Statistiques sur générations WCRG et comparaisons avec d'autres modèles

Jean-Eric Campagne, Etienne Lempereur

31 août 2023

Table des matières

1. Introduction

Dans cette note est exposé une série de statistiques calculées sur des images générées par WCRG (Guth et al. 2023) en les comparant aux mêmes statistiques calculées non

seulement avec les images ayant servies au training du modèle, mais aussi pour un jeu de données avec d'autres modèles comme un DGAN et un modèle micro-locale.

Brièvement, les données utilisées sont les suivantes:

WL-1 : la motivation première était de comparer les images de Weak Lensing [1] issues d'un modèle Deep GAN de la référence (Mustafa et al. 2019) avec un modèle WCRG entrainé à partir des mêmes données d'entrainement que le réseau génératif. Donc, pour le training les auteurs nous ont fourni 100,000 images 128x128 pixels, ainsi qu'un lot d'images générées par leur réseau CosmoGAN. Ainsi, nous n'avons donc pas ré-entrainé un DGAN à cette occasion. Nous avons extrait 5,000 images [2] pour l'entrainement d'un modèle WCRG (noté par la suite WCRG-WL1). Nous détaillons un peu plus le traitement dans la section **??**.

WL-2 & ϕ_4 : A la suite, de la comparaison entre le modèle WCRG-WL1 et le DGAN, nous nous sommes interrogés sur ce que les modèles WCRG entrainés sur des données WL mentionnées dans le papier (Guth et al. 2023). Nous avons donc repris ces images d'entrainement et synthétisées pour calculer les statistiques. Idem nous avons utilisé les images d'entrainement et synthétisées par WCRG ϕ_4 ($\beta = \{0.5, 0.68, 0.76\}$ pour calculer les dites statistiques. Cela est détaillé dans la section **??**

Les statistiques utilisées sont principalement issues du code python *scattering_transform* développé par Siho Cheng [3] et utilisé entres autres dans les articles (Cheng & Ménard 2021; Cheng et al. 2023). Cela concerne, le power, le bi et le tri spectra et aussi les coefficients de corrélation C_{00} et C_{01}. Concernant les données de WL, la statistique du comptage de pics a également été utilisée.

Concernant la modélisation WCRG, le code python utilisé est celui du repository https://github.com/Elempereur/WCRG/ mentionné dans la publication afférente.

1. La référence utilise des résultats de la simulation N-corps GadGet2.
2. C'est la statistique utilisée dans la référence (Guth et al. 2023)
3. https://github.com/SihaoCheng/scattering_transform commit Fri Jun 2 11:14:53

2. Modélisation WCRG sur les données WL-1

2.1 Phase d'apprentissage

Avant l'entrainement du modèle WCRG, un traitement des données que l'on peut qualifier de "gaussianisation" a été effectué. En effet, sur la figure 1, on peut remarquer que la distribution des valeurs des pixels est mono-mode et peut être transformée en distribution normale par la méthode Cox-Box suivie d'une standardisation classique [4]. Le résultat de la transformation globale est illustré sur la figure 2. Cette transformation globale (inversible) est appliquée sur tous les lots de d'images (entrainement et génération) du DGAN.

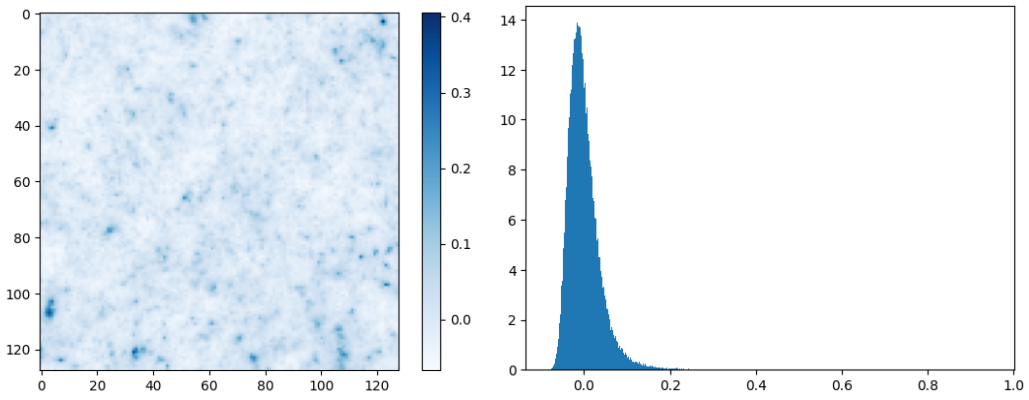

FIGURE 1 – Exemple d'une image du lot WL-1 (gauche) et la distribution afférente des valeurs de pixels (droite).

Concernant l'optimisation du modèle WCRG [5], la démarche est très proche de celle documenté sur le repository du code, en particulier le notebook *Learning_Models.ipynb* moyennant quelques adaptations décrites ci-après.

4. Si x est la valeur d'un pixel original alors primo par l'optimisation Cox-Box on trouve $\lambda^* \approx -0.22$, et $x' = (x^{\lambda^*} - 1)/\lambda^*$ suit une loi gaussienne, puis la standardisation classique permet d'obtenir une distribution normale $x' \sim \mathcal{N}(0, 1)$.

5. Notons que l'optimisation du modèle pourrait sans doute se passer de l'opération de gaussianisation préalable, mais l'idée était que cela pourrait aider *a priori* de ne pas avoir à traiter des queues de distributions trop asymétriques.

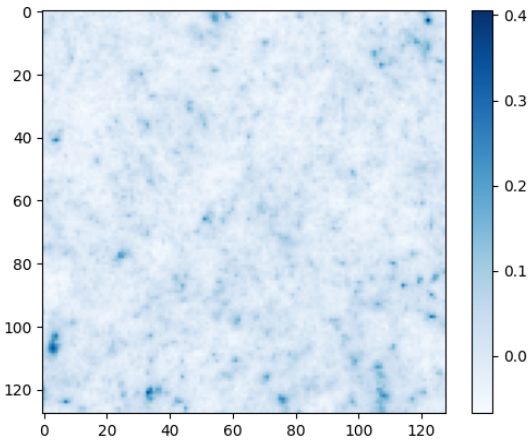

FIGURE 2 – Distribution des valeurs de pixels de la figure 1 après application d'une transformation Cox-Box de paramètre $\lambda \approx -0.22$ et d'une standardisation classique.

- Les ondelettes utilisées sont de la familles Debauchies d'ordre 4 avec un padding périodique.
- Pour l'ansatz $E(x_J)$ ($J = 7$, $L = 1$) a été utilisée la fonction

```
def ANSATZ_NoCondi(L,centers,sigma,shifts,shifts_sym = False):
    """Non conditionnal ansatz for direct estimation of energy, a scalar
    potential (sigmoids) + a quadratic potential

    Parameters:
    L (int): system size = L*L
    centers (tensor): position of the centers of the sigmoids
    sigma (tensor) : width of the sigmoids
    shifts (list of tuples) : spatial shifts for the quadratic potential,
    carefull (0,0) is already taken into account, do not add here
    shifts_sym (Bool) : if True, the shifts are not symetrized
    """
```

  avec 20 sigmoïdes régulièrement espacées entre le min et le max de la distribution des pixels $\approx [-100, 100]$, et par ailleurs shifts $= ()$). L'optimisation SGD est menée après avoir normalisé le hessien.
- A toutes les échelles $L > 1$, nous avons utilisé la fonction

```
def ANSATZ_Wavelet(W,L,centers,sigma,mode,shifts,shifts_sym = False):
""" Conditionnal ansatz for conditonal Energy \bar E(\bar x_j\vert x{j})
```

```
    estimation with a wavelet transform,with a scalar potential (sigmoids) + a
    quadratic potential

    Parameters:
    W (Wavelet) : Wavelet to perfom fast wavelet transform
    L (int): system size ( of x_{j-1}) = L*L
    centers (tensor): position of the centers of the sigmoids
    sigma (tensor) : width of the sigmoids
    shifts (list of tuples) : spatial shifts for the quadratic potential,
    carefull (0,0) is already taken into account, do not add here
    shifts_sym (Bool) : if True, the shifts are not symetrized
"""
```

Tout comme pour E_J, l'optimisation SGD est menée après avoir normalisé le hessien.

- A l'échelle $L = 2$, 30 sigmoïdes régulièrement espacées dans l'intervalle $[-75, 75]$ sont utilisées, tandis que extend = 1.0. Les autres paramètres sont mode = 'All' et shifts = ().
- A l'échelle $L = 4$, 30 sigmoïdes sont également utilisées mais cette fois réparties uniformément selon les quantiles avec $q_{min} = 10^{-5}$, $q_{max} = 1.0$ et extend = 1.0. De plus mode = 'All' tandis que shifts = ((1,0),(0,1),(1,1)) conjointement à shifts_sym = True.
- A l'échelle $L = 8$, on utilise 40 sigmoïdes réparties uniformément selon les quantiles avec $q_{min} = 10^{-5}$, $q_{max} = 1.0$ et extend = 1.0. Par ailleurs outre mode = 'All', nous avons utilisé la fonction suivante pour définir les shifts tels que shifts =shift_modif(L//4).

```
def shift_modif(n):
    shifts =[]
    for i in range(-n,n):
        for j in range(-n,n):
            if i==0 and j==0:
                pass
            else:
                shifts.append((i,j))
    return shifts
```

- A l'échelle $L = 16$, se sont 40 sigmoïdes uniformément réparties sur l'intervalle $[-30, 30]$ que l'on utilise avec extend = 1.0. Par ailleurs mode = 'All' et shifts

=shift_modif(L//4).

- A l'échelle $L = 32$, on utilise comme précédemment 40 sigmoïdes uniformément réparties sur l'intervalle $[-20, 20]$ (extend = 1.0). Les paramètres extend, mode et shifts sont les mêmes valeurs pour les deux premiers et définition pour le dernier que pour l'échelle précédente.
- A l'échelle $L = 64$, on utilise 30 sigmoïdes réparties uniformément selon les quantiles avec $q_{min} = 10^{-5}$, $q_{max} = 1.0$ et extend = 1.0. De plus si mode = 'All' comme précédemment, par contre shifts = shift_modif(L//8).
- Enfin à l'échelle $L = 128$ (taille des images d'entrainement), on utilise comme 30 sigmoïdes uniformément réparties sur l'intervalle $[-7.0, 5.5]$ (extend = 0.25), et si mode = 'All' comme précédemment, par contre shifts = shift_modif(L//16).
- Les paramètres qui sont particulièrement délicats affectant la synthèse des images aux différentes échelles, sont ceux qui définissent les centres des sigmoïdes (l'intervalle $[min, max]$ pour linspace_centers, ou $[q_{min}, q_{max}]$ pour quantile_centers. La méthode shift_modif a été utilisée in fine mais ne change pas vraiment les résultats par rapport à l'usage de shift_quad_Sym utilisée de prime abord. Les optimisations SGD sont souvent reprises pour un peu tuner les paramètres tels que le nombre d'époques num_epochs et le learning rate lr. Parfois deux étapes d'optimisation ont été pratiquées, notamment avec une valeur de lr plus petite pour la seconde étape.

Une fois les paramètres "optimisés" (par ex. les sigmoïdes), les différents potentiels appris $\bar{v}_j(x_{j-1})$ sont présentés sur la figure 3.

2.2 Phase de synthèse

Une fois le modèle appris, on peut regarder à chaque échelle la qualité des synthèses. En fait, à dire vrai, les paramètres qui définissent les sigmoïdes à toutes les échelles ont été ajustés à la main largement pour que les histogrammes de contrôles montre une bonne adéquation entre images d'origines et images de synthèse. La technique de sampling est en tout point identique à celle mise en œuvre pour l'article. Ce qui peut changer concerne le nombre de steps et la taille de chaque step. *Même si la suite pourrait paraître fastidieuse et répétitive, dans cette note une certaine exhaustivité s'impose néanmoins surtout que les codes n'ont pu être tournés sur notebooks partageables.*

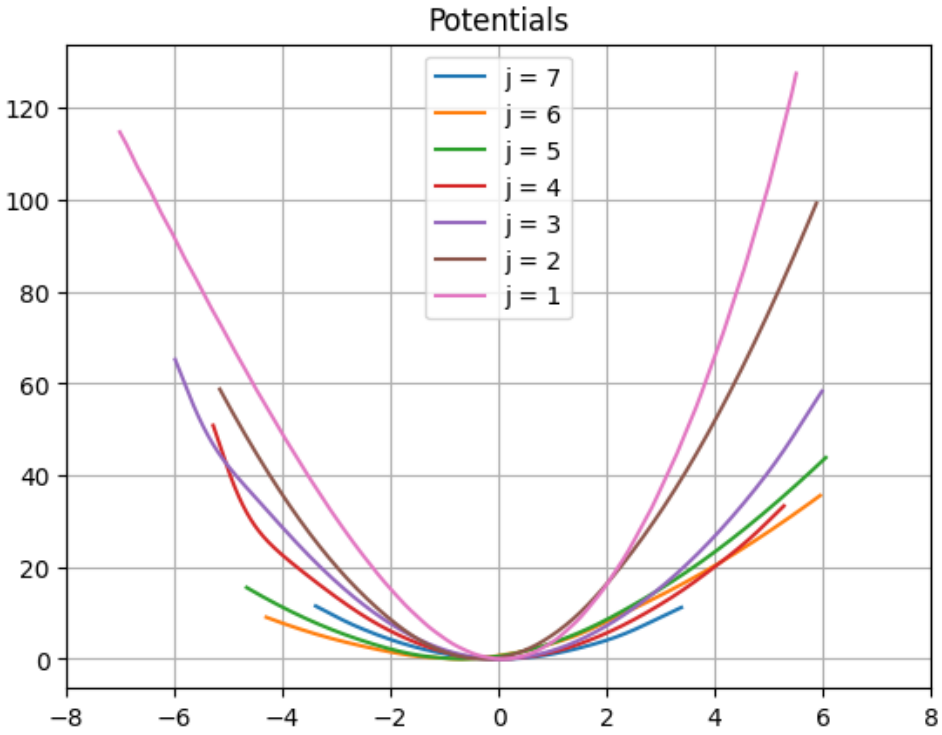

FIGURE 3 – Potentiels scalaires équivalents $\bar{v}_j$ appris à chaque échelle j pour les images de WL-1.

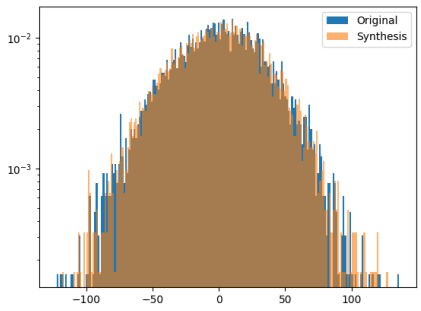

FIGURE 4 – Comparaison des valeurs de pixels à l'échelle $L = 1$.

A l'échelle $L = 1$, la comparaison des distributions de valeurs de pixels entre données d'entrainement et synthétisées est montrée sur la figure 4.

Aux échelles suivantes, outre la comparaison des distributions des valeurs de pixels des images originales et synthétisées, on peut également questionner la distribution des cartes de détails hautes fréquences (horizontales/verticales/diagonales) obtenues par décomposition en ondelettes.

Les résultats aux échelles $L \in [1, 128]$ sont présentés sur les figures de 5 à 11. Pour des raisons tenant à la taille mémoire des GPUs utilisés, le nombre de cartes générées jusqu'à l'échelle 16 incluse est de 5000, tandis que pour les échelles supérieures on ne peut disposer que de 500 cartes.

A l'échelle $L = 128$, on dispose de cartes synthétisées "finales" qui si tout se passe au mieux doivent ressembler comme deux gouttes d'eau aux cartes utilisées pour l'entrainement (ou cartes "originales"). La figure 12 donne quelques exemples de cartes originales et synthétisées. Il est indéniable que le modèle a appris quelque chose. Maintenant, dans l'article (Guth et al. 2023), il est montré non seulement des cartes synthétisées, la distribution des valeurs de pixels [6] mais aussi la comparaison du spectre de puissance. Ce dernier point a motivé l'usage de statistiques afin de caractériser les distributions des pixels et de comparer ces statistiques obtenues sur les cartes originales et les synthétisées. Ceci est présenté dans la section suivante.

6. nb. le modèle de l'article a été optimisé directement sur les cartes de WL-2 sans transformation préalable donc on ne peut comparer les modèles. Cela sera sans doute à revoir ultérieurement.

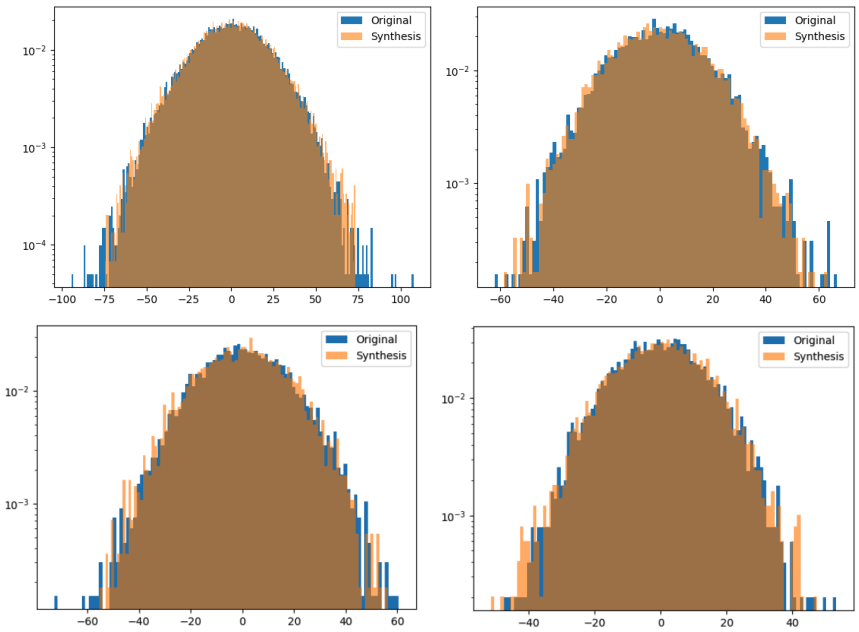

FIGURE 5 – Comparaison à l'échelle $L = 2$ entre cartes de type "originale/entrainement" et de type "synthètisée": distribution des valeurs de pixels de cartes "totales" (haut-gauche), de détails "H" (haut-droite), de détails "V" (bas-gauche) et de détails "D" (bas-droite).

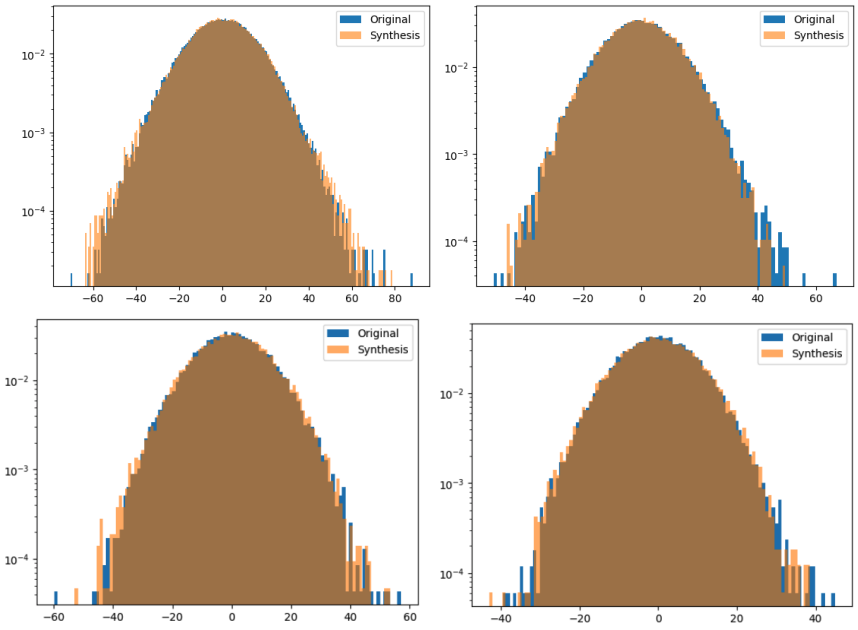

FIGURE 6 – Echelle $L = 4$: légende identique à la figure 5.

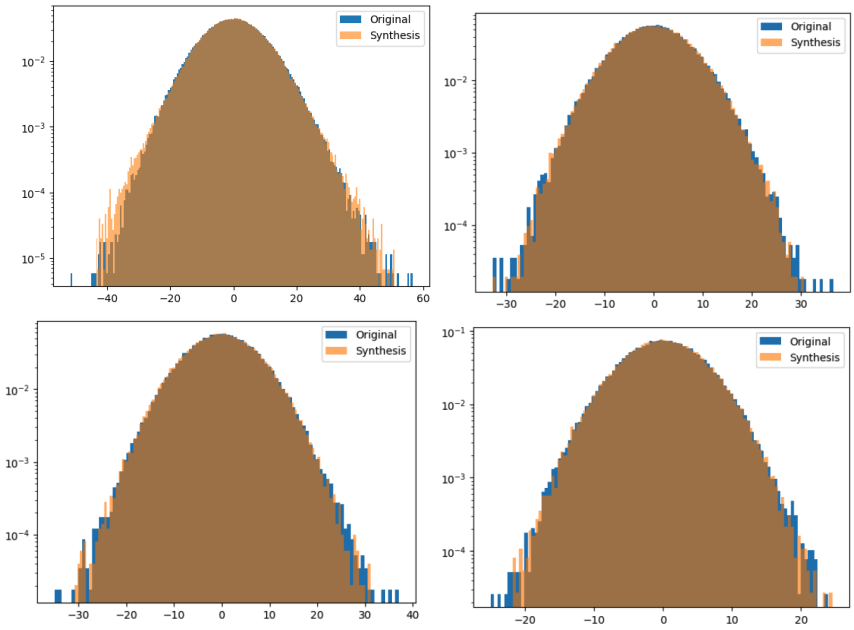

FIGURE 7 – Echelle $L = 8$: légende identique à la figure 5.

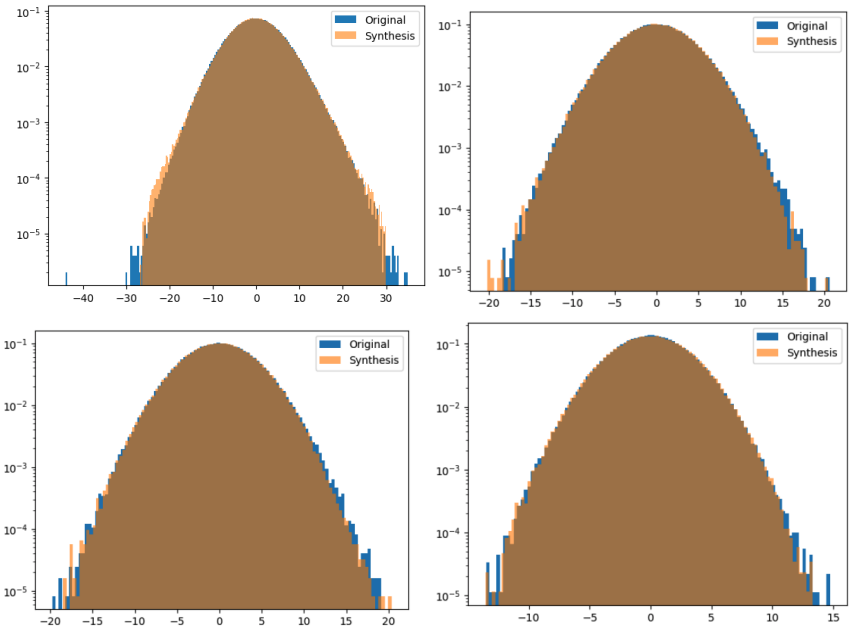

FIGURE 8 – Echelle $L = 16$: légende identique à la figure 5.

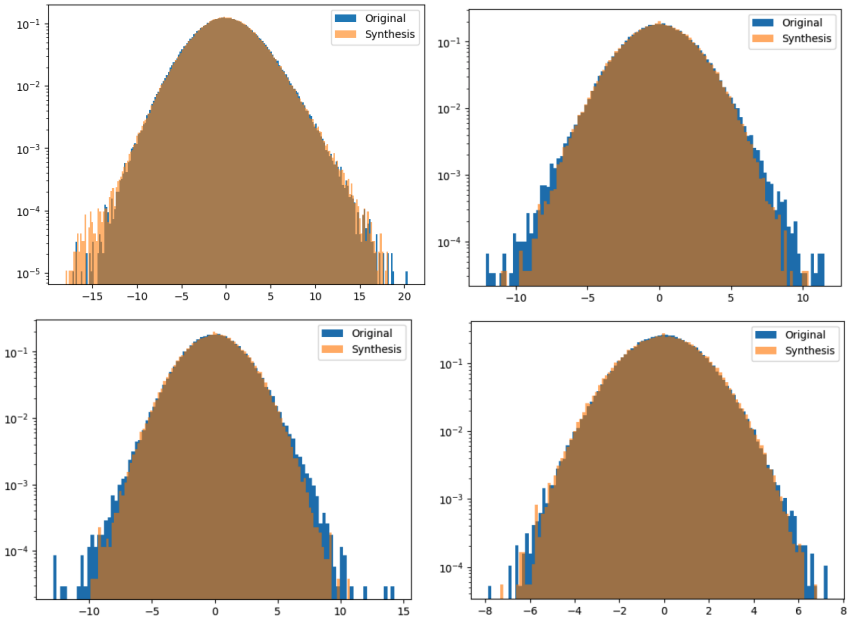

FIGURE 9 – Echelle $L = 32$: légende identique à la figure 5.

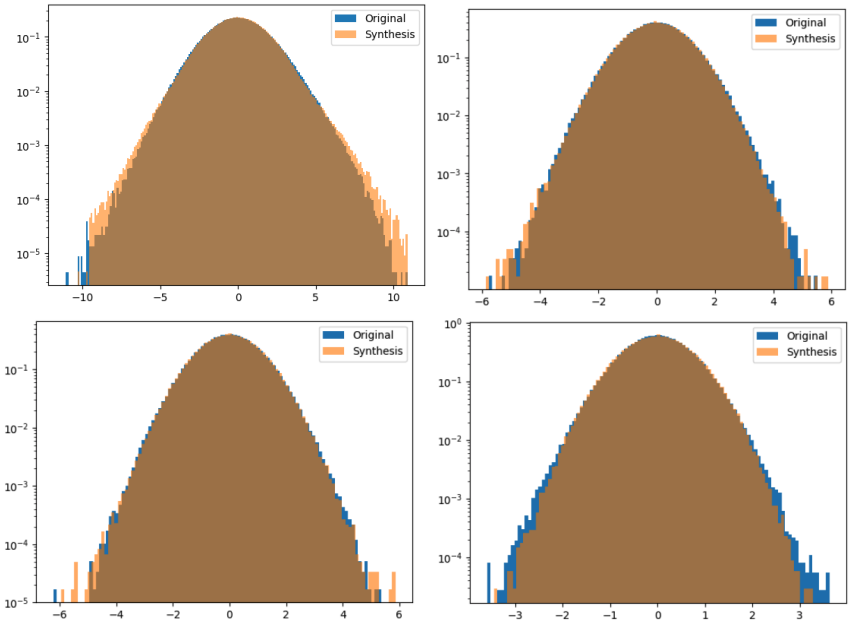

FIGURE 10 – Echelle $L = 64$: légende identique à la figure 5.

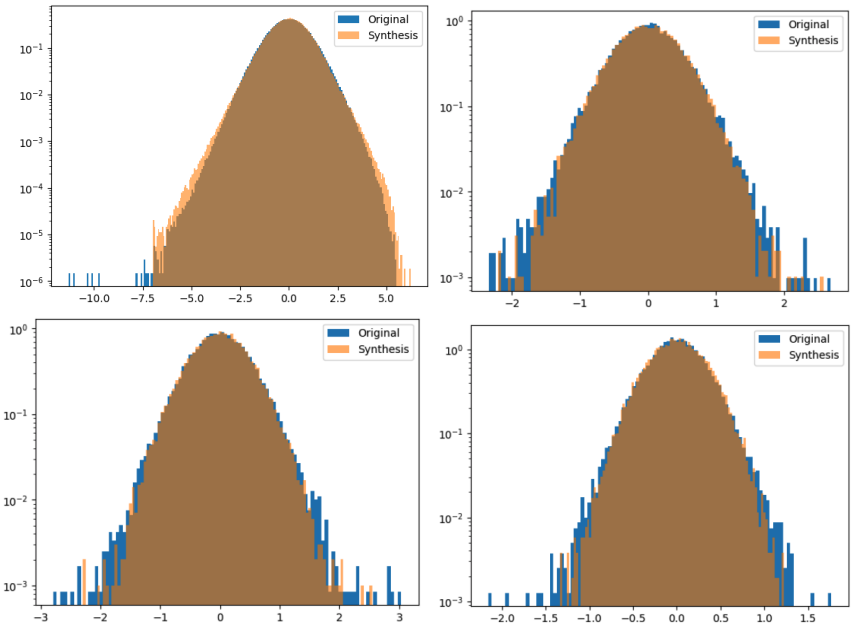

FIGURE 11 – Echelle $L = 128$ (la plus grande possible): légende identique à la figure 5.

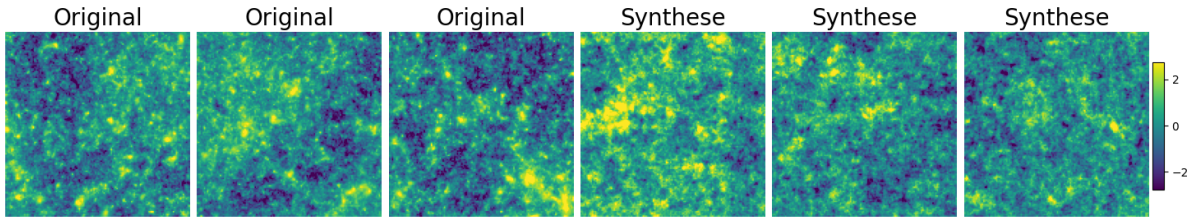

FIGURE 12 – Exemples de cartes WL-1 utilisées pour l'entrainement et synthétisées une fois le modèles WCRG optimisé.

2.3 Usage de statistiques d'ordre supérieures

Un certain nombre de statistiques ont été utilisées afin d'aller au-delà de l'aspect visuel jugeant de la qualité de la synthèse des cartes de WL-1. Comme mentionné dans l'introduction, en premier lieu il a été utilisé le code développé par Siho Cheng afin de calculer:

- les classiques spectre de puissance, *bi*-spectre et *tri*-spectre;
- mais aussi les versions *isotropiques* "C01_iso" et "C11_iso" des corrélations définies selon

$$C01 = \langle (I * \psi_2)(|I * \psi_1| * \psi_2)^* \rangle / factor \quad (1)$$

$$C11 = < (|I * \psi_1| * \psi_3)(|I * \psi_2| * \psi_3)^* \rangle / factor \quad (2)$$

avec I la carte de champ, ψ_i des ondelettes de Morlet à différentes échelles et orientations et la choix de normalisation est

$$factor = L2(I * \psi_1) * L2(I * \psi_2) \quad (3)$$

En pratique pour fixer un peu les idées l'obtention des différents spectres et coefficients "C01_iso" et "C11_iso" s'effectue schématiquement selon les étapes (si "maps" correspond à un lot de cartes):

```
bins_pw = 30
Nmaps,M,N = maps.shape
J = int(np.log2(min(M,N))) - 1
pw, kr  = scattering.get_power_spectrum(maps,
              k_range=np.logspace(0,np.log10(M/2*1.4),
```

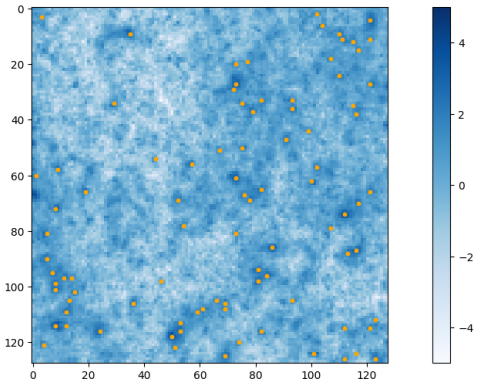

FIGURE 13 – Exemples de la recherche de pics au dessus d'un seuil ici pris à 2.

```
                bins_pw+1))
bi_calc = scattering.Bispectrum_Calculator(M,N,
                k_range=np.logspace(0,np.log10(M/2*1.4), J-1))
tri_calc = scattering.Trispectrum_Calculator(M,N,
                k_range=np.logspace(0,np.log10(M/2*1.4), J-1))

st_calc = scattering.Scattering2d(M, N, J=6, L=4)

cov_coef = st_calc.scattering_cov(maps)
select_and_index = get_scattering_index(J, L, normalization='P00', C11_criteria='j2>=
    j1')
c01 = cov_coef['C01_iso'][:,select_and_index['select_2_iso']]
c11 = cov_coef['C11_iso'][:,select_and_index['select_3_iso']]
```

Afin de satisfaire des contraintes de taille de mémoire, les spectres/coefficients ont été calculés sur des batchs de cartes et ensuite on en a tiré moyennes et écarts-types (barres d'erreur dans les histogrammes).

Concernant les données de WL, on a utilisé une statistique d'ordre supérieur qui s'est répandue dans certains articles récents comme par exemple la référence (Lanzieri et al. 2023) et les références incluses, concerne le comptage de "pics" au dessus d'un seuil. Pour se faire le code de D. Lanizieri et al. a été utilisé[7]. Un exemple pour un seuil de 2 est donné sur la figure 13.

7. https://github.com/LSSTDESC/DifferentiableHOS.git

3. Conclusion

4. Annexe

Références

Cheng, S. & Ménard, B. 2021, arXiv e-prints, arXiv:2112.01288

Cheng, S., Morel, R., Allys, E., Ménard, B., & Mallat, S. 2023, arXiv e-prints, arXiv:2306.17210

Guth, F., Lempereur, E., Bruna, J., & Mallat, S. 2023, arXiv e-prints, arXiv:2306.00181

Lanzieri, D., Lanusse, F., Modi, C., et al. 2023, arXiv e-prints, arXiv:2305.07531

Mustafa, M., Bard, D., Bhimji, W., et al. 2019, Computational Astrophysics and Cosmology, 6, 1