\pi(x)}{q(x)}$$

ce qui donne l'estimateur

---

2. en notant  $w^i$  le poids associé à  $x^i$

$$E_{\pi}[\phi(x)] \approx \frac{1}{N} \sum_{i=1}^N \frac{w^i}{\sum_{l=1}^N w^l} \phi(x^i)$$

Cet estimateur est biaisé, mais est consistant et satisfait le théorème de la limite centrale. L'avantage de son utilisation est qu'il permet tout de même d'utiliser des densités dont on ne connaît pas la constante de normalisation.

Néanmoins, un des gros désavantages de l'*importance sampling* en général est qu'il s'avère peu pratique car il est difficile de trouver de bons *proposals*, surtout lorsque la dimension du problème croît. Etant donné que nos distributions cibles sont une séquence de distributions, nous voulons utiliser les méthodes d'*importance sampling* de manière séquentielle.

#### 4.2.3 Sequential Importance Sampling

On note  $\pi_t(x_{1:t}) = \frac{1}{Z_t} \gamma_t(x_{1:t})$  notre distribution cible, où  $\pi_t = p(x_{1:t}|y_{1:t})$ ,  $\gamma_t(x_{1:t}) = p(x_{1:t}, y_{1:t})$ , et  $Z_t = p(y_{1:t})$ . Notre problème étant séquentiel, utiliser du SIS (*Sequential Importance Sampling*) [2] est plus efficient pour générer des échantillons car il prend en compte la structure du modèle. De plus, il a une complexité computationnelle fixe à tout temps, alors que si l'on voulait simuler directement de  $\pi_t(x_{1:t})$ , on aurait une complexité au moins linéaire en t.

On sélectionne un *proposal* ayant une structure auto-régressive :

$$q_t(x_{1:t}) = q_{t-1}(x_{1:t-1})q_t(x_t|x_{1:t-1})$$

On peut alors obtenir l'échantillon  $x_{1:t}^i$  en réutilisant  $x_{1:t-1}^i$  et en lui concaténant  $x_t^i$  simulé depuis  $q_t(x_t|x_{t-1}^i)$ . On calcule les poids de manière récursive :

$$w_t(x_{1:t}) = \frac{\gamma_t(x_{1:t})}{q_t(x_{1:t})} = \frac{\gamma_{t-1}(x_{1:t-1})}{q_{t-1}(x_{1:t-1})} \frac{\gamma_t(x_{1:t})}{\gamma_{t-1}(x_{1:t-1})q_t(x_t|x_{1:t-1})}$$

c'est-à-dire,

$$w_t(x_{1:t}) = w_{t-1}(x_{1:t-1}) \frac{\gamma_t(x_{1:t})}{\gamma_{t-1}(x_{1:t-1})q_t(x_t|x_{1:t-1})}$$

Le désavantage de cette méthode est que le poids maximal s'approche de 1 quand t augmente, ce qui implique que les autres poids tendent vers 0 et que l'on va donc approcher notre distribution cible

uniquement avec une valeur<sup>3</sup>.

Pour remédier à ce problème, on introduit une étape de rééchantillonnage qui permet d'obtenir un nouveau jeu de particules équi-pondéré.

### 4.3 Monte Carlo séquentiel

Etant donné le problème de dégénérescence des poids, on aimeraient pouvoir trouver un moyen de générer des échantillons ayant des poids égaux à partir de notre échantillon ayant des poids très inégaux. On peut alors effectuer un rééchantillonage où chaque particule va avoir dans le nouvel échantillon une multiplicité proportionnelle au poids qui la représente. On tire donc des indices ancêtres de telle sorte que

$$P(A_t = i | (x_t^j, w_t^j)_{j=1}^n) = \frac{w_t^i}{\sum_l w_t^l}$$

Puis on sélectionne les particules indiquées par ces ancêtres. On obtient un nouvel échantillon où toutes les particules ont le même poids mais leur multiplicité n'est pas toujours égale à 1.

$$(x_i, w_i)_{i=1}^n \implies (x_i^{a_i}, \frac{1}{n})_{i=1}^n$$

Le Monte carlo séquentiel va être constitué d'une combinaison de SIS et de rééchantillonage<sup>4</sup>.

On va propager dans le temps une collection de  $N$  échantillons pondérés qui représentent une approximation de la densité cible.

$$\gamma_t^N(dx_{1:t}) = \sum_{i=1}^{i=N} \frac{w_t^i}{\sum_l w_t^l} \delta_{x_{1:t}^i}(dx_{1:t})$$

#### 4.3.1 Implémentation d'un algorithme de SMC

L'implémentation du SMC se base sur les étapes précédemment décrites. L'algorithme consiste donc en 4 étapes majeures [2] :

- *resample* : On choisit  $N$  particules à  $t - 1$  en conservant les plus prometteuses. C'est-à-dire que l'on tire des indices de particules par rapport aux poids qui leur sont attribués. Ces indices indiquent la multiplicité de chaque particule dans le nouvel échantillon.

---

3. Ce problème s'appelle la dégénérescence des poids

4. Le SMC est utilisé pour échantillonner séquentiellement une séquence de distributions cibles. On réserve le nom filtre à particules lorsque l'on s'intéresse aux distributions cibles  $(p(x_t|y_{1:t}))_t$  ou  $(p(x_{0:t}|y_{1:t}))_t$ .

- *propagate* : On génère de nouveaux échantillons à  $t$  conditionnellement aux parties rééchantillonées.
- *concatenate* : On concatène cette nouvelle valeur à la chaîne  $(x_1, \dots, x_{t-1})$  pour obtenir un échantillon de taille  $t$ .
- *weight* : On pondère l'échantillon pour corriger l'erreur entre le *proposal* et la cible. On normalise les poids avant de passer à l'itération suivante.

L'avantage du SMC est de prendre en compte l'information disponible des temps précédents pour construire un proposal plus pertinent pour le temps suivant.

Les sous sections suivantes présentent les algorithmes mis en place lors de mon stage.

#### 4.3.2 Bootstrap particle filter

L'algorithme de SMC le plus simple est le *Bootstrap particle filter* (BPF) [3]. On rappelle que l'on souhaite approximer la *filtering distribution*

$$p(x_t|y_{1:t}) = \frac{p(y_t|x_t)p(x_t|y_{1:t-1})}{p(y_t|y_{1:t-1})}$$

De plus, on sait que

$$p(x_t|y_{1:t-1}) = \int p(x_t|x_{t-1})p(x_{t-1}|y_{1:t-1})dx_{t-1}$$

On cherche à approximer la distribution d'intérêt par une somme pondérée de diracs. Si l'on cherche à utiliser de l'*importance sampling* pour  $p(x_t|y_{1:t})$ , il va falloir évaluer l'intégrale découlant de  $p(x_t|y_{1:t})$ , ce qui n'est pas possible analytiquement. L'intégrale peut néanmoins être approximée avec de l'IS qui utilise pour distribution cible la *filtering distribution* à  $t-1$ . L'idée est donc de partir du principe que l'on a une approximation à  $t-1$  constituée par  $(x_{t-1}^i, w_{t-1}^i)_{i=1}^N$ . On a

$$\hat{p}^N(x_{t-1}|y_{1:t-1}) = \sum_{i=1}^N w_{t-1}^i \delta_{x_{t-1}^i}(x_{t-1})$$

où  $N$  représente le nombre de particules,  $w$  les poids et  $x$  les particules. En insérant cela dans l'intégrale on obtient

$$\hat{p}^N(x_t|y_{1:t-1}) = \int p(x_t|x_{t-1})\hat{p}^N(x_{t-1}|y_{1:t-1})dx_{t-1} = \sum_{i=1}^N w_{t-1}^i p(x_t|x_{t-1}^i)$$

On peut alors évaluer

$$p(x_t|y_{1:t}) \approx \frac{p(y_t|x_t)}{p(y_t|y_{1:t-1})} \sum_{i=1}^N w_{t-1}^i p(x_t|x_{t-1}^i)$$

On choisit pour notre importance sampler une distribution similaire, c'est-à-dire

$$q(x_t|y_{1:t}) = \sum_{i=1}^N w_{t-1}^i q(x_t|x_{t-1}^i, y_t)$$

On peut générer des échantillons  $\tilde{x}_{t-1}$  en rééchantillonant parmi nos particules  $x_{t-1}^i$  selon les poids  $w_{t-1}^i$ . On obtient des particules ayant des poids égaux (car on va dupliquer ou supprimer des particules selon leur importance, encodée par les poids) et on peut approximer notre distribution selon la formule suivante

$$\hat{p}^N(x_{t-1}|y_{1:t-1}) = \frac{1}{N} \sum_{i=1}^N w_{t-1}^i \delta_{\tilde{x}_{t-1}^i}(x_{t-1})$$

On obtient donc des poids qui représentent à quel point chaque particule est utile pour l'approximation de la distribution cible, ce qui est encodé par la multiplicité des particules lors du rééchantillonage.

On peut définir la fonction de poids comme

$$w(x_t, y_t) = \frac{p(y_t|x_t) \sum_{i=1}^N w_{t-1}^i p(x_t|x_{t-1}^i)}{\sum_{j=1}^N w_{t-1}^j q(x_t|x_{t-1}^j, y_t)}$$

On peut alors évaluer et normaliser les  $w_t^i = w(x_t^i, y_t)$  pour chaque particule ( $i = 1 \dots N$ ), et on obtient un système de particules approximant  $p(x_t|y_{1:t})$ . Le calcul de la fonction de poids peut néanmoins être computationnellement coûteuse. En choisissant le proposal comme

$$q(x_t|y_{1:t}) = \hat{p}^N(x_t|y_{1:t-1}) = \sum_{i=1}^N w_{t-1}^i p(x_t|x_{t-1}^i)$$

la fonction de poids devient  $w(x_t, y_t) = p(y_t|x_t)$ , ce qui est bien plus facile à évaluer.

L'algorithme [1] [3] détaille les étapes du *bootstrap particle filter*.

Le BPF est l'algorithme le plus simple du SMC, mais, il utilise les idées principales régissant les algorithmes de SMC : l'*importance sampling* et le rééchantillonage sont utilisés de manière séquentielle pour approximer une séquence de distributions de probabilité. Les algorithmes présentés ci après sont des variantes du BPF qui ont pour but d'améliorer la précision des approximations.

---

**Algorithm 1** Bootstrap Particle Filter
 

---

- 1: **Initialisation** ( $t = 1$ ):
- 2: Echantillonner  $x_i^1 \sim \mu(x_1)$ .
- 3: Calculer  $\bar{w}_i^1 = p(y_1|x_i^1)$  et normaliser,  $w_i^1 = \frac{\bar{w}_i^1}{\sum_{j=1}^N \bar{w}_j^1}$ .
- 4: **for**  $t = 2$  to  $T$  **do**
- 5:   **Rééchantillonage** : Générer le système de particules équipondérées  $\{\bar{x}_i^{t-1}, \frac{1}{N}\}_{i=1}^N$  en rééchantillonant  $\{x_i^{t-1}, w_i^{t-1}\}_{i=1}^N$  de manière multinomiale.
- 6:   **Propagation** : Echantillonner  $x_i^t \sim p(x_t|\bar{x}_i^{t-1})$ .
- 7:   **Pondération** : Calculer  $\bar{w}_i^t = p(y_t|x_i^t)$  et normaliser,  $w_i^t = \frac{\bar{w}_i^t}{\sum_{j=1}^N \bar{w}_j^t}$ .
- 8: **end for**=0

---

#### 4.3.3 Auxiliary particle filter

Le choix de *proposal* pour le BPF est sous-optimal, car lors de l'étape de propagation, la valeur de  $y_t$  n'est pas prise en compte. En prenant en compte cette valeur, on augmente nos chances de simuler des échantillons plus utiles. En repartant du fait que l'on rééchantillonne nos particules, on peut définir des indices des 'ancêtres'  $a_{t-1}^i$  pour chaque particule i. On peut alors s'intéresser à la distribution jointe de  $(x_t, a_t)$ . On obtient [3] que

$$q(x_t, a_t | y_{1:t}) = w_{t-1}^{a_t} q(x_t | w_{t-1}^{a_t}, y_t)$$

et que

$$w(x_{t-1}, x_t, y_t) = \frac{p(y_t, x_t)p(x_t | x_{t-1})}{q(x_t | x_{t-1}, y_t)}$$

Pour l'*auxiliary particle filter*, on veut lors de l'étape de rééchantillonage prendre en compte la valeur de  $y_t$ <sup>5</sup>. On définit une fonction  $\nu$  telle que  $\nu_{t-1}^i = \nu(x_{t-1}^i, y_t)$ <sup>6</sup> (de manière similaire à une fonction de poids). Alors, on va rééchantillonner nos particules (c'est-à-dire choisir des ancêtres) tels que

$$P(A_t = i | (x_{t-1}^j, w_{t-1}^j)_{j=1..N}) = \frac{w_{t-1}^i \nu_{t-1}^i}{\sum_{l=1}^M w_{t-1}^l \nu_{t-1}^l}$$

L'idée est donc de choisir une fonction  $\nu$  qui soit large si la probabilité d'observer  $y_t$  connaissant  $x_{t-1}$  à t est grande. La fonction de poids est également modifiée car il faut prendre en compte l'effet du rééchantillonage avec  $\nu$ , donc

$$w(x_{t-1}, x_t, y_t) = \frac{p(y_t, x_t)p(x_t | x_{t-1})}{\nu(x_{t-1}, y_t)q(x_t | x_{t-1}, y_t)}$$

---

5. On prend seulement en compte  $y_{1:t-1}$

6. En choisissant  $\nu_{t-1}^i = w_{t-1}^i$  et  $q(x_t | x_{t-1}, y_t) = p(x_t | x_{t-1})$ , on retombe sur le *Bootstrap particle filter*

De nouveau, cette fonction de poids nous permet d'effectuer l'étape de propagation. Cette approche par les variables auxiliaires est surtout utile pour définir le *fully adapted auxiliary particle filter*.

#### 4.3.4 Fully adapted particle filter

En choisissant  $\nu(x_{t-1}, y_t) = p(y_t|x_{t-1})$ , tous les poids de pondération sont égaux à 1<sup>7</sup>, c'est à dire que le *fully adapted SMC sampler* correspond à un choix optimal de poids de rééchantillonage et de *proposal* avec une variance incrémentale dans les poids de 0. De plus, il permet de satisfaire la condition d'avoir une valeur large si la probabilité d'observer  $y_t$  connaissant  $x_t$  est élevée.

Il est en général difficile d'implémenter l'algorithme de manière exacte car on ne peut pas calculer  $\nu_{t-1}$  et /ou simuler selon  $q_t$ . Il est possible de calculer les *proposals* optimaux localement lorsque  $p(y_t|x_t)$  est conjugué à  $p(x_t|x_{t-1})$ .

### 4.4 En grande dimension

Les algorithmes standards présentés ci dessus souffrent de la *curse of dimensionality*, c'est-à-dire qu'il faut augmenter exponentiellement la capacité de calcul lorsque la dimension augmente pour garder des résultats pertinents. Ceci limite grandement l'application à des systèmes comportant une grande dimensionnalité. Néanmoins, il apparaît en pratique que les dépendances sont locales au niveau spatial, ce qui pousse à décomposer le processus de filtrage en plusieurs filtrages locaux de plus petite dimension que l'on va ensuite combiner [5], [6]. Cette stratégie s'illustre avec le *block particle filter* [5], le *nested SMC* [7] [2], ou encore le *divide and conquer algorithm* [5] [6]. Je détaille ci-après ceux sur lesquels j'ai travaillé.

#### 4.4.1 Divide and Conquer

L'idée de la méthode est dans le cas de distributions multivariées de séparer les variables en groupes disjoints, et pour chacun de ces groupes de définir une distribution cible. Ces distributions sont plus faciles à échantillonner et les solutions seront ensuite fusionnées pour approximer la distribution cible initiale. Pour chaque étape intermédiaire, on conserve des groupes pondérés de particules indépendants qui sont fusionnés, puis propagés comme lors de SMC standard [5].

On s'intéresse à la densité cible  $\gamma_t(x_t) = p(x_t|y_{1:t}) = g_t(y_t|x_t) \int f_t(x_t|x_{t-1})p(x_{t-1}|y_{1:t-1})dx_{t-1}$  où l'on utilise des particules pour approximer à chaque temps  $\gamma_{t-1}$  et remplacer dans l'intégrale

---

7. car on choisit également un *proposal* optimal :  $q(x_t|x_{t-1}, y_t) = p(x_t|x_{t-1}, y_t)$

$p(x_{t-1}|y_{1:t-1}) = \gamma_{t-1}(x_{t-1})$  par l'approximation. On définit un arbre pour nos séquences de dimension D, qui va permettre aux particules d'évoluer des feuilles à la racine lors de l'algorithme. Pour des dimensions pouvant se décomposer comme  $2^N$ , on peut alors créer un arbre binaire comme montré dans la figure ci dessous [5] [5].

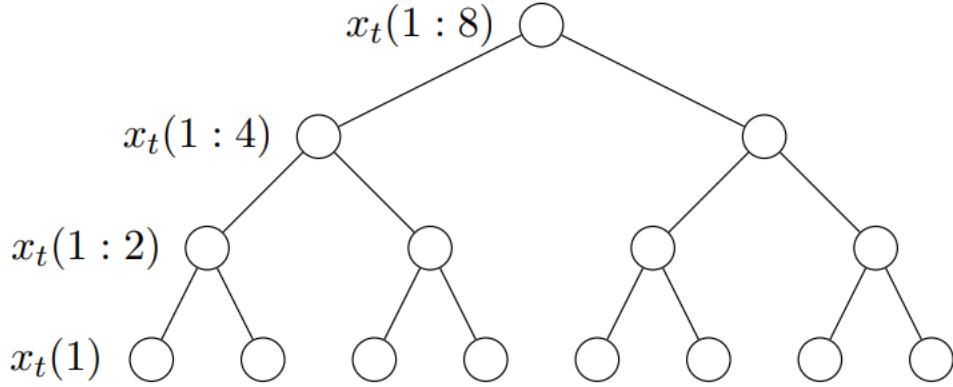


FIGURE 5 – Arbre pour  $d = 8$

Les feuilles représentent les valeurs pour chaque dimension à  $t$  (c'est à dire  $x_t(1), x_t(2), x_t(3) \dots x_t(D)$ ). Pour chaque feuille, on effectue de l'importance sampling avec des particules indépendantes les unes des autres pour obtenir une approximation pondérée. Pour chaque noeud qui n'est pas une feuille, on fusionne les particules des enfants à gauche et à droite pour obtenir une approximation de  $\gamma_{t,enfants}$ . Cette approximation n'est pas unidimensionnelle mais de la dimension de l'addition des dimensions des approximations des enfants à droite et à gauche<sup>8</sup>.

$$\gamma_{t,enfants}^N = \frac{1}{N^2} \sum_{n1} \sum_{n2} w_{t,enfant1}(z_{t,enfant1}^{n1}) w_{t,enfant2}(z_{t,enfant2}^{n2}) \delta_{(z_{t,enfant1}^{n1}, z_{t,enfant2}^{n2})}$$

On pondère ensuite ces approximations pour se ramener à  $\gamma_{t,u}$  où  $u$  fait référence au noeud considéré.

On a

$$m_{t,u}(z_{t,enfants}) = \frac{\gamma_{t,u}(z_{t,enfants})}{\gamma_{t,enfants}(z_{t,enfants})}$$

Finalement, pour chaque paire n1, n2, on obtient

$$\tilde{w}_{t,u}^{n1,n2} = w_{t,enfant1}(z_{t,enfant1}^{n1}) w_{t,enfant2}(z_{t,enfant2}^{n2}) m_{t,u}(z_{t,enfants})$$

où  $z_{t,enfants}$  représente la concaténation des approximations dans les enfants du noeud. On rééchan-

8. On effectue une approximation de la *mesure produit*

tillone ensuite nos particules selon ces poids et on leur assigne un poids égal à 1.

Il faut bien garder en tête que lorsque l'on fusionne nos particules et nos estimations, on augmente la dimension de ces dernières. On passe d'une estimation ponctuelle dans les feuilles à une estimation de dimension 2 à la première jointure, puis 4 à la deuxième... etc. De même, nos particules qui étaient uni-dimensionnelles deviennent bi-dimensionnelles, puis quadri-dimensionnelles ... etc

En effectuant cet algorithme récursif et en utilisant à chaque temps l'approximation du temps précédent, on obtient à la racine une approximation de  $\gamma_t(x_t)$ .

Pour des dimensions qui ne se décomposent pas en  $2^n$ , on peut créer un algorithme qui décompose la séquence selon un arbre presque binaire. Si l'on a des à priori sur quelles dimensions s'influencent entre elles, on peut également créer un arbre fidèle à ces relations.

#### 4.4.2 Nested sequential Monte Carlo

Le but du NSMC (*Nested sequential Monte Carlo*) [7] [2] est également d'approximer le *proposal* localement optimal dans le cas où  $x$  est de grande dimension. Pour cela, on effectue un SMC interne dans le sens spatial de  $x_t$ . Cette étape de filtrage se décompose en

- *forward filtering* : On choisit  $M$  particules internes pour chacune de nos  $N$  particules externes et on effectue un filtrage spatial pour chaque  $x_t$  en partant de  $x_{t,1}$ , puis  $x_{t,1:2} \dots$  jusqu'à  $x_{t,1:d}$  où  $d$  est la dimension. On se sert du modèle de transition pour passer de la dimension  $k$  à  $k+1$  jusqu'à arriver à la dimension  $d$ .
- *backward sampling* : L'étape précédente nous a fourni  $M$  particules pondérées approximant  $p(x_{t,1}), p(x_{t,1:2}|y_{t,1:2}) \dots$  Néanmoins, nous souhaitons conserver uniquement  $N$  particules externes, alors, on va d'une part rééchantillonner ces particules externes en fonction de leur poids, et d'autre part conserver l'état interne associé à chacune. On crée alors une nouvelle particule externe dimension par dimension en échantillonant parmi ses particules internes en fonction des poids associés à chaque dimension. C'est-à-dire que pour chaque composante, on sélectionne avec un rééchantillonage multinomial la particule la plus prometteuse et on extrait la valeur pour la composante correspondante. On remonte dans le sens décroissant (de  $D$  à 1) et pondérant les particules au fur et à mesure que l'on progresse.

Il est important de noter que l'on utilise lors de cet algorithme au total  $N \cdot M$  particules. Lors de la comparaison avec d'autres algorithmes, on donnera à tous le même budget computationnel, c'est à dire  $N \cdot M$  particules pour le BPF ou le FAPF.

## 5 Résultats

Le code permettant d'obtenir les résultats présentés dans cette section est disponible sur github : [https://github.com/candicebaud/liu\\_smc](https://github.com/candicebaud/liu_smc).

### 5.1 Implémentation sur modèles linéaires gaussiens

La première partie de mon stage s'est articulée sur la compréhension des techniques de filtre à particules expliquées plus haut. J'ai donc implémenté ces techniques théoriques dans le cas de modèles linéaires gaussiens. Ces derniers présentent l'avantage de pouvoir implémenter le filtre de Kalman qui permet d'obtenir une solution analytique optimale sur l'état de  $X$ . Ainsi, il est facile d'évaluer la précision de nos algorithmes en comparant leurs résultats avec ceux du filtre de Kalman.

#### 5.1.1 Modèle linéaire Gaussien

En dimension 1, modèle linéaire Gaussien s'écrit

$$x_t = ax_{t-1} + \omega_t \text{ où } \omega_t \sim N(0, \sigma_X^2)$$

$$y_t = bx_t + v_t \text{ où } v_t \sim N(0, \sigma_Y^2)$$

où  $a, b, \sigma_X^2, \sigma_Y^2$  sont des scalaires,  $x$  et  $y$  sont des scalaires à chaque temps  $t$ .

En dimension D arbitraire, il s'écrit de la manière suivante :

$$x_t = Ax_{t-1} + \omega_t \text{ où } \omega_t \sim N(0, Q)$$

$$y_t = Bx_t + v_t \text{ où } v_t \sim N(0, R)$$

$A, B, Q, R$  sont des matrices, et  $x$  et  $y$  sont des vecteurs à chaque temps  $t$ .

#### 5.1.2 Filtre de Kalman

Le filtre de Kalman est une méthode de filtrage utilisée pour estimer l'état d'un système dynamique à partir de mesures bruitées. Il utilise les propriétés des lois normales afin d'approximer à chaque itération la valeur de la variable latente et l'erreur de prédiction. Pour voir la dérivation complète, se référer à [3] et [8].

Pour mettre en place l'algorithme, on utilise donc deux étapes principales : l'étape de prédiction et l'étape de correction.

- Étape de prédiction :
  - . On utilise le modèle dynamique du système pour prédire l'état futur du système.
  - . On estime l'état futur et son incertitude en utilisant l'estimation précédente et la dynamique du système.
- Étape de mise à jour :
  - . On compare les nouvelles mesures à l'estimation prédictive pour calculer le résidu.
  - . On utilise le résidu, l'estimation prédictive et les matrices de covariance pour calculer le gain de Kalman.
  - . On met à jour l'estimation de l'état en combinant l'estimation prédictive avec les nouvelles mesures en utilisant le gain de Kalman.
  - . On met à jour la matrice de covariance pour refléter l'incertitude réduite de l'estimation.

Le filtre de Kalman est un filtre optimal dans le cadre d'un système linéaire gaussien, ce qui signifie qu'il fournit l'estimation la plus précise possible de l'état du système en présence de bruit gaussien.

### 5.1.3 Résultats des différents algorithmes

#### *Bootstrap Particle filter*

Dans un premier temps, j'implémente le *Bootstrap Particle filter* en dimension 1. Pour mieux comprendre comment fonctionne l'algorithme, la figure [6a] permet d'observer pour 3 particules comment celles-ci évoluent (pour  $T = 50$ ), et comment leur moyenne pondérée par les poids (en bleu) approxime les valeurs données par le filtre de Kalman (en noir). J'ai utilisé dans l'équation du modèle linéaire gaussien les paramètres suivants :

- Nombre de particules :  $N = 3$
- $a = 0.1$
- $b = 2$
- Des bruits gaussiens  $N(0, 1)$

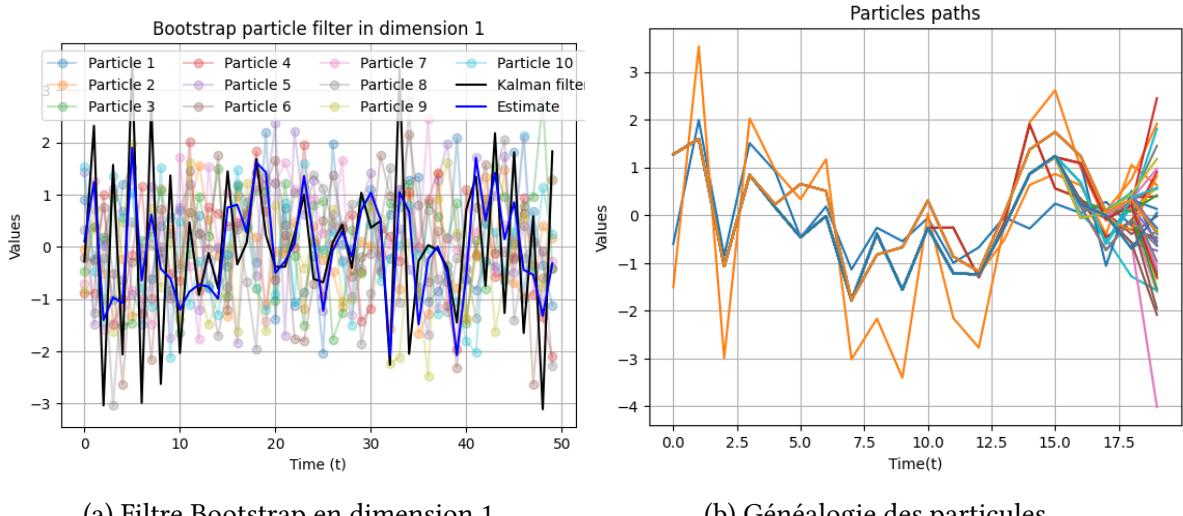


FIGURE 6 – Bootstrap particle filter - dimension 1

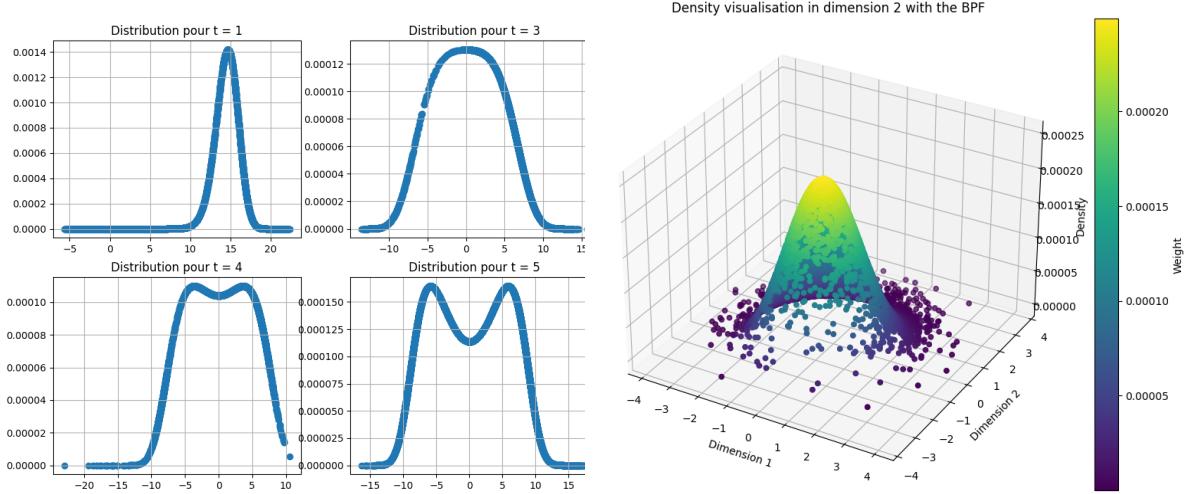
De plus, il est possible de visualiser le chemin parcouru par les particules. Comme expliqué dans la partie théorique, les particules sont sélectionnées (dupliquées ou supprimées) selon leur importance, puis on simule de nouveaux échantillons selon la loi du proposal en se basant sur les particules conservées. Il est donc possible et intéressant de tracer la généalogie des particules pour voir à partir de quoi celles-ci ont été générées. La figure [6b] montre la généalogie de 40 particules pour  $T = 20$ , avec les même paramètres pour la modélisation du processus que pour la figure [6a]. Le phénomène illustré est la dégénérescence de la généalogie. On se rend compte que nos 40 particules obtenues au temps  $T = 20$  ne proviennent en réalité au départ que de 3 particules. Le tracé de la généalogie du processus présenté en figure [6a] est disponible en annexe [27] et permet d'entrevoir que les 3 particules que l'on observe à la fin proviennent toutes de la même particule. Le tracé est effectué en remontant dans le temps : c'est à dire que l'on regarde d'où proviennent les particules à  $T = 20$ , puis on retrace la généalogie en remontant petit à petit. Lors de la dernière étape de sélection, une des particules avait un poids tellement important qu'elle a évincé les autres. Ainsi toutes les particules proviennent de cette particule et donc du même ancêtre. La morale est que si l'on prend trop peu de particules, on se retrouve très rapidement avec une seule particule 'plausible' qui élimine les autres lors de la sélection. En prenant 40 particules, on observe déjà une très forte éviction [6b].

Enfin, il est intéressant de visualiser les lois  $p(x_t|y_{1:t}) \forall t$ . Je simule cette fois ci un modèle [3] qui n'est pas linéaire :

$$x_1 \sim N(0, 1)$$

$$x_{t+1} = 0.5x_t + 25 \frac{x_t}{1+x_t^2} + 8\cos(1.2t) + v_t \text{ où } v_t \sim N(0, 0.5)$$

$$y_t = \frac{1}{20}x_t^2 + e_t \text{ où } e_t \sim N(0, 0.5)$$



(a) Distributions de  $p(x_t|y_{1:t})$  obtenues avec le BPF en dimension 1  
(b)  $p(x_1|y_1)$  en dimension 2, modèle linéaire gaussien

FIGURE 7 – Bootstrap particle filter - densités

Les résultats présentés sur la figure [7a] permettent de visualiser les distributions pour différentes valeurs de  $t$  du modèle non linéaire. La figure [7b] montre elle la densité obtenue en dimension 2 avec un modèle linéaire gaussien. On peut également visualiser la distribution obtenue avec le BPF par rapport à celle obtenue avec le filtre de Kalman<sup>9</sup>. Ces résultats sont disponibles en annexe [35a, 35b]

#### *Bootstrap Particle filter vs Kalman filter*

Comme expliqué précédemment, dans le cas de modèles linéaires gaussiens, le filtre de Kalman donne la meilleure approximation possible. Le *Bootstrap particle filter* est censé converger vers le filtre de Kalman lorsque l'on augmente le nombre de particules. Intuitivement, avec un nombre de particules infini, on est capable de simuler toutes les particules possibles, et donc d'avoir une infinité de diracs pondérés qui permettent de connaître le processus, et sa densité 'en tout point'. Ainsi, en ayant suffisamment de particules, on peut approximer idéalement notre processus, tout en gardant en tête que l'on a du bruit sur nos observations. On va donc converger vers la meilleure approximation possible avec du bruit qui correspond au filtre de kalman.

La figure [8a] permet de visualiser la performance de l'algorithme pour un nombre de particules allant de 1 à 100 en moyennant sur 10 rounds pour chaque nombre de particules. On évalue la différence entre le filtre de Kalman et le BPF avec le RMSE (*root mean square error*). Celle-ci est décroissante au taux  $\frac{1}{N}$  [3], [2].

9. Le filtre de Kalman donne la moyenne et la matrice de covariance de la loi gaussienne suivie par les observations.

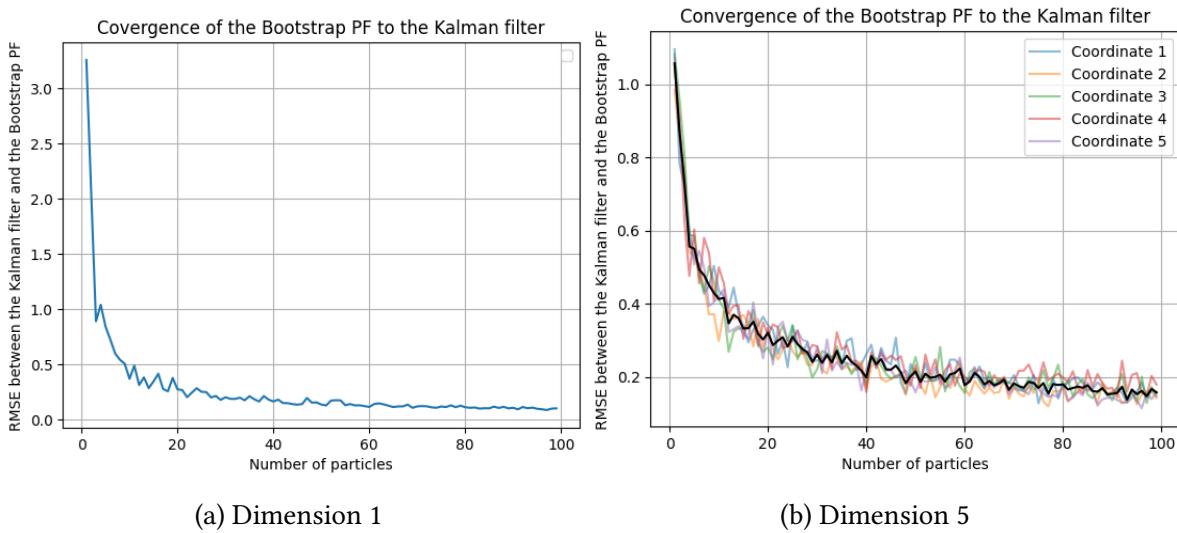


FIGURE 8 – Performances du Bootstrap PF comparativement au filtre de Kalman

Il est également possible de visualiser les résultats en plus grande dimension. J'ai représenté [8b] pour une dimension 5 le RMSE entre la solution donnée par le filtre de Kalman. Pour chaque coordonnée, j'ai calculé le RMSE entre le BPF et le filtre de Kalman. La courbe noire, elle, représente le RMSE moyen entre les coordonnées. On observe à nouveau une tendance décroissante qui montre la convergence du BPF vers la solution optimale donnée par le filtre de Kalman.

Enfin, la convergence en  $\frac{1}{N}$  est confirmée en traçant ces graphes en échelle log-log<sup>10</sup>. Ces graphes sont disponibles en annexe [28, 29].

Fully adapted particle filter

De la même manière que pour le *Bootstrap particle filter*, on peut tracer l'évolution des particules, les performances et l'écart avec la solution donnée par le filtre de Kalman. Les algorithmes ne diffèrent pas dans leur structure, mais seulement dans une optimisation de l'information disponible.

## *Différents niveaux de bruit*

Les résultats présentés précédemment couvraient des processus avec des termes d'erreur gaussiens centrés, de variance 1<sup>11</sup>. Ici, on s'intéresse à ce qu'il se passe lorsque l'on modifie les niveaux de bruit dans nos simulations.

En prenant une forte variance dans l'équation régissant l'évolution de  $x$ , on s'attend à simuler des particules diffuses lors de l'étape de propagation. En prenant une forte variance dans l'équation

10. C'est à dire  $\log(\text{RMSE})$  en fonction de  $\log(N)$

11. En dimension D, on prend le vecteur nul et la matrice identité

régissant l'évolution de  $y$ , on part du principe que notre erreur de mesure est élevée, ce qui va influencer la manière dont on pondère nos résultats. Le *fully adapted particle filter* utilise la mesure de  $y$  et l'incorpore lors de la sélection de particules. Ainsi, on s'attend à ce qu'en ayant une mesure plus précise de  $y$ , le filtre présente une bien meilleure précision que le *Bootstrap particle filter*.

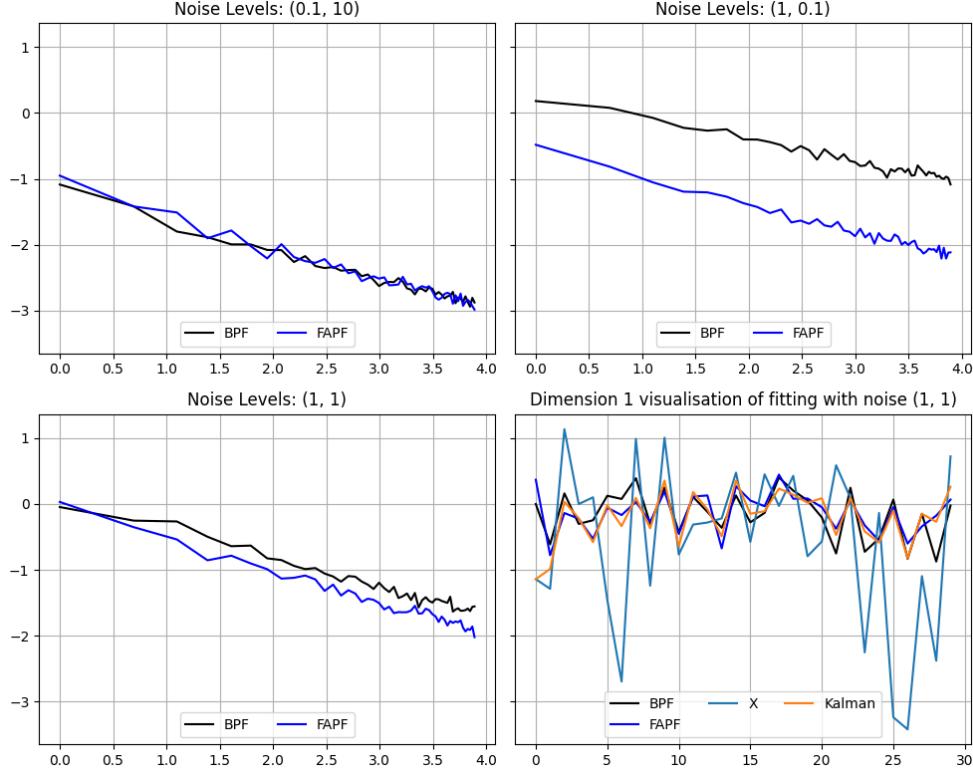


FIGURE 9 – RMSE pour différents niveaux de bruit et fitting

La figure [9] présente les résultats. Le modèle simulé est de dimension 5, avec  $A = 0.2I_d$ ,  $B = 0.4I_d$ ,  $T = 30$ . Le niveau de bruit est multiplié par la matrice identité et est donné de cette manière (niveau de bruit pour  $x$ , niveau de bruit pour  $y$ ). Les RMSE sont tracés en échelle log-log, ce qui explique les valeurs négatives. Cela permet de visualiser la vitesse de convergence des algorithmes. Le panneau en bas à droite présente un *fitting* des données.

Les points importants à noter sont les suivants. Lorsque l'erreur de mesure est très faible, la convergence vers la solution donnée par le filtre de Kalman est bien meilleure pour le *Fully adapted particle filter*. Lorsque l'erreur de mesure est assez importante, les deux algorithmes ont des performances similaires.

Ces résultats sont intuitifs étant donné que le *Fully adapted particle filter* est une amélioration du BPF car il prend en compte l'information de mesure pour ajuster la sélection de particules. Si l'information est de mauvaise qualité, la sélection n'est pas améliorée.

### Effective sample size

Les algorithmes présentés précédemment suivaient les étapes classiques d'un algorithme de SMC [1] [3], [2] c'est-à-dire, rééchantillonage, propagation et pondération. Comme observé sur les figures [6] représentant les généralogies, on peut se retrouver, selon les paramètres dans des cas où toutes nos particules sont issues du même ancêtre, ce que l'on ne souhaite pas. On peut définir l'ESS (*Effective sample size*) qui permet de voir si les poids sont dégénérés ou non :

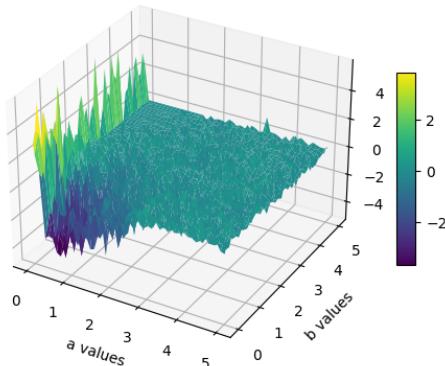
$$ESS_t = \frac{1}{\sum_{i=1}^N (w_t^i)^2}$$

Si tous les poids sont égaux à  $\frac{1}{N}$ , l'ESS vaut  $N$ . Si tous les poids sont égaux à 0 sauf un qui vaut 1, l'ESS vaut 1. On peut alors implémenter un rééchantillonage adaptatif, c'est-à-dire, que l'on ne va rééchantillonner que lorsque l'ESS est inférieur à un seuil, par exemple  $\frac{N}{2}$ . En effectuant cette vérification sur les poids, on diminue le risque de se retrouver avec une généralogie unique.

On s'intéresse donc au nombre d'ancêtres pour des paramètres donnés et un nombre de particules. L'algorithme du BPF contient de l'aléatoire lors de l'étape de propagation qui tire des échantillons selon la loi  $p(x_t|x_{t-1})$  et donc il convient de comparer le nombre moyen d'ancêtres sur plusieurs simulations. On choisit 10 *rounds* et l'on teste plusieurs paramétrages. Il apparaît sans surprise qu'en augmentant le nombre de particules, on augmente le nombre d'ancêtres [32].

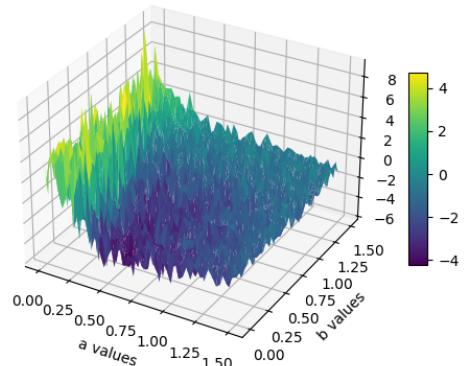
De plus, dans la majeure partie des cas, le BPF adaptatif permet de conserver une plus grande diversité de particules, mais pour certaines valeurs de  $a, b$ , et des variances des processus, le BPF classique peut être légèrement plus avantageux.

Adaptative BPF vs Classical BPF, N = 100, T = 15, sdX = 1, sdY = 1



(a)  $a, b$  évoluant entre 0 et 5

Adaptative BPF vs Classical BPF, N = 100, T = 15, sdX = 1, sdY = 1



(b) Zoom entre 0 et 1.5

FIGURE 10 – BPF adaptatif vs BPF classique

Pour des variances unitaires, on trace la différence de nombre d'ancêtres entre le BPF adaptatif et le

BPF classique [10]. Pour  $a$  et  $b$  entre 0 et 5, on se retrouve majoritairement au dessus de la surface d'équation  $z = 0$  ce qui signifie que le BPF adaptatif permet d'obtenir plus d'ancêtres que le BPF classique.

En modifiant les variances des processus, on obtient également des graphes très différents qui sont disponibles en annexe [33, 34]. Lorsque la variance du processus mesuré est faible, le BPF adaptatif est systématiquement et significativement meilleur que le BPF classique. Lorsque la variance du processus latent est faible, le BPF adaptatif est à nouveau majoritairement meilleur que le BPF classique surtout pour de faibles valeurs de  $b$ .

### Divide and Conquer

Pour remédier à la *curse of dimensionality*<sup>12</sup> [7][36], on implémente l'algorithme DAC. Cette méthode pour rappel divise en arbres nos séquences pour décomposer le problème en sous-problèmes de petite dimension. On utilise toujours le RMSE comme mesure. Les résultats obtenus montrent une diminution du RMSE en  $\frac{1}{N}$  lorsque l'on augmente le nombre de particules quelque soient les autres paramètres. Les performances du DAC ne sont toutefois pas systématiquement meilleures que celles du BPF et dépendent fortement du niveau de bruit, ainsi que de l'évolution du système (A,B).

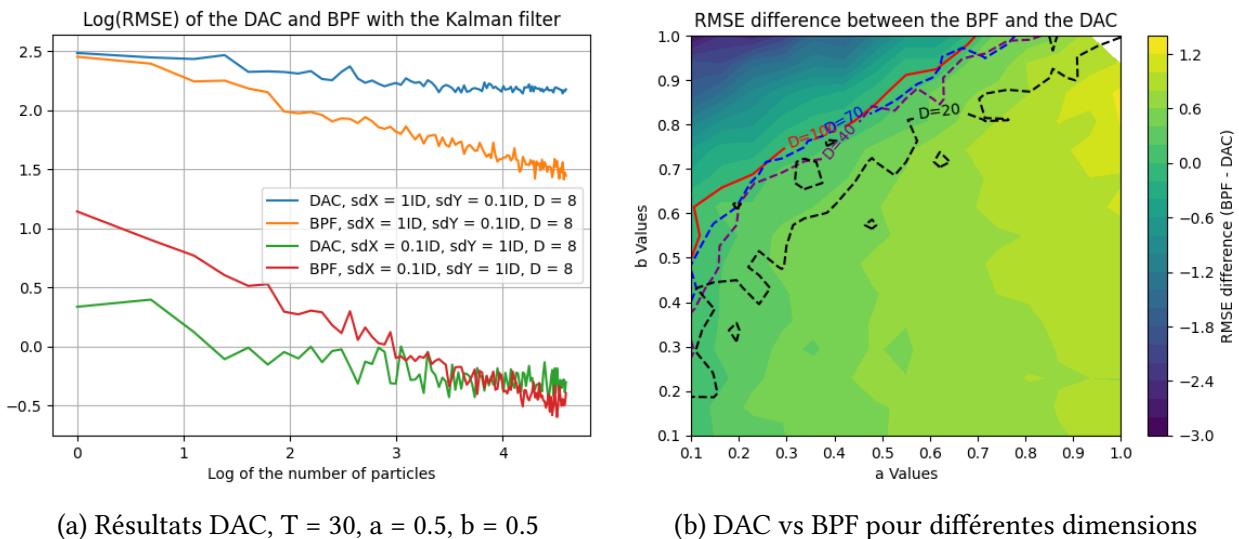


FIGURE 11 – Résultats DAC

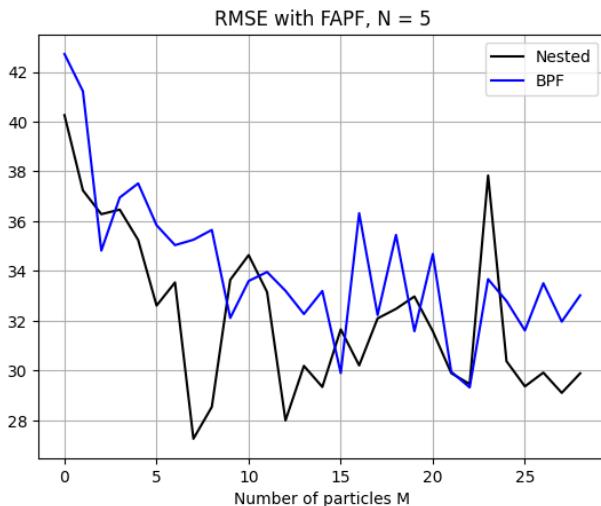
La figure [11a] permet d'observer la convergence vers le filtre de Kalman lorsque le nombre de particules augmente. Elle permet également de voir que pour différents paramètres régissant le modèle, la convergence est plus ou moins rapide, et également que le DAC n'est pas toujours meilleur que le

12. Lorsque la dimension augmente, il faudrait augmenter le nombre de particules de manière exponentielle pour conserver une précision comparable.

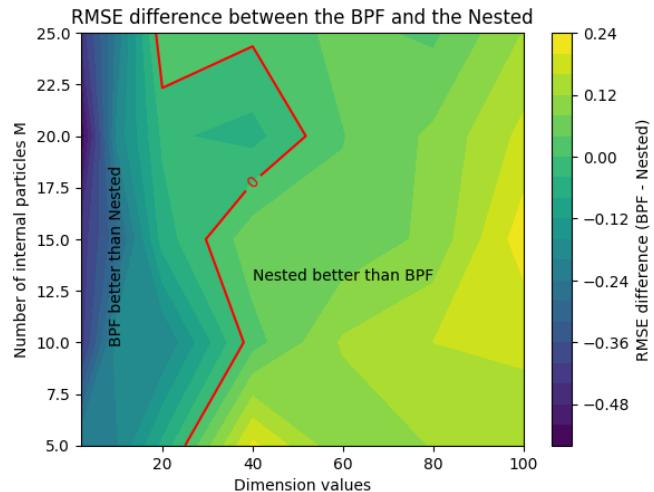
BPF. La figure [11b]<sup>13</sup> permet de visualiser ces relations avec les paramètres des matrices, et permet également d'observer une baisse de l'efficacité du BPF comparativement au DAC alors que la dimension augmente. On observe le même phénomène en modifiant les variances des processus X et Y. Selon le modèle, le BPF peut s'avérer plus avantageux mais quoiqu'il arrive, lorsque la dimension augmente, le DAC gagne en performance comparativement au BPF. De plus amples graphiques sont disponibles en annexe [39, 40, 41, 37, 38].

### Nested SMC

Une deuxième approche en grande dimension est le nested SMC. Dans les algorithmes les plus simples (BPF et FAPF), on ne prenait en compte que l'évolution temporelle des vecteurs complets. Le *Nested SMC* décompose le problème spatio-temporel en un problème spatial pour chaque temps. Il permet donc de gagner en efficacité quand la dimension augmente, malgré un temps de computation extrêmement important. En gardant des matrices diagonales<sup>14</sup>, on observe alors que le *Nested SMC* et le BPF ont des performances similaires en petite dimension mais que le *Nested SMC* se révèle bien meilleur quand la dimension augmente. Pour une dimension inférieure à 10 et un nombre de particules internes élevé, le BPF est plus performant et plus rapide.



(a) Résultats pour  $D = 40$ ,  $a = 0.2$ ,  $b = 1.1$



(b) Nested vs BPF

FIGURE 12 – Résultats Nested SMC

La figure [12a] montre la convergence du BPF ainsi que du *Nested SMC* lorsque le nombre de particules internes augmente. Les algorithmes ont le même budget computationnel, c'est à dire,  $N \cdot m$  particules pour le BPF pour chaque nombre  $m$  de particules internes du *Nested SMC*. En dimension

13. Les paramètres considérés sont  $N = 30$ ,  $T = 10$ , variances unitaires des processus

14. C'est-à-dire pas de dépendances spatiales

40, on observe que le *Nested SMC* a de meilleurs résultats que le BPF en moyenne, et que les deux quantités convergent vers le FAPF. La figure [12b] montre l'amélioration apportée par le *Nested SMC* lorsque la dimension augmente et lorsque le nombre de particules augmente. Cette figure confirme l'apport du *Nested SMC* comparativement au BPF. Même avec peu de particules, lorsque la dimension augmente, le *Nested SMC* devient rapidement meilleur que le BPF [43, 44]. Il est également possible de visualiser les dépendances au modèle en annexe [42].

### *Nested vs DAC*

Le but d'implémenter des algorithmes comme le *Nested SMC* et le DAC est d'améliorer la précision en grande dimension. Nous avons vu dans les sections précédentes que le BPF est battu par le DAC et le *Nested SMC* dans la plupart des cas. Mais alors, qui choisir entre le *Nested SMC* et le DAC ? Deux aspects importants sont à prendre en compte : la précision et le temps de calcul. La figure [13] montre que le DAC bat le *Nested SMC* en terme de précision pour les dimensions et nombres de particules considérées. Le temps de calcul est également plus faible pour le DAC bien qu'il soit conséquent. Celui-ci présente donc plus d'avantages dans le cas linéaire gaussien.

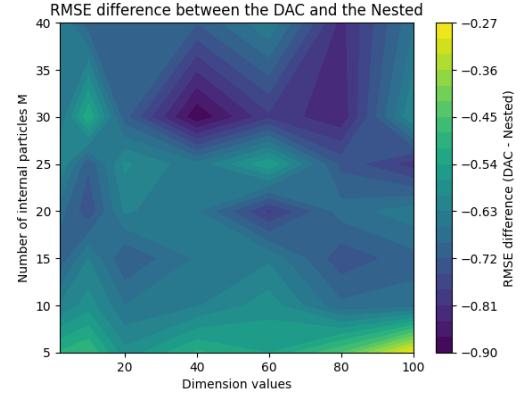


FIGURE 13 – Nested vs DAC

## 5.2 Application aux modèles météorologiques

### 5.2.1 Modèle de Lorenz96

Le modèle Lorenz96 est un modèle météorologique ayant été conçu pour capturer les aspects essentiels de la dynamique des fluides atmosphériques de manière simplifiée, tout en conservant les interactions complexes entre les variables. Un certain nombre de variables (représentant des points spatiaux sur une grille) interagissent les unes avec les autres. Chaque variable évolue au fil du temps en fonction de l'influence des autres variables, créant ainsi un réseau complexe de connexions. Cette interaction peut donner lieu à des comportements chaotiques, où de petites variations initiales peuvent conduire à des résultats radicalement différents à long terme. Cela illustre le fameux "effet papillon", où de petits changements peuvent avoir des conséquences majeures.

Le modèle déterministe s'écrit ainsi :

$$\begin{aligned}\frac{dx_{t+1}(i)}{dt} &= (x_t(i+1) - x_t(i-2))x_t(i-1) - x_t(i) + F \\ x_t(-1) &= x_t(D-1), \quad x_t(0) = x_t(D), \quad x_t(D+1) = x_t(1) \\ x_0 &\sim N(F \cdot I_D, \sigma^2 \cdot I_D)\end{aligned}$$

$F$  représente le 'forcing', c'est une constante. Pour appliquer des algorithmes de filtrage avec particules, il est nécessaire d'avoir des processus stochastiques. On ajoute un bruit gaussien à nos équations et on définit  $y$  le processus que l'on observe :

$$\begin{aligned}x_{t+1} &= f(x_t) + v_t, \quad v_t \sim N(0, \sigma^2 I_d) \\ y_t &= Hx_t + u_t, \quad u_t \sim N(0, \delta^2 I_d)\end{aligned}$$

où  $f$  représente l'évolution déterministe montrée plus haut.

### 5.2.2 Ensemble Kalman filter

Le modèle de Lorenz96 n'est pas linéaire, il n'est donc pas possible d'implémenter le filtre de Kalman traditionnel. Néanmoins, il est possible de linéariser l'équation d'évolution de  $x$  en utilisant un développement de Taylor à l'ordre 1 [9]. On peut écrire

$$f(x_t) \approx f(\hat{x}_{t|t}) + \left. \frac{\partial f(x)}{\partial x} \right|_{x=\hat{x}_{t|t}} (x_t - \hat{x}_{t|t})$$

Et réécrire  $F_t = \left. \frac{\partial f(x)}{\partial x} \right|_{x=\hat{x}_{t|t}}$ , permettant d'écrire finalement

$$x_{t+1} = f(\hat{x}_{t|t}) - F_t \hat{x}_{t|t} + F_t x_t + v_t$$

On se retrouve alors dans le cas d'un modèle linéaire gaussien en  $x_t$ . On obtient alors le *Extended Kalman filter* [26]. Ce filtre n'est pas une solution analytique mais consiste en une autre manière d'effectuer nos approximations. Nous allons le comparer aux filtres à particules en prenant comme indicateur le RMSE avec les vraies valeurs du processus X. Ceci permet d'apprécier la capacité à prévoir les valeurs de X, ce qui est ce que l'on cherche à faire en météorologie.

### 5.2.3 Différents algorithmes considérés et calculs

#### *Bootstrap particle filter*

Dans un premier temps, on peut se pencher sur le BPF [1] qui reste l'algorithme le plus basique. Les calculs pour les étapes importantes sont :

- Propagation :  $p(x_t|x_{t-1}^i) = N(f(x_{t-1}^i), \sigma^2 I)$
- Pondération :  $p(y_t|x_t^i) = N(Hx_t^i, \delta^2 I_d)$

#### *Fully adapted particle filter*

Dans un second temps, on peut s'intéresser au FAPF qui consiste en un APF [24] avec un choix spécifique de  $q$  et de  $\nu$  :  $q(x_t|x_{t-1}, y_t) = p(x_t|x_{t-1}, y_t)$ ,  $\nu(x_{t-1}, y_t) = p(y_t|x_{t-1})$ . Dans le détail, on a en utilisant la formule de Bayes

$$p(x_t|x_{t-1}, y_t) = \frac{p(y_t|x_t)p(x_t|x_{t-1})}{p(y_t|x_{t-1})}$$

De plus,

$$p(x_t|x_{t-1}) = N(x_t|f(x_{t-1}), \sigma^2 I_d) \quad p(y_t|x_t) = N(Hx_t, \delta^2 I_d)$$

On obtient directement<sup>15</sup>

$$p(y_t|x_{t-1}) = N(y_t|Hf(x_{t-1}), H\sigma^2 H^T + \delta^2 I_d)$$

Finalement, on a<sup>16</sup>

$$p(x_t|x_{t-1}, y_t) = N(x_t|\hat{x}_t, \hat{P}_t)$$

$$\hat{x}_t = f(x_{t-1}) + K_t(y_t - Hf(x_{t-1})) \quad \hat{P}_t = Q - K_t H Q \quad K_t = Q H^T (H Q H^T + R)^{-1}$$

#### *Divide-and-Conquer*

Enfin, j'ai pu implémenter la méthode *DAC* dans le cadre spécifique du modèle de Lorenz.

- **Initialisation** : Comme expliqué, pour générer le modèle de Lorenz avec du bruit, on initialise les particules suivant une loi  $N(F \cdot I_D, \sigma^2 \cdot I_D)$ . On fait de même pour la génération de particules

15. en remarquant que  $y_t = Hx_t + u_t = H(f(x_{t-1}) + v_{t-1}) + u_t$

16. avec  $Q$  et  $R$  qui représentent les matrices de covariance des processus  $x$  et  $y$

dans les feuilles à l'initialisation. On calcule les poids en remontant l'arbre pour obtenir une approximation de  $\gamma_0(1 : D)$

- **Itération** : On propage nos anciennes particules selon la définition du modèle de Lorenz pour obtenir  $f(x_{t-1}^i)$ , puis l'on échantillonne selon  $N(f(x_{t-1}^i)_u, \sigma^2)$  à chaque feuille u. On calcule les poids dans les feuilles selon la formule  $w_{t,u}^i = p(y_{t,u}|z_{t,u}^i) = N(y_{t,u}|[Hz_t]_u, \sigma^2 I_d)$  où  $z_{t,u}^i$  correspond à l'échantillon venant de la particule i. On peut alors remonter dans l'arbre et calculer  $m_{t,u} = \frac{p(y_{t,Cu}|z_{t,Cu})}{p(y_{t,l(u)}|z_{t,l(u)})p(y_{t,r(u)}|z_{t,r(u)})} * \frac{\sum_n p(z_{t,Cu}|z_{t-1,Cu}^n)}{\sum_n p(z_{t,l(u)}|z_{t-1,l(u)}^n) \sum_n p(z_{t,r(u)}|z_{t-1,r(u)}^n)}$ . Quand on a remonté l'arbre, on rééchantillonne nos particules en fonction des poids et on passe au temps suivant.

Explicitement, les quantités sont les suivantes :

$$\begin{aligned} p(y_{t,Cu}|z_{t,Cu}) &= N(y_{t,Cu}|[Hz_t]_{Cu}, \delta^2 I) & p(y_{t,l(u)}|z_{t,l(u)}) &= N(y_{t,l(u)}|[Hz_t]_{l(u)}, \delta^2 I) \\ p(z_{t,Cu}|z_{t-1,Cu}^n) &= N(z_{t,Cu}|[f(z_{t-1}^n)]_{Cu}, \sigma^2 I) & p(z_{t,l(u)}|z_{t-1,l(u)}^n) &= N(z_{t,l(u)}|[f(z_{t-1}^n)]_{l(u)}, \sigma^2 I) \end{aligned}$$

$z_t$  correspond aux particules générées à t, Cu aux enfants du noeud u, r(u) aux enfants à droite et l(u) aux enfants à gauche.

#### 5.2.4 Résultats

La figure [14] présente les RMSE des différents filtres avec les vraies valeurs de X. Les résultats présentent les RMSE des filtres pour un nombre de particules allant de 20 à 1000, avec un pas d'échantillonnage de 10. Il apparaît que les RMSE du BPF [45] et du DAC [46] sont décroissants à des taux respectifs de  $N^{-0.05}$  et  $N^{-0.2}$ , tandis que le FAPF a une précision qui semble constante à partir de 200 particules. Un zoom sur le nombre de particules allant uniquement jusqu'à 150 est disponible en annexe [47]. Le DAC s'avère être de loin le meilleur algorithme en terme de précision suivi de l'EKF.

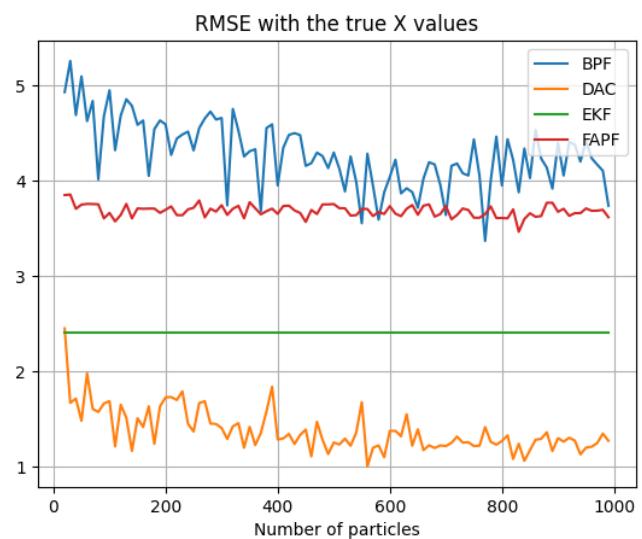


FIGURE 14 – Précision des filtres pour le modèle de Lorenz

## 6 Conclusion

En conclusion, les filtres à particules sont particulièrement utiles pour trouver la loi d'un processus à partir de modèles spatio-temporels. Pour des problèmes de petite dimensionnalité, les algorithmes de base ont des performances satisfaisantes et ne subissent que très peu la dégénérescence des poids. En revanche, lorsque la dimension augmente, il faut innover pour trouver une manière de conserver des résultats précis.

Un principe en mathématiques est de diviser un problème trop grand dimensionnellement en plein de problèmes de plus petite dimension et ensuite de trouver une manière de regrouper les solutions. C'est exactement ce que font le *Nested SMC* et le *Divide and Conquer SMC*. Dans le premier cas, on filtre en premier lieu nos observations dans la dimension spatiale avec un premier jeu de particules, puis on filtre dans la dimension temporelle avec un deuxième jeu de particules issu du premier. Dans le second cas, on divise nos distributions en arbres presque binaires et l'on approxime des valeurs unidimensionnelles qu'on regroupe ensuite en répondant.

Les performances de ces algorithmes dépendent grandement du problème considéré, des dépendances spatio-temporelles, et de la connaissance des lois marginales. Dans des modèles linéaires gaussiens, nous pouvons calculer des solutions analytiques ce qui permet de facilement évaluer nos approximations. Le DAC et le *Nested SMC* apportent alors une réelle amélioration face au BPF, notamment quand la dimension est importante. Dans le cas du modèle de Lorenz, nous avons choisi une discrétisation temporelle de taille 1 ce qui permet de retrouver des modèles gaussiens et donc de comparer nos résultats à l'EKF. Nous observons alors que le DAC est le plus performant des trois algorithmes implémentés, suivi du FAPF, puis du BPF. Ceci est d'autant plus vrai que la dimension du problème augmente.

L'extension directe de ce projet est la mise en place de ces algorithmes sur les vraies données météorologiques grâce à des centres de supercomputation comme le NSC (*National Supercomputer Center*) présent à l'université de Linköping et mis à disposition du SMHI. Au vu des temps de computation pour les différents algorithmes, et du volume de données à traiter, il est nécessaire d'utiliser ces centres de computation.

## 7 Bonus : Effectuer un stage à l'étranger

### 7.1 Géographie

J'ai effectué durant quatre mois mon stage à LiU, l'université de Linköping dans le comté d'Östergötland. La carte permet de visualiser la proximité avec Norrköping, et Stockholm, villes dans lesquelles l'université comporte des campus secondaires.

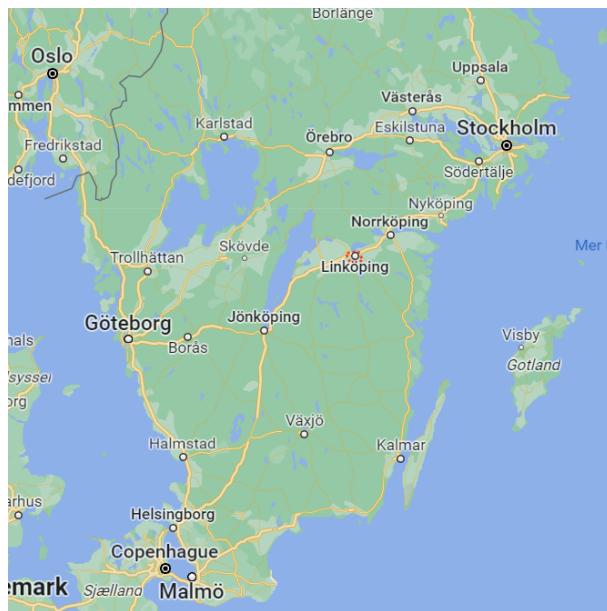


FIGURE 15 – Situation géographique de Linköping

Stockholm, Göteborg et Malmö sont les trois plus grandes villes de Suède en terme de population, avec respectivement 1 252 020, 510 491 et 258 020 habitants ce qui indique une forte concentration de la population suédoise dans le tiers le plus au sud du pays.

Les pays frontaliers de la Suède sont le Danemark au Sud, la Norvège à l'Ouest, et la Finlande à l'Est. La Suède est le plus peuplé des pays scandinaves avec 10 300 000 habitants, comparé à 5 à 6 millions pour le Danemark, la Norvège et la Finlande.

D'un point de vue architectural et urbain, les villes suédoises sont plus étendues et les bâtiments plus bas qu'en France. Pour un même nombre d'habitants, la surface d'une ville suédoise est bien plus grande car les villes comportent de nombreux parcs et forêts en leur sein<sup>17</sup>. Les maisons traditionnelles sont construites en bois et peintes avec des couleurs vives et chaleureuses.

17. En l'occurrence, pour aller à l'université, je traversais une réserve naturelle.

### 7.1.1 Quelques photos de villes scandinaves

*Linköping*



(a) *Stortorget* : la grande place



(b) Au bord de la rivière



(a) *Gamla Linköping* : la vieille ville



(b) *Domkyrka* : la cathédrale de Linköping

*Stockholm, la capitale*



(a) *Gamla stan*



(b) *Fjäderholmarna* dans l'archipel

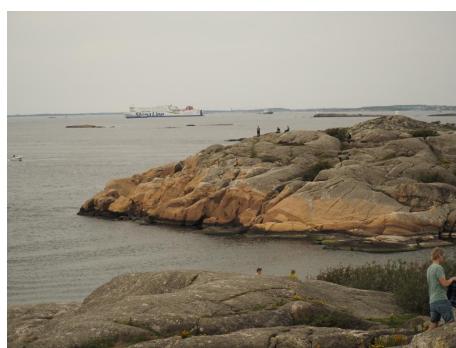
(a) *Stadshuset* : la mairie

(b) Dans le centre ville

### *Göteborg*

(a) *Haga*, le quartier branché dans la vieille ville

(b) Un lac à quelques minutes de la ville



(a) L'archipel



(b) Le jardin aux nénuphars

## 7.2 Gastronomie

Si la France est connue et se revendique comme le pays de la gastronomie, la Suède n'est pas en reste avec de nombreuses spécialités locales et coutumes.

### 7.2.1 Le fika

Le *fika* désigne deux moments dans la journée, respectivement à 10h le matin et à 15h l'après-midi. C'est une pause durant laquelle on retrouve ses collègues dans la *fikarum* afin de prendre un thé, un café, et une petite friandise. En ville, de nombreuses *konditori* et salons de thé proposent également un *fika*. Le *fika* est une véritable institution suédoise, durant laquelle tout le monde se met en pause au même moment pour couper avec la journée de travail. Les pâtisseries les plus courantes sont les *kanellbullar*, des roulés à la cannelle, et leurs variantes tels que les *kanelflätör* (tresses à la cannelle), ou encore les *kardemummabullar* à la cardamome ; les *kokosbollar* (boules au cacao saupoudrées de coco) ou encore les *punschrulle* (sorte de marzipan avec du chocolat).



FIGURE 22 – Kanellbulle

En hiver, les suédois sont également adeptes du *gröt*, qui consiste en une bouillie au lait épicé, sur lequel on rajoute du lait froid et du miel, du sucre, ou de la confiture, en particulier d'airelles. Le *gröt* se prépare habituellement avec de l'avoine (*havregrynsgröt*), mais peut également se préparer avec du riz (*risgrynsgröt*, ressemblant au riz au lait) pour de plus grandes occasions, par exemple à Noël.

### 7.2.2 Spécialités salées atypiques

Dans les spécialités salées atypiques, on peut mentionner le fameux *surstromming*, hareng mariné pendant de longs mois, surtout mangé dans le Nord, où il est pêché. Le hareng (*sill*) et le saumon (*lax*) sont globalement les poissons les plus populaires, et l'on trouve des saumons entiers fumés à l'ancienne dans de nombreuses poissonneries. Les Suédois mangent assez léger le soir en effectuant

des *mackor* (ou *smörgås*) qui consistent en des sandwichs avec du pain, des tomates, des concombres, du fromage, et souvent des préparations de poisson à la place du beurre comme le *räkost*.



FIGURE 23 – Räkost (gauche) et smörgåstårta (droite)

Pour les repas en famille, on peut également acheter un *smörgåstårta*, qui consiste en un 'gâteau en sandwich', contenant du saumon, de l'aneth, des tomates, des concombres et de la mayonnaise dans la majorité, mais pouvant également contenir d'autres garnitures telles que le jambon, des crevettes, des oeufs etc.

Du côté de la viande, les boulettes (*köttbullar*) accompagnées de pommes de terre restent le plat traditionnel le plus courant ; mais l'on trouve également de nombreuses variantes de saucisses (*korv*), ainsi que des plat à base de viande de renne ou d'élan.

## 8 Annexes

### 8.1 Algorithmes

---

**Algorithm 5.2:** Auxiliary particle filter (APF)

---

```

1 Initialization ( $t = 1$ ):
2   Sample  $x_1^i \sim q(x_1 | y_1)$ .
3   Compute the importance weights  $\bar{w}_1^i = p(y_1 | x_1^i) \mu(x_1^i) / q(x_1^i | y_1)$  and
      normalize,  $w_1^i = \bar{w}_1^i / \sum_{j=1}^N \bar{w}_1^j$ .
4 for  $t = 2$  to  $T$  do
5   Compute the adjustment multipliers  $\nu_{t-1}^i = \nu(x_{t-1}^i, y_t)$ 
6   Resampling: Resample  $\{x_{t-1}^i\}_{i=1}^N$  with probabilities proportional to
       $\{w_{t-1}^i \nu_{t-1}^i\}_{i=1}^N$  to generate the equally weighted particle system
       $\{\bar{x}_{t-1}^i, 1/N\}_{i=1}^N$ .
7   Propagation: Sample  $x_t^i \sim q(x_t | \bar{x}_{t-1}^i, y_t)$ .
8   Weighting: Compute  $\bar{w}_t^i = \omega(\bar{x}_{t-1}^i, x_t^i, y_t)$  and normalize,  $w_t^i = \bar{w}_t^i / \sum_{j=1}^N \bar{w}_t^j$ .
9 end
```

---

FIGURE 24 – APF, extrait de [3]

---

**Algorithm 2** dac\_smc( $t$ ) for  $t \geq 1$ . Given  $(\{z_{t-1,\mathfrak{R}}^n\}_{n=1}^N) := \text{dac\_smc}(t-1)$ .

---

```

1: for  $u$  leaf node do
2:   Initialize: draw  $z_{t,u}^n \sim N^{-1} \sum_{n=1}^N K_{t,u}(z_{t-1,\mathfrak{R}}^n, \cdot)$  and compute  $w_{t,u}^n$  as in (8) for all
       $n \leq N$ .
3: end for
4: for  $u$  non-leaf node do
5:   Recurse: set  $(\{z_{t,v}^n, w_{t,v}^n\}_{n=1}^N) := \text{dac\_smc}(t, v)$  for  $v$  in  $\{\ell(u), r(u)\}$  and obtain  $\gamma_{C_u}^N$  in (9).
6:   Merge: compute the mixture weights  $m_{t,u}^{(n_1, n_2)}$  in (10) and  $\tilde{w}_{t,u}^{(n_1, n_2)}$  for all  $n_1, n_2 \leq N$ .
7:   Resample: draw  $\{\tilde{z}_{t,u}^n\}_{n=1}^N$  using weights  $\tilde{w}_{t,u}^{(n_1, n_2)}$  and set  $w_u^n = 1$  for all  $n \leq N$ .
8:   Update: set  $z_{t,u}^n = \tilde{z}_{t,u}^n$  for all  $n \leq N$ .
9: end for
10: Output  $(\{z_{t,\mathfrak{R}}^n\}_{n=1}^N)$ .
```

---

FIGURE 25 – DAC, extrait de [5]

---

**Algorithm 2.1 (Extended Kalman Filter (EKF))**


---

Consider Model 2.1. An approximate sub-optimal estimate for the filter density function  $p(x_t|Y_t)$ , obtained by linearization, is recursively given according to

$$\hat{p}(x_t|Y_t) = \mathcal{N}(x; \hat{x}_{t|t}, P_{t|t}), \quad (2.22a)$$

$$\hat{p}(x_{t+1}|Y_t) = \mathcal{N}(x; \hat{x}_{t+1|t}, P_{t+1|t}), \quad (2.22b)$$

where

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t (y_t - h(\hat{x}_{t|t-1}, t)), \quad (2.23a)$$

$$P_{t|t} = P_{t|t-1} - K_t H_t P_{t|t-1}, \quad (2.23b)$$

$$\hat{x}_{t+1|t} = f(\hat{x}_{t|t}, t), \quad (2.23c)$$

$$P_{t+1|t} = F_t P_{t|t} F_t^T + Q_t, \quad (2.23d)$$

$$K_t = P_{t|t-1} H_t^T (H_t P_{t|t-1} H_t^T + R_t)^{-1}, \quad (2.23e)$$

with initial values  $\hat{x}_{1|0} = \bar{x}_1$  and  $P_{1|0} = \bar{\Pi}_1$ . Furthermore,  $F_t$  and  $H_t$  are defined by

$$F_t = \frac{\partial f(x, t)}{\partial x} \Big|_{x_t=\hat{x}_{t|t}} \quad H_t = \frac{\partial h(x, t)}{\partial x} \Big|_{x_t=\hat{x}_{t|t-1}} \quad (2.24)$$


---

FIGURE 26 – EKF, extrait de [9]

## 8.2 Figures

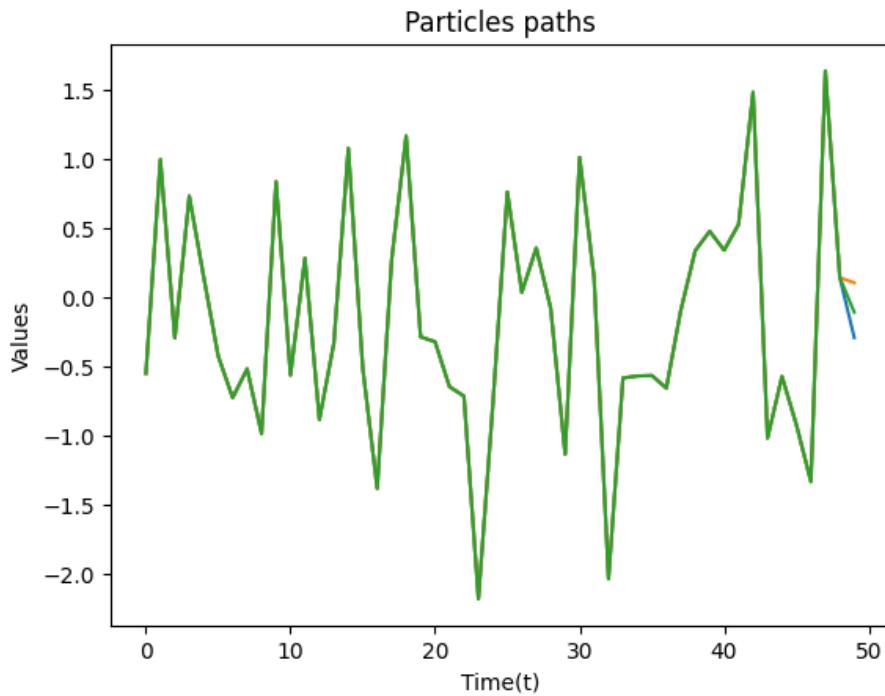


FIGURE 27 – Généalogie du processus présenté en figure 6a

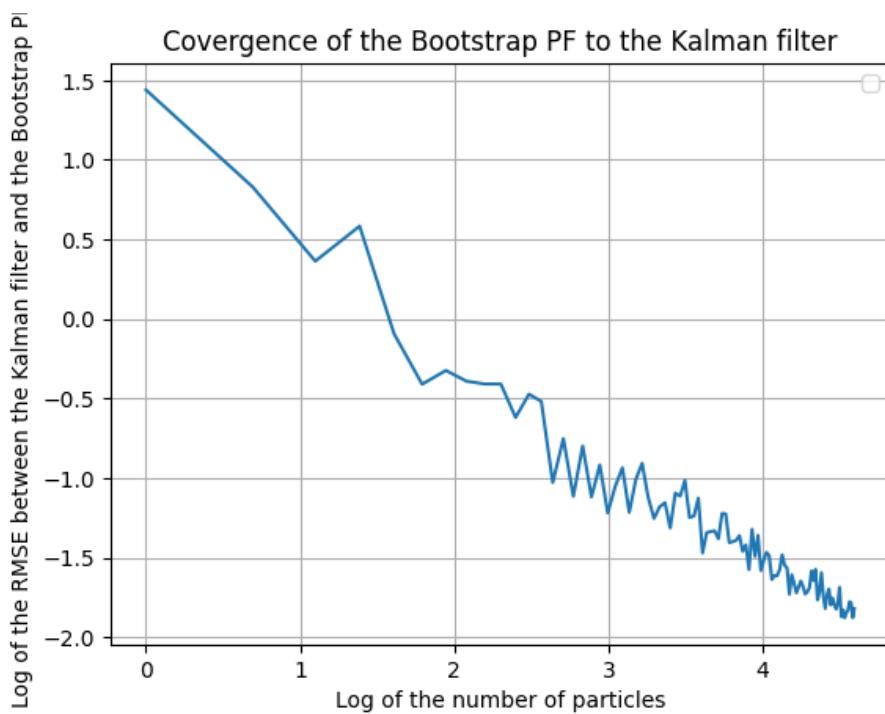


FIGURE 28 – Log-log RMSE en dimension 1

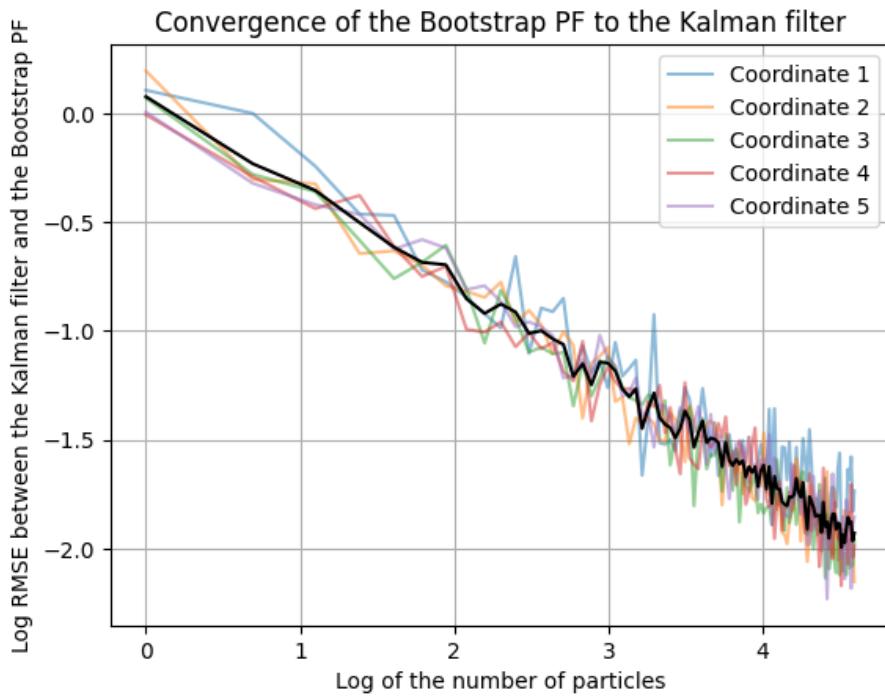


FIGURE 29 – Log-log RMSE en dimension 5

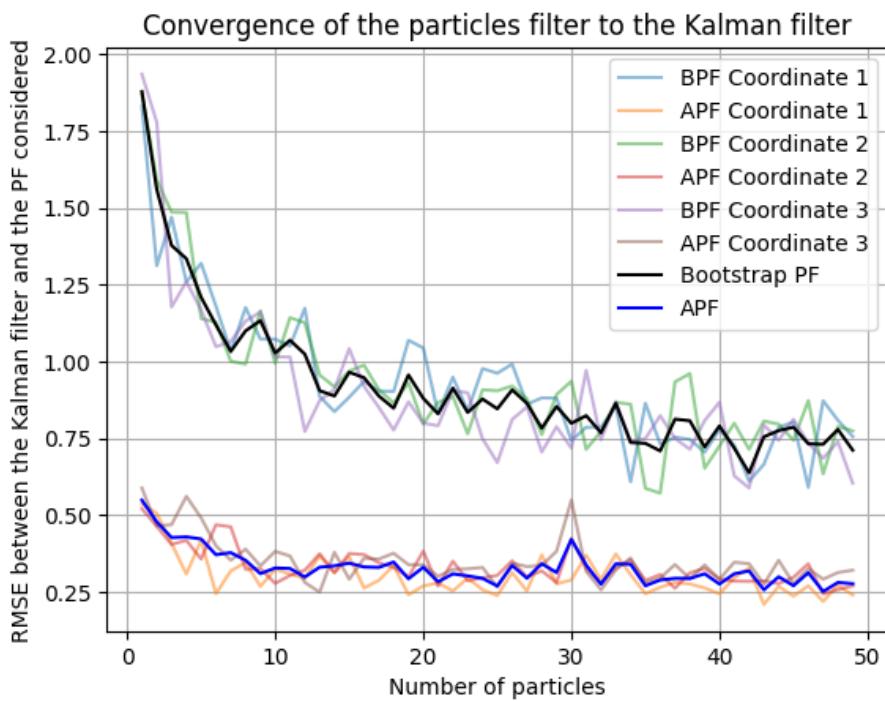


FIGURE 30 – Apf vs BPF avec une erreur de mesure faible

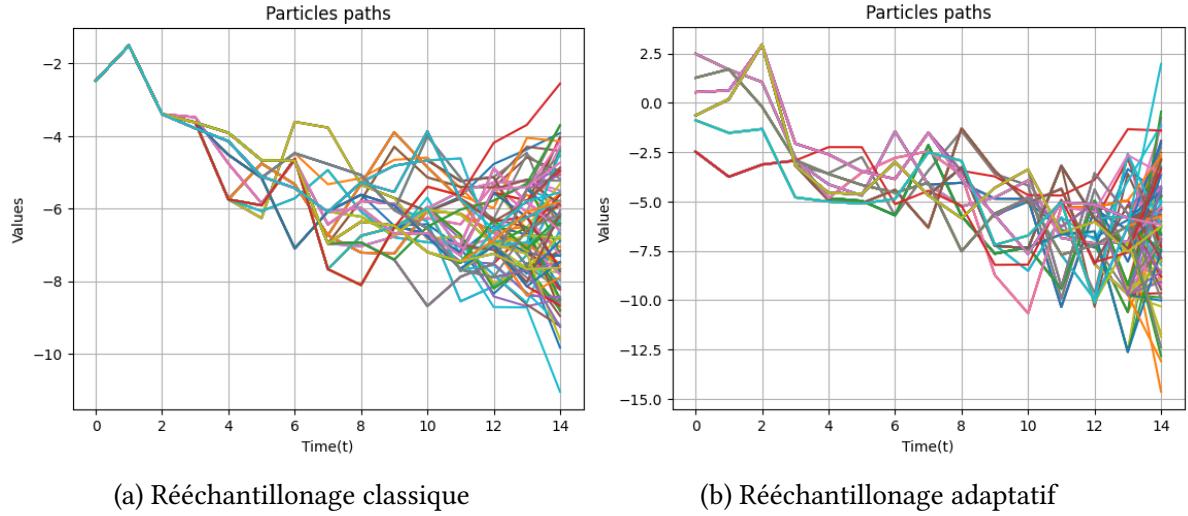


FIGURE 31 – Généalogie des particules

La figure [31] présente les généralogies pour l'algorithme classique [31a] et l'algorithme adaptatif [31b]. Les paramètres utilisés sont les suivants :  $T = 15$ ,  $N = 100$ ,  $a = 1$ ,  $b = 0.5$ , erreurs de variance unitaire.

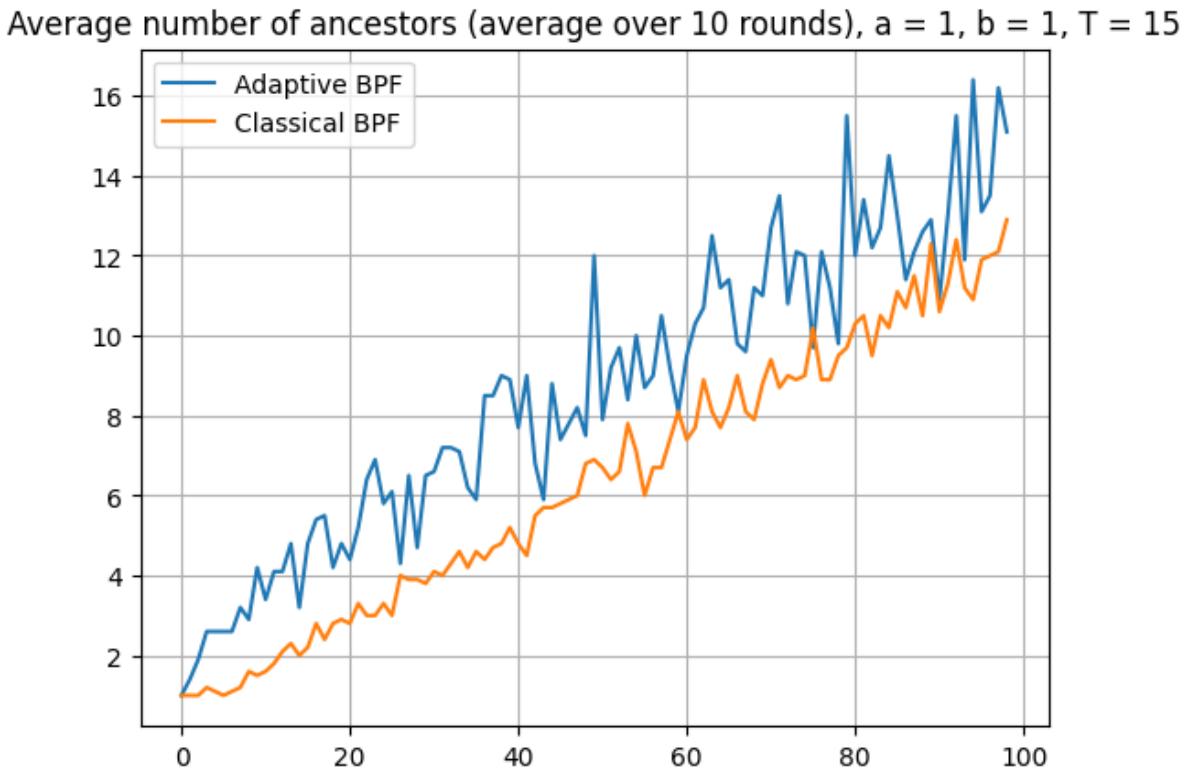


FIGURE 32 – Nombre d'ancêtres

Adaptative BPF vs Classical BPF, N = 100, T = 15, sdX = 0.1, sdY = 1

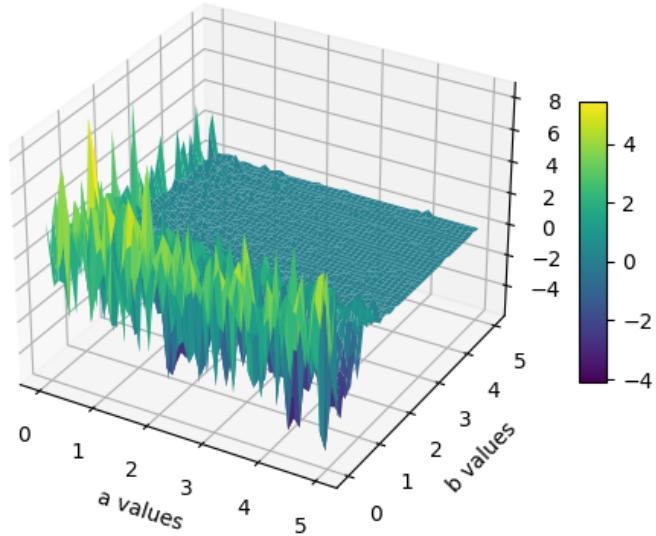


FIGURE 33 –  $sdX = 0.1, sdY = 1$

Adaptative BPF vs Classical BPF, N = 100, T = 15, sdX = 1, sdY = 0.1

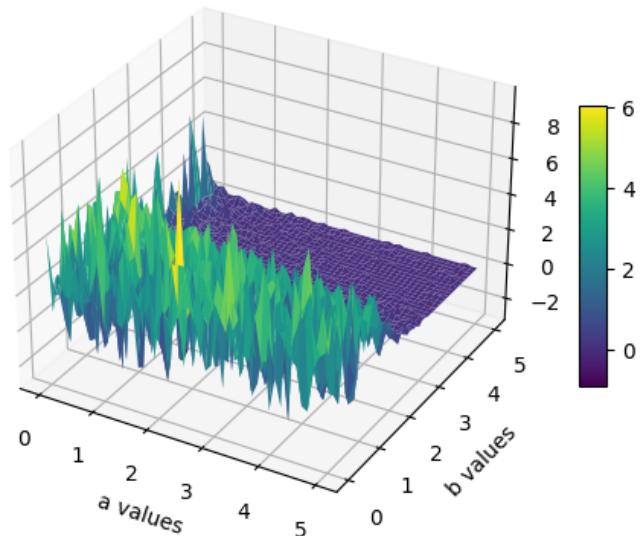


FIGURE 34 –  $sdX = 1, sdY = 0.1$

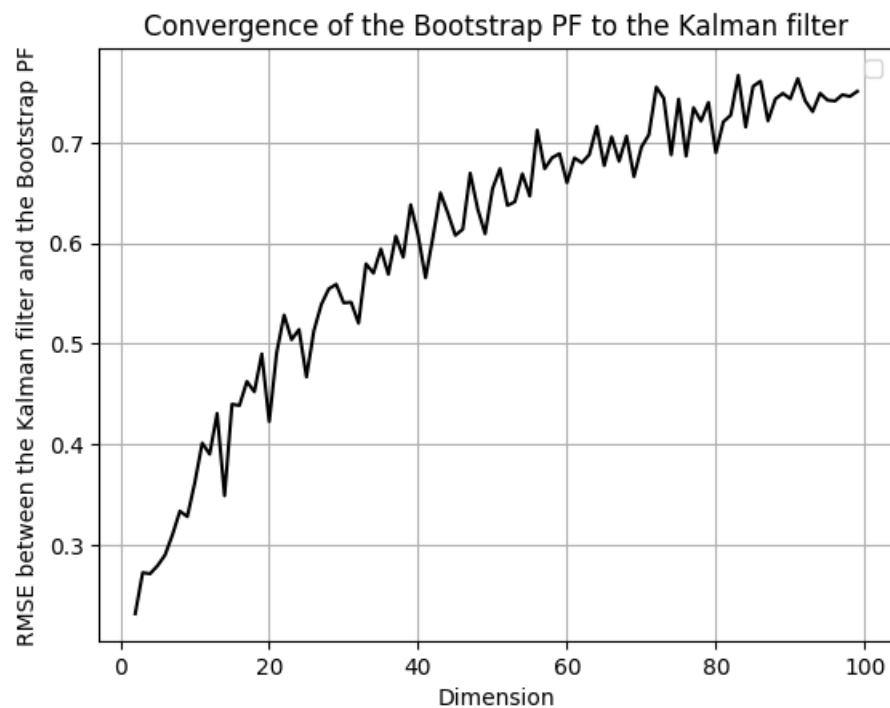
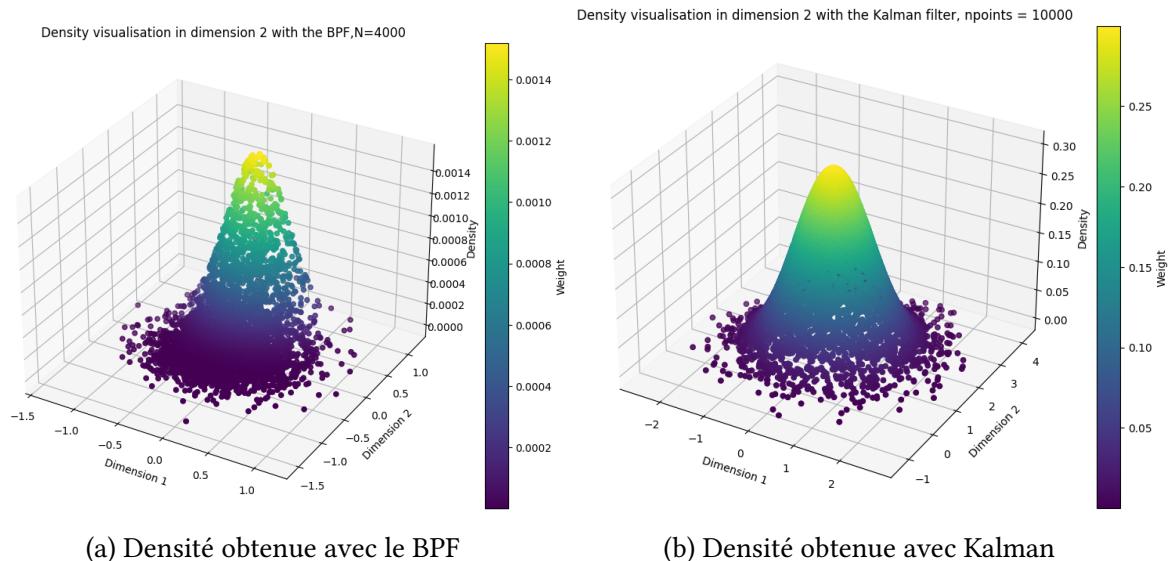


FIGURE 36 – N = 20, T=10, sdX = 1, sdY= 1, a = 0.2, b = 0.4

Plus la dimension augmente, et plus notre BPF est inefficient à même nombre de particules. On considère D allant de 2 à 100.

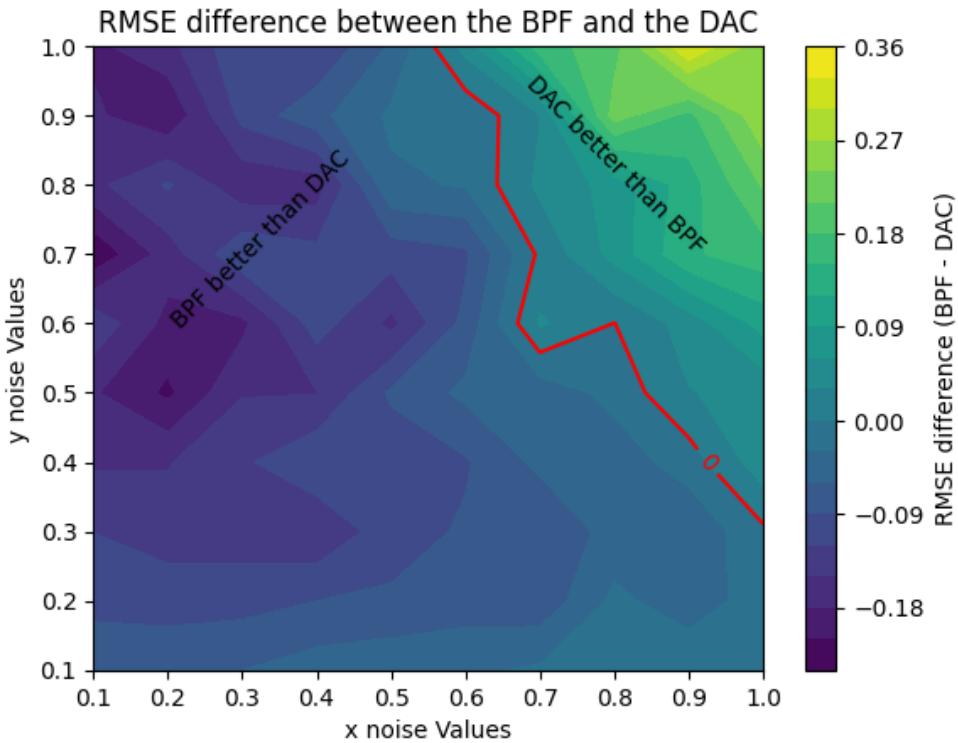


FIGURE 37 – DAC BPF comparaison quand les variances changent, dimension 10,  $a = 0.9$ ,  $b = 0.1$

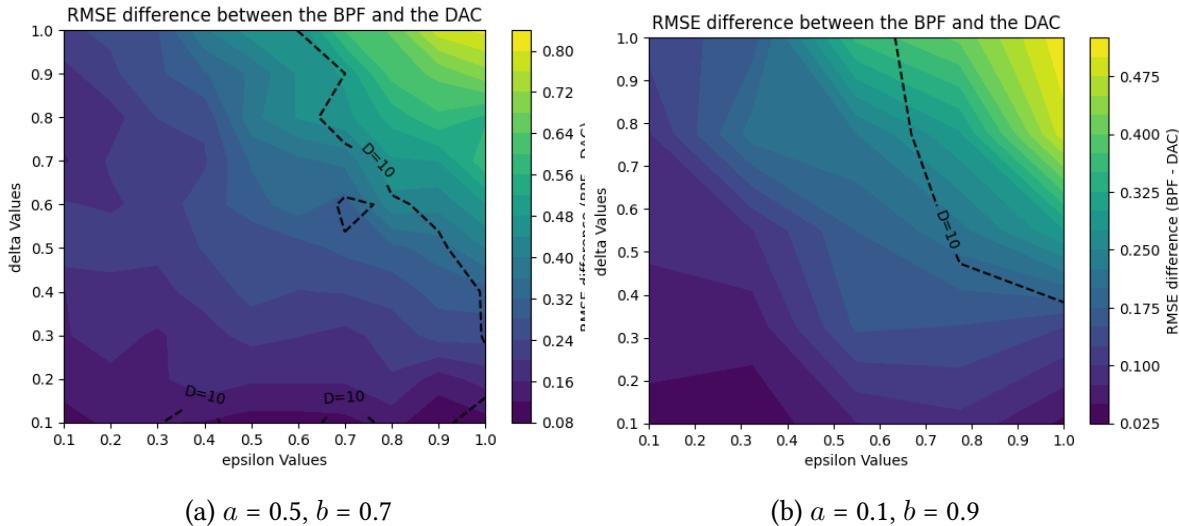


FIGURE 38 – DAC BPF comparaison quand les variances changent et les dimensions changent (dimension 10, 40 et 100 représentées sur ce graphe)

Il est intéressant de noter que pour les dimensions supérieures à 10, il n'y a aucun point tel que le BPF est meilleur que le DAC. Le fond correspond à  $D = 100$ .

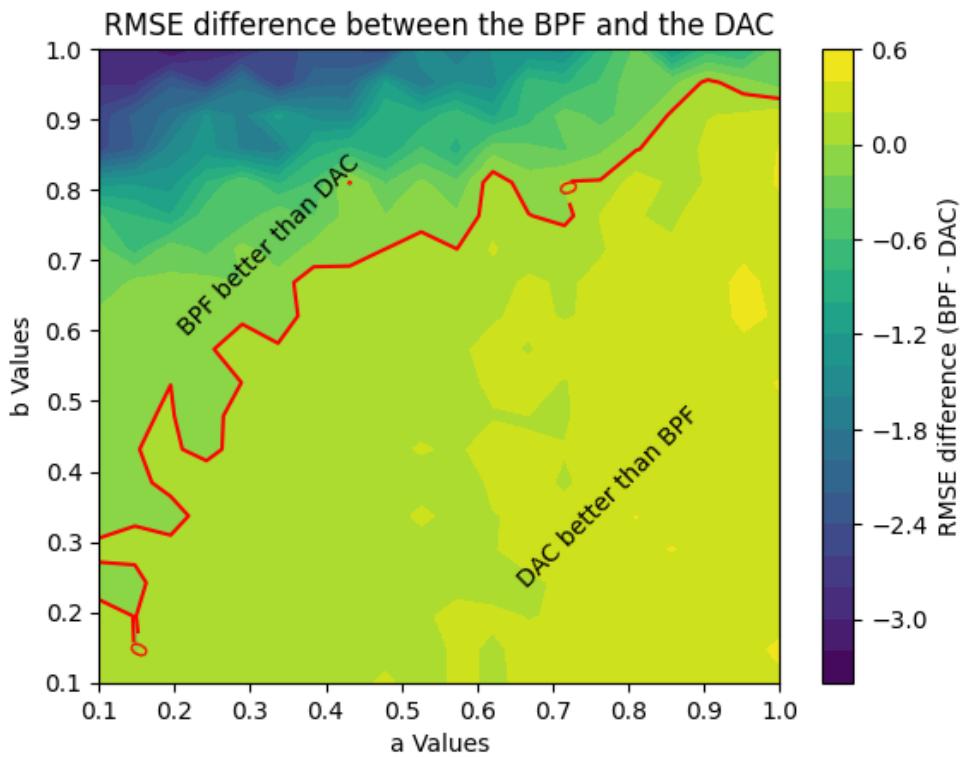


FIGURE 39 – DAC BPF comparaison quand  $a$  et  $b$  changent, dimension 10, variances unitaires

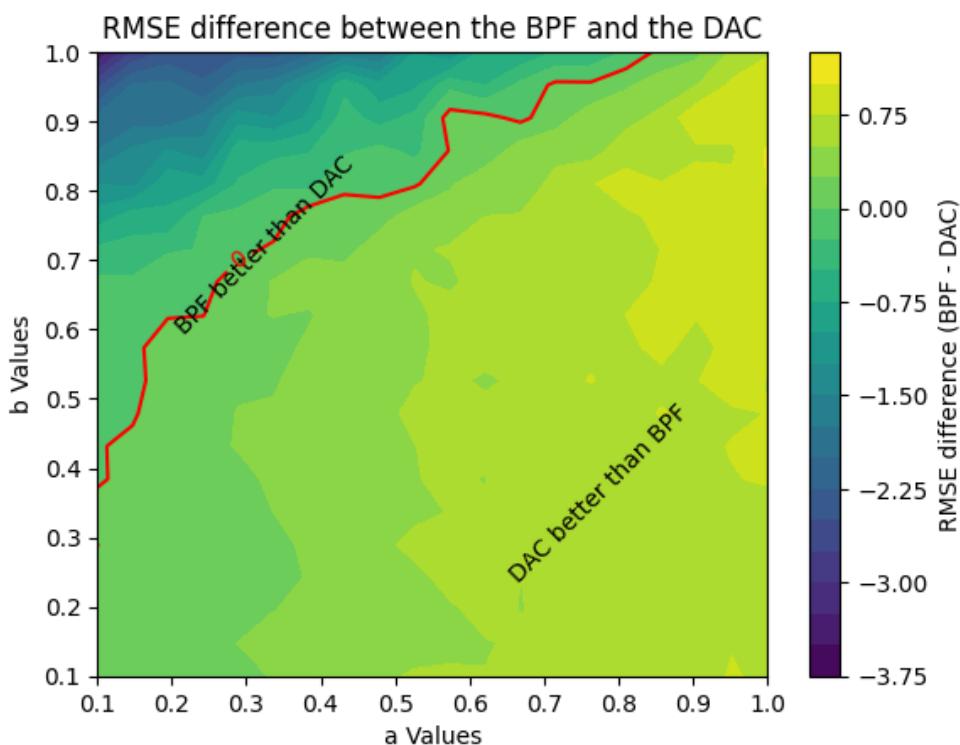


FIGURE 40 – DAC BPF comparaison quand  $a$  et  $b$  changent, dimension 40, variances unitaires

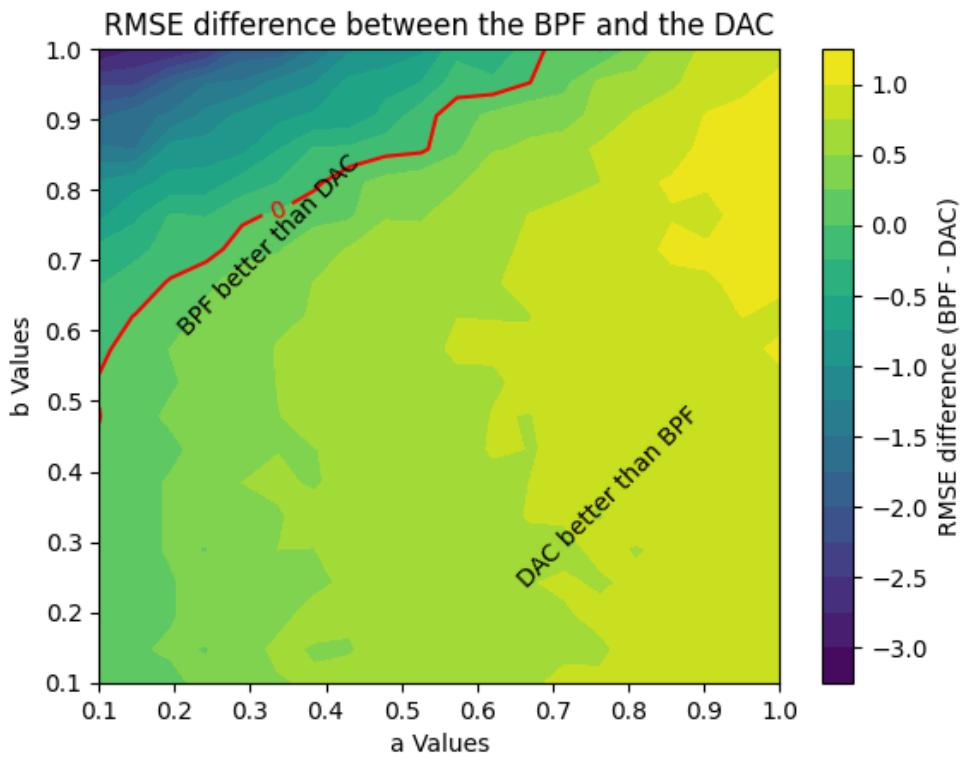


FIGURE 41 – DAC BPF comparaison quand  $a$  et  $b$  changent, dimension 100, variances unitaires

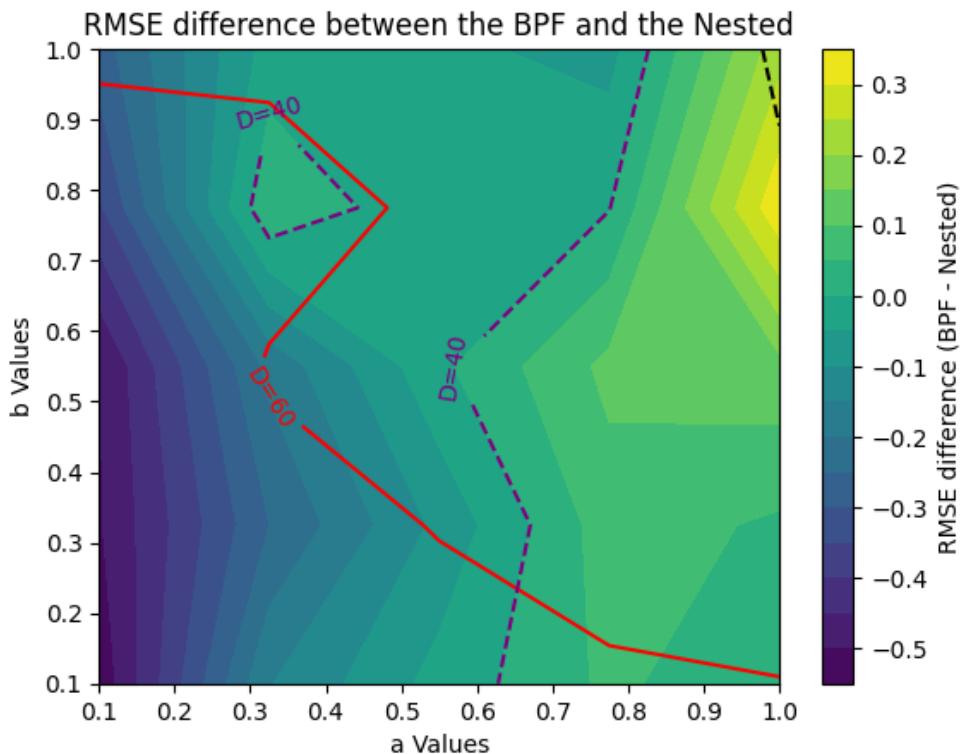


FIGURE 42 – Nested BPF comparaison quand  $a$  et  $b$  changent, variances unitaires

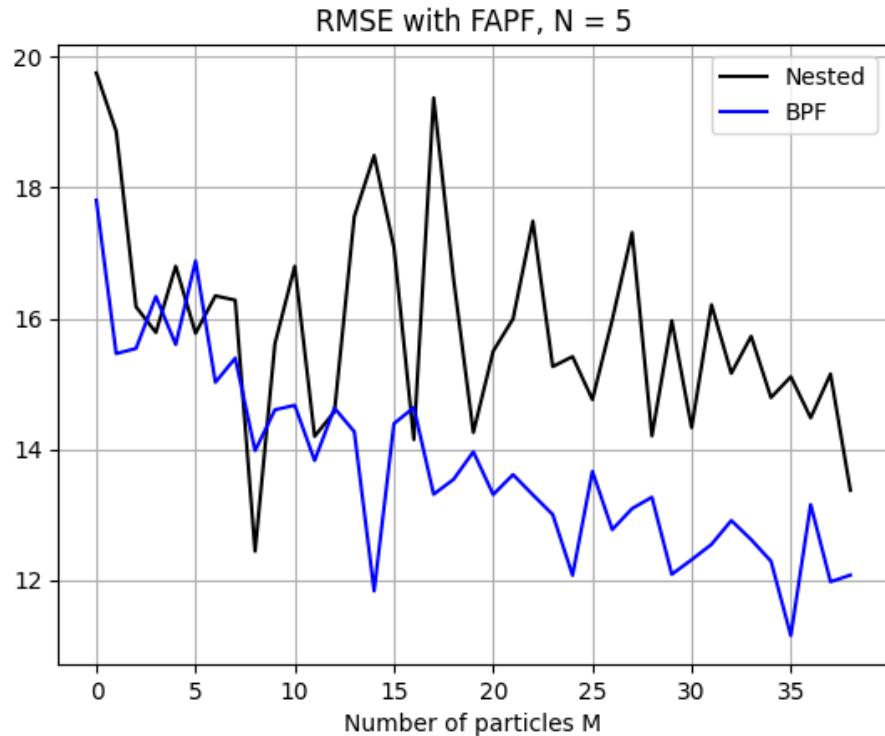


FIGURE 43 – Dimension 20

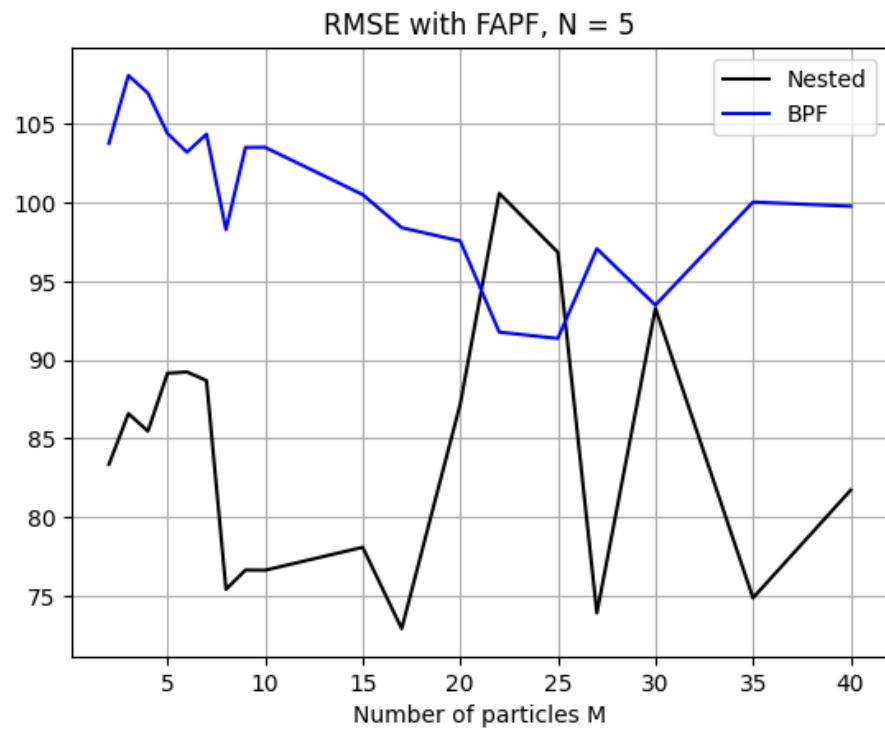


FIGURE 44 – Dimension 100

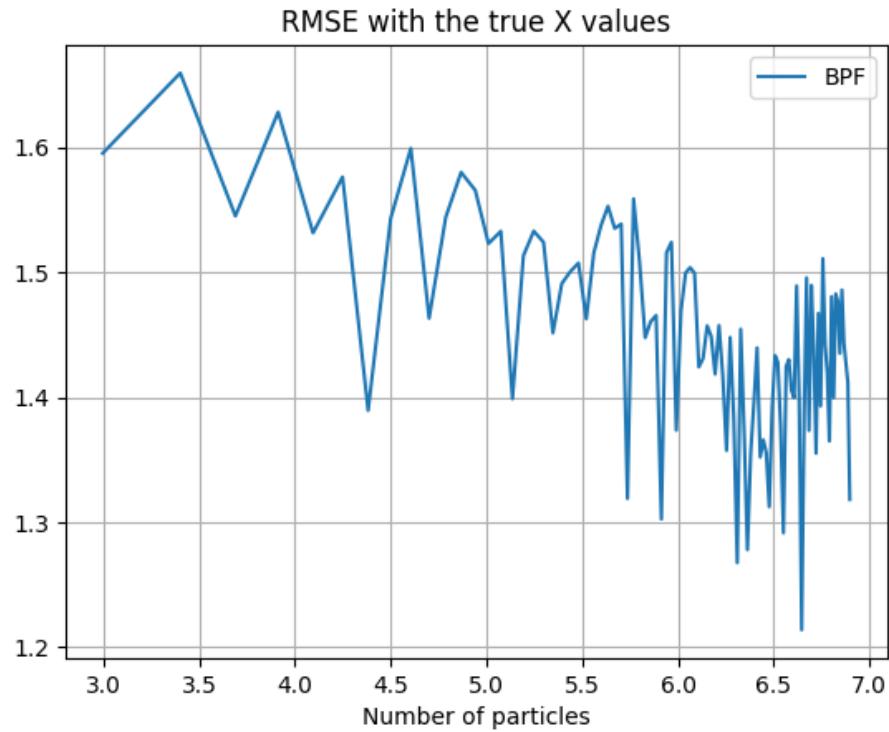


FIGURE 45 – RMSE du BPF, modèle de Lorenz, D = 15, T = 10, F = 8, sdX = 0.1, sdY = 1

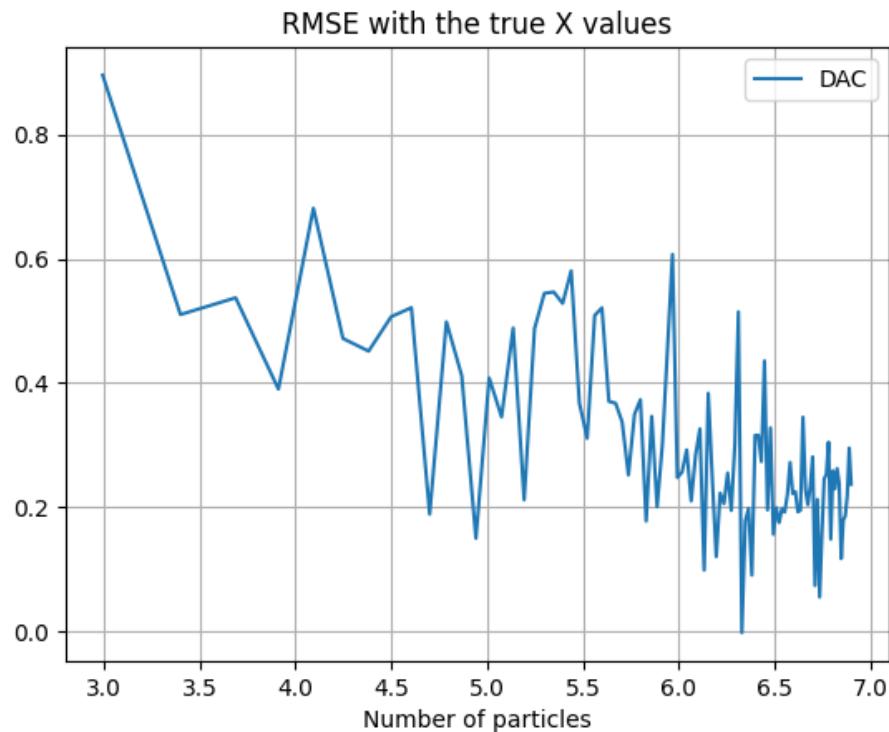


FIGURE 46 – RMSE du DAC, modèle de Lorenz, D = 15, T = 10, F = 8, sdX = 0.1, sdY = 1

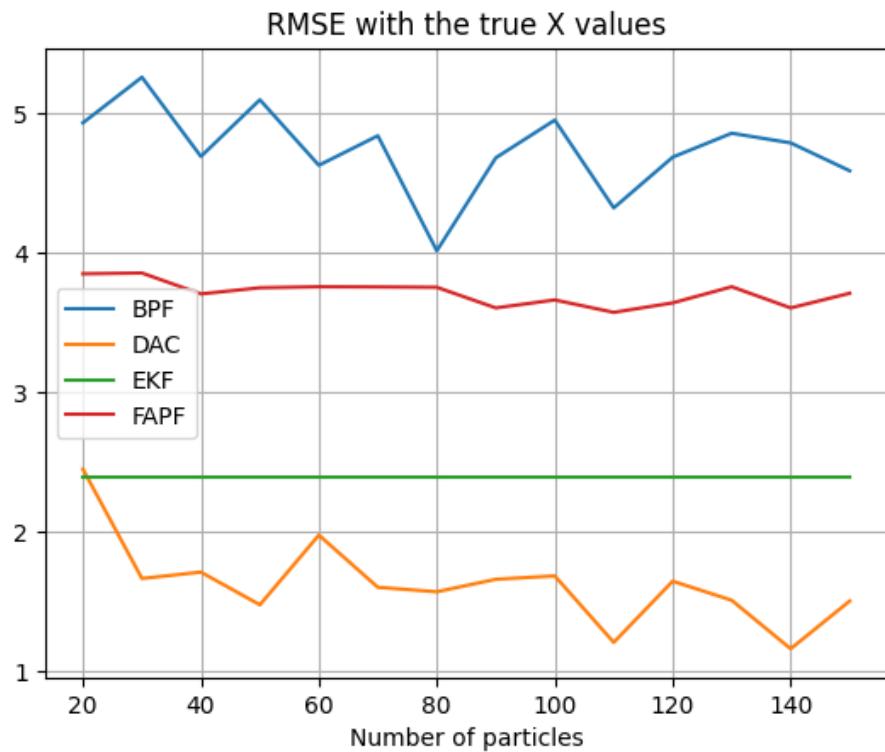


FIGURE 47 – RMSE des filtres pour N allant jusqu'à 150, modèle de Lorenz, D = 15, T = 10, F = 8, sdX = 0.1, sdY = 1

# Table des figures

2	Résumé des données . . . . .	3
3	Approximation de la densité par des diracs pondérés . . . . .	4
4	State space model, extrait de [3] . . . . .	5
5	Arbre pour $d = 8$ . . . . .	15
6	Bootstrap particle filter - dimension 1 . . . . .	19
7	Bootstrap particle filter - densités . . . . .	20
8	Performances du Boostrap PF comparativement au filtre de Kalman . . . . .	21
9	RMSE pour différents niveaux de bruit et fitting . . . . .	22
10	BPF adaptatif vs BPF classique . . . . .	23
11	Résultats DAC . . . . .	24
12	Résultats Nested SMC . . . . .	25
13	Nested vs DAC . . . . .	26
14	Précision des filtres pour le modèle de Lorenz . . . . .	29
15	Situation géographique de Linköping . . . . .	31
22	Kanellbulle . . . . .	34
23	Räkost (gauche) et smörgåstårta (droite) . . . . .	35
24	APF, extrait de [3] . . . . .	36
25	DAC, extrait de [5] . . . . .	36
26	EKF, extrait de [9] . . . . .	37
27	Généalogie du processus présenté en figure 6a . . . . .	38
28	Log-log RMSE en dimension 1 . . . . .	38
29	Log-log RMSE en dimension 5 . . . . .	39
30	Apf vs BPF avec une erreur de mesure faible . . . . .	39
31	Généalogie des particules . . . . .	40
32	Nombre d'ancêtres . . . . .	40
33	$sdX = 0.1, sdY = 1$ . . . . .	41
34	$sdX = 1, sdY = 0.1$ . . . . .	41
36	$N = 20, T=10, sdX = 1, sdY= 1, a = 0.2, b = 0.4$ . . . . .	42
37	DAC BPF comparaison quand les variances changent, dimension 10, $a = 0.9, b = 0.1$ . . . . .	43
38	DAC BPF comparaison quand les variances changent et les dimensions changent (dimension 10, 40 et 100 représentées sur ce graphe) . . . . .	43
39	DAC BPF comparaison quand $a$ et $b$ changent, dimension 10, variances unitaires . . . . .	44

40	DAC BPF comparaison quand $a$ et $b$ changent, dimension 40, variances unitaires . . . . .	44
41	DAC BPF comparaison quand $a$ et $b$ changent, dimension 100, variances unitaires . . . . .	45
42	Nested BPF comparaison quand $a$ et $b$ changent, variances unitaires . . . . .	45
43	Dimension 20 . . . . .	46
44	Dimension 100 . . . . .	46
45	RMSE du BPF, modèle de Lorenz, D = 15, T = 10, F = 8, sdX = 0.1, sdY = 1 . . . . .	47
46	RMSE du DAC, modèle de Lorenz, D = 15, T = 10, F = 8, sdX = 0.1, sdY = 1 . . . . .	47
47	RMSE des filtres pour N allant jusqu'à 150, modèle de Lorenz, D = 15, T = 10, F = 8, sdX = 0.1, sdY = 1 . . . . .	48

## Liste des tableaux

## Références

- [1] P. le Moigne, “Surfex scientific documentation,” vol. 8.1, pp. 107–224, 2018. [Online]. Available : <https://www.umr-cnrm.fr/surfex/spip.php?rubrique11>
- [2] C. Naesseth, F. Lindsten, and T. Schön, “Elements of sequential monte carlo,” *arXiv*, no. 1612.09162v1, p. 307–392, 2019.
- [3] T. Schön and F. Lindsten, “Learning of dynamical systems,” August 2017.
- [4] N. Chopin and O. Papaspiliopoulos, *An introduction to sequential Monte Carlo*. Springer, 2020.
- [5] F. R. Crucinio and A. M. Johansen, “A divide and conquer sequential monte carlo approach to high dimensional filtering,” 2022.
- [6] F. Lindsten, A. M. Johansen, C. A. Naesseth, B. Kirkpatrick, T. B. Schön, J. A. D. Aston, and A. Bouchard-Côté, “Divide-and-conquer with sequential monte carlo,” *Journal of Computational and Graphical Statistics*, vol. 26, no. 2, pp. 445–458, apr 2017. [Online]. Available : <https://doi.org/10.1080%2F10618600.2016.1237363>
- [7] C. Naesseth, F. Lindsten, and T. Schön, “High-dimensional filtering using nested sequential monte carlo,” *arXiv*, no. 1612.09162v1, 2016.
- [8] H. Masnadi-Shirazi, A. Masnadi-Shirazi, and M.-A. Dastgheib, “A step by step mathematical derivation and tutorial on kalman filters,” 2019.
- [9] T. Schön, “Solving nonlinear state estimation problems using particle filters,” 2010.