ENSAE Paris

Deep Learnnig

Quant GANs: deep generation of financial time series

Etudiants :
Béline Aubergeon, Paul Petit &
Steven Worick

Auteurs :
Magnus Wiese, Robert Knobloch,
Ralf Korn & Peter Kretschmer

Mai 2022

Table des matières

1 Introduction

Le papier *Quant GANs*, écrit par Magnus Wiese, Robert Knobloch, Ralf Korn & Peter Kretschmer en 2020, introduit et formalise un nouveau modèle répondant à la problématique de la génération de séries temporelles financières synthétiques. La génération de scénarios de marché est l'un des challenges les plus importants et persistant en finance quantitative bien que nécessaire dans de nombreux domaines d'application que ce soit pour la création de portefeuilles d'actifs, le hedging ou le pricing d'option, le backtest de stratégies d'investissement... Alors que les techniques quantitatives les plus utilisées aujourd'hui reposent sur des simulations de Monte-Carlo, des approches utilisant des réseaux de neurones commencent à émerger avec notament le Bernoulli RBM ou les CVAE. Celle proposée par les auteurs repose sur les *Generative Adversarial Networks* (*GANs*), combinaison d'un générateur et d'un discriminateur utilisant les *Temporal Convolutional Networks* (*TCNs*). Le modèle est capable de prendre en compte les dépendances de long terme et de reproduire les propriétés fondamentales des séries comme les quatres premiers moments statistiques, l'autocorrélation, la présence de clusters de volatilités, faisant de cette approche une méthodologie particulièrement adaptée à la problématique de modélisation de données financières.

Au travers de ce projet, nous présenterons l'article *Quant GANs : Deep Generation of Financial Time Series*. Dans un premier temps nous décrirons le modèle, puis nous nous intéresserons à l'application proposée par les auteurs et enfin nous montrerons les résultats obtenus par notre réplication de l'algorithme.

2 Description du modèle GAN

En finance, la performance d'un actif est estimée par son rendement relatif : $R_t = \frac{S_t - S_{t-1}}{S_{t-1}}$ ou son rendement logarithmique $R_t = log(S_t) - log(S_{t-1})$. Ce derniers présentent plusieurs faits stylisés :

- Des queues de distribution épaisses que la loi normale,
- Une plus forte concentration des valeurs autour de la moyenne par rapport à la loi normale,
- Des clusters de volatilités,
- Un effet de levier correspondant à une corrélation négative entre la volatilités des rendements et les rendements eux-mêmes,
- Les rendements sont non-corrélés mais pas indépendants.

L'objectif du papier que nous étudions est alors de générer de façon synthétique des rendements tout en prenant en considération l'ensemble de leurs caractéristiques. Nous allons montrer comment le modèle permet la reproduction de ces différents faits après avoir introduit les réseaux de neuronnes *NN* permettant la construction du générateur *Stochastic Volatility Neural Networks* (*SVNNs*).

2.1 Architecture du réseau

2.1.1 Temporal Convolutional Networks (*TCN*)

Les *TCNs* sont des réseaux de neurones résultants de la combinaison des *Convolutional Neural Networks* (*CNNs*) et des *Recurrent Neural Network* (*RNNs*). Ils permettent de réaliser des prévisions sur des processus stochastiques qui sont dans notre cas des log-rendements décomposés en un processus de volatilité stochastique. Les convolutions causales utilisées par les *TCNs* diffèrent des convolutions traditionnelles du fait que l'output repose exclusivement sur les valeurs passées et présentes et ne prennent pas en compte les valeurs

futures. Pour les construire nous utilisons alors des convolutions causales dilatées, définies par le quadruplet (N_I, N_O, K, D) (dimension de l'input, dimension de l'output, taille du kernel, dilatation).

Soient $X \in \mathbb{R}^{N_1 \times T}$ et $W \in \mathbb{R}^{K \times N_1 \times N_O}$, une couche convolutionnelle causale avec dilatation D est une fonction $w : \mathbb{R}^{N_1 \times T} \to \mathbb{R}^{N_O \times (T - D(K-1))}$ telle que :

$$w(X)_{m,t} := (W *_D X)_{m,t} + b_m = \sum_{i=1}^{K} \sum_{j=1}^{N_I} W_{i,j,m}.X_{j,t-D(K-I)} + b_m$$

L'avantage principal de cette fonction est de pouvoir prendre en compte les valeurs passées sans augmenter le nombre de poids grâce au facteur de dilatation. En effet, puisque l'objectif est d'intégrer les dépendances de long terme, il serait possible d'augmenter la taille du filtre mais cela allongerait inévitablement le nombre de poids à optimiser dans le modèle le rendant de fait plus difficile à calibrer. D'autre part, l'un des ses inconvénients est de ne pas prendre en compte les valeurs intermédiaires. C'est pourquoi les auteurs décident d'empiler différentes couches convolutionnelles cachées L avec différents facteurs de dilatation. Nous parlons alors de *Vanilla TCNs*.

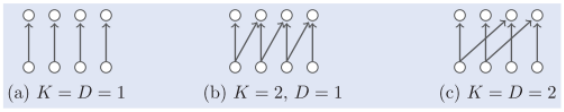


FIGURE 1 – Couche convolutionnelle causale dilatée avec différents paramètres de dilatation D et tailles de kernel K.

Contrairemennt aux réseaux de neurones standards, *Multilayer Perceptron* (*MLPs*), qui sont obtenus par composition de transformations affines utilisants des fonctions d'activation $f(x, \theta) = a_{L+1} \circ f_L \circ ... \circ f_1(x)$, les *Vanilla TCNs* sont eux obtenus par composition de couches convolutives causales dilatées avec des fonctions d'activation. Si l'on considère L couches cachées, nous pouvons écrire la fonction $f : \mathbb{R}^{N_O \times T_O} \to \mathbb{R}^{N_{L+1} \times T_L}$ telle que :

$$f(X, \theta) = w \circ \psi_L \circ ... \circ \psi_1(X)$$

où : $\psi_l : \mathbb{R}^{N_{l-1} \times T_{l-1}} \to \mathbb{R}^{N_l \times T_l}$ est un bloc module (fonction lipschitz), $w : \mathbb{R}^{N_L \times T_L} \to \mathbb{R}^{N_{L+1} \times T_L}$ une 1×1 couche convolutionnelle et $T_0, L, N_0, ..., N_{L+1}, S_l \in \mathbb{N}$ tel que $T_l := T_{l-1} - S_l$ donnant $T_L - T_0 = \sum_{l=1}^{L} S_l$.

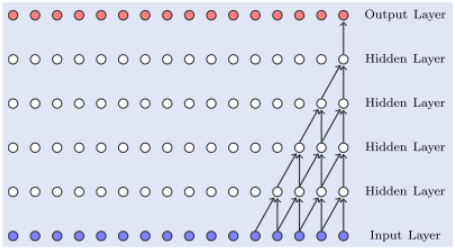


FIGURE 2 – Vanilla TCN avec 4 couches cachées, taille de kernel = 2, dilatation D=1

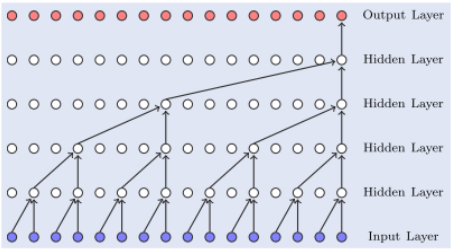


FIGURE 3 – Vanilla TCN avec 4 couches cachées, taille de kernel = 2, dilatation D=2

Nous appellons *receptive field size (RFS)* la quantité $T^{(f)} := 1 + \sum_{l=1}^{L} S_l$ correspondant à la taille de la séquence que le *TCN* reçoit en input. Le réseau ne peut donc prendre en compte qu'un passé limité dans l'historique de la série temporelle.

Les principaux avantages des *TCNs* par rapport aux autres réseaux de neurones sont :

- La possibilité de réaliser les prévisoins en parallèle,
- La possibilité de controler la mémoire du processus en modifiant la taille du *RFS*, le facteur de dilatation, le nombre de couches ou la taille du kernel,
- Des gradients stables,
- Peu de mémoire requise pour l'entrainenemt puisque les filtres sont partagés au travers des couches et la backpropagation ne dépend que dela profondeur du réseau,
- La possibilité de faire varier le nombre de séries en input.

A l'inverse, leurs principaux inconvénients sont :

- Le stockage de la donnée lors de l'évaluation car contrairement aux*RNNs* toute l'historique doit être utilisée,
- La reparamétrisation du modèle nécessaire lorsque l'application doit porter sur un nouveau domaine.

2.1.2 Generative Adversarial Networks (GANs)

On note les processus stochastiques paramétré par θ de la manière suivante :

$$X_{s:t,\theta} = (X_{s,t}, ..., X_{t,\theta}) \in \mathbb{R}^{N_X \times (t-s+1)}$$

L'appelation Generative Adversarial Networks (*GANs*) (*Goodfellow & al., 2014* [3]) fait référence à une classe d'algorithmes apprenant un modèle génératif d'un échantillon d'une variable aléatoire, c'est-à-dire un ensemble de données, ou la distribution d'une variable aléatoire elle-même.. Dans notre cas nous nous intéressons à un modèle stochastique en temps discret défini par le *TCN*. Soient N_Z et $N_X \in \mathbb{N}$ et $(\Omega, \mathcal{F}, \mathbb{P})$ un espace probabiliste. On suppose que les variables aléatoires X et Z appartiennent respectivement à $\mathbb{R}^{N_X}$ et $\mathbb{R}^{N_Z}$ et on note leur distribution $\mathbb{P}_X$ et $\mathbb{P}_Z$.

Dans le cadre des *GANs*, on appelle $(\mathbb{R}^{N_Z}, \mathcal{B}(\mathbb{R}^{N_Z}))$ et $(\mathbb{R}^{N_X}, \mathcal{B}(\mathbb{R}^{N_X}))$ *latent* et *data measure space*. Z représente le bruit a-priori et X la variable aléatoire cible. L'objectif des GANs est d'entrainer le réseau g : $\mathbb{R}^{N_Z} \times \Theta^{(g)} \to \mathbb{R}^{N_X}$ tel que la variable aléatoire induite $g_\Theta(\mathrm{Z}) := g_\Theta \circ \mathrm{Z}$ avec les paramètres $\theta \in \Theta^{(g)}$ et la variable aléatoire cible X aient la même distribution. *Goodfellow & al.* [3] proposent la modélisation adverse *GANs* pour les *NNs* en introduisant le générateur et le discriminateur de la manière suivante :

- Générateur : Soit g : $\mathbb{R}^{N_Z} \times \Theta^{(g)} \to \mathbb{R}^{N_X}$ un réseau avec pour paramètres $\Theta^{(g)}$. La variable aléatoire

X définie par :

$$X : \Omega \times \Theta^{(g)} \to \mathbb{R}^{N_X}$$
$$(\omega, \theta) \mapsto g_\theta(Z(\omega))$$

est appelée variable aléatoire générée. La variable aléatoire X_θ est à partir du générateur g avec le paramètre θ.

— Discriminateur : Soit $d : \mathbb{R}^{N_X} \times \Theta^{(d)} \to \mathbb{R}$ un réseau avec pour paramètres $\eta \in \Theta^{(d)}$ et la fonction sigmoid $\sigma : \mathbb{R} \mapsto [0,1] : x \mapsto \frac{1}{1+e^{-x}}$. On appelle discriminateur, une fonction d : $\mathbb{R}^{N_X} \times \Theta^{(d)} \mapsto [0,1]$ définie par d : $(x, \eta) \mapsto \sigma \circ d_\eta(x)$.

L'objectif du générateur g est de créer un échantillon (X) tel que le discriminateur soit incapable de classifier cet échantillon comme synthétique ou réel. Nous obtenons alors le *GAN objective* de la manière suivante :

$$\min_{\theta \in \Theta^{(g)}} \max_{\eta \in \Theta^{(d)}} \mathcal{L}(\theta, \eta)$$

où : $\mathcal{L}(\theta, \eta) := \mathbb{E}[log(d_\eta(X))] + \mathbb{E}[log(1 - d_\eta(g_\theta(Z)))] = \mathbb{E}[log(d_\eta(X))] + \mathbb{E}[log(1 - d_\eta(\tilde{X}_\theta))]$.

Les paramètres (θ, η) sont entrainés en alternant la mise à jour de leurs gradients entre le générateur et le discriminateur.

Les *GANs* sont donc un outil intéressant puisqu'à partir d'une combinaison de réseau de neurones, ils permettent de créer des données synthétiques ayant les mêmes propriétés que la distribution des variables d'origine sans que nous ayons à définir un *prior*. Par ailleurs contrairement à des modèles comme les RBM, aucune *Markov Chain Monte Carlo* n'est requise pour la génération ce qui permet un gain de temps.

Cependant, les *GANs* ont aussi des inconvénients. Il est en effet difficile de définir le bon moment pour arréter l'entrainement des 2 réseaux. On utilise alors des méthodes d'évaluation quantitatives (définis dans la Section 2.3) afin d'estimer la performance du modèle. Il est également possible lors de l'entrainement que le modèle se retrouve dans un optimum local. Enfin, il n'y a pas de répresentation de la fonction de densité du générateur et comme avec les autres *NNs*, le modèle génératif n'est pas inversible ce qui ne nous permet pas à partir d'un échantillon généré d'obtenir la variable latente.

2.2 Stochastic Volatility Neural Network (SVNN)

Pour l'implémentation du Quant GANs, les auteurs utilisent un *Stochastic Volatility Neural Network SVNN*), aussi appelé *log return neural process* (*log return NP*). Celui-ci est utilisé par le générateur et est défini de la manière suivante :

Soient Z $= (Z_t)_{t \in \mathbb{Z}}$, $\mathbb{R}^{N_Z}$ un bruit blanc gaussien iid, $g^{(TCN)} : \mathbb{R}^{N_Z \times T^{(g)}} \times \Theta^{(TCN)} \to \mathbb{R}^{2N_X}$ un *TCN* avec un *RFS* $T^{(g)}$ et un réseau $g^{(e)} : \mathbb{R}^{N_Z} \times \Theta^{(e)} \to \mathbb{R}^{N_X}$. Soient les paramètres $\alpha \in \Theta^{(TCN)}$ et $\beta \in \Theta^{(e)}$ et le processus stochastique suivant :

$$\mathrm{R} : \Omega \times \mathbb{Z} \times \Theta^{(TCN)} \times \Theta^{(e)} \to \mathbb{R}^{N_X}$$
$$(\omega, \mathrm{t}, \alpha, \beta) \to [\sigma_{t,\alpha} \bigodot \epsilon_{t,\beta} + \mu_{t,\alpha}](\omega)$$

avec $\odot$ un produit de Hadamard, et le processus h

$$\begin{aligned} h_t &:= g_\alpha^{(TCN)}(Z_{t-T^{(g)}:(t-1)}) \\ \sigma_{t,\alpha} &:= |h_{t,1:N_X}| \\ \mu_{t,\alpha} &:= h_{t,(N_X+1):2N_X} \\ \epsilon_{t,\beta} &:= g_\beta^{(e)}(Z_t) \end{aligned}$$

Ce processus est appelé *log-return NP* et son architecture *SVNN*. Les *NPs* $\sigma_\alpha := (\sigma_{t,\alpha})_{t\in\mathbb{Z}}$, $\mu_\alpha := (\mu_{t,\alpha})_{t\in\mathbb{Z}}$, et $\epsilon_\beta := (\epsilon_{t,\beta})_{t\in\mathbb{Z}}$ sont appelés respectivement volatilité, drift et innovation *NP*.

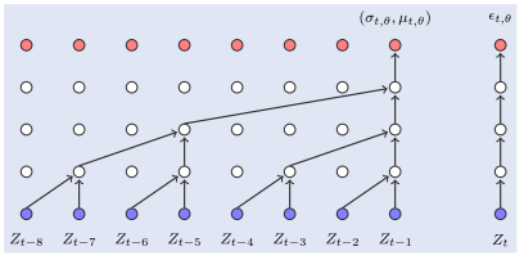


FIGURE 4 – SVNN architecture

On observe que le drift et la volatilité sont $\mathcal{F}_{t-1}^Z$ mesurables alors que l'innovation est $\mathcal{F}_t^Z$ par construction.

L'un des principaux défi comme nous l'avons vu est de prendre en considération les faits stylisés caractéristiques des rendements. L'appartenance de *log-return NP* à l'espace $\mathcal{L}^p(\mathbb{R}^{\mathbb{N}_\mathbb{X}})$ implique l'existence de tous les moments statistiques alors que les séries financières ont des queues de distribution importantes. Afin d'y remédier et de générer des queues épaisses, les auteurs appliquent une transformée de Lambert $W \times F_X$.

$$Y = \frac{X-\mu}{\sigma} exp\left(\frac{\delta}{2} \times \left(\frac{X-\mu}{\sigma}\right)^2\right)\sigma + \mu$$

où $\delta \in \mathbb{R}$, μ et σ sont la moyenne et l'écart type de X estimés par maximum de vraisemblance. Pour $\delta > 0$, nous obtenons des queues plus épaisses pour Y que pour X. Cependant, cette transformation étant appliquée au log-rendements, les *log-return NP* sont optimisés pour approximer la transformée inverse Lambert W du processus des log-rendements.

Afin d'évaluer des options avec un *log-return NP*, il est nécessaire de passer par la probabilité risque neutre. On définit alors le *log-return NP* risque neutre $R_{t,\theta}^M$ par

$$R_{t,\theta}^M := R_{t,\theta} - log(\mathbb{E}[exp(\sigma\epsilon_{t,\theta} + \mu)]_{\sigma=\sigma_{t,\theta}, \mu=\mu_{t,\theta}})$$

où $R_{t,\theta} = \sigma_{t,\theta}\epsilon_{t,\theta} + \mu_{t,\theta}$. Le prix spot risque neutre est ainsi $S_{t,\theta}^M = S_0 exp\left(\sum_{s=1}^t (R_{s,\theta} - log(h(\sigma_{s,\theta}, \mu_{s,t}))\right)$.

L'un des principaux problèmes de cette approche est l'estimation des paramètres du modèle précédent puisque les trajectoires du risque neutres ne sont pas observables. Comme nous le savons, le discriminateur est utilisé pour faire la distinction entre les séries financières réelles et les séries générées. Par conséquent, il n'est pas possible d'entraîner le générateur-discriminateur de la même manière pour le pricing d'option

que pour les séries financières. Une solution pourrait être d'approximer le prix des options en générant de trajectoires risque neutre aveec Monte Carlo.

Le fait de contraindre soit la volatilité soit les innovations *NP*, permet le représentation de diverses processus stochastique bien connus tels que les modèles *CIR, GARCH, Black & Scholes*. Par exemple, en imposant $\epsilon_{t,\theta} \sim \mathcal{N}(0,1)$, nous obtenons $S^M_{t,\theta} = S_0 exp\left(\sum_{s=1}^{t}\left[\sigma_{s,\theta}\epsilon_{s,\theta} - \frac{\sigma^2_{s,t}}{2}\right] + rt\right)$ qui est similaire à la distribution risque neutre de *Black & Scholes* des prix spot avec ici une volatilité qui n'est pas constante du fait de l'utilisation du générateur de volatilité.

Avant d'utiliser le discriminateur, les séries temporelles doivent être prétraitées. La Figure ci-dessous montre les 5 étapes du prétraitement.

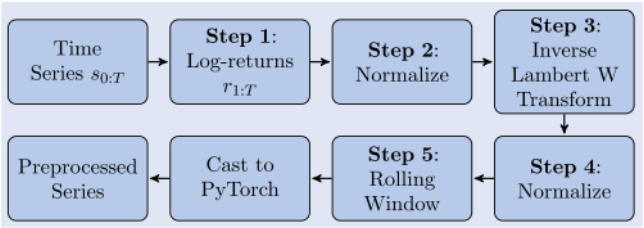


FIGURE 5 – Preprocessing des séries temporelles

2.3 Application et évaluation du modèle

Afin de tester les capacités du *Quant GAN*, les auteurs proposent de modéliser les log-rendements de l'indice S&P 500 avec la décomposition *C-SVNN* et un *RFS* $T^{(g)} = 127$. Le processus des rendements est donné par :

$$R_{t,\theta} = \sigma_{t,\theta}\epsilon_{t,\theta} + \mu_{t,\theta}$$

avec une volatilité *NP* $\sigma_{t,\theta}$, un drift *NP* $\mu_{t,\theta}$ et une innovation *NP* $\epsilon_{t,\theta}$ contrainte d'être i.i.d. $\mathcal{N}(0,1)$. Le processus latent est $Z_t \overset{\text{i.i.d}}{\sim} \mathcal{N}(\mathbf{0}, \mathbb{I})$ *avec* $\mathrm{N}_Z = 3$. Le processus d'innovation prend la forme $\epsilon_{t,\theta} = Z_{t,1}$ pour tout $t \in \mathbb{Z}$.

2.3.1 Évaluation de la précision du modèle

Afin d'évaluer la qualité du modèle et savoir à quel moment arrêter l'entrainement, les auteurs combinent plusieurs mesures. Pour chacune d'entre elles, la distance entre les données générées et les données réelles est calculée. Si pour l'ensemble des mesures la distance est inférieure à un certain seuil, on considère l'entrainement comme terminé. Les mesures utilisées par les auteurs sont les suivantes :

2.3.2 Distance earth mover (EMD)

Décrit la quantité de masse de probabilité qui doit être déplacée pour transformer $\mathbb{P}^h$ la distribution historique en $\mathbb{P}^g$ la distribution générée :

$$\text{EMD}(\mathbb{P}^h, \mathbb{P}^g) = \inf_{\pi \in \Pi(\mathbb{P}^h, \mathbb{P}^g)} \mathbb{E}_{(X,Y)\sim\pi}[\|X - Y\|]$$

où $\Pi(\mathbb{P}^h, \mathbb{P}^g)$ désigne l'ensemble des distributions jointes avec les marginales $\mathbb{P}^h$ et $\mathbb{P}^g$.

2.3.3 Métrique DY

$$\text{DY(t)} = \sum_x |\log P_t^h(A_{t,x}) - \log P_t^g(A_{t,x})|$$

où P_t^h et P_t^g désignent les densités empiriques historiques et générées et $A_{t,x}$ est un partitionnement des données réelles historiques.

2.3.4 Score ACF

$$\text{ACF}(f) := \left\| C(f(r_1 : T)) - \frac{1}{M} \sum_{i=1}^{M} C(f(r_{1:T,\theta}^{(i)})) \right\|_2$$

où $C : \mathbb{R}^T \to [-1, 1]^S : r_{1:T} \mapsto (\mathcal{C}(1; r), ..., \mathcal{C}(S; r))$ est la fonction d'autocorrélation du rendement r jusqu'au retard $S \leq T - 1$.

2.3.5 Score "Leverage effect"

Permet de comparer la dépendance temporelle de la série historique et celle de la série générée.

$$\left\| L\left(r_{1:T}\right) - \frac{1}{M} \sum_{i=1}^{M} L\left(r_{1:T,\theta}^{(i)}\right) \right\|_2$$

où $L\left(r_{1:T}\right) = Corr(r_{t+\tau}^2, r_t)$.

3 Réplication du modèle

Notre implémentation du modèle a été réalisée en *Python* dans un notebook *Google Colaboratory* grâce à la bibliothèque *PyTorch*. Il est recommandé d'utiliser un GPU lors de l'entrainement du modèle.

3.1 Implémentation du modèle

L'implémentation consiste à reproduire le *TCN* et le *GAN* présentés en sections 2.1.1 et 2.1.2. Comme les auteurs, nous utilisons les données de l'indice *S&P 500* du 05/01/2009 au 31/12/2018 récupérées sur *Yahoo Finance*. Nous réalisons le preprocessing présenté en Figure 5 qui consiste à calculer les log-rendements de la série, les normaliser, réaliser la transformation inverse de Lambert-W permettant d'éliminer les queues trop épaisses, diviser par la valeur maximale de la série transformée toutes les données et normaliser à nouveau. Nous modifions alors la série afin d'avoir une suite de fenêtre glissante de taille du $RFS = 127$.

3.2 Difficultés d'implémentation

Certaines difficultés ont été rencontrées lors de la réplication du modèle, en particulier :

— La transformation de Lambert-W : Cette transformation n'est pas disponible par défaut en *Python*. Nous avons donc traduit le package *LambertW* disponible sur *R* en suivant les explication de M. Georg [2]. Nous obtenons une transformation des log-rendements se rapprochant d'une gaussienne (Figure 6b).
— L'initialisation et implémentation du *TCN* et du *GAN* : Peu d'informations sont données par les auteurs afin d'initialiser des blocs temporels et du modèle *TCN*. Nous avons donc lu d'autres articles tels que I. Goodfellow[3], S. Bai[1] et en particulier E. Lezmi[4] afin de paramétrer au mieux ces éléments.
— Le modèle *C-SVNN* n'a pas été implémenté et nous nous contentons d'une approche *TCN pur* avec un processus risque neutre des log-rendements *NP* $R^M_{t,\theta} = R_{t,\theta} - log(h(\sigma_t, \theta, \mu_{t,\theta})) + r$.

Par ailleurs nous avons comparé deux méthodes de pré-processing avec et sans transformation Lambert-W. Le cas sans transformation nous donne des résultats très satisfaisant assez rapidement et avec des seuils relativement larges pour les mesures. Dans le cas avec transformation inverse Lambert-W, les valeurs générées produisent quelques valeurs extrêmes aberrantes faisant réaliser des sauts (crachs) délirants aux trajectoires de la séries. Pour remédier à ce problème nous avons réduit les *learning rates* des *TCNs*, augmenté la taille des mini-batchs et réduit les seuils des mesures.

3.3 Résultats

Dans notre implémentation nous évaluons le modèle grâce à différentes mesures qui sont :
— Les quatres premiers moments statistiques : moyenne, variance, skewness et kurtosis,
— La métrique *DY*,
— Le score *ACF*,
— Le score *leveraged effect*.
Nous n'avons pas implémenté la distance *Earth mover*. Les résultats des métriques du modèle sont disponibles dans le Tableau suivant. Celui-ci montre bien les difficultés que nous avons eu avec le modèle *C-SVNN*.

Transformation	Ecart absolu	Kurtosis	Skewness	ACF	ACF absolu	Leverage effect
log-rendement	0.150	0.967	0.057	0.248	1.045	0.327
log-rendement x Lambert W	0.309	15.646	0.251	0.442	1.042	0.299

TABLE 1 – Score des log-rendements.

L'ensembles des graphiques de nos résultats sont disponibles en Annexe B et C. Ils montrent d'une part que les séries financières et leurs principales caractéristiques peuvent être répliqués par les modèles *GANs* grâce à l'utilisation des *TCNs* avec des distributions des log-rendements synthétiques proches de la distribution réelle. D'autre part, ils soulignent les difficultés rencontrées avec le modèle qui utilisent la transformation log-rendement et Lambert W c'est à dire la génération de valeurs extrêmes aberrantes (lié probablement au manque d'entrainement) faussant l'ensemble des métriques.

4 Conclusion

Le modèle *Quant GANs* présente un nouvelle méthode permettant de simuler des rendements d'actions en utilisant deux réseaux de neurones convolutifs temporels en compétition l'un contre l'autre. Cette approche nouvelle en la matière est au vu des résultats une alternative plus que crédible aux simulations de Monte-Carlo et autres méthodes traditionnelles. En effet, le modèle de génération de rendements proposés par les auteurs pur *TCN* permet de reproduire les faits stylisés des séries. Bien que nous ayons eu des difficultés avec la transformation de Lambert-W, nos résultats restent consistants. Nous trouvons très intéressante la possibilité de controler l'arrêt de l'entrainement avec différentes mesures définies par l'utilisateur, ce qui n'est pas le cas pour des modèles type Bernoulli RBM. Il pourrait maintenant être intéressant de tester l'efficacité de l'algorithme avec un modèle multivarié permettant de controler les relations entre les séries, comme la structure de corrélation, avec différentes mesures de dépendances linéaire ou non.

Références

[1] S. Bai et al. "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling". In : *arXiv :1803.01271* (2018).

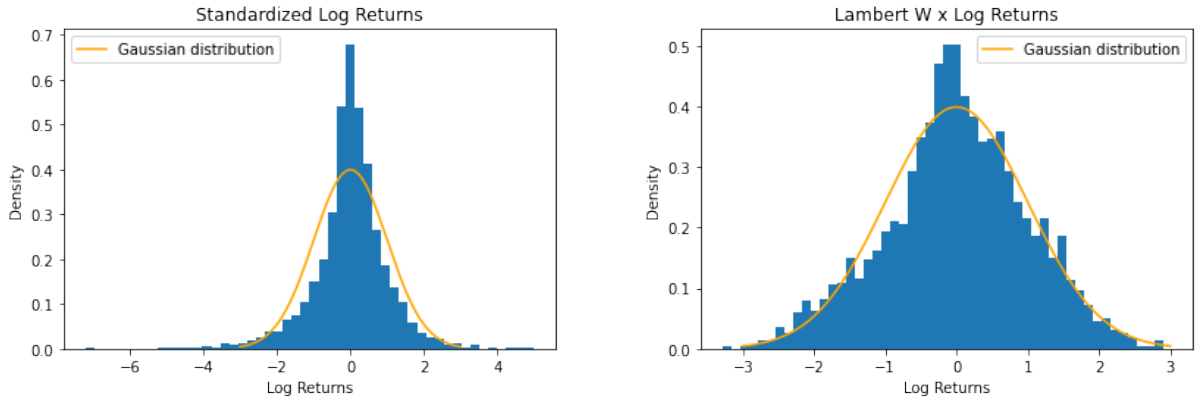
[2] M. Georg. "Lambert W random variables - a new family of generalized skewed distributions with applications to risk estimation". In : *The Annals of Applied Statistics* (2010).

[3] I. Goodfellow et al. "Generative adversarial nets". In : *http ://papers.nips.cc/paper/ 5423-generative-adversarial-nets.pdf* (2014).

[4] E. Lezmi et al. "Improving the Robustness of Trading Strategy Backtesting with Boltzmann Machines and Generative Adversarial Networks". In : *arXiv* (2020).

Appendices

A Annexe S&P 500



(a) Log-rendements.

(b) Lambert-W x Log-rendements.

FIGURE 6 – Distribution des rendements de la série S&P 500.

B Annexe transformation log-rendements

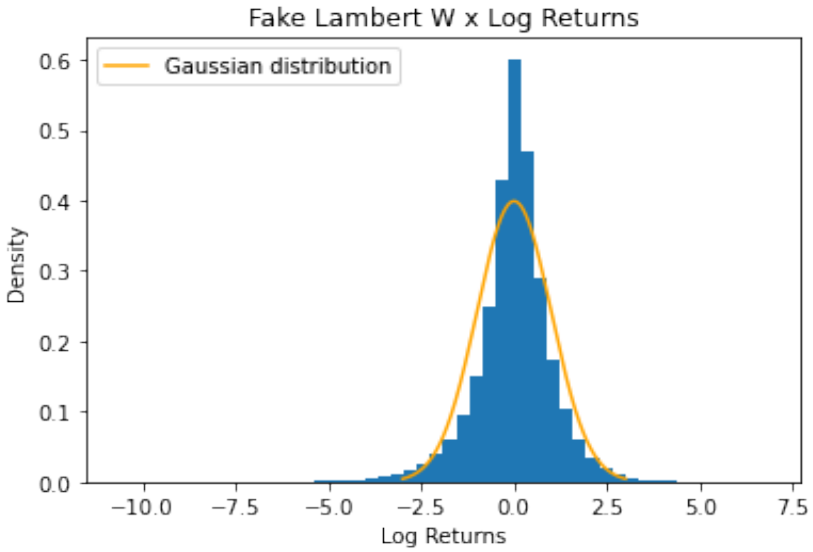


FIGURE 7 – Distribution des log-rendements synthétiques.

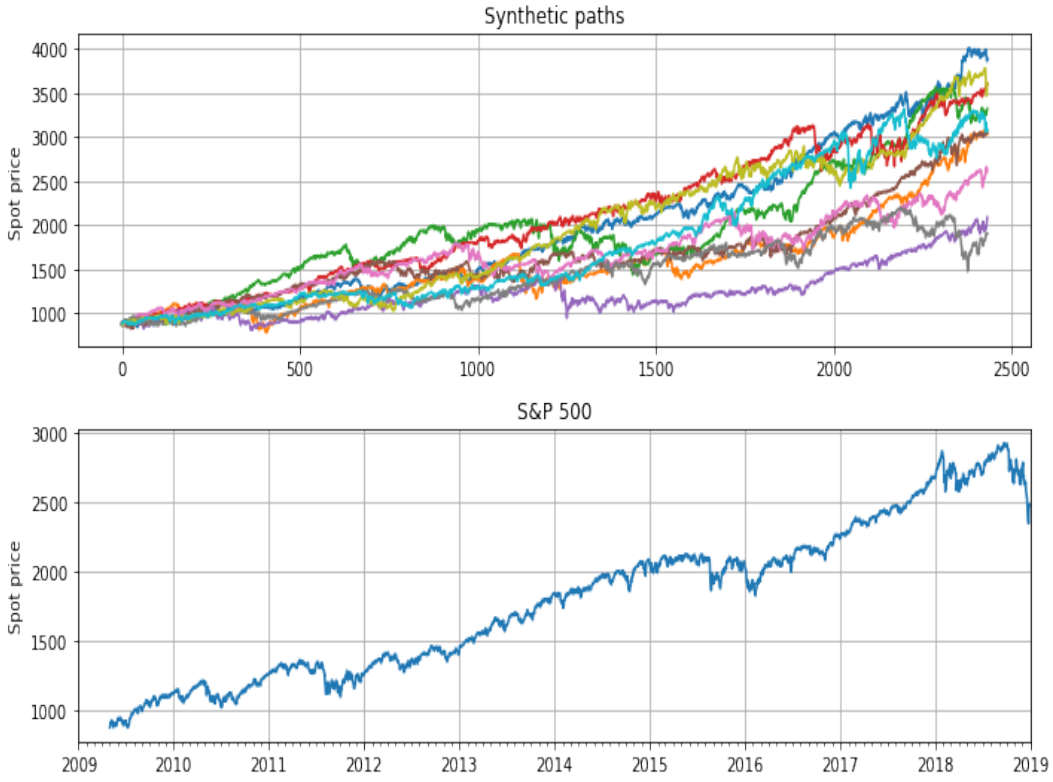


FIGURE 8 – Dix Trajectoires synthétiques.

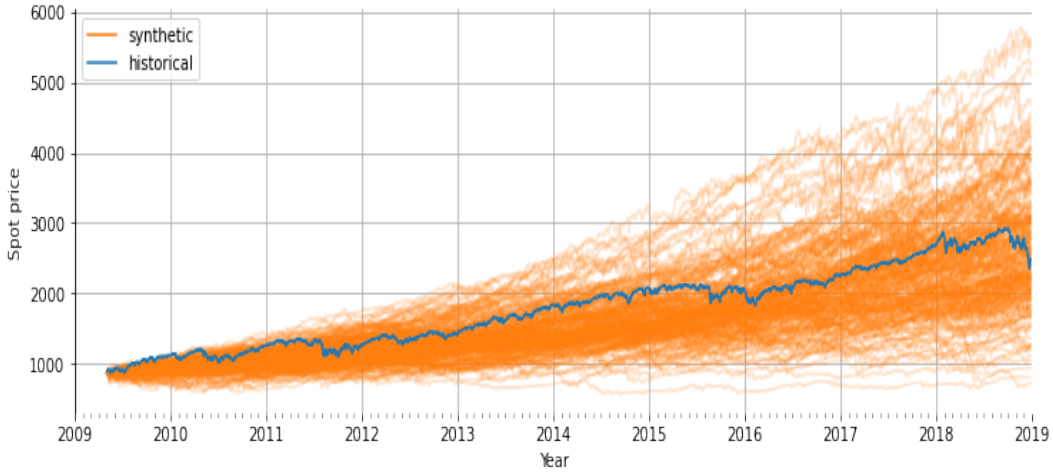
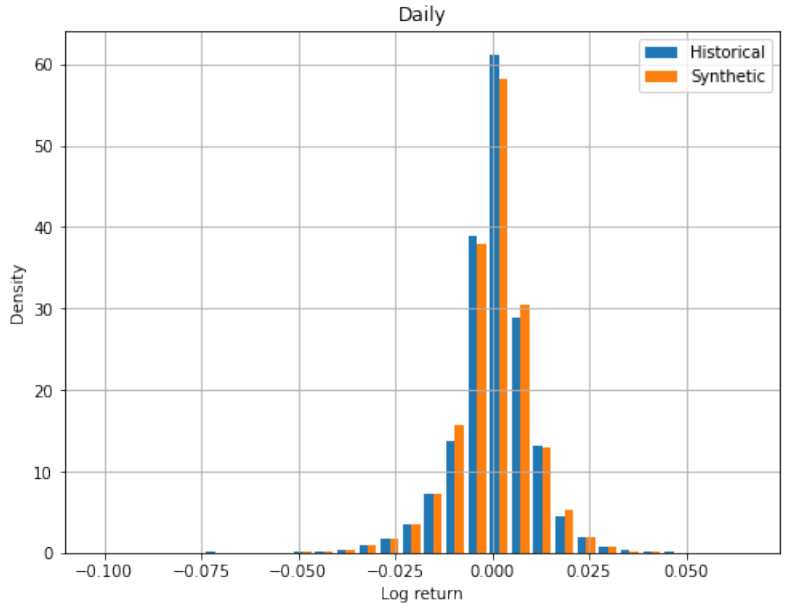
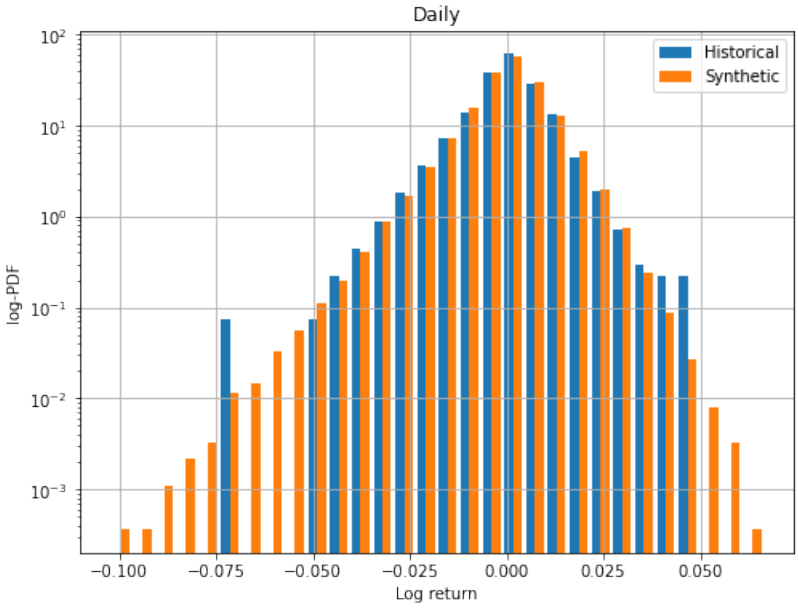


FIGURE 9 – Trajectoires synthétiques.

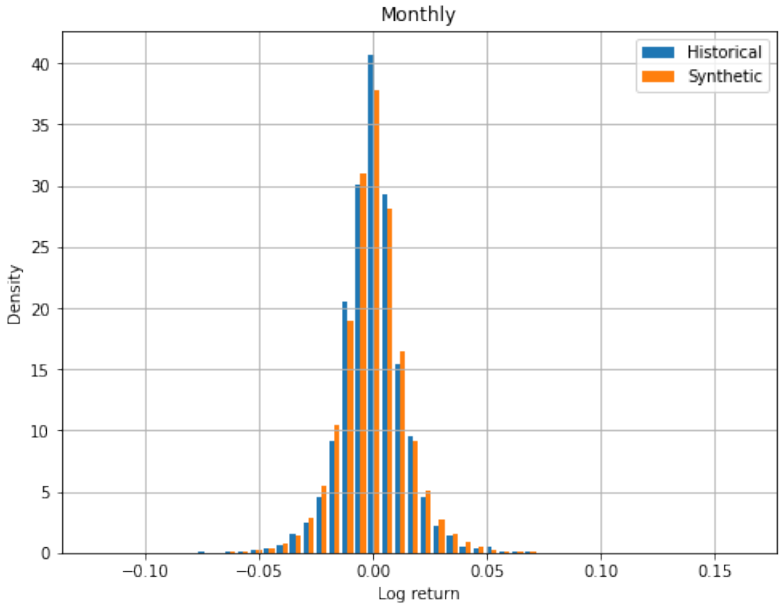


(a) Distribution des log-rendements synthétiques .

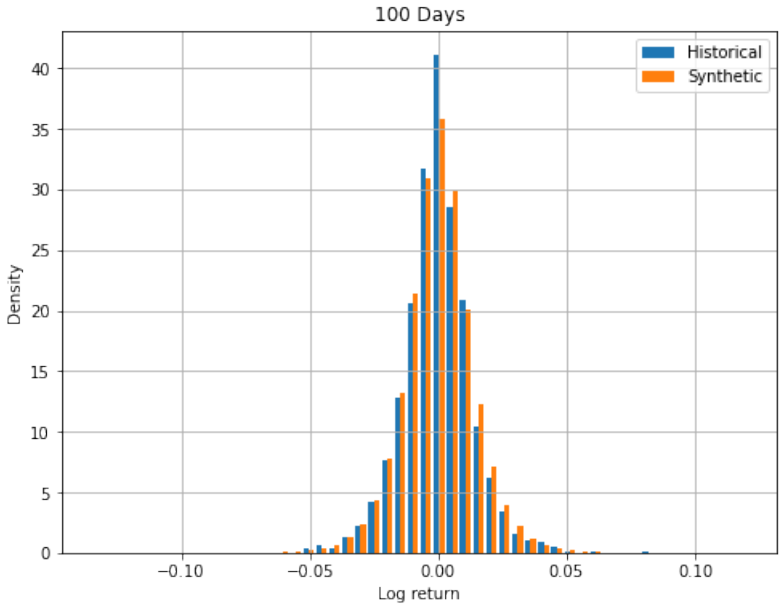


(b) Distribution journalières des log-rendements synthétiques (log densité).

FIGURE 10 – Données synthétiques transformation log-rendements pour la série *S&P 500* (2).



(a) Distribution des log-rendements mensuels synthétiques.



(b) Distribution des log-rendements synthétiques 100 jours.

FIGURE 11 – Données synthétiques transformation log-rendements pour la série S&P 500 (3).

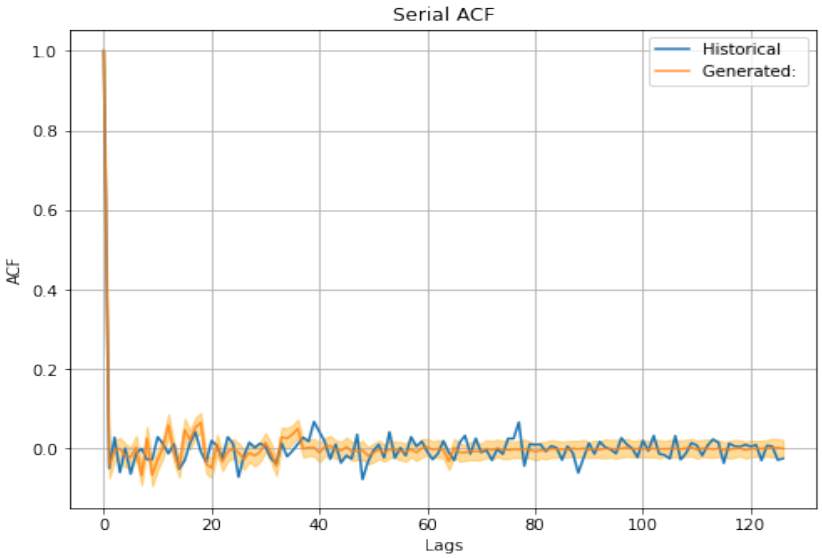


FIGURE 12 – Serial ACF des log-rendements.

C Annexe transformation log-rendements x Lambert W

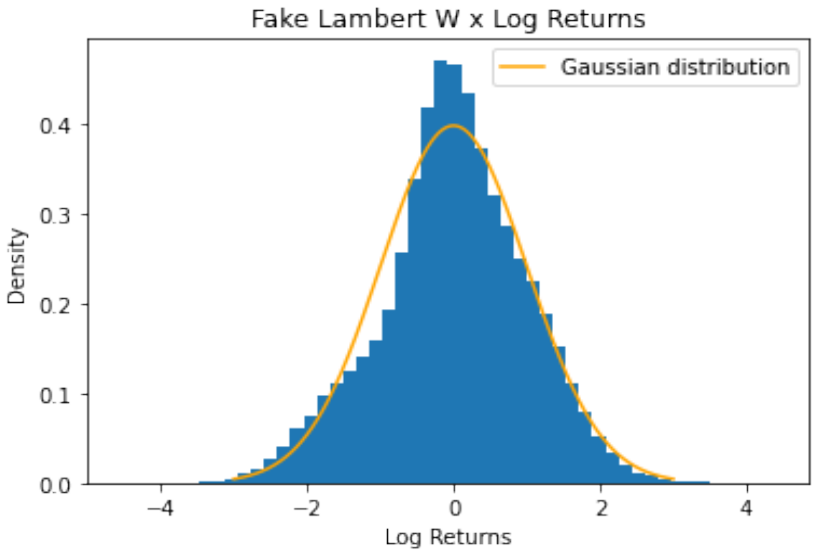


FIGURE 13 – Distribution des log-rendements synthétiques.

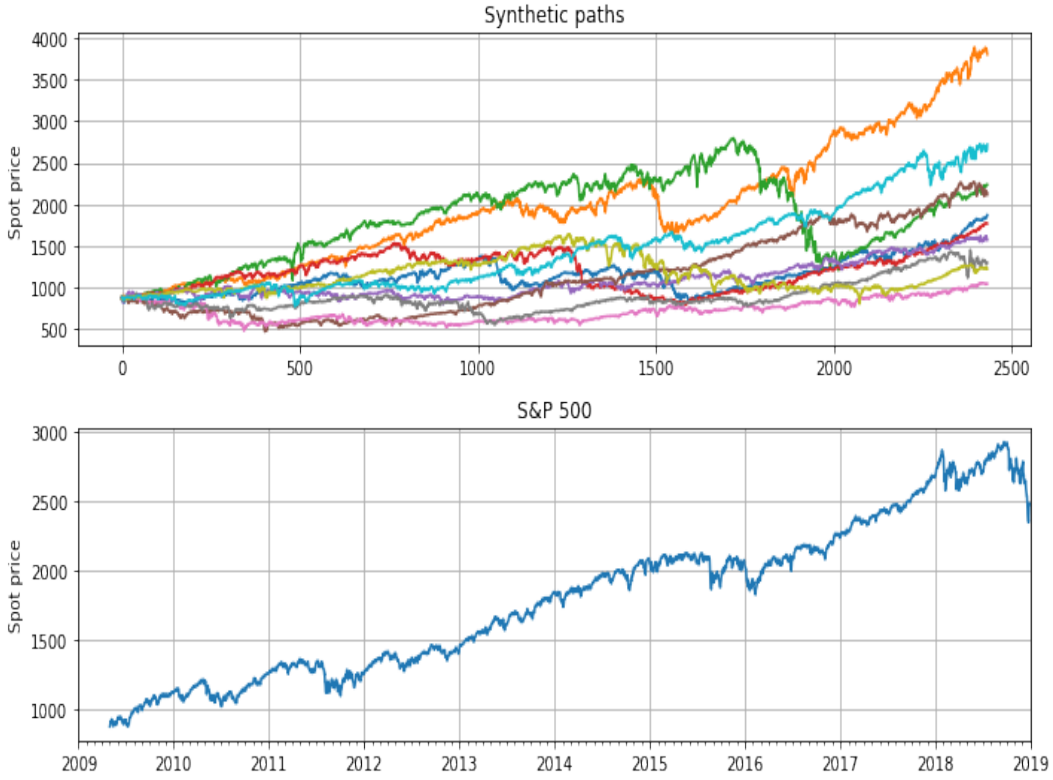


FIGURE 14 – Dix Trajectoires synthétiques.

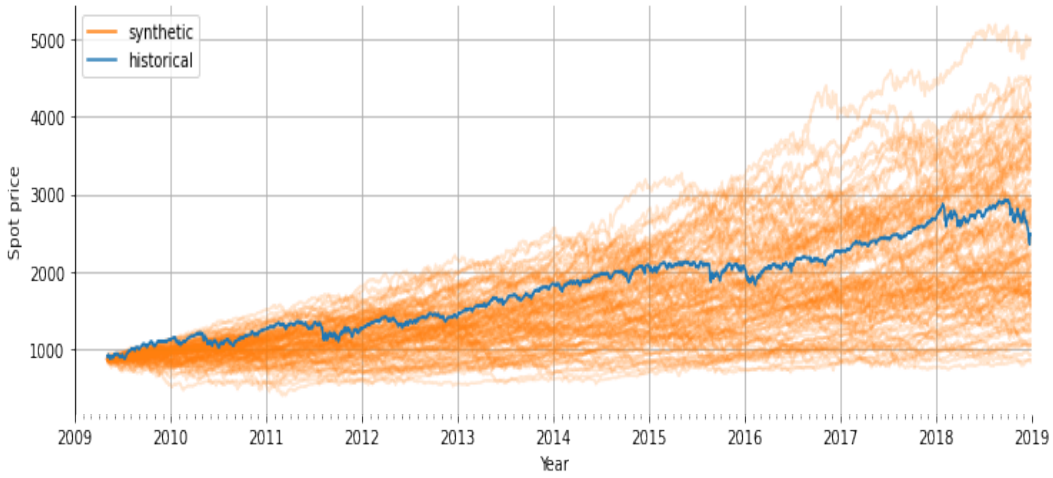


FIGURE 15 – Trajectoires synthétiques.

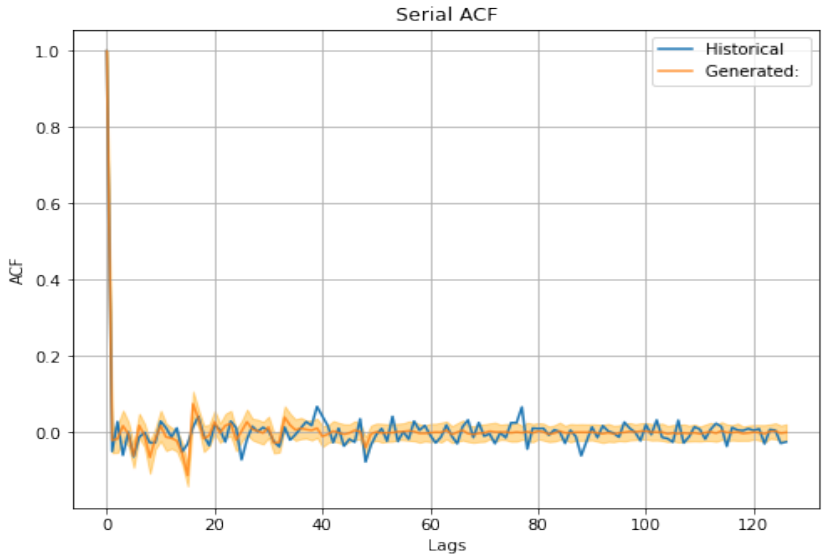
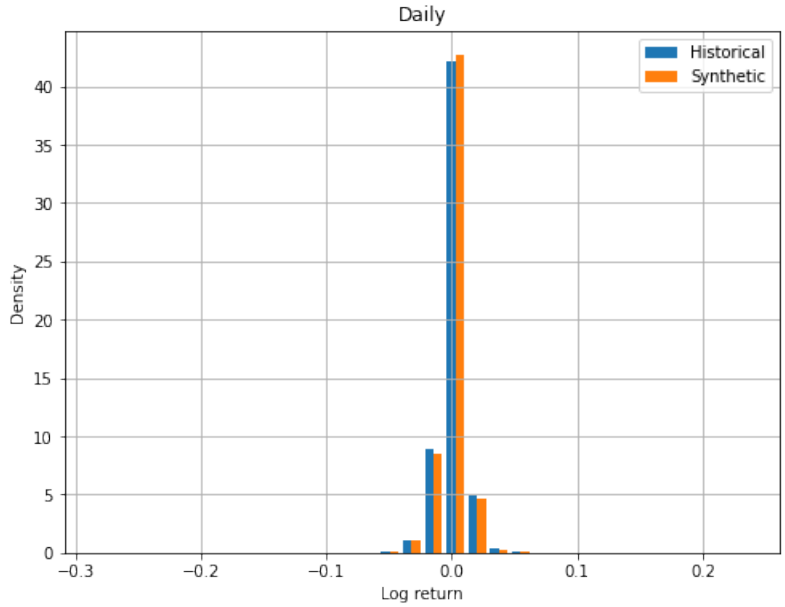
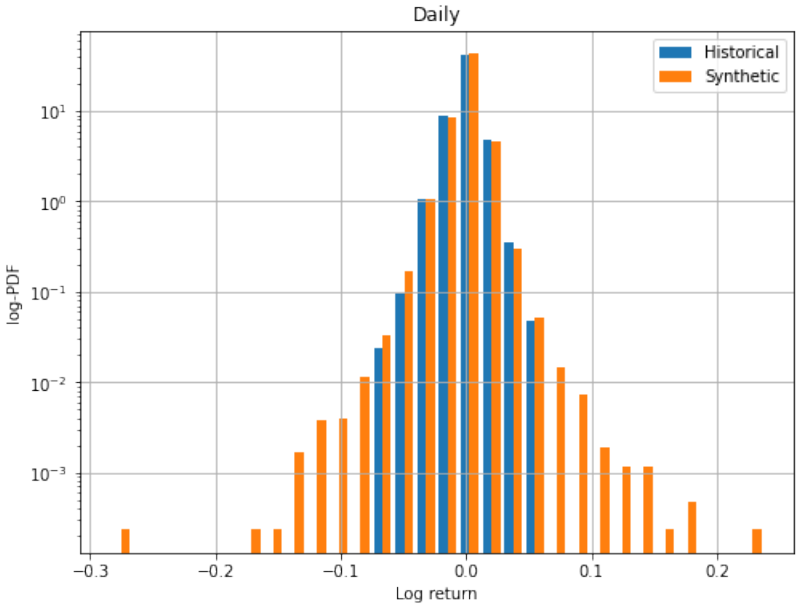


FIGURE 16 – Serial ACF des log-rendements pour la transformation log-rendements x Lambert W.

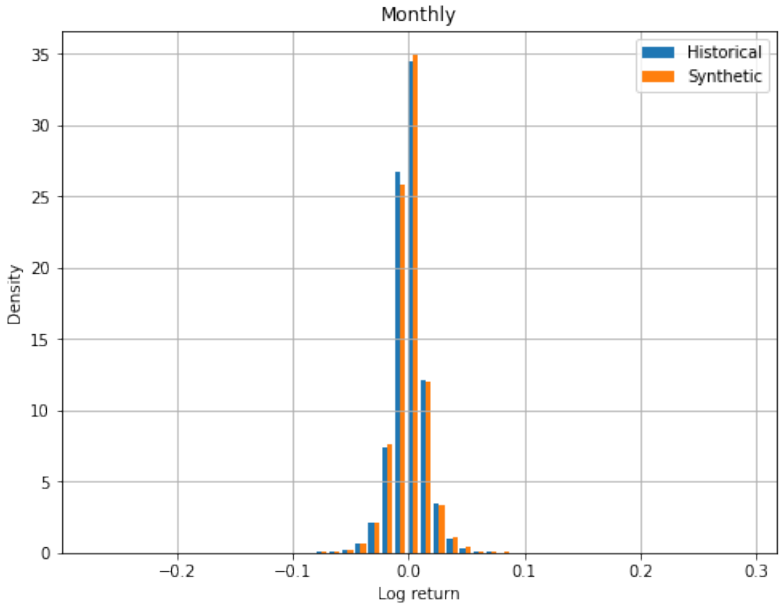


(a) Distribution des log-rendements synthétiques .

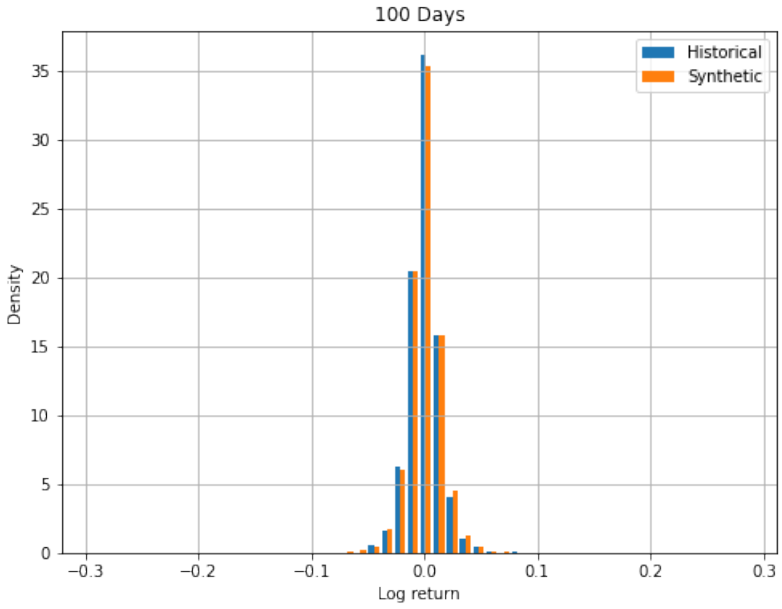


(b) Distribution journalières des log-rendements synthétiques (log densité).

FIGURE 17 – Données synthétiques transformation log-rendements x Lambert W pour la série S&P 500 (2).



(a) Distribution des log-rendements mensuels synthétiques.



(b) Distribution des log-rendements synthétiques 100 jours.

FIGURE 18 – Données synthétiques transformation log-rendements x Lambert W pour la série S&P 500 (3).