

Rapport de stage

Analyse statistique d'un écoulement turbulent



ALLAGLO Nikita

L3 Physique



Le 14 juillet 2023

Table des matières

1	Introduction	1
2	Contexte	1
2.1	Turbulence hydrodynamique	1
2.1.1	Nombre de Reynolds	1
2.1.2	Cascade turbulente	2
2.1.3	Dissipation anormale	3
2.1.4	Traitemet statistique	4
2.2	Géométrie et simulations numériques	4
2.2.1	Géométrie de von Kármán	4
2.2.2	Code SFEMaNS	4
2.3	Faller <i>et al.</i> , Journal of Fluid Mechanics 914 , 2 (2021)	5
2.3.1	Données analysées	5
2.3.2	Corrections et nouvelles simulations	5
3	Module d'analyse statistique	6
3.1	Code von-Karman-PostProcess	6
3.1.1	Valeurs moyennes	7
3.1.2	Lois de probabilité	8
3.1.3	Localisation des statistiques	9
3.2	Gestion de la mémoire	9
3.2.1	Restructuration des données	9
3.2.2	Lectures en parallèle	10
3.2.3	Tâches de travail (workloads) de type probabilité jointe	10
4	Résultats et discussion	12
4.1	Comparaison avec Faller <i>et al.</i> [1]	12
4.1.1	Comparaison avec les résultats de DNS	13
4.1.2	Comparaison avec les résultats expérimentaux	14
4.2	Comparaison entre \mathcal{D}_ℓ^u et \mathcal{D}_ℓ^I	14
4.2.1	Probabilités jointes	15
4.2.2	Corrélations	15
5	Conclusion	16
Annexes		17
A	\mathcal{D}^u vs ω	17
A.1	Full	17
A.2	Penal	18
A.3	Interior	19
A.4	Bulk	19
B	Statistiques inter-champs	20
C	Optimisation de Python sous GPUs	31
C.1	Travailler sur un cluster de GPUs	31
C.1.1	Architecture	31
C.1.2	CUDA	31
C.2	Optimisation de Python	32
C.2.1	CuPy	32
C.2.2	Benchmark	33
D	Matériel supplémentaire workloads type probabilité jointe	33
D.1	Détails techniques workloads GPUs	33
D.2	Déploiement parallèle sous CPUs	34

Table des figures

1	La turbulence vue comme une cascade de tourbillons des grandes vers les petites échelles	2
2	Géométrie de von Kármán	5
3	Densités de probabilité jointe pour les données expérimentales de [1]	6
4	Densités de probabilité jointe pour les données simulées de [1]	7
5	Statistiques de \mathcal{D}^I et ω aux échelles $\ell = 1.06\eta$ et $\ell = 26.5\eta$ avec pénalisation des turbines	13
6	Statistiques de \mathcal{D}^I et ω aux échelles $\ell = 1.06\eta$ et $\ell = 26.5\eta$ dans la région bulk	14
7	Évènements simultanés entre $\mathcal{D}^I (= \mathcal{D}^{DR})$ et \mathcal{D}^u	15
8	Corrélations entre $\mathcal{D}^I (= \mathcal{D}^{DR})$ et \mathcal{D}^u	15
9	Statistiques de \mathcal{D}^u et ω aux échelles $\ell = 1.06\eta$ et $\ell = 26.5\eta$ dans l'espace entier	17
10	Statistiques de \mathcal{D}^u et ω aux échelles $\ell = 1.06\eta$ et $\ell = 26.5\eta$ dans l'espace pénalisé des turbines	18
11	Statistiques de \mathcal{D}^u et ω aux échelles $\ell = 1.06\eta$ et $\ell = 26.5\eta$ dans l'espace extérieur aux turbines	19
12	Statistiques de \mathcal{D}^u et ω aux échelles $\ell = 1.06\eta$ et $\ell = 26.5\eta$ dans le sous-espace central au volume	20
13	$\mathcal{D}_{\ell=1.06\eta}^v$ vs ω	21
14	$\mathcal{D}_{\ell=1.06\eta}^v$ vs H	21
15	$\mathcal{D}_{\ell=1.06\eta}^v$ vs u	21
16	$\mathcal{D}_{\ell=1.06\eta}^v$ vs $\mathcal{D}_{\ell=1.06\eta}^u$	22
17	$\mathcal{D}_{\ell=1.06\eta}^v$ vs $\mathcal{D}_{\ell=26.5\eta}^u$	22
18	$\mathcal{D}_{\ell=1.06\eta}^v$ vs $\mathcal{D}_{\ell=26.5\eta}^I$	22
19	$\mathcal{D}_{\ell=1.06\eta}^v$ vs $\mathcal{D}_{\ell=26.5\eta}^v$	23
20	$\mathcal{D}_{\ell=26.5\eta}^v$ vs ω	23
21	$\mathcal{D}_{\ell=26.5\eta}^v$ vs u	23
22	$\mathcal{D}_{\ell=26.5\eta}^v$ vs $\mathcal{D}_{\ell=1.06\eta}^u$	24
23	$\mathcal{D}_{\ell=26.5\eta}^v$ vs $\mathcal{D}_{\ell=26.5\eta}^u$	24
24	$\mathcal{D}_{\ell=1.06\eta}^I$ vs H	24
25	$\mathcal{D}_{\ell=1.06\eta}^I$ vs u	25
26	$\mathcal{D}_{\ell=1.06\eta}^I$ vs $\mathcal{D}_{\ell=1.06\eta}^v$	25
27	$\mathcal{D}_{\ell=1.06\eta}^I$ vs $\mathcal{D}_{\ell=26.5\eta}^v$	25
28	$\mathcal{D}_{\ell=1.06\eta}^I$ vs $\mathcal{D}_{\ell=26.5\eta}^u$	26
29	$\mathcal{D}_{\ell=1.06\eta}^I$ vs $\mathcal{D}_{\ell=26.5\eta}^I$	26
30	$\mathcal{D}_{\ell=26.5\eta}^I$ vs u	26
31	$\mathcal{D}_{\ell=26.5\eta}^I$ vs $\mathcal{D}_{\ell=26.5\eta}^v$	27
32	$\mathcal{D}_{\ell=26.5\eta}^I$ vs $\mathcal{D}_{\ell=1.06\eta}^u$	27
33	$\mathcal{D}_{\ell=1.06\eta}^u$ vs u	27
34	$\mathcal{D}_{\ell=26.5\eta}^u$ vs u	28
35	$\mathcal{D}_{\ell=26.5\eta}^u$ vs $\mathcal{D}_{\ell=1.06\eta}^u$	28
36	ω vs u	28
37	H vs ω	29
38	H vs u	29
39	H vs $\mathcal{D}_{\ell=26.5\eta}^v$	29
40	H vs $\mathcal{D}_{\ell=1.06\eta}^u$	30
41	H vs $\mathcal{D}_{\ell=26.5\eta}^u$	30
42	H vs $\mathcal{D}_{\ell=26.5\eta}^I$	30
43	Benchmarks CPU vs GPU sur Python	33

Liste des tableaux

1	Paramètres décrivant les données utilisées dans l'article [1]	6
---	---	---

Liste des scripts

1	Script 1 : Chunking des données délayées sur 14 threads	10
2	Script 2 : Workloads séquentiels sous GPUs	10
3	Script 3 : Recombinaison des probabilités jointes	12
4	Script 4 : Archétype de script de soumission d'un job GPU	31

1 Introduction

La turbulence est une branche de recherche en pleine activité en mécanique des fluides. Expérimentée au quotidien dans une fumée de cigarette par exemple, du fait de son origine hautement non-linéaire, elle demeure un casse-tête auquel se heurtent et physiciens et mathématiciens. Dans une ère où les théories statistiques sont omniprésentes en recherche fondamentale, l'approche la plus rapide permettant de mieux éclairer les *voies sombres de la turbulence* est précisément de passer par l'analyse statistique. En couplant ainsi ces méthodes statistiques à des théories novatrices de transferts inter-échelles [2], on aboutit à un cadre de travail idéal pouvant potentiellement démêler les phénomènes complexes intrinsèques aux équations régissant la mécanique des fluides. Le présent stage s'est en conséquence articulé autour de trois axes majeurs : dans un premier temps, nous avons saisi les rudiments de la théorie, en partant des idées phénoménologiques de Kolmogorov pour parvenir aux liens entre turbulence et irrégularité du champ de vitesse. Dans un second temps, assisté par les infrastructures de calcul haute performance offertes par le *Laboratoire Interdisciplinaire des Sciences du Numérique* où nous avons effectué notre stage, nous avons développé un module de traitement des données permettant d'analyser efficacement des bases de données d'écoulement turbulent de l'ordre du TeraByte. En dernier lieu, à l'aide de ce module, nous avons pu mettre en évidence quelques inexactitudes hypothétisées dans les résultats décrits dans un article écrit par l'équipe encadrante [1] (issu de la thèse d'un ancien doctorant).

2 Contexte

2.1 Turbulence hydrodynamique

2.1.1 Nombre de Reynolds

La turbulence est un régime d'écoulement caractérisé par un champ de vitesse chaotique. Quiconque n'a jamais pris l'avion a probablement déjà expérimenté le phénomène. Bien que les équations régissant le mouvement des fluides soient déterministes, un écoulement turbulent possède une très forte sensibilité aux conditions initiales - ce qui rend le système quasi-imprédictible à temps long. On sait qu'il existe encore des résultats non démontrés à ce jour en mécanique des fluides (cf. problème d'existence et de régularité des solutions des équations de Navier-Stokes de l'Institut Clay) et la recherche en turbulence n'échappe pas à cette loi : Richard Feynman est allé jusqu'à qualifier la turbulence comme le plus important problème non-réolu de la physique classique. On s'attend naturellement à ce que, face à un état si complexe, on puisse chercher à catégoriser les écoulements par leur nature turbulente ou non. Un paramètre de contrôle existe à ce propos : *le nombre de Reynolds* qui prend forme en considérant les équations de Navier-Stokes adimensionnées. Pour un fluide incompressible, nous avons :

$$\partial_t u_i + u_j \partial_j u_i = -\frac{1}{\rho} \partial_i p + \nu \partial_j \partial_j u_i + f_i \quad (1)$$

$$\partial_i u_i = 0, \quad (2)$$

où \mathbf{u} est la vitesse du fluide, ρ sa densité de masse, p la pression, ν la viscosité cinématique et \mathbf{f} la résultante des forces externes. En incorporant la longueur caractéristique L et vitesse caractéristique U de l'écoulement de telle sorte que $\partial_i = \frac{1}{L} \partial'_i$, $u_i = U u'_i$, $p = \rho U^2 p'$ ¹ et $\partial_t = \frac{U}{L} \partial'_t$, on obtient (pour Navier-Stokes) :

$$\frac{U^2}{L} \partial'_t u'_i + \frac{U^2}{L} u'_j \partial'_j u'_i = -\frac{U^2}{L} \partial'_i p' + \nu \frac{U}{L^2} \partial'_j \partial'_j u'_i + f_i \quad (3)$$

$$\Rightarrow \partial'_t u'_i + u'_j \partial'_j u'_i = -\partial'_i p' + \frac{1}{\text{Re}} \partial'_j \partial'_j u'_i + \frac{L}{U^2} f_i \quad (4)$$

avec Re le nombre de Reynolds défini comme

$$\boxed{\text{Re} = \frac{UL}{\nu} \sim \frac{|(\mathbf{u} \cdot \nabla) \mathbf{u}|_{\text{caract.}}}{|\nu \nabla^2 \mathbf{u}|_{\text{caract.}}}}. \quad (5)$$

Lorsque $\text{Re} \ll 1$, l'équation devient quasi-linéaire puisque le terme en laplacien domine. Dans ce cas, on retrouve un semblant d'équation de diffusion $\frac{\partial X}{\partial t} \sim \nu \frac{\partial^2 X}{\partial t^2}$, on comprend alors bien que la viscosité agit comme un agent **diffusif**; il est impossible qu'il y ait apparition de tourbillons dans l'écoulement. Dans le cas où $\text{Re} \gg 1$, le terme non-linéaire est dominant. On peut alors montrer que si, avec certaines conditions initiales, l'écoulement était parallèle pour $\text{Re} \ll 1$, il y

1. une pression est une énergie par unité de volume

a plutôt apparition de tourbillons à différentes échelles pour $\text{Re} \gg 1$ et donc un chaos progressif. La dérivée particulaire permet de comprendre ce phénomène : c'est la vitesse elle-même qui "se transporte" d'où la non-linéarité. C'est ce caractère alors qui mène au chaos par sensibilité locale. Nous verrons dans la section §2.1.2 suivante que c'est précisément une hiérarchie d'échelles qui permet de comprendre - avec les mains - le concept de turbulence.

2.1.2 Cascade turbulente

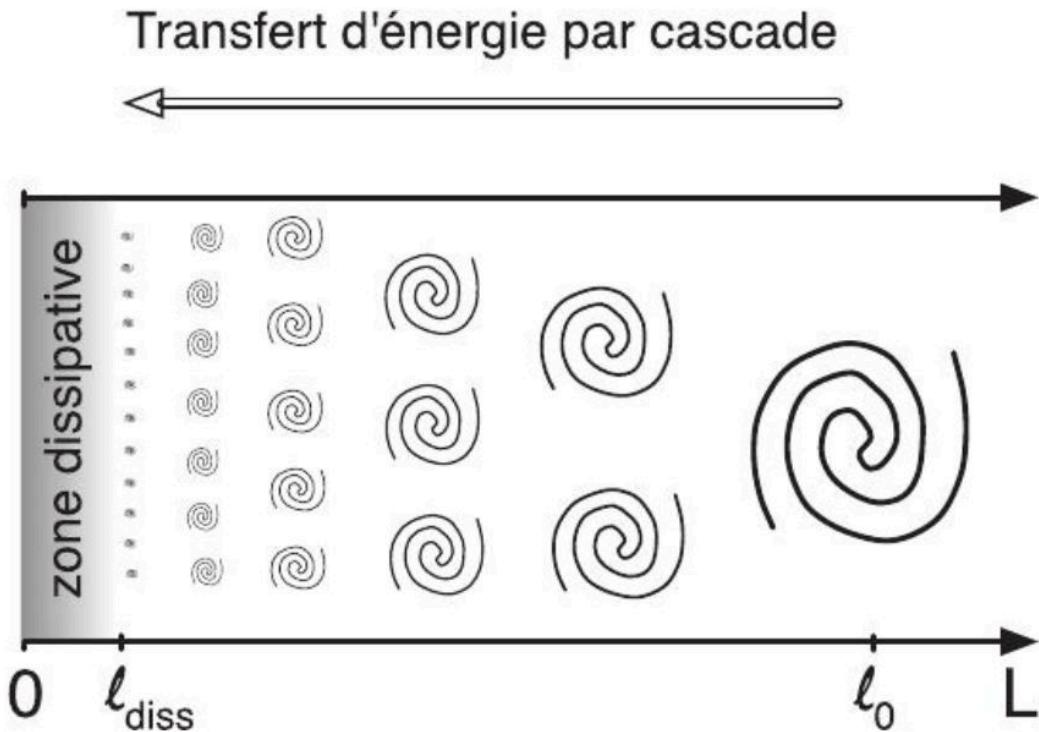


FIG. 1. La turbulence vue comme une cascade de tourbillons des grandes vers les petites échelles. $\ell_{\text{diss}} \ll \ell \ll \ell_0$ représente la zone inertie. Figure reprise de [3].

La turbulence peut être comprise par des arguments phénoménologiques. Lewis Fry Richardson fut le premier à mettre en évidence ce que l'on nomme aujourd'hui la *cascade turbulente* sous les vers suivants :

*Big whirls have little whirls that feed on their velocity,
and little whirls have lesser whirls and so on to viscosity.*

Comme le schématisé la figure 1, il existe une cascade énergétique permettant le transfert d'énergie des gros tourbillons aux plus petits - à condition d'être dans une échelle suffisamment grande pour que la viscosité ne puisse pas encore dissiper et suffisamment petite pour être aveugle aux phénomènes à grande échelle (l'injection d'énergie dans le système par exemple) : on parle de *zone inertie*. C'est ainsi dans cette zone que la physique non-linéaire demeure influente. Dans la suite de ce rapport, la limite inférieure des échelles de la zone inertie sera appelée *échelle de Kolmogorov* et notée η . L'idée de cascade de tourbillons repose en toute rigueur sur une vision très simplifiée puisque, dans la réalité, on observe plutôt des tubes de vorticité imbriqués les uns dans les autres[3]. Kolmogorov a développé une série d'articles dans le courant des années 1940 permettant d'appuyer cette idée de cascade de turbulence [4]. Il a notamment obtenu des lois universelles (indépendantes de la viscosité) **valables sur toute la zone inertie** pour une turbulence statistiquement homogène et isotrope vérifiées par l'expérience : on a en idée les densités spectrales d'énergie qui affichent une dépendance en $k^{-5/3}$ sur la zone inertie. Même s'il existe toute une théorie permettant de démontrer cela dans le cas d'une turbulence statistiquement homogène et isotrope, on peut obtenir ce résultat par simple analyse dimensionnelle. Nous avons vu comment, à partir de modèles jouets, on pouvait aboutir à certaines propriétés remarquables de turbulence. Il s'agit désormais de pouvoir généraliser à une turbulence non homogène et anisotrope, pouvant expliquer des propriétés émergentes du chaos comme l'intermittence mais surtout pouvant capter des singularités.

2.1.3 Dissipation anormale

La question de la régularité/des singularités est une branche de recherche très active en mécanique des fluides (il suffit de penser au prix de l’Institut Clay). Le problème se pose notamment lorsque l’on prend en considération la *0ème loi de la turbulence*.

Zéroième loi de la turbulence

La puissance injectée moyenne (ou le taux moyen de variation de l’énergie cinétique) au sein d’un fluide tend vers une constante non-nulle à viscosité rigoureusement nulle.

On parle d’anomalie dissipative (ou de dissipation anormale) car la viscosité est (normalement) la seule grandeur pouvant dissiper de l’énergie : s’il n’existe rien qui se “nourrit” de l’énergie injectée, comment expliquer un taux moyen de dissipation non-nul ? Cela suggère l’existence d’un autre mécanisme intervenant dans les écoulements très turbulents ($\text{Re} \rightarrow \infty$). À partir des équations de Navier-Stokes, on peut démontrer l’expression exacte du taux moyen de dissipation dans un fluide [3] :

$$\frac{\partial \langle E \rangle}{\partial t} = \rho v \langle \omega^2 \rangle, \quad (6)$$

avec $\langle E \rangle$ l’énergie cinétique moyenne par unité de volume et $\boldsymbol{\omega} = \text{rot}(\mathbf{u})$ la vorticité. Si $v \rightarrow 0$, la 0ème loi de la turbulence implique que $\langle \omega^2 \rangle \rightarrow \infty$. Un rotationnel infini provient d’un champ de vitesse irrégulier. Les irrégularités seraient alors responsables de l’anomalie dissipative : c’est la conjecture d’Onsager. On comprend alors tout l’intérêt de développer une théorie incorporant de façon fondamentale le caractère singulier des grandeurs physiques comme la vitesse ou la vorticité.

Cette théorie est basée sur une formulation faible des équations décrivant les fluides prenant précisément en compte ce qui est *non-lisse* [2, 5]. On s’attend bien à rencontrer des concepts inhérents à la théorie des distributions, les fonctions-test en particulier seront vitales dans cette formulation faible. Les fonctions-test seront définies comme les $\varphi > 0 \in \mathcal{C}^\infty$ à support compact dans \mathbb{R}^3 et s’intégrant à l’unité. On peut démontrer alors que, pour ℓ réel, $\varphi_\ell(\xi) := \frac{1}{\ell^3} \varphi\left(\frac{\xi}{\ell}\right)$ converge (au sens des distributions) vers un delta de Dirac. On comprend alors que le principal but de ces fonctions est de lisser les quantités irrégulières par une opération de convolution. Pour une échelle ℓ , on définit pour un champ (scalaire ou vectoriel) X sa version *filtrée* comme :

$$\bar{X}^\ell(\mathbf{r}) = \int_{\mathbb{R}^3} \varphi_\ell(\xi) X(\xi + \mathbf{r}) d^3\xi \quad (7)$$

Ces outils permettent alors de caractériser les transferts d’énergie inter-échelle : on applique tout d’abord l’opération (7) aux équations de Navier-Stokes puis on prend le produit scalaire du résultat obtenu avec la quantité $\rho \frac{\mathbf{u}}{2}$. Dans un second temps, on prend le produit scalaire de $\rho \bar{\mathbf{u}}^\ell$ avec les équations de Navier-Stokes non filtrées (1). En sommant les deux résultats, on obtient l’équation d’évolution de l’énergie cinétique pseudo-filtrée $\tilde{E}_\ell := \rho \frac{\mathbf{u} \cdot \bar{\mathbf{u}}^\ell}{2}$ [6] :

$$\partial_t \tilde{E}_\ell + \nabla \cdot J_\ell^{\text{NS}} + \mathcal{D}_\ell^v + \mathcal{D}_\ell^u = \mathcal{P}_\ell, \quad (8)$$

avec la définition suivante des termes de dissipation :

$$J_\ell^{\text{NS}} = \frac{\rho}{2} ((\mathbf{u} \cdot \bar{\mathbf{u}}^\ell) \mathbf{u} - v \nabla (\mathbf{u} \cdot \bar{\mathbf{u}}^\ell)) + \frac{1}{2} (\bar{\mathbf{u}}^\ell p + \mathbf{u} \bar{p}^\ell), \quad (9)$$

$$\mathcal{P}_\ell = \frac{\rho}{2} (\mathbf{u} \cdot \bar{\mathbf{f}}^\ell + \bar{\mathbf{u}}^\ell \cdot \mathbf{f}), \quad (10)$$

$$\mathcal{D}_\ell^v = \rho v \partial_i \bar{u}_j^\ell \partial_i u_j, \quad (11)$$

$$\mathcal{D}_\ell^u = \frac{\rho}{2} (\mathbf{u} \cdot \bar{u}_i \partial_i \mathbf{u}^\ell - u_i \mathbf{u} \cdot \partial_i \bar{\mathbf{u}}^\ell). \quad (12)$$

Tous les termes de l’équation (8) mesurent les transferts entre l’échelle ℓ et toutes les échelles hiérarchiquement supérieures. J_ℓ^{NS} est un courant de déplacement d’énergie (du fait de la divergence) ; il n’est donc pas vraiment intéressant pour tout ce qui a trait à la dissipation. \mathcal{D}_ℓ^v est la dissipation visqueuse. \mathcal{D}_ℓ^u est la dissipation anormale : ce terme ne disparaît pas dans la limite des échelles infinitésimales [5]. Notez toutefois que cette dérivation est très récente : à la place de \mathcal{D}_ℓ^u , on considère plutôt dans la littérature la dissipation inertuelle notée \mathcal{D}^I ou \mathcal{D}^{DR} . Ces dernières sont reliées

par un terme de divergence :

$$\mathcal{D}_\ell^I = \frac{\rho}{2} \boldsymbol{\nabla} \cdot (\overline{\mathbf{u} u^2}^\ell - \overline{u^2} \overline{\mathbf{u}}^\ell) + \mathcal{D}_\ell^u \quad (13)$$

Nous expliquons en section §2.3.2 les raisons d'un tel changement.

Avant de mentionner ce que nous faisons explicitement avec toute cette théorie, il est bon de rappeler l'essence aléatoire de l'écoulement turbulent. On ne peut pas juste étudier les champs d'intérêt du point de vue de l'espace ou du temps ; il est nettement plus pertinent de traiter l'ensemble du point de vue statistique.

2.1.4 Traitement statistique

Il faut imaginer reproduire la même expérience N fois, au lieu de considérer une grandeur $X(\mathbf{r}, t)$, on considère plutôt la moyenne d'ensemble des N réalisations $\{X_i(\mathbf{r}, t)\}_i$:

$$\langle X(\mathbf{r}, t) \rangle_N = \frac{1}{N} \sum_i X_i(\mathbf{r}, t). \quad (14)$$

Du point de vue numérique, en raison du coût des ressources computationnelles, on ne relance évidemment pas la même simulation N fois, on propage plutôt temporellement les solutions des équations de Navier-Stokes, et on dispose alors d'un ensemble de *snapshots* ("photo instantanée") prises à intervalles de temps très rapprochés. En théorie des systèmes dynamiques, on peut alors montrer qu'en attendant plus longtemps qu'un temps appelé *temps de corrélation*, on aboutit à une configuration qui n'a plus vraiment de lien avec la condition initiale. On peut alors considérer N expériences "répétées" comme N snapshots suffisamment espacées temporellement.

Par ailleurs, plutôt que de chercher à savoir si un champ prend une valeur particulière, il est plus pertinent de considérer la **probabilité** qu'il prenne cette valeur particulière. Dans ce cas, la position et le temps ne nous intéressent plus vraiment, ces derniers deviennent eux-mêmes en quelque sorte des "expériences" aléatoires. Le volume complet représente donc déjà un espace d'intérêt pour les statistiques **spatiales**. Cet espace concaténé avec lui-même autant de fois qu'il y a de snapshots devient l'**espace des expériences aléatoires** : à cet espace, on peut associer des statistiques **d'ensemble**. Chaque champ peut ensuite prendre des valeurs qui lui sont propres pour chaque expérience aléatoire : il y a autant de champs que d'**espaces des réalisations**. On insiste grandement durant ce stage sur le concept de densités de probabilité jointe : il s'agit de la densité de probabilité d'avoir 2 événements simultanément.

Dans la présente section, nous avons introduit le contexte de travail théorique très général qu'est celui de la turbulence. Le stage que j'ai effectué ayant été avant tout numérique, nous expliciterons désormais l'écoulement considéré ainsi que le cadre du traitement de données.

2.2 Géométrie et simulations numériques

2.2.1 Géométrie de von Kármán

Nous considérons dans ce stage une turbulence dans une géométrie de von Kármán. L'écoulement est généré dans un cylindre de rayon R par deux turbines tournant en sens opposés à fréquence angulaire égale. Deux configurations sont d'intérêt : *CONTRA* et *ANTI-CONTRA* représentées respectivement par le signe + et - sur la figure 2. L'écoulement est réputé pour sa turbulence fortement inhomogène et anisotrope. Il représente une expérience majeure pour les nouvelles théories de turbulence.

2.2.2 Code SFEMaNS

SFEMaNS (*Spectral/Finite Element code for Maxwell and Navier-Stokes equations*) est un code hautement parallélisé qui permet de résoudre les équations de Navier-Stokes, les équations de Maxwell et leur couplage (magnétohydrodynamique) [7]. Il se base sur une discrétisation spatiale hybride : des éléments finis (dans le plan méridien (r, z)) ainsi qu'une décomposition en modes de Fourier en azimut. Toutes les simulations mentionnées dans ce rapport - la base de données traitée par exemple - sont issues de calculs effectués avec SFEMaNS. Nous détaillons plus les paramètres de simulations en sections §2.3.1 et §2.3.2.

Le cadre de travail étant dorénavant bien défini, la problématique du stage en lui-même repose sur un article que nous introduisons dans la section suivante.

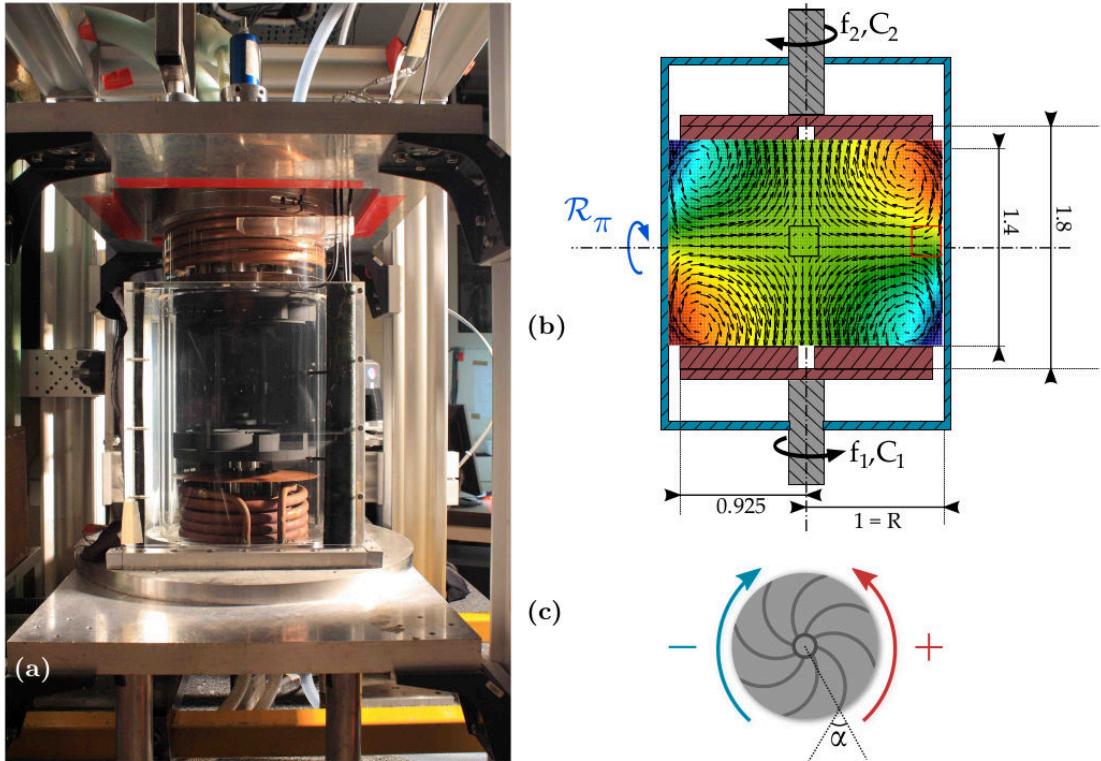


FIG. 2. Géométrie de von Kármán. (a) Photographie de la manipulation effectuée au SPEC CEA-Saclay. (b) Vue schématique du montage et champ de vitesse moyen temporellement obtenu par vélocimétrie par imagerie de particules stéréoscopique. (c) Vue du dessus d'une turbines (constituées d'un disque et de 8 pales courbées) avec les 2 sens de rotation possibles : en bleu (-) la configuration ANTI-CONTRA et en rouge (+) la configuration CONTRA.

Figure tirée de [2].

2.3 Faller *et al.*, Journal of Fluid Mechanics 914, 2 (2021)

2.3.1 Données analysées

En préambule de considérations plus dynamiques (intermittence), Hugues Faller effectue une analyse statistique de la turbulence de l'écoulement de von Kármán [1]. Il dispose d'une base de données en simulation numérique directe ainsi que de données provenant d'expériences. Les mesures expérimentales sont issues du carré noir central de la figure 2-(b) tandis que les données computationnelles proviennent du volume entier excluant les turbines. La discrétisation SFEMaNS consiste en une grille de points en coordonnées cylindriques. Il y a 255 modes de Fourier décomposant la coordonnée angulaire et donc 509 plans- θ . En chaque plan, le code associe la même triangulation en (r, z) générant 732 016 points. Il y a donc pour une snapshot $\sim 3.73 \times 10^8$ points. 21 snapshots décorrélatés forment l'ensemble des expériences répétées. Les paramètres associés aux écoulements de von Kármán étudiés sont en outre listés dans la table 1.

Les densités de probabilité jointe entre le champ \mathcal{D}_ℓ^I et $\omega = |\boldsymbol{\omega}|$ obtenues pour les données expérimentales et numériques sont rapportées dans les figures 3 et 4. Quel est le but d'observer ces quantités ? Il suffit de se rappeler de la section §2.1.3 : s'il existe un mécanisme à l'origine de la dissipation anormale, alors il est obligatoirement relié aux singularités de la vorticité. Faller obtient notamment un étrange jet de corrélations dans la gamme de densité de probabilité autour de 10^{-5} (voir la figure 4) qui est restée inexpliquée à ce jour. *Dans tout ce rapport, nous ferons spécifiquement référence à ce jet par l'appellation oreille de lapin.*

2.3.2 Corrections et nouvelles simulations

On arrive à un des questionnements majeurs de mon stage ; à savoir **quelle est l'origine de l'oreille de lapin** ? Mes superviseurs ont établi une liste d'erreurs dans les calculs SFEMaNS de Faller qui mettent en doute la validité des résultats obtenus :

- $\nabla \cdot \boldsymbol{u}$ n'est pas tout à fait égal à 0 ;
- le filtrage est mauvais sur l'axe du cylindre ;

Cas	F (Hz)	Points de grille	Re	ϵ (adim)	η (mm)	Δx (mm)
A	5	89×65	$3.1 \cdot 10^5$	0.045	0.016	2.1
B	5	77×79	$3.1 \cdot 10^5$	0.045	0.016	0.49
C	5	162×157	$3.1 \cdot 10^5$	0.045	0.016	0.24
D	1	77×80	$4.1 \cdot 10^4$	0.045	0.073	0.49
E	1.2	151×174	$5.8 \cdot 10^3$	0.045	0.32	0.24
T-1	5	$149 \times 103 \times 20$	$3.1 \cdot 10^5$	0.045	0.016	0.35
T-2	1	$139 \times 101 \times 20$	$6.3 \cdot 10^4$	0.045	0.054	0.35
T-3	0.5	$148 \times 103 \times 20$	$3.1 \cdot 10^4$	0.045	0.09	0.35
T-4	0.1	$149 \times 100 \times 20$	$6.3 \cdot 10^3$	0.045	0.3	0.35
DNS	$\frac{1}{2\pi}$	$400 \times 800 \times 509$	$6 \cdot 10^3$	0.045	0.37	0.1-0.4

TABLE 1. Paramètres décrivant les données utilisées dans l'article [1]. F est la fréquence de rotation des turbines en Hz; Re est le nombre de Reynolds basé sur F et le rayon du cylindre; ϵ est la dissipation énergétique totale adimensionnée; η est l'échelle de Kolmogorov; et Δx représente la résolution spatiale dans les expériences (dénotées de A à T-4) et les simulations numériques directes (dénotées DNS).

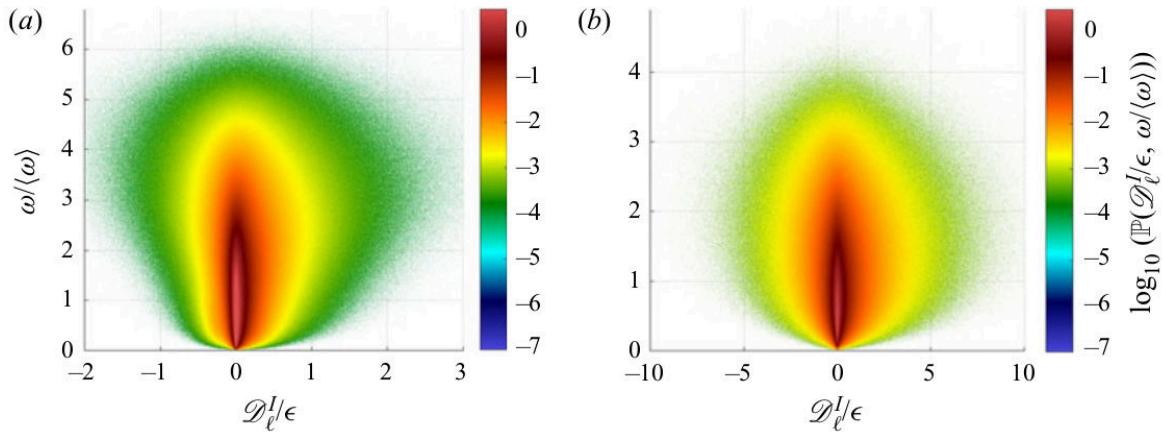


FIG. 3. Densités de probabilité jointe pour les données expérimentales. (a) T-4 : $\ell = 3.2\eta$ (3×10^4 snapshots). (b) T-2 : $\ell = 17.9\eta$ (1.02×10^4 snapshots). Figure tirée de [1].

- les conditions aux limites sont questionnables;
- \mathcal{D}^I comportant un terme de divergence, son intégration apporte des effets indésirables. En effet, les conditions aux limites peuvent créer des quasi-irrégularités et \mathcal{D}^I peut les capturer par le théorème de la divergence. Or on s'intéresse exclusivement aux transferts inter-échelle ici, c'est-à-dire aux transferts qui ne dépendent pas des conditions aux limites.

Sur la base de ces pistes, nous disposons d'une nouvelle base de données corrigées avec rigoureusement les mêmes paramètres que ceux de Faller. Nous pouvons finalement introduire dans la suite l'analyse statistique des données en question - sujet qui est au coeur de mon stage.

3 Module d'analyse statistique

L'objectif du stage étant d'étudier la statistique de l'écoulement de von Kármán turbulent, mon travail a consisté à développer un code dédié à cette analyse en utilisant le langage Python.

3.1 Code von-Karman-PostProcess

Nous avons programmé un module d'analyse statistique permettant de calculer toutes les grandeurs mentionnées dans la section §2.1.4 : [von-Karman-PostProcess](#). En particulier, nous avons cherché à optimiser les performances du dit module en implémentant une version accélérée sous la branche gpus du projet (voir section §C pour des explications sur l'utilisation de GPUs sur Python). Des exemples d'utilisation sont rapportés dans un [tutoriel](#) partagé dans le projet. Cette section décrit les mathématiques sous-jacentes à toutes ces fonctions. On considère dans toute la suite un champ

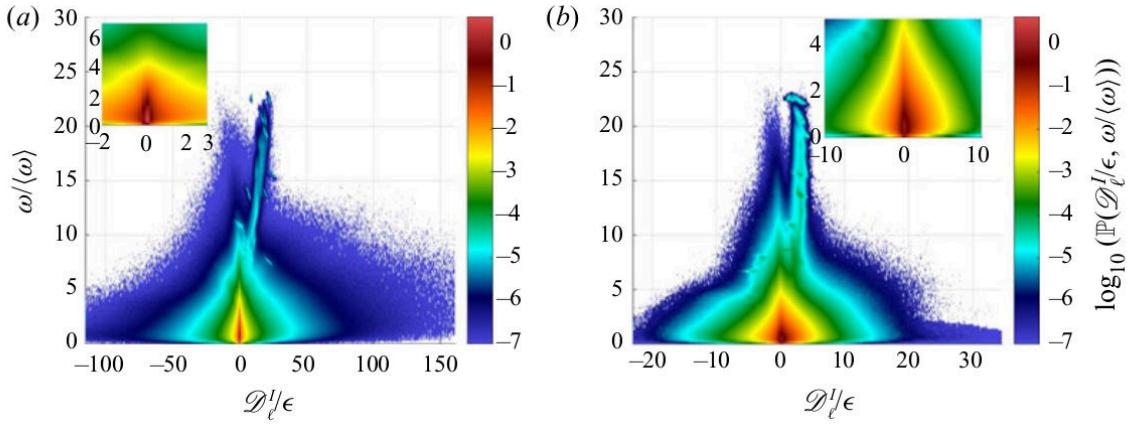


FIG. 4. Densités de probabilité jointe pour les données simulées. (a) $\ell = 1.06\eta$. (b) $\ell = 26.5\eta$. Figure tirée de [1].

scalaire et continu quelconque A défini sur un volume Ω décrit temporellement pendant une durée T . La base de données (classe Database) fournit la structure de données d'ordre 3 suivante : $A_{r,p}^t$ avec r qui indice un point de la grille, p le plan et t le temps.

3.1.1 Valeurs moyennes

On définit le moment **spatial** de A à l'ordre $n \neq 1$ comme

$$\langle A^n \rangle_\Omega(t) = \frac{1}{\text{Vol } \Omega} \int_\Omega (A(\mathbf{r}, t) - \langle A \rangle_\Omega(t))^n d^3\mathbf{r}, \quad (15)$$

avec $\text{Vol } \Omega = \int_\Omega d^3\mathbf{r}$. Bien évidemment, il faut discréteriser l'équation (15). La mesure d'intégration $d^3\mathbf{r}$ est stockée dans un tableau w_r . On va expliciter le calcul de la moyenne qu'on généralisera ensuite pour un moment quelconque. Nous obtenons :

$$\langle A \rangle_\Omega(t) = \frac{\sum_{p \in P} \sum_{r \in \text{mesh}} A_{r,p}^t w_r}{\sum_{p \in P} \sum_{r \in \text{mesh}} w_r}, \quad (16)$$

qui se généralise à l'ordre n en

$$\langle A^n \rangle_\Omega(t) = \frac{\sum_{p \in P} \sum_{r \in \text{mesh}} (A_{r,p}^t - \langle A \rangle_\Omega(t))^n w_r}{\sum_{p \in P} \sum_{r \in \text{mesh}} w_r}. \quad (17)$$

Seule l'opération (16) est déjà implémentée sous `numpy.average`. Il suffit donc d'appliquer (16) sur le champ $(A_{r,p}^t - \langle A \rangle_\Omega(t))^n$ pour obtenir le moment à l'ordre n .

Les statistiques **temporelles** s'encodent selon le même principe. Les statistiques sont ici définies selon :

$$\langle A^n \rangle_T(\mathbf{r}) = \frac{1}{T} \int_T (A(\mathbf{r}, t) - \langle A \rangle_T(\mathbf{r}))^n dt, \quad (18)$$

La discréttisation est immédiate après ce qui précède :

$$\langle A^n \rangle_T(\mathbf{r}, p) = \frac{1}{N_T} \sum_{t \in T} \left(A_{\mathbf{r}, p}^t - \langle A \rangle_T(\mathbf{r}, p) \right)^n \quad (19)$$

On notera bien que, dans un cas, on obtient un tenseur d'ordre 1 et, dans l'autre, un tenseur d'ordre 2.

3.1.2 Lois de probabilité

Afin de calculer la fonction de densité de A , il faut encore procéder à l'usage d'estimateurs. Il s'agit ici essentiellement de "compter" combien de fois les données égalisent une certaine valeur a . Dans le cas continu, on considère plutôt un intervalle centré sur a et de largeur choisie δa . Il y a une légère subtilité : A étant accessible non pas localement mais autour d'un élément de volume, une mesure (un compte) doit par conséquent être pondéré par le jacobien associé. Formellement, l'estimateur f^* de la fonction de densité de A est enfin :

$$f^*(a) = \frac{1}{N \delta a} \sum_{t \in T} \sum_{p \in P} \sum_{\mathbf{r} \in \text{mesh}} H\left(\frac{\delta a}{2} - |A_{\mathbf{r}, p}^t - a|\right) w_r, \quad (20)$$

avec ($H(x) = 1$ si $x \geq 0$ et 0 sinon) et N le nombre de mesures TOTALES. Pour que f^* soit bien une densité de **probabilité**, il faut que les poids soient normalisés au sens $w_r \leftarrow \frac{w_r}{\sum_{t \in T} \sum_{p \in P} \sum_{\mathbf{r} \in \text{mesh}} w_r}$. L'opération (20) est implémentée sous `numpy.histogram`.

Le principe est semblable lorsque l'on considère 2 champs A et B :

$$f^*(a, b) = \frac{1}{N \delta a \delta b} \sum_{t \in T} \sum_{p \in P} \sum_{\mathbf{r} \in \text{mesh}} H\left(\frac{\delta a}{2} - |A_{\mathbf{r}, p}^t - a|\right) H\left(\frac{\delta b}{2} - |B_{\mathbf{r}, p}^t - b|\right) w_r^2. \quad (21)$$

Il faut néanmoins utiliser dans ce cas `histogram2d`. En anticipation de la prochaine section, on notera bien qu'en aucune façon on devrait avoir $f^*(a, b) \neq f^*(a)f^*(b)$. En effet, même s'il est vrai que l'on échantillonne bi-dimensionnellement les grandeurs pour calculer les pdfs (*probability density function*), c'est plutôt les échantillons de **l'espace des expériences aléatoires** qui importent ! L'expression (21) parcourt bien le dit espace une seule fois et non deux fois.

On appelle f_α la densité de probabilité *marginale* d'un champ α . La densité de probabilité jointe de deux événements A et B quant à elle sera notée $f_{A,B}$. L'intuition probabilistique permet de quantifier la probabilité d'un événement A étant donnée la réalisation d'un événement B comme :

$$\mathbb{P}(B|A) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(A)}. \quad (22)$$

Pour des variables à densité, l'idée est évidemment généralisable :

$$f_{B|A}(a, b) = \frac{f_{A,B}(a, b)}{f_A(a)}. \quad (23)$$

Pour échantillonner cette loi, il suffit d'obtenir $f_{A,B}(a, b)$. En effet, $f_A(a)$ étant une loi marginale, on peut se passer d'effectivement calculer la pdf simple. Échantillonner (23) revient donc à une simple division de tableaux élément par élément. L'espérance conditionnelle découle enfin :

$$\mathbb{E}(B|A) = \int_{\Omega_B} b \cdot f_{B|A}(a, b) db \text{ (variable continue)}; \quad (24)$$

$$\mathbb{E}(B|A) = \sum_b b \cdot \frac{f_{A,B}(a, b)}{f_A(a)} \delta b \text{ (variable échantillonnée).} \quad (25)$$

On a donc vu qu'en général $f_{A,B} = f_A \cdot f_{B|A}$ avec $f_{B|A}$ la probabilité que B se réalise sachant que A s'est réalisé. A et B sont **indépendants** si $f_{A,B} = f_A \cdot f_B$. Cela revient à dire que trouver b n'influe pas sur le fait de trouver a : il n'existe donc bien aucune corrélation entre A et B .

Le coefficient de corrélation de A et B n'est pas vraiment pertinent pour nous car il ne quantifie pas **localement** les

corrélations. Il est plus pertinent de considérer la corrélation :

$$C(A, B) = \frac{f_{A,B}}{f_A f_B} \quad (26)$$

qui mesure localement si la loi de Bayes ($f_{A,B} = f_A f_B$) s'applique ou pas. Cette corrélation $C(A, B)$ varie entre 0 (si le numérateur vaut zéro) et l'infini. Elle vaut 1 quand les évènements sont indépendants, $C(A, B) < 1$ quand les variables sont anti-corrélées et $C(A, B) > 1$ quand les variables sont corrélées.

Pour calculer (26), on pourrait croire qu'il suffit d'appeler la fonction `histogram2d` et `histogram` de `numpy` et procéder à une division élément par élément mais ce n'est pas la méthode la plus efficace. Avec la seule fonction de probabilité jointe, on peut obtenir toutes les grandeurs. Les densités marginales peuvent en effet être obtenues à partir de la probabilité jointe : la probabilité d'avoir juste A égalise la probabilité d'avoir A et B **quel que soit** B . Mathématiquement,

$$f_A(a) = \int_{\Omega_B} f_{A,B}(a, b) db \text{ variable continue} \quad (27)$$

$$f_A(a) = \sum_b f_{A,B}(a, b) \delta b \text{ variable échantillonnée.} \quad (28)$$

La corrélation locale peut donc être obtenue à partir de la seule probabilité jointe.

3.1.3 Localisation des statistiques

En plus des statistiques d'ensemble, on peut comparer les statistiques entre sous-espaces du volume de l'écoulement. En l'occurrence, en prévision des résultats rapportés en section §4, on considère trois sous-espaces pertinents :

- **bulk** : $|z| \leq 0.1, r \leq 0.1$ - pertinent pour la comparaison avec les résultats des expériences
- **interior** : $|z| \leq 0.69$ - pour considérer l'écoulement en dehors des pales
- **penal** : parce que le champ de vitesse associé au fluide n'existe pas dans la région des turbines (qui sont solides), le code SFEMaNS fournit un champ $\text{penal} = I_{r,p}^t$ indiquant le poids effectif des points échantillonnes (par-delà le jacobien). En voyant un peu trop ce champ comme une indicatrice, on pourrait croire qu'il suffit de multiplier tous les champs d'intérêt par ce dernier. C'est faux : il faut plutôt pondérer le poids statistique par ce dernier afin d'effectivement comptabiliser ou non un point. Dans le cas contraire, on modifie la valeur du champ en lui-même, ce qui perturbe inévitablement les statistiques.

Pour toute fonction retournant des statistiques $\text{func}(w_r)$ dépendant de la base de données à travers le poids statistique, la pénalisation implique la transformation globale du code :

$$\text{func}(w_r) \rightarrow \text{func}\left(I_{r,p}^t w_r\right). \quad (29)$$

On notera qu'on peut désormais définir un nouveau poids statistique $\tilde{w}_{r,p}^t = I_{r,p}^t w_r$ qui est un tenseur d'ordre 3 selon la structure actuelle de la base de données. On considère qu'un point appartient bien à l'écoulement si $\text{penal} > 0.8$ et, dans ce cas, on fixe la valeur de penal à 1. Dans le cas contraire, la valeur est imposée à 0.

3.2 Gestion de la mémoire

Ce stage fut une occasion pour moi d'entrouvrir une porte sur le monde du *High Performance Computing* (HPC) : bases de données massives, clusters, supercalculateurs, programmation dirigée GPUs, etc. Le traitement de données massives peut s'avérer en outre une étape cruciale dans le déroulé d'un projet de recherche en physique. Une partie importante de mon travail a ainsi été de pouvoir appliquer les statistiques développées dans la section §3.1 sur une grande base de données. Cette section technique explique comment j'ai dû restructurer la base de données à cet effet. Elle peut être ignorée en première lecture si le lecteur s'intéresse aux résultats physiques décrits dans la section §4.

3.2.1 Restructuration des données

La base de données fournit des champs très gourmands en termes de ressources : pour un champ 1D, il faut compter 77 GB et donc le triple pour trois dimensions. Construire un workload sur deux champs entiers est possible mais nécessite au minimum 150 GB de RAM pour deux champs 1D et 300 GB pour un champ 1D et un champ 3D. Réserver 300 GB de RAM est possible sur Lab-IA, la machine dont on dispose : il y a 2 noeuds sur 12 offrant cette possibilité (qui ont

384 GB de mémoire). Le temps d'attente sera cependant plus long qu'à l'accoutumée puisqu'il faudra réserver un noeud quasiment entier.

La structure des données étant nativement ordonnée et de surcroît d'ordre 3 ($A_{r,p}^t$ avec r indiquant la grille, p le plan et t le champ instantané), un moyen rapide de remédier au problème est de restructurer les données.

Le moyen le plus trivial et en même temps le plus intelligent est de découper chaque champ temporellement : on génère alors 28 paquets ordonnés de taille variant entre 2.33 GB et 8.33 GB. Par-delà l'évidente optimisation en ce qui concerne la gestion de la mémoire, on peut dorénavant comparer les statistiques entre ensembles décorrélés. Si l'on générerait plutôt les paquets aléatoirement (par exemple en mélangeant temps et espace), aucun résultat intermédiaire ne serait exploitable puisque seule la recombinaison ultime des paquets fournirait le résultat initialement recherché.

3.2.2 Lectures en parallèle

La base de données étant *delayée*, les données sont effectivement importées sur la RAM uniquement si on donne cette instruction. Cette façon de procéder permet de charger en mémoire uniquement ce qui nous sert, chose qui est très commode pour une restructuration des données. À ce propos, le module Dask permet d'effectuer des opérations en parallèle sur des objets délayés. Selon les dimensions des données, on peut alors optimiser le chargement de ces données en choisissant un nombre de coeurs adéquat. En pratique, il s'agit de faire en sorte que chaque thread puisse s'occuper indépendamment d'une partie ni trop grosse ni trop petite des données totales (il est important que le nombre de "chunks" ainsi formés soit entier). Le chargement des données est enfin optimisé en chargeant 14 chunks indépendamment à l'aide des instructions suivantes :

Script 1: Chunking des données délayées sur 14 threads

```
from vkcupp import Stats
import dask.array as da

db_path = '...'
pp = Stats(db_path, 'penal')          # getting the PostProcessing module
data = pp.db                           # database dictionary
P, N = pp.dims[2], pp.dims[3]         # p,r dimensions in data struct

field = data[key]['field']            # field is delayed at this point
field = field.rechunk((1,1,P,N//14)) # field is chunked into 14 blocks
field = field.compute()               # parallelized-loading on the RAM
```

3.2.3 Tâches de travail (workloads) de type probabilité jointe

Des détails techniques de la mise en place des workloads sont donnés en section §D.1. L'idée étant à terme de recombiner les histogrammes, il est très important que les intervalles échantillonnes sur chaque workload soient identiques. La méthode `Stats.joint_pdf` que j'ai codée détermine nativement les intervalles automatiquement ; cela demeure problématique puisque les intervalles varient entre chaque ensemble. Pour calculer des probabilités jointes, il faut alors en premier lieu calculer les intervalles des champs totaux et les fournir à chaque workload. On comprend vite que le problème de la mémoire demeure pour ce calcul : la solution est naturellement de procéder à cette même restructuration de données. Concrètement, il s'agit de lancer un job récupérant l'étendue (range) de chaque workload. L'ensemble des workloads partitionnant les données totales, en concaténant les intervalles ainsi obtenus, l'étendue (range) globale est simplement constituée du min et du max de l'ensemble $\{\{A_{\min}^t, A_{\max}^t\}_t\}$ aplati.

Nous donnons dans le script 2 un script permettant de calculer la densité de probabilité jointe entre les champs $\mathcal{D}_{t=1.06\eta}^I$ et ω avec champ de pénalisation à l'aide d'un GPU.

Script 2: Workloads séquentiels sous GPUs

```
import os
from vkcupp import Stats
import cupy as cp
```

```

import dask.array as da

db_path = '...'
pp = Stats(db_path, 'penal')           # getting the PostProcessing module
data = pp.db                           # database dictionary
P, N = pp.dims[2], pp.dims[3]         # p,r dimensions in data struct

# Defining the rechunked+delayed fields
DI_delayed = data['D002_DR']['field'].rechunk((1,1,P,N//14))
w_delayed = data['omega']['field'].rechunk((3,1,P,N//14))

# Run params
bags = 28
bins = 2000
w_min, w_max = 0, 402.535165          # computed externally
DI_min, DI_max = -1.893386, 3.594616  # computed externally
ranges = [[DI_min, DI_max], [w_min, w_max]]
os.mkdir('slices')                     # for saving purposes
s = slice(None)

# Sequential workloads
for t in range(bags):
    print('########################################')
    print('Frame %d' %t)

    # Loading the sliced fields
    pp.set_penal(s, [t], s, s)          # automatically sent to CUDA
    DI, w = [x[:, [t], :, :].compute() for x in [DI_delayed, w_delayed]]
    print('Loaded all fields')

    # Transferring to CUDA
    DI, w = [cp.array(x, copy=False) for x in [DI, w]]
    print('Transferred all fields to CUDA')

    # CUDA calculations
    w = pp.norm(w)
    print('Computed w norm')

    DI_w, edges = pp.joint_pdf(DI, w_norm, bins, ranges, log=False, \
                                save='slices/iter_%d' % (t))

    print('Computed joint_pdf')

```

Le script 2 sauvegarde alors les densités de probabilité jointes dans des fichiers binaires; la suite des opérations est naturellement de correctement recombiner les résultats. Les slices temporelles partageant l'espace des expériences aléatoires, la procédure correcte est de sommer toutes les réalisations bin par bin. Quid de la normalisation? Si les pdfs sommées sont déjà normalisées, alors on vérifie pour chaque histogramme H^t (les éléments de surface sont identiques

pour tous puisqu'on a imposé une étendue (range) commune) :

$$\sum_{i,j} H_{i,j}^t W_{i,j} = 1 \quad (30)$$

$$\Rightarrow \sum_t \sum_{i,j} H_{i,j}^t W_{i,j} = N_{\text{bags}} \quad (31)$$

$$\Rightarrow \sum_{i,j} \tilde{H}_{i,j} W_{i,j} = 1, \text{ avec } \tilde{H}_{i,j} = \frac{1}{N_{\text{bags}}} \sum_t H_{i,j}^t. \quad (32)$$

Le script 3 permet enfin d'obtenir les histogrammes complets.

Script 3: Recombinaison des probabilités jointes

```
from vkcupp import Stats
import numpy as np

db_path = '...'
pp = Stats(db_path, 'penal')           # getting the PostProcessing module
bins = 2000
bags = 28

# Gathering all partial joint pdfs
joints = []
for t in range(bags):
    hist = np.fromfile('slices/iter_%d' % (t))
    DI_w = pp.load_reshaped(hist, bins)
    joints.append(DI_w)

# Full hist
DI_w_full = (np.sum(np.array(joints), axis=0)/bags)
```

4 Résultats et discussion

Nous décrivons dans cette section les résultats obtenus avec le module de traitement statistique appliqué sur la base de données restructurée. Nous allons comparer nos résultats à ceux publiés par l'équipe [1].

4.1 Comparaison avec Faller *et al.* [1]

Nous comparons d'abord avec les résultats issus des simulations numériques directes (Direct Numerical Simulations, DNS), puis avec les résultats issus de l'expérience von Kármán située au laboratoire SPEC (CEA, Saclay) sous la direction de Bérengère Dubrulle.

4.1.1 Comparaison avec les résultats de DNS

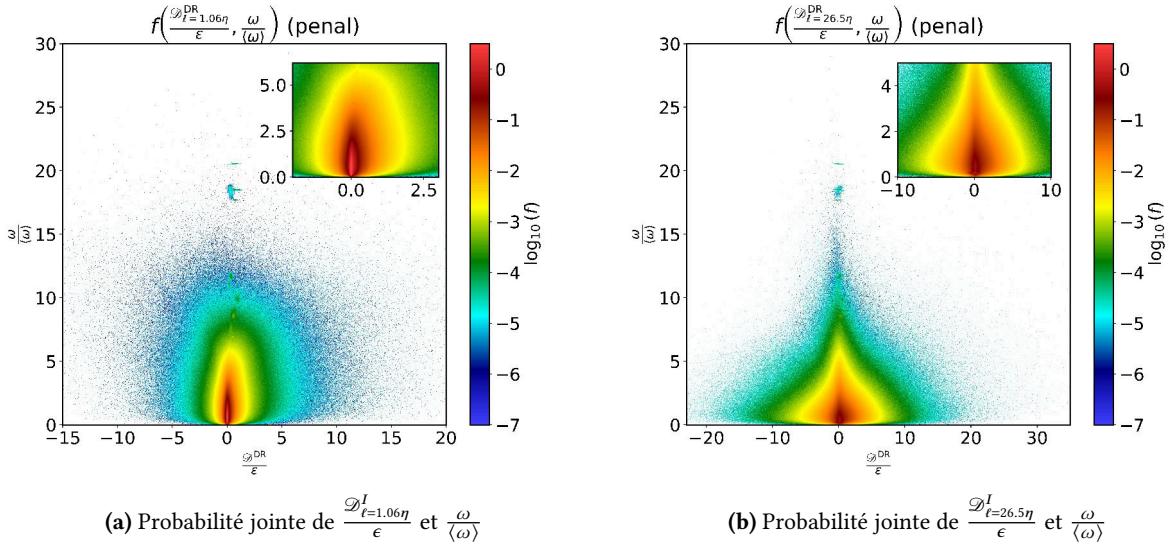


FIG. 5. Statistiques de \mathcal{D}^I et ω aux échelles $\ell = 1.06\eta$ et $\ell = 26.5\eta$ avec pénalisation des turbines

La figure 5 rapporte les densités de probabilité jointe pour les champs \mathcal{D}^I et ω pour deux échelles de filtrage distinctes. Près de l'échelle de Kolmogorov (fig. 5-a), la première observation à faire est la différence majeure d'étendue de $\frac{\mathcal{D}_{\ell=1.06\eta}^I}{\epsilon}$ avec les résultats de Faller (cf. figure 4). Dans notre cas, les évènements hors de l'intervalle $[-15, 20]$ (normalisé par ϵ) sont quasiment improbables. Faller parvient à capturer l'étendue $[-100, 150]$. Cette différence n'est pas présente pour la grande échelle de filtrage (fig. 5-b) où nos étendues demeurent à peu près similaires même si nous ne parvenons que difficilement à reproduire des probabilités inférieures à 10^{-5} . La difficulté à aller chercher des probabilité très faibles peut s'expliquer par un argument numérique : la taille des *bins*. Les graphiques de Faller donnent l'impression d'être beaucoup moins résolus que les nôtres (10000×10000). Des bins trop étroits ont l'avantage de la résolution mais peuvent malgré tout manquer à rendre compte de *tendances*. Autrement dit, pour des évènements effectivement plus rares, il vaut mieux avoir une plus grosse réserve permettant au moins de capturer un certain nombre de réalisations. En capturant seulement un bin infinitésimal, on ne peut presque rien visualiser.

En ce qui concerne l'etroitesse de l'abscisse de la probabilité jointe de la figure 5-a, le fait qu'elle apparaisse seulement à très faible échelle suggère que c'est un problème lié au filtrage sur l'axe, seule source de différence liée à ℓ .

Cependant, les zooms présentés en insert dans la figure 5 présentent une certaine similarité avec les résultats de [1] (au moins pour la grande échelle).

Le point essentiel à désormais relever est **l'absence totale d'oreille de lapin**. Il existe certains structures ressemblant à des "blobs" mais qui n'apparaissent pas quand nous calculons les probabilités jointes dans les autres sous-espaces (bulk et interior). Nous pouvons en déduire que ces blobs sont des artefacts liés à l'écoulement entre les pales des turbines. L'origine de l'oreille de lapin est à notre niveau d'analyse multi-factorielle : la divergence non-nulle ainsi que les mauvaises conditions aux bords ont pu toutes les deux participer à cette structure corrélée.

Pour conclure cette comparaison entre résultats numériques, nous retiendrons surtout que les bonnes statistiques s'étaisnt moins à basse échelle et sont donc plus localisées. Les transferts vers les échelles de plus en plus basses sont plus regroupés vers des valeurs plus faibles des transferts inter-échelle.

4.1.2 Comparaison avec les résultats expérimentaux

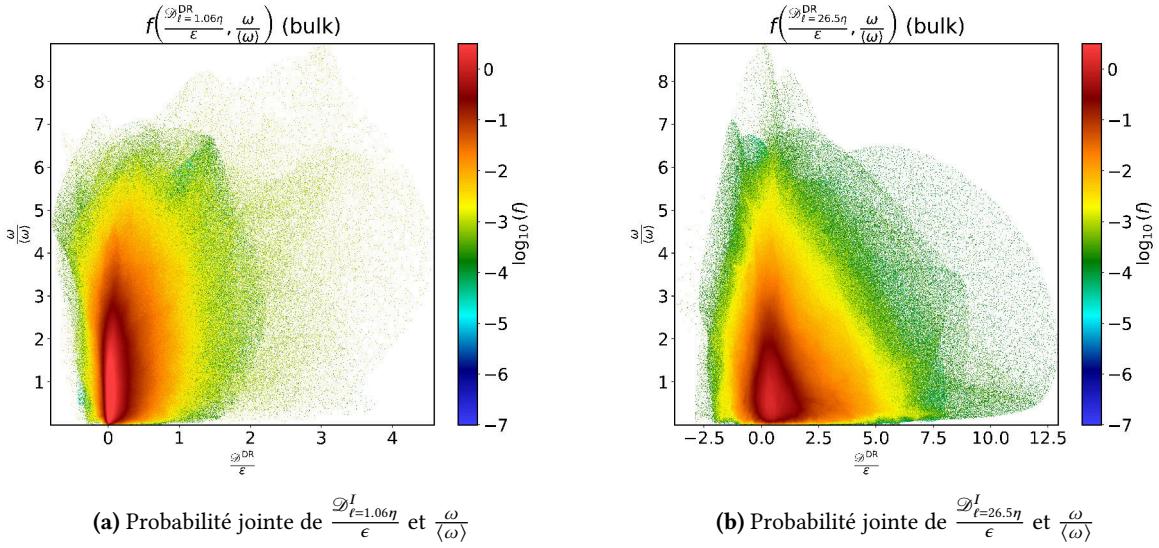


FIG. 6. Statistiques de \mathcal{D}^I et ω aux échelles $\ell = 1.06\eta$ et $\ell = 26.5\eta$ dans la région bulk

La figure 6 représente les mêmes quantités que la figure 5 sauf qu'on considère désormais la région bulk uniquement (on reproduit les mesures des expérimentateurs en quelque sorte). Même si les données expérimentales formant la figure 3 ont des échelles de Kolmogorov ainsi que des échelles de filtrage différentes, nous pouvons au moins mentionner quelques idées intéressantes émergeant de la comparaison. Même si nous parvenons à accéder à des intervalles de probabilité similaire, nos structures préservent un caractère dissymétrique contrairement aux statistiques des expérimentateurs. Au centre du cylindre, on s'attendrait plutôt à une turbulence statistiquement quasi-homogène et quasi-isotrope car la zone de mesure est loin des turbines (qui entraînent le fluide) et des parois (qui imposent les conditions aux limites). Cela semble dire que la dissymétrie perdure même pour une turbulence homogène et isotrope. Ce point nécessite de plus amples investigations.

Dans cette section, nous avons cherché à identifier les différentes causes d'écart (cf. section §2.3.2) entre les résultats de [1] et nos résultats. Il nous semble que le problème à l'axe a bien été identifié, les soucis liés aux conditions aux limites et à la non-divergence du champ de vitesse restent plus insaisissables. Nous nous intéressons maintenant plus en détail à la distinction entre l'ancienne et la nouvelle dissipation anormale dérivée analytiquement.

4.2 Comparaison entre \mathcal{D}_ℓ^u et \mathcal{D}_ℓ^I

Comme souligné dans la section §2.1.3, ces deux quantités diffèrent d'un terme en gradient. Nous allons mesurer l'impact de cette différence.

4.2.1 Probabilités jointes

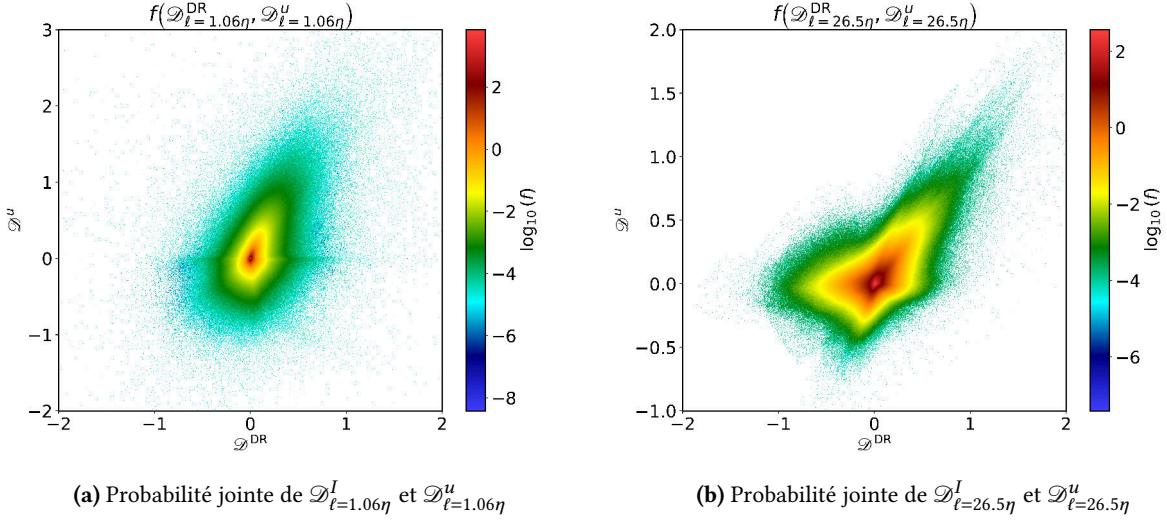


FIG. 7. Événements simultanés entre $\mathcal{D}^I (= \mathcal{D}^{DR})$ et \mathcal{D}^u

Nous pouvons commencer par comparer les deux champs du point de vue de leur probabilité conjointe de réalisation. La figure 7 rend compte de leur densité jointe pour les deux échelles déjà rencontrées précédemment. On remarque, très grossièrement, un contour ovoïde incliné sur la droite $y = x$ pour ces deux échelles. La déformation en $y = x$ suggère bien que les deux termes disposent d'une nature très similaire. À basse échelle, on retrouve un véritable "oeuf"; à grande échelle, on observe une déformation type "aile de papillon" comportant des lignes de dilatation quasiment dans toutes les directions. Cependant, on remarque que, pour $\ell \approx \eta$, il existe une ligne de dilatation pour \mathcal{D}^u autour de 0. On comprend alors que \mathcal{D}^I capture plus d'événements qu'il n'en devrait : le terme en divergence contribue davantage que les simples transferts.

4.2.2 Corrélations

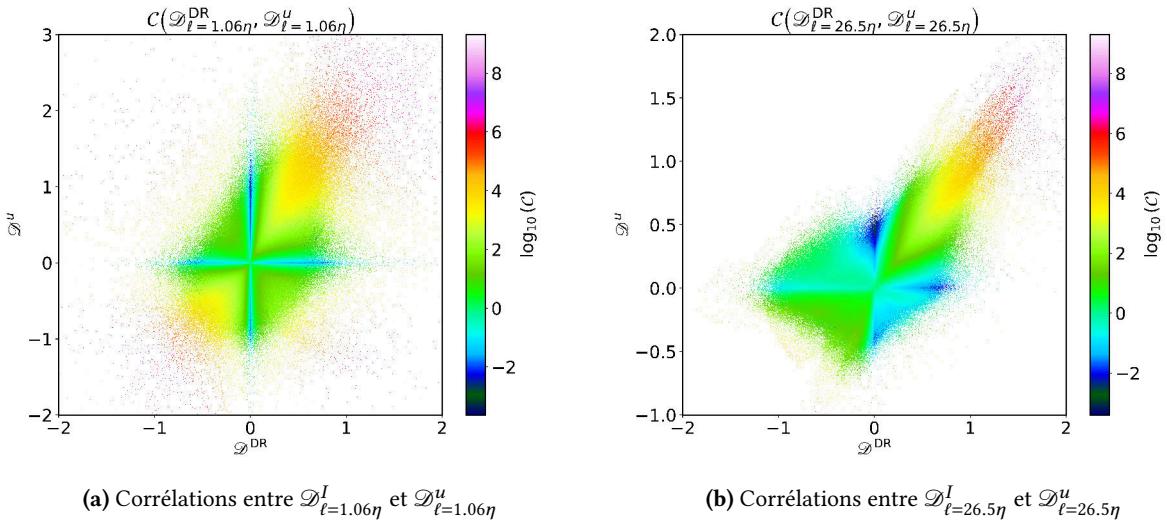


FIG. 8. Corrélations entre $\mathcal{D}^I (= \mathcal{D}^{DR})$ et \mathcal{D}^u

La figure 8 complète la figure 7 en considérant les corrélations définies par l'équation (26). L'avantage est criant : on n'a plus besoin de "deviner" les motifs (patterns) derrière la probabilité, ils apparaissent d'eux-mêmes. Ainsi, on voit directement que les lignes de dilatation (les traces bleues dans les graphes) sont anti-correlées (car $C(A, B) < 1$) et d'autant plus brouillées que l'échelle du filtrage est grande. Que signifie l'anti-corrélation ? Il y a anti-corrélation en (a, b) lorsque la probabilité que A vaille a devient d'autant plus faible lorsque la probabilité que B vaille b augmente (ou

l'inverse). Si on circule par exemple le long de l'axe $\mathcal{D}_{\ell=1.06\eta}^u = 0$ de la figure 8-a, en s'éloignant de l'origine, on capte de plus en plus d'événements rares de \mathcal{D}^I tandis qu'on est d'autant plus sûr que \mathcal{D}^u vaut bel et bien 0. Cela s'interprète par le fait qu'il y a un surplus d'événements qui ne veulent pas dire grand chose avec \mathcal{D}^I : ils proviennent précisément du terme de divergence. On retrouve les valeurs de $C(A, B) > 1$ le long de la bissectrice $y = x$ qui montrent la corrélation entre les deux termes.

Nous avons vu dans cette section comment l'influence du terme du divergence pouvait rapidement être mise en évidence par les corrélations. Elles demeurent un outil de visualisation très pratique. Pour cause, sans aucune considération analytique, nous avons pu mettre en évidence qualitativement le terme supplémentaire différentiant \mathcal{D}^I et \mathcal{D}^u .

5 Conclusion

Parmi les nombreux défis à relever en turbulence, la démonstration de la conjecture d'Onsager est fondamentale. Pour cause, son lien avec la structure même des équations régissant la mécanique des fluides pourrait faire avancer plusieurs problèmes à la fois. Couplée à la théorie des distributions, il existe une théorie moderne de la turbulence permettant de répondre aux questions à ce sujet. Le code SFEMaNS, puissant logiciel de simulation (magnéto-)hydrodynamique nous donne alors accès à d'immenses bases de données fournissant les grandeurs d'intérêt prédites par la théorie. Nous avons alors développé un module d'analyse statistique permettant de rendre compte des statistiques inhérentes à un écoulement turbulent. Par ailleurs, dans l'environnement avéré de calcul haute performance du laboratoire, nous avons également appris à gérer la mémoire sur des architectures différentes de la façon la plus optimale possible. En conclusion, nous avons pu efficacement déployer nos algorithmes afin d'analyser les statistiques d'un écoulement de von Kármán turbulent. À ce propos, nous avons pu pointer du doigt le caractère erroné d'anciens calculs. En particulier, l'étrange jet de corrélations issu de [1] a été caractérisé comme factice. Également, l'intérêt de la nouvelle dissipation anormale de [6] a pu être mis en exergue par mon analyse statistique. La prochaine étape serait alors de porter un regard plus attentif sur cette dernière (annexe A). Enfin, il faudrait analyser les corrélations entre toute paire de champ évoluant dans le fluide (annexe B), afin de mettre en évidence de nouveaux caractères singuliers dans l'optique de faire avancer la théorie de la turbulence.

Du point de vue du développement personnel, ce stage m'a permis d'entrer en contact avec deux outils fondamentaux - à mon sens - dans la poursuite d'une carrière en physique numérique/théorique : à la fois je me suis confronté à une théorie complexe par son cadre mathématique, mais j'ai surtout cherché à supporter cette complexité dans l'environnement plus pratique du calcul haute performance. Même si beaucoup de technicités informatiques surgissent à la lecture de ce document, je crois avoir toujours cherché à élaborer des procédures permettant de faciliter un futur travail d'analyse statistique.

Annexes

A \mathcal{D}^u vs ω

Les dimensions des histogrammes sont 10000×10000 .

A.1 Full

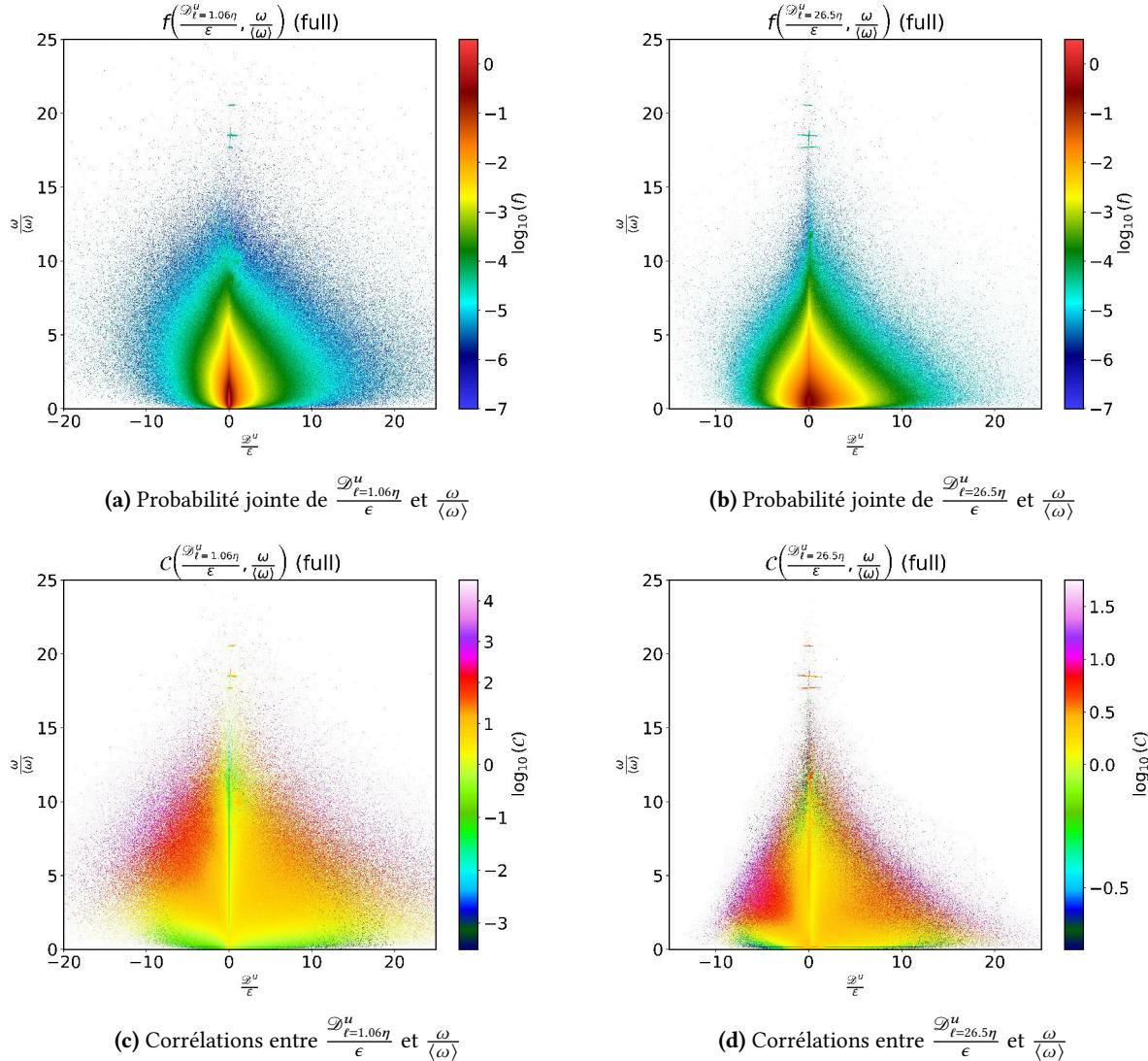


FIG. 9. Statistiques de \mathcal{D}^u et ω aux échelles $\ell = 1.06\eta$ et $\ell = 26.5\eta$ dans l'espace entier

A.2 Penal

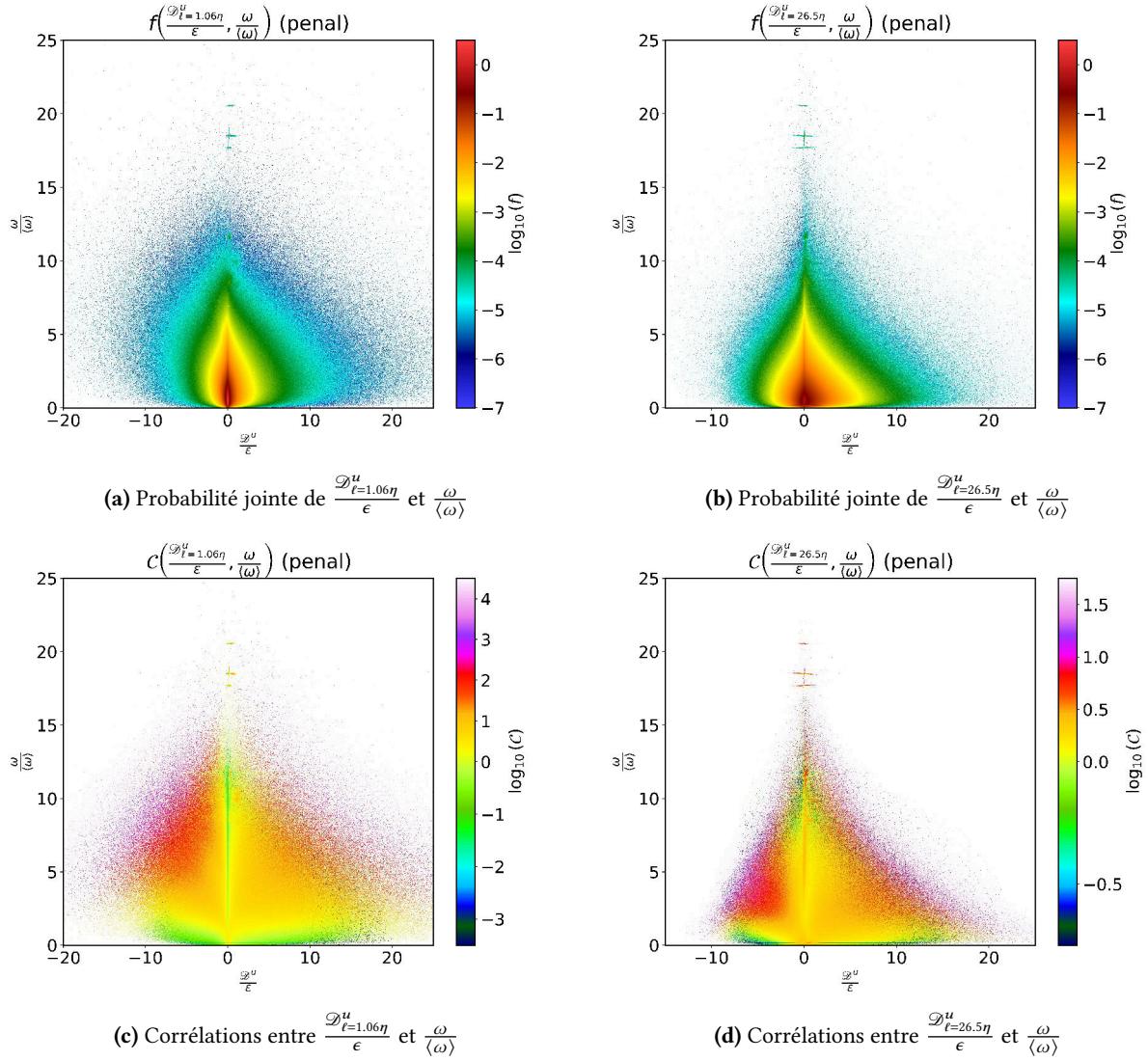


FIG. 10. Statistiques de \mathcal{D}^u et ω aux échelles $\ell = 1.06\eta$ et $\ell = 26.5\eta$ dans l'espace pénalisé des turbines

A.3 Interior

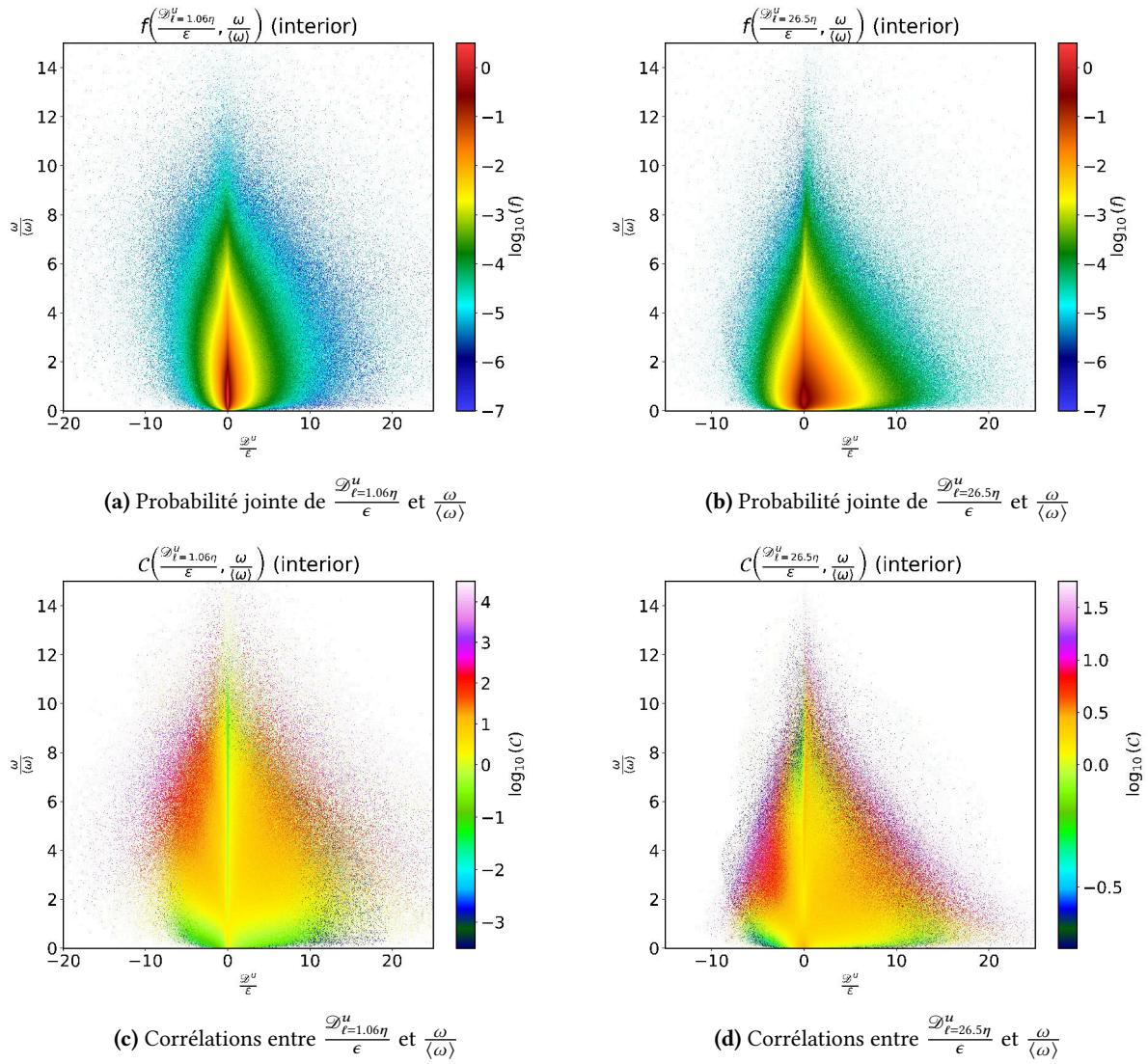


FIG. 11. Statistiques de \mathcal{D}^u et ω aux échelles $\ell = 1.06\eta$ et $\ell = 26.5\eta$ dans l'espace extérieur aux turbines

A.4 Bulk

Notez que les figures 6 (a) et (b) (en comparaison) sont issus d'histogrammes 2000×2000 .

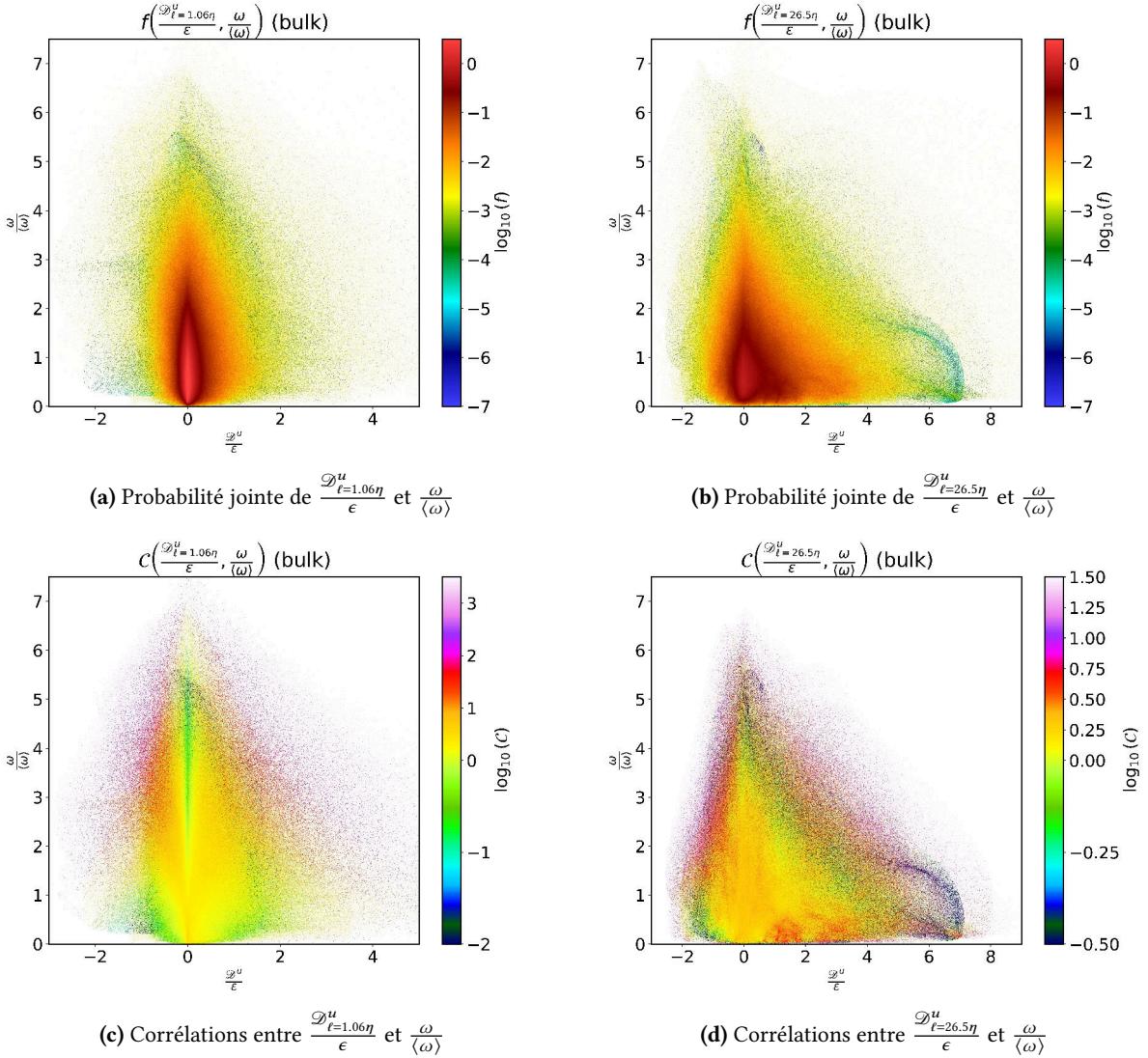


FIG. 12. Statistiques de \mathcal{D}^u et ω aux échelles $\ell = 1.06\eta$ et $\ell = 26.5\eta$ dans le sous-espace central au volume

B Statistiques inter-champs

Les statistiques présentées dans cette section sont issues d'histogrammes 2000×2000 dans le sous-espace "penal". Les paires $(\mathcal{D}_{\ell=1.06\eta;26.5\eta}^I, \mathcal{D}_{\ell=1.06\eta;26.5\eta}^u)$, $(\mathcal{D}_{\ell=1.06\eta;26.5\eta}^I, \omega)$ et $(\mathcal{D}_{\ell=1.06\eta;26.5\eta}^u, \omega)$ sont représentées sur les figures 7/8, 5 et 10.

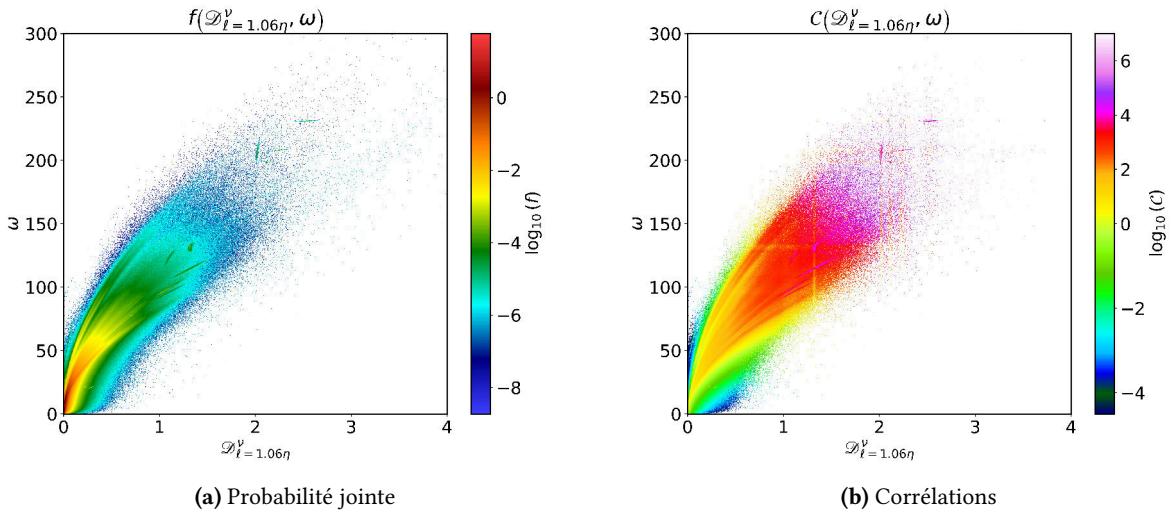


FIG. 13. $\mathcal{D}_{\ell=1.06\eta}^{\nu}$ vs ω

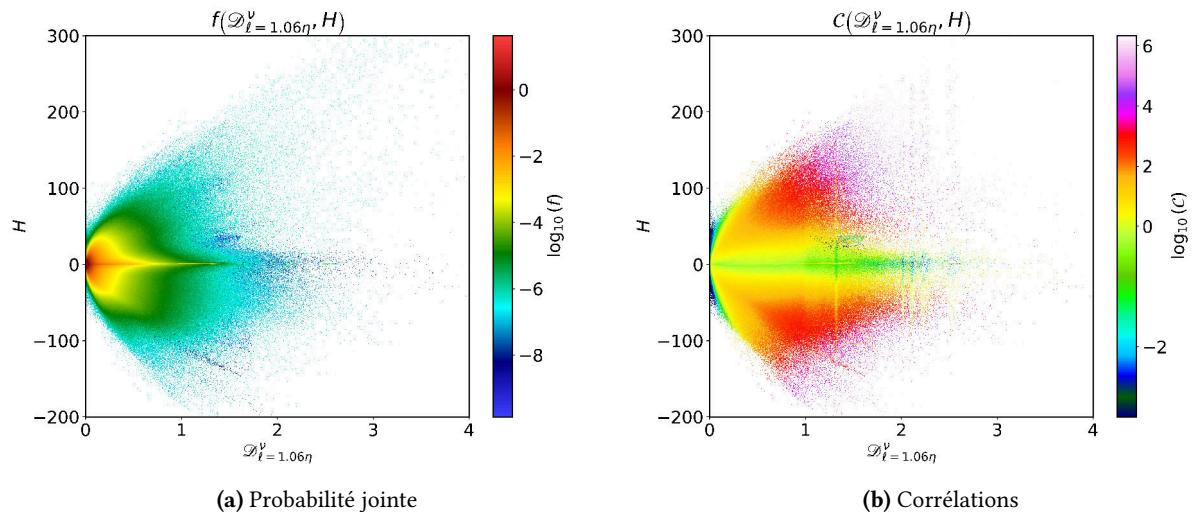


FIG. 14. $\mathcal{D}_{\ell=1.06\eta}^{\nu}$ vs H

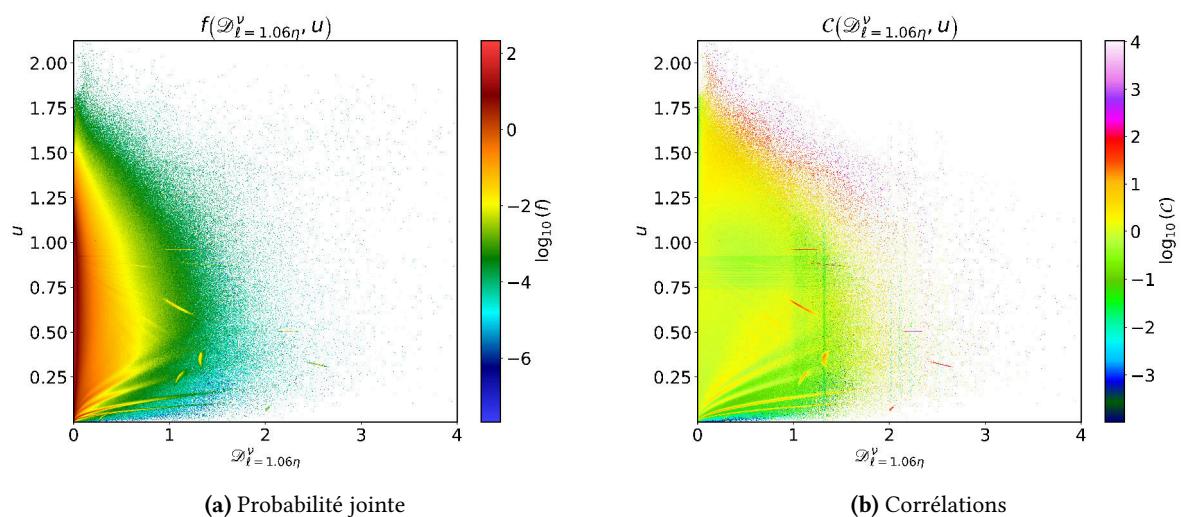


FIG. 15. $\mathcal{D}_{\ell=1.06\eta}^{\nu}$ vs u

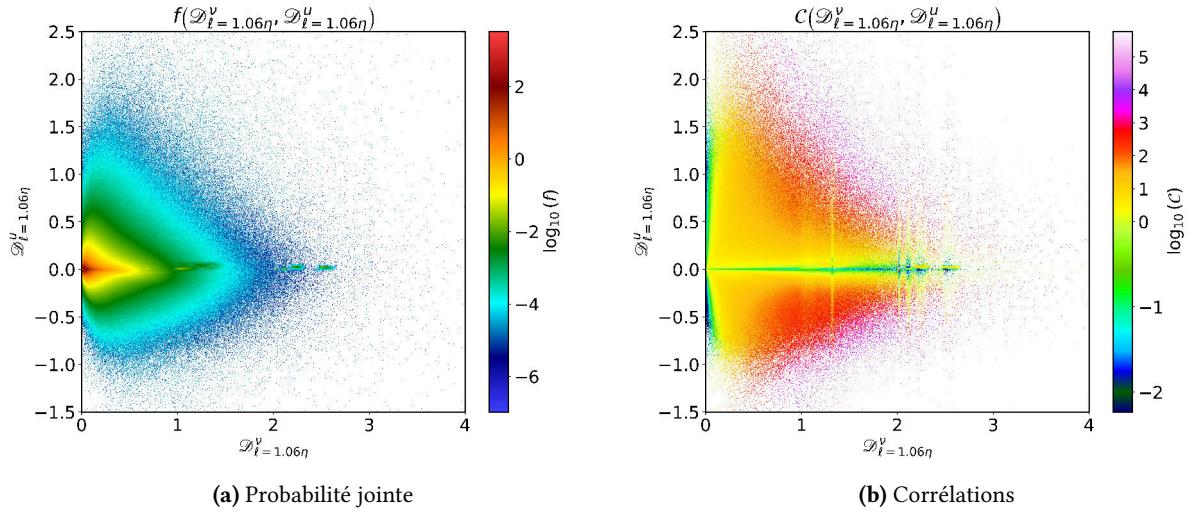


FIG. 16. $D_{\ell=1.06\eta}^v$ vs $D_{\ell=1.06\eta}^u$

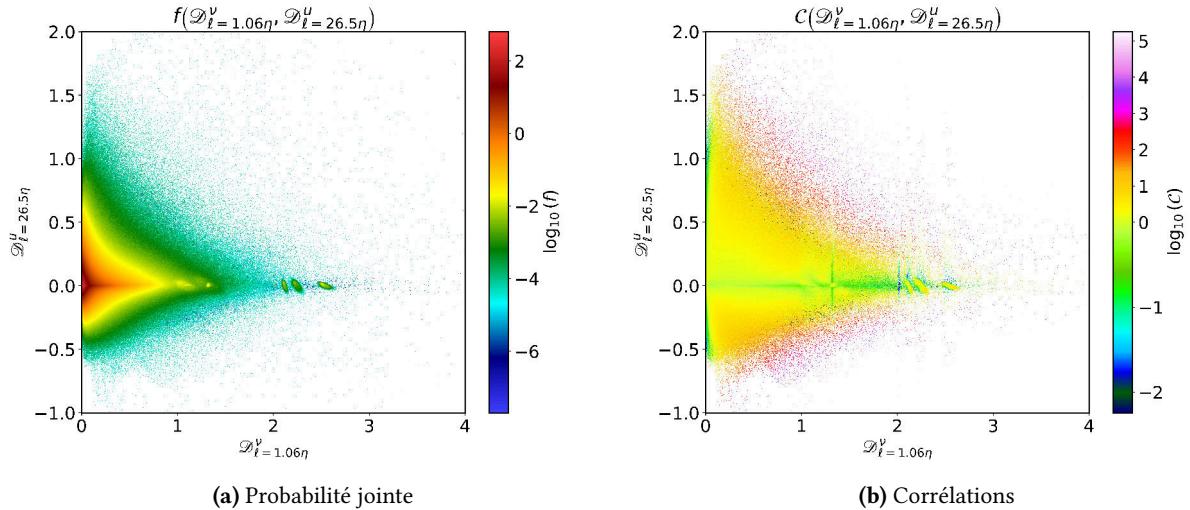


FIG. 17. $D_{\ell=1.06\eta}^v$ vs $D_{\ell=26.5\eta}^u$

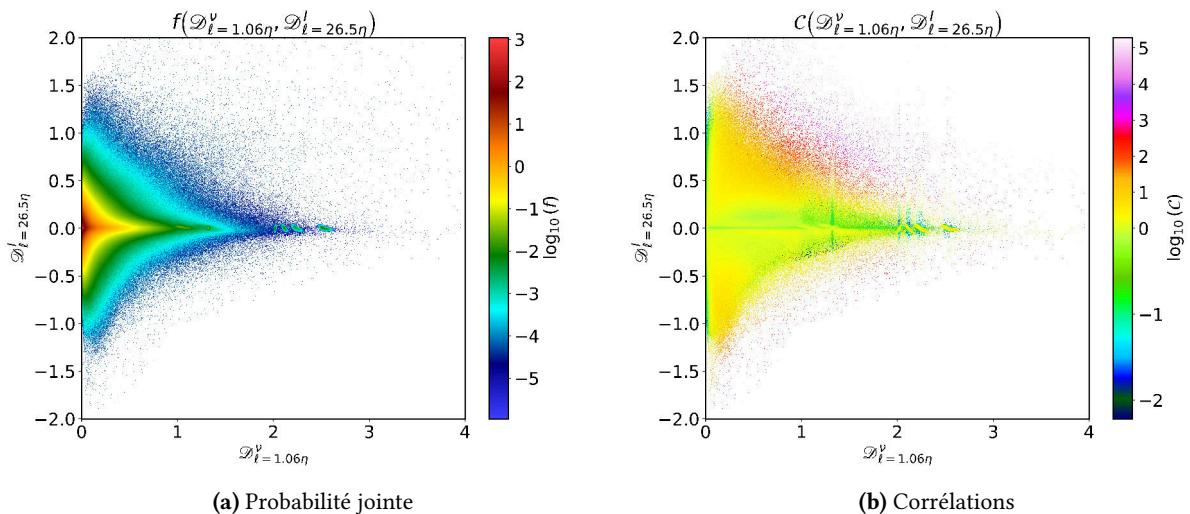


FIG. 18. $D_{\ell=1.06\eta}^v$ vs $D_{\ell=26.5\eta}^l$

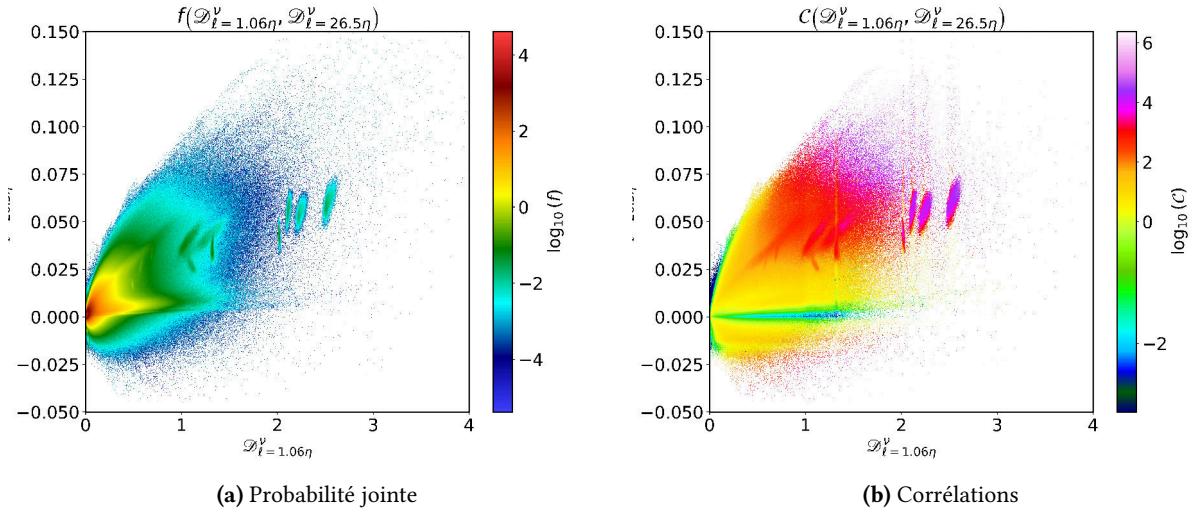


FIG. 19. $D_{\ell=1.06\eta}^{\nu}$ vs $D_{\ell=26.5\eta}^{\nu}$

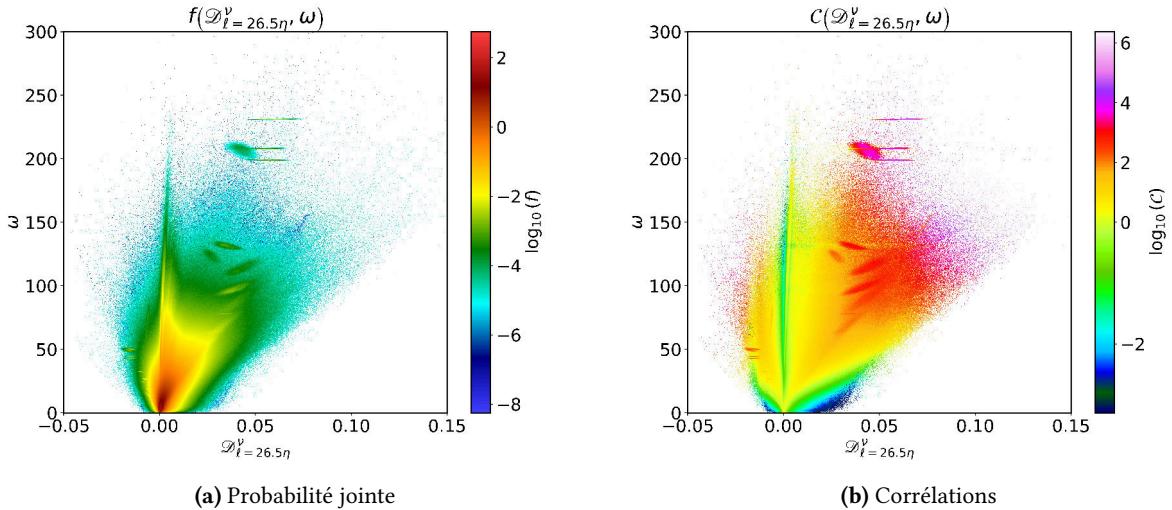


FIG. 20. $D_{\ell=26.5\eta}^{\nu}$ vs ω

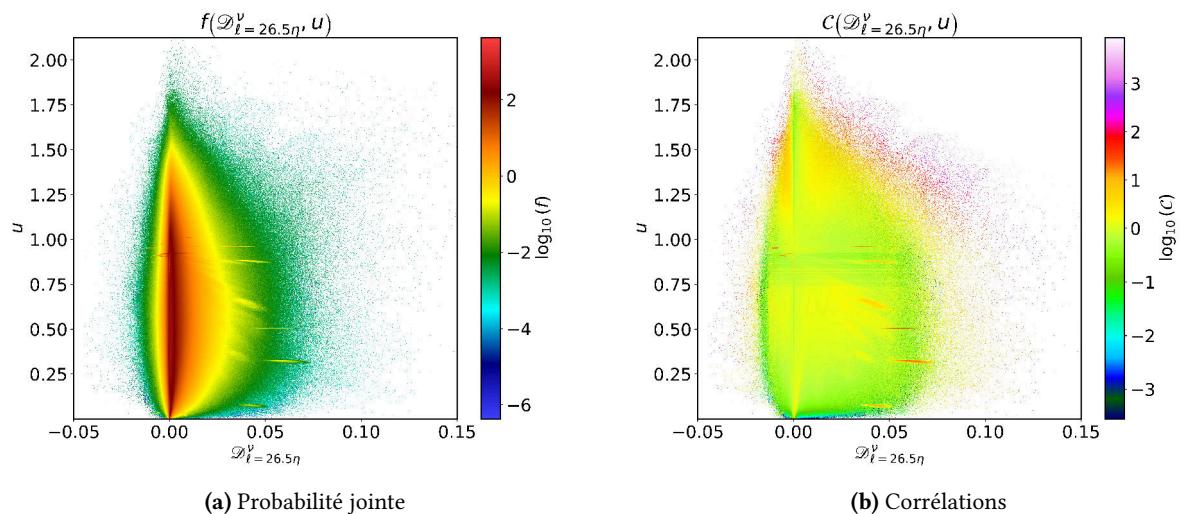


FIG. 21. $D_{\ell=26.5\eta}^{\nu}$ vs u

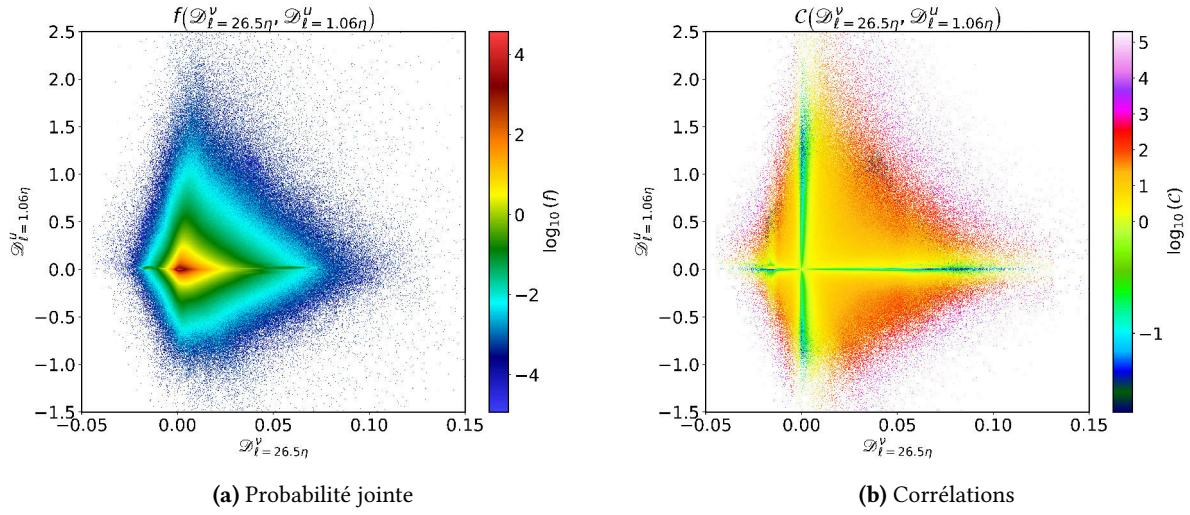


FIG. 22. $D_{\ell=26.5\eta}^v$ vs $D_{\ell=1.06\eta}^u$

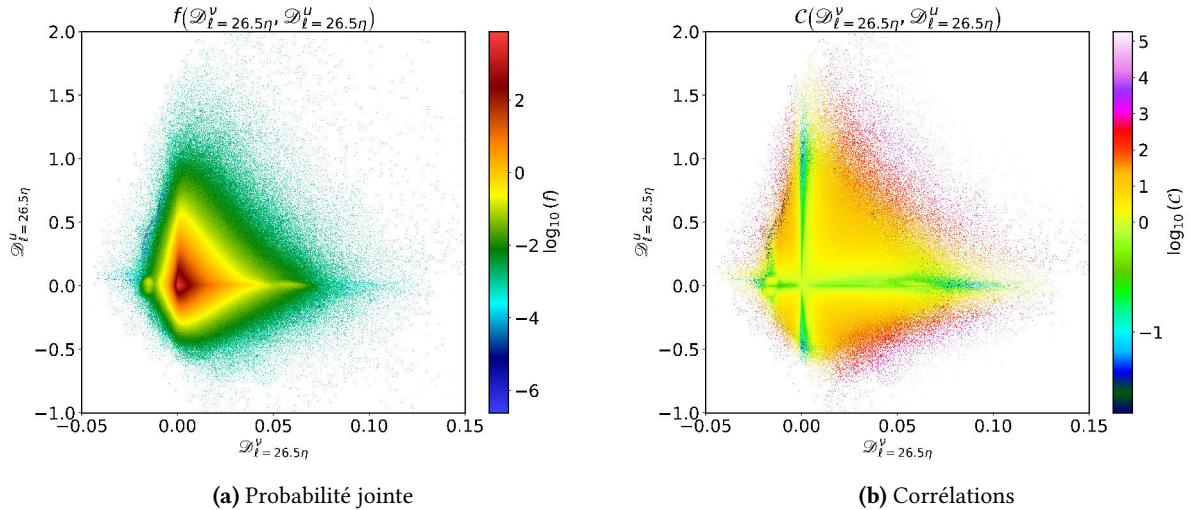


FIG. 23. $D_{\ell=26.5\eta}^v$ vs $D_{\ell=26.5\eta}^u$

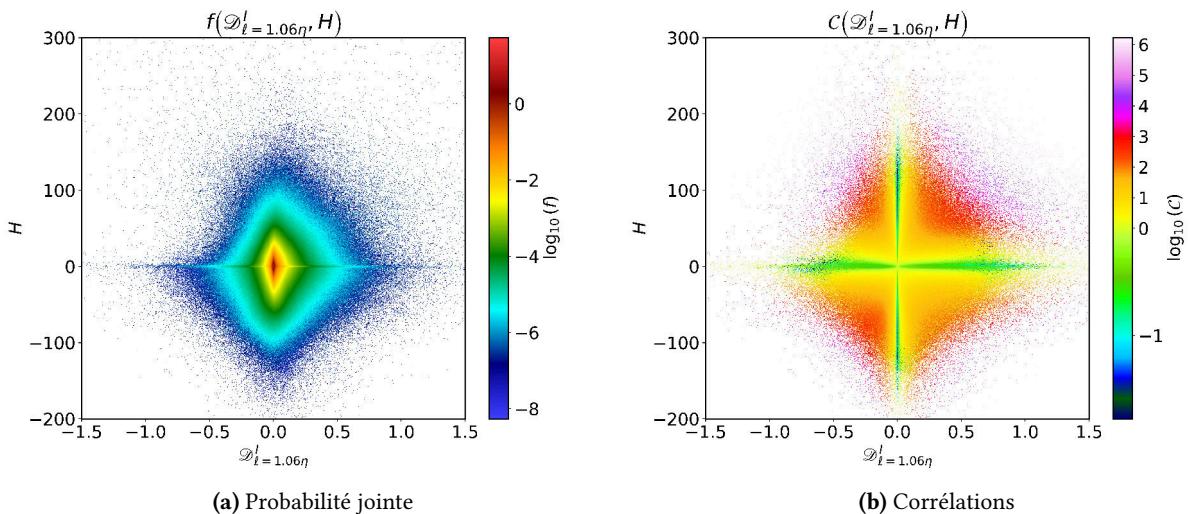


FIG. 24. $D_{\ell=1.06\eta}^I$ vs H

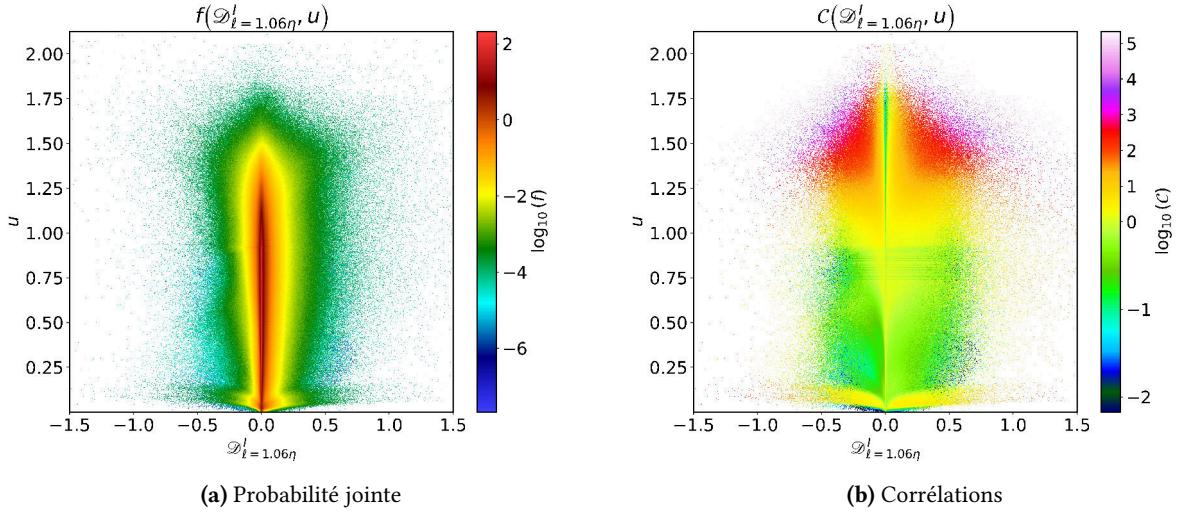


FIG. 25. $D_{\ell=1.06\eta}^I$ vs u

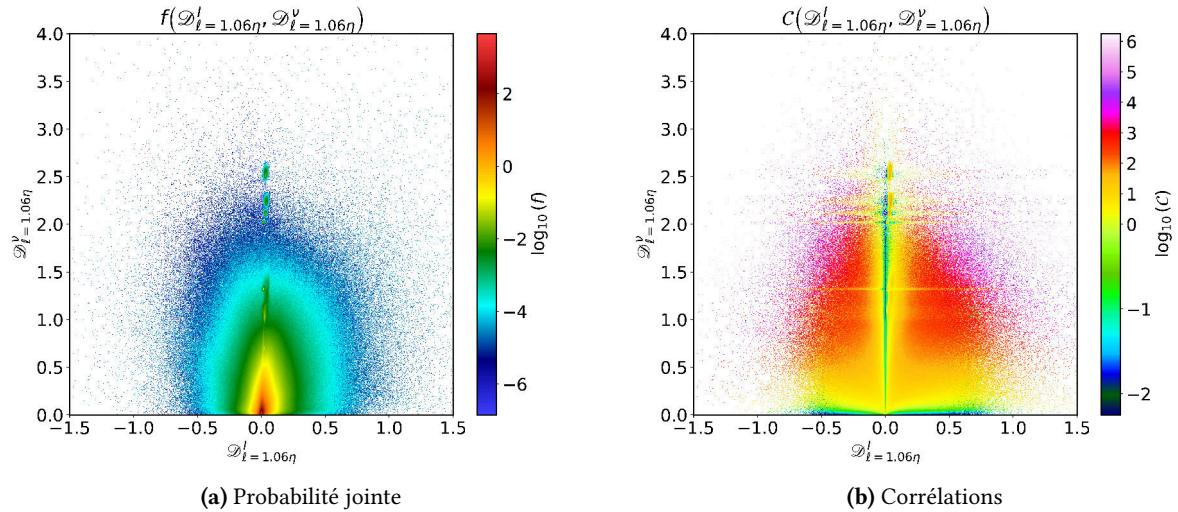


FIG. 26. $D_{\ell=1.06\eta}^I$ vs $D_{\ell=1.06\eta}^\nu$

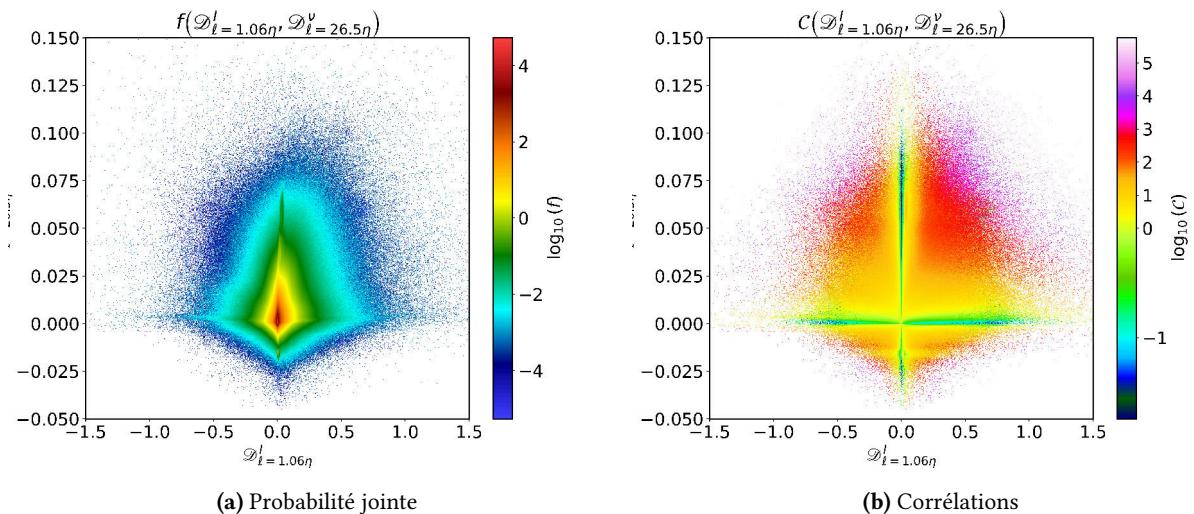


FIG. 27. $D_{\ell=1.06\eta}^I$ vs $D_{\ell=26.5\eta}^\nu$

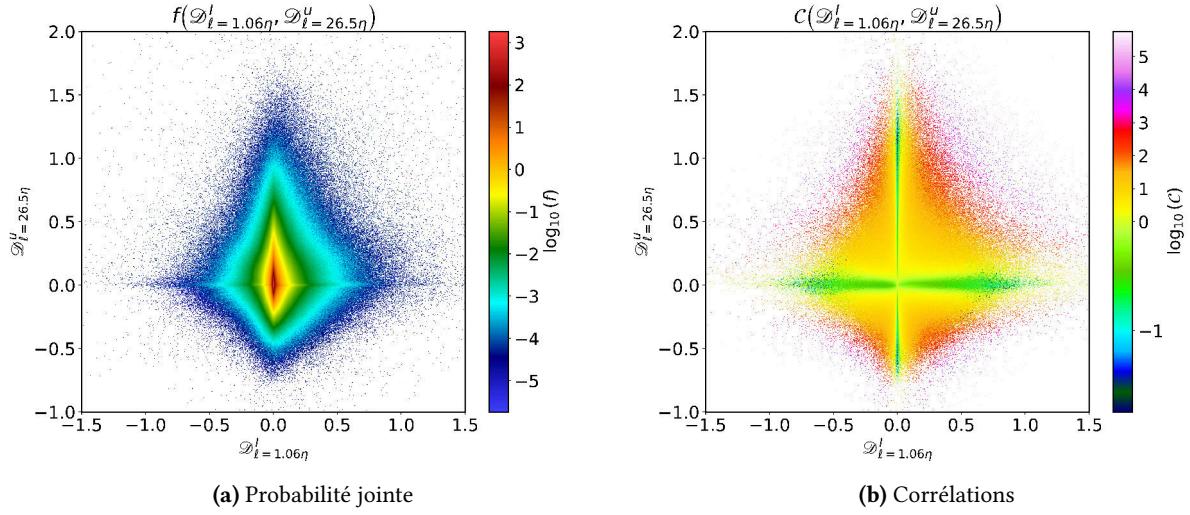


FIG. 28. $D_{\ell=1.06\eta}^I$ vs $D_{\ell=26.5\eta}^U$

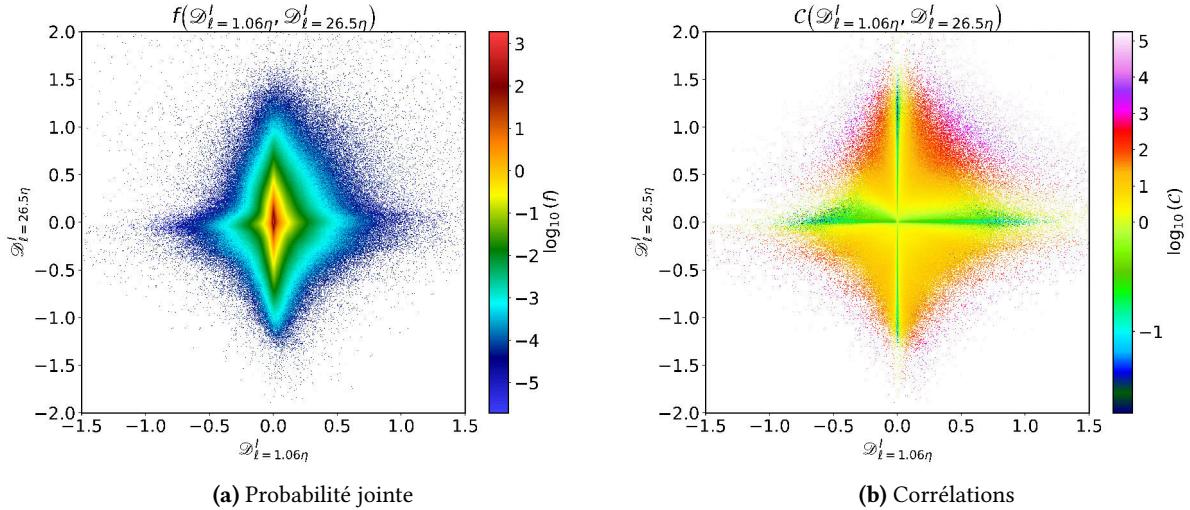


FIG. 29. $D_{\ell=1.06\eta}^I$ vs $D_{\ell=26.5\eta}^I$

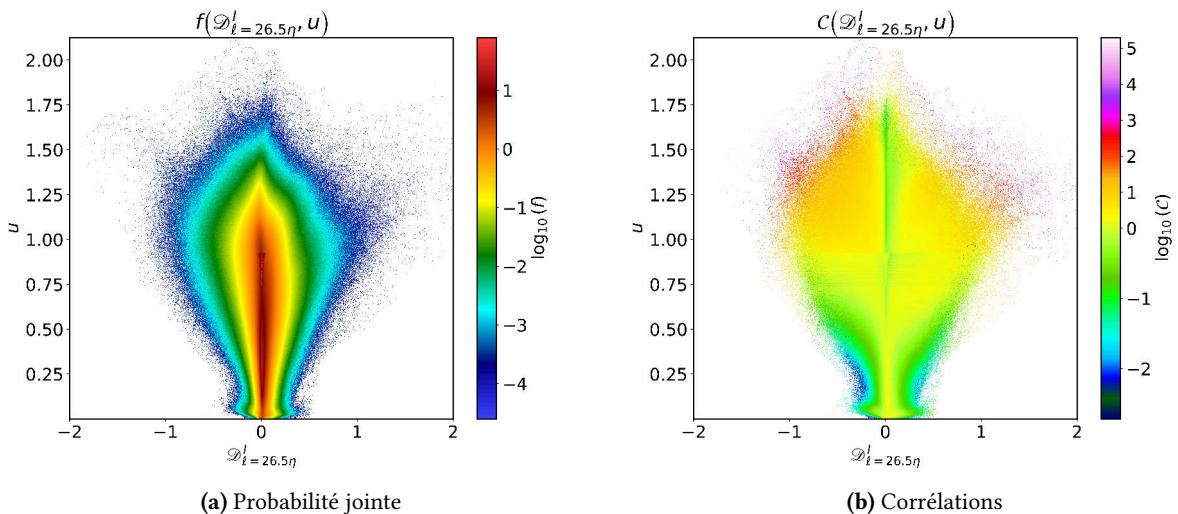


FIG. 30. $D_{\ell=26.5\eta}^I$ vs u

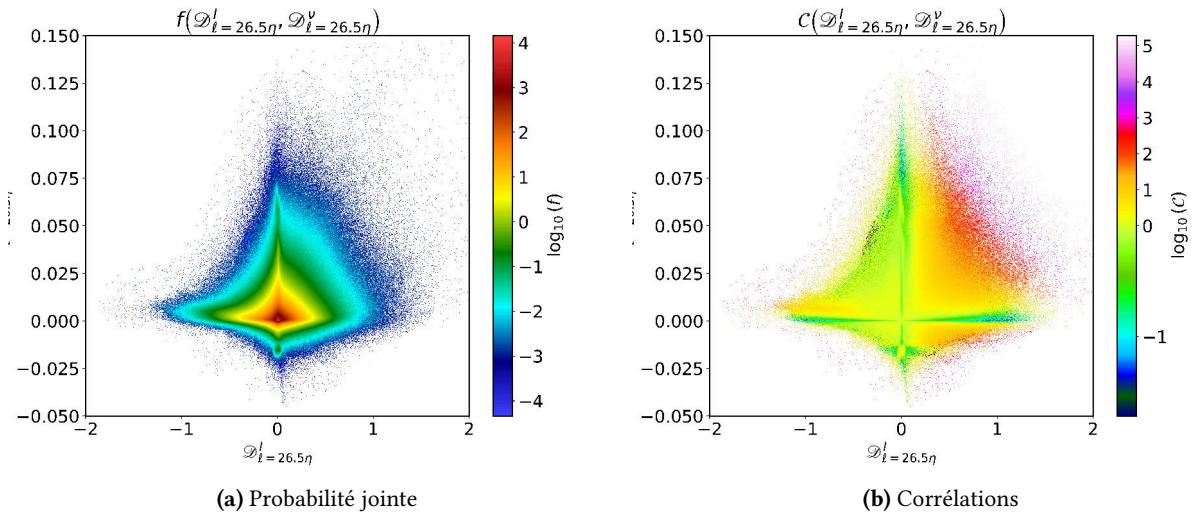


FIG. 31. $D_{\ell=26.5\eta}^I$ vs $D_{\ell=26.5\eta}^V$

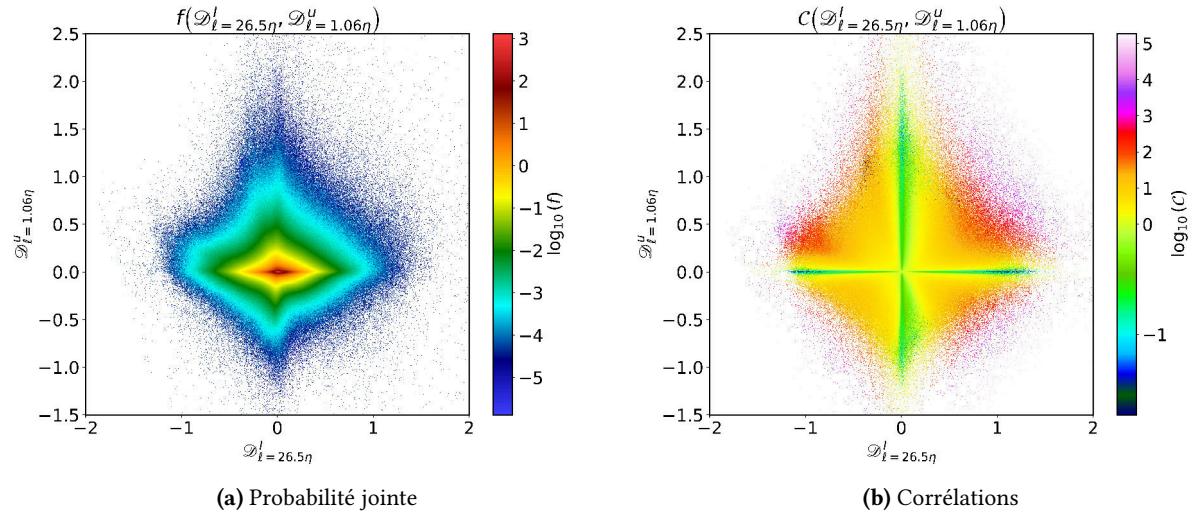


FIG. 32. $D_{\ell=26.5\eta}^I$ vs $D_{\ell=1.06\eta}^u$

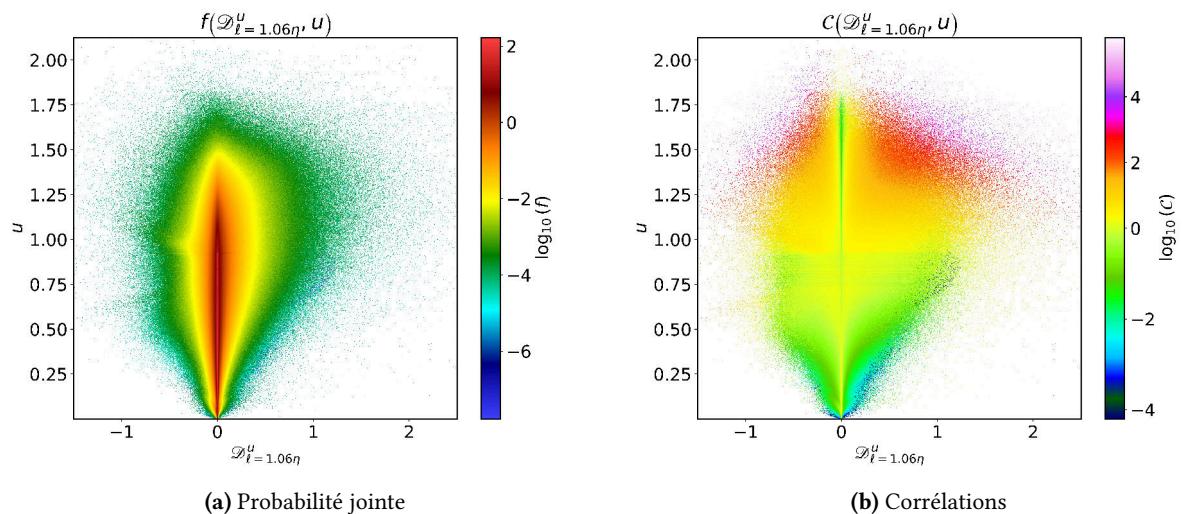


FIG. 33. $D_{\ell=1.06\eta}^u$ vs u

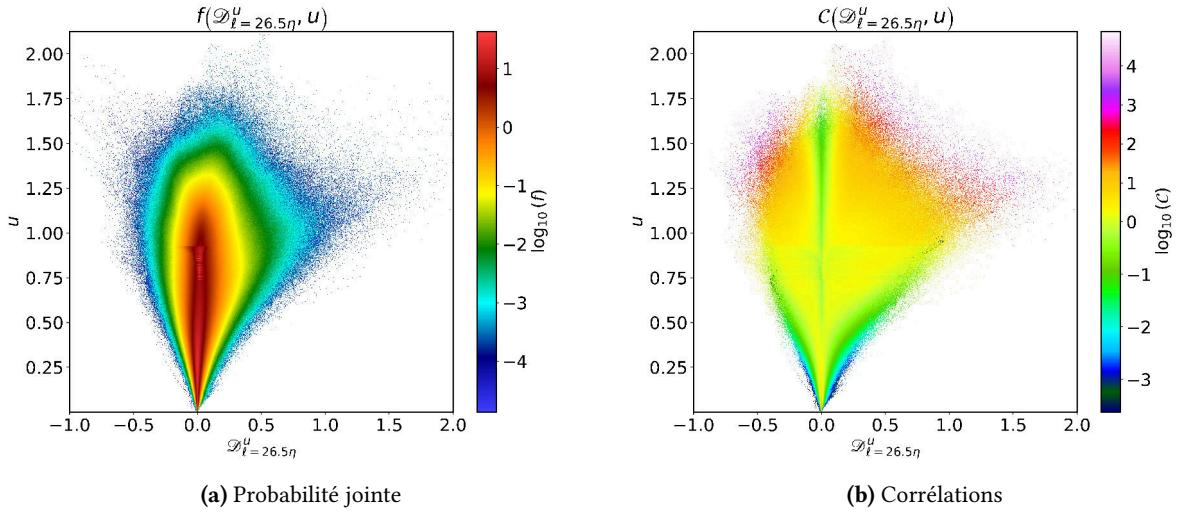


FIG. 34. $\mathcal{D}_{\ell=26.5\eta}^u$ vs u

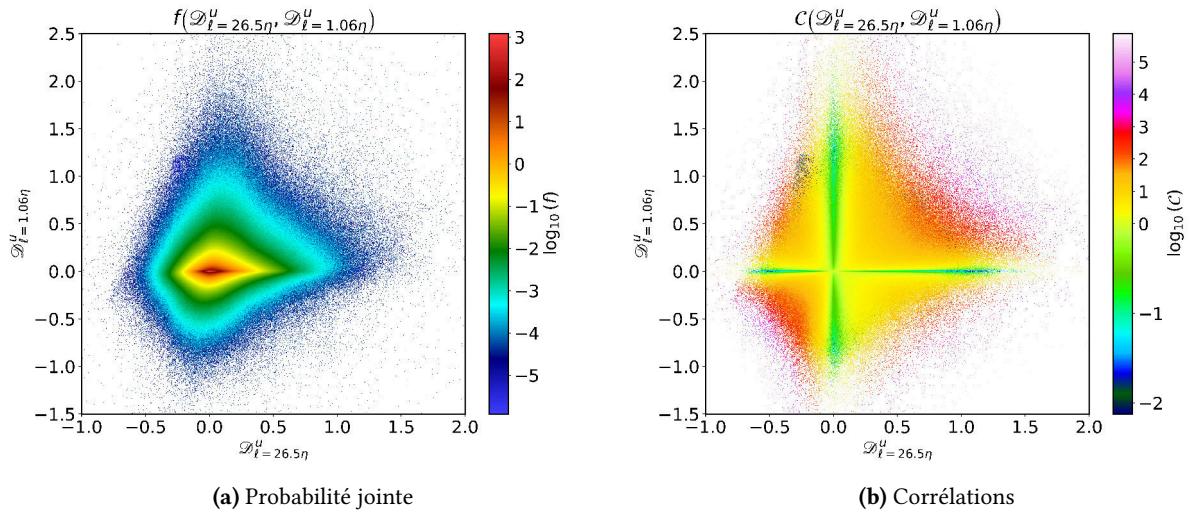


FIG. 35. $\mathcal{D}_{\ell=26.5\eta}^u$ vs $\mathcal{D}_{\ell=1.06\eta}^u$

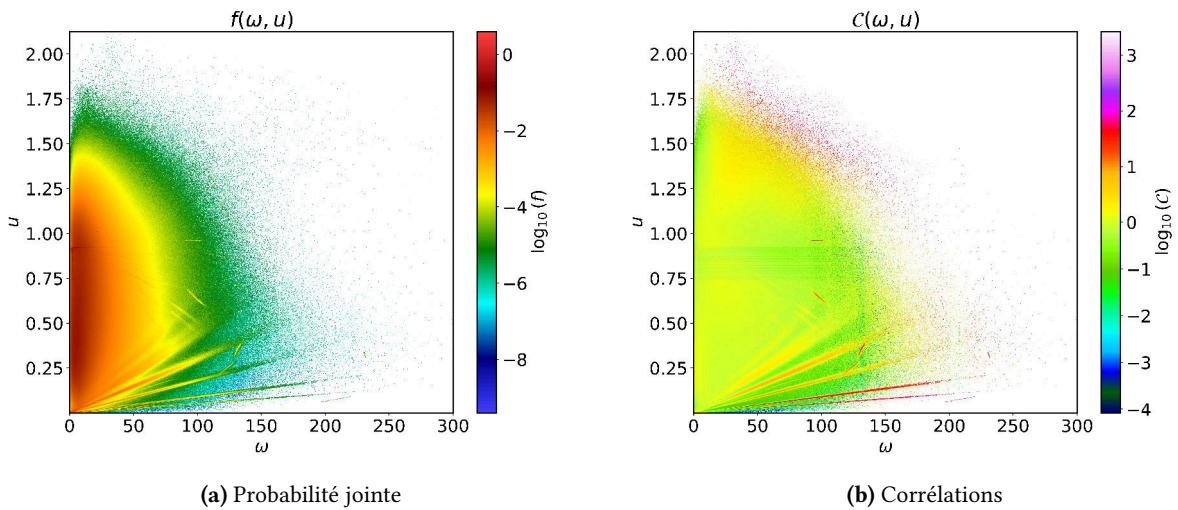


FIG. 36. ω vs u

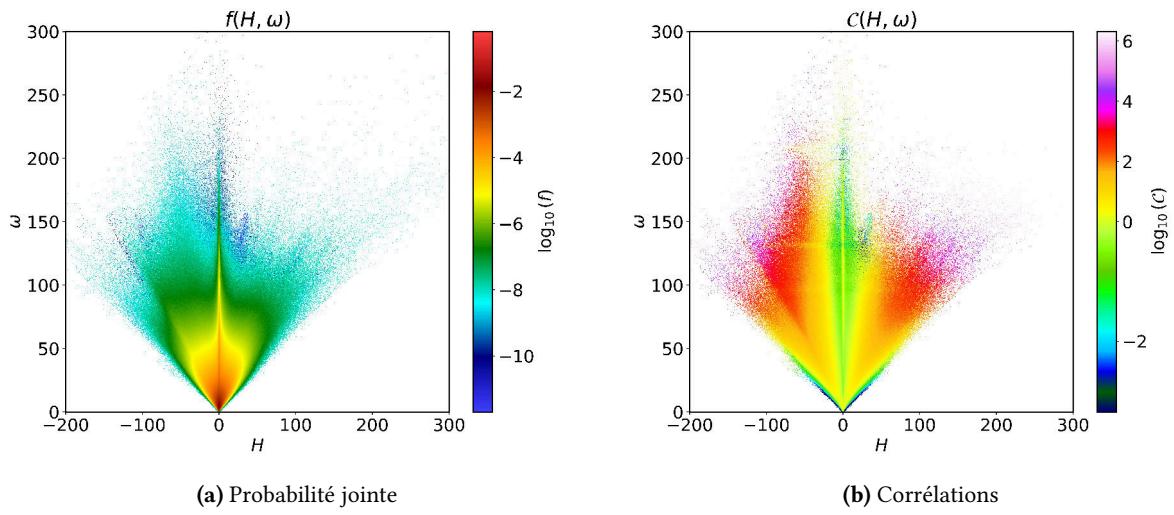


FIG. 37. H vs ω

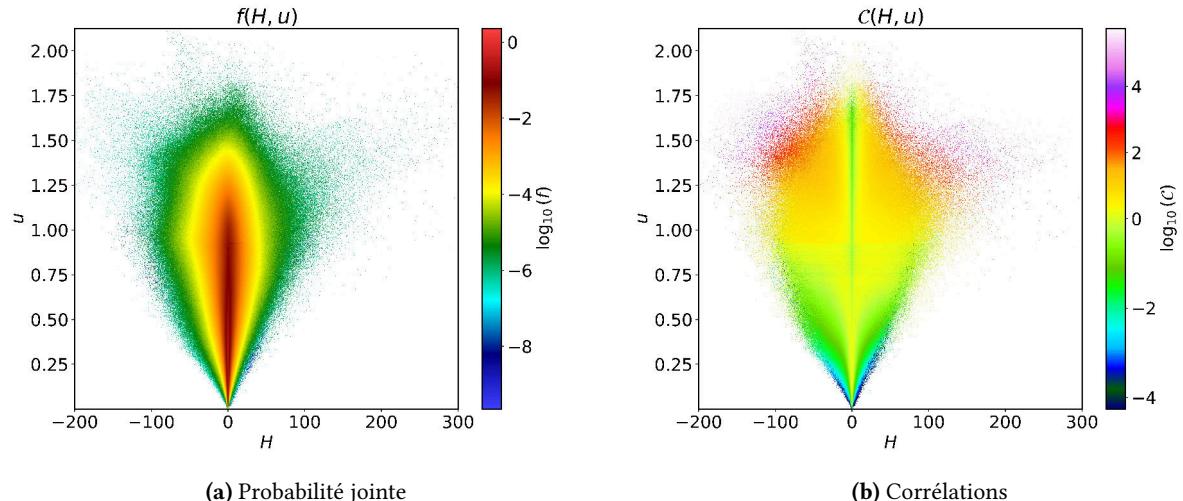


FIG. 38. H vs u

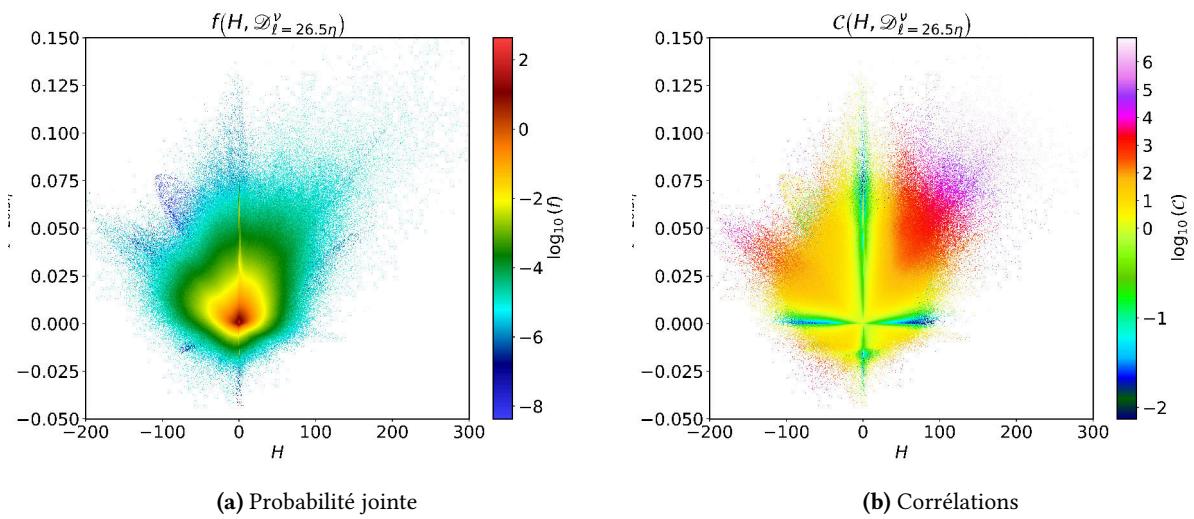


FIG. 39. H vs $\mathcal{D}_l^v = 26.5\eta$

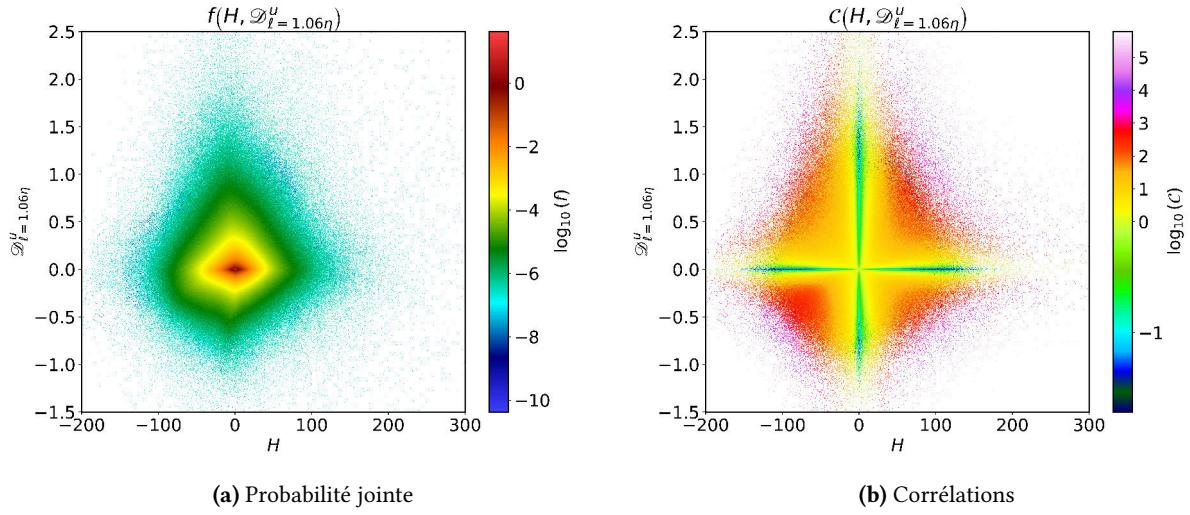


FIG. 40. H vs $\mathcal{D}_{\ell=1.06\eta}^u$

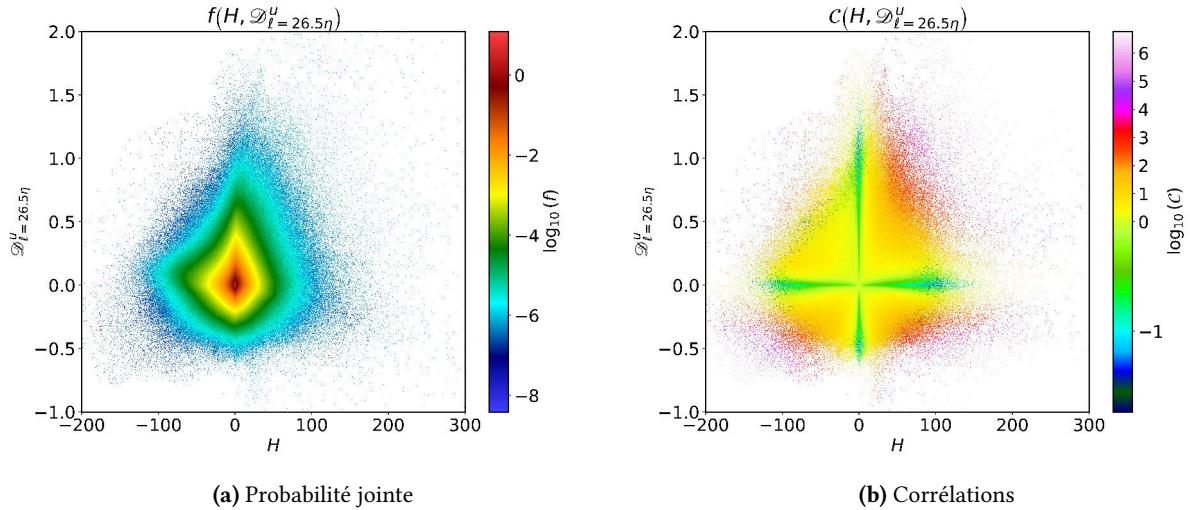


FIG. 41. H vs $\mathcal{D}_{\ell=26.5\eta}^u$

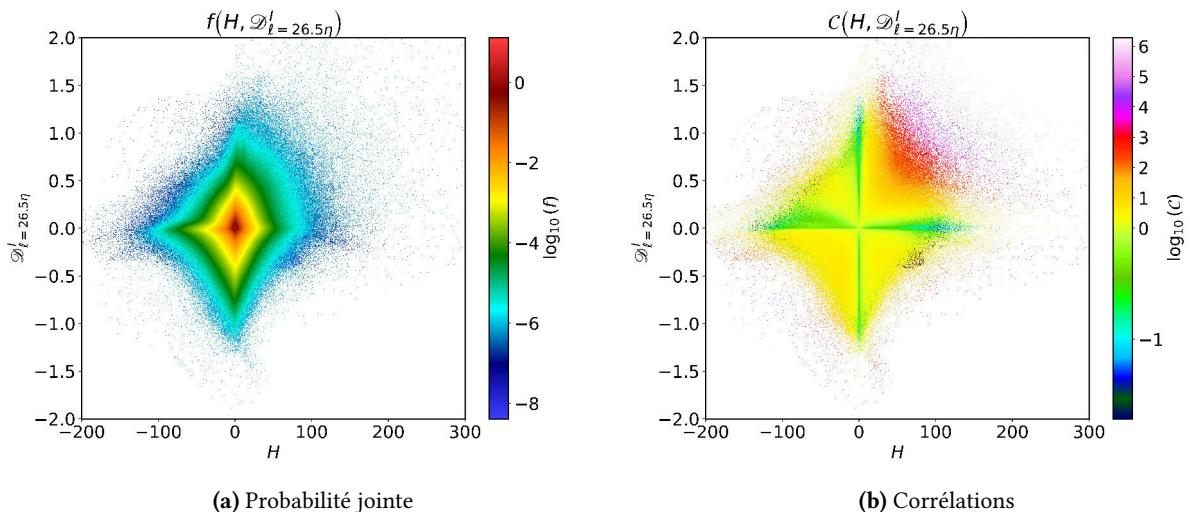


FIG. 42. H vs $\mathcal{D}_{\ell=26.5\eta}^I$

C Optimisation de Python sous GPUs

Pour l'utilisateur accoutumé aux clusters de CPUs, la transition sur une architecture type GPUs peut nécessiter un certain temps d'adaptation. Un utilisateur précipité peut cependant souhaiter rapidement lancer ses scripts sans vraiment se soucier de ce qui se passe au sein des supercalculateurs. Il faut toutefois s'assurer que les ressources mises à sa disposition sont intelligemment exploitées. **De surcroît, les GPUs sont réputés bien mieux performer pour manipuler linéairement des tableaux à très hautes dimensions** (typiquement les calculs statistiques que j'effectue). Étant donné que la documentation native de Lab-IA est très limitée et qu'il est facile de se perdre dans un tumulte d'informations, cette note introduira donc rapidement les concepts fondamentaux associés pour l'utilisation de GPUs sous Python.

C.1 Travailler sur un cluster de GPUs

C.1.1 Architecture

Lab-IA est un cluster de GPUs : il y existe donc une concommittance entre CPUs et GPUs. Les premiers servent les seconds dans un rôle de **distributeurs** au sein d'un job. Étant loin d'être un expert en HPC, je m'arrêterai là pour ce qui est de l'architecture-type de ce genre de clusters.

Comme pour les clusters de CPUs, avant d'y travailler, il faut réserver une partition par le biais de l'ordonnanceur *Slurm*. Le script 4 explicite les instructions minimales à fournir dans un script de soumission pour un job orienté GPUs.

Script 4: Archétype de script de soumission d'un job GPU

```
#!/bin/bash
#SBATCH --gres=gpu:1
[...]
```

On pourrait alors penser que, comme avec les CPUs, il suffit alors de paralléliser son code d'intérêt convenablement à l'aide de librairies existantes et la machinerie peut directement être lancée ! Malheureusement, sans une optimisation supplémentaire, le code se servira uniquement des coeurs mis à disposition du GPU (dans l'exemple du script 4, il y en a un seul par défaut). Tout ça pour dire que le passage au travers d'une interface est crucial avec les GPUs : il s'agit en l'occurrence de *CUDA*.

Si vous n'avez jamais modifié vos algorithmes de façon à activer les librairies de CUDA, les GPUs ne seront pas utilisés. **Ne rajoutez pas l'option `-gres=gpu :1` dans votre script de soumission** : il y a beaucoup moins de GPUs que de coeurs-CPU sur Lab-IA (33 contre 300 !) ; vous pouvez donc quasiment en tout temps lancer votre calcul sur un noeud même si tous les GPUs y sont utilisés.

C.1.2 CUDA

Nous n'allons pas entrer dans le détail des architectures GPGPU (*general-purpose computing on graphics processing units*) ; il suffit de savoir que CUDA est une technologie développée par Nvidia afin d'"activer" un GPU en unité de calculs. Plusieurs versions de CUDA sont pré-installées sur les noeuds de Lab-IA (11.2 par défaut) : cela suffit amplement à notre niveau actuel. Étant donné que nous traitons ici de Python uniquement, nous mentionnons également l'installation des librairies CUDA sous Python. À la distribution CUDA utilisée, il faut associer un *toolkit* comportant toutes les librairies de parallélisation GPGPU de Nvidia. Parce que nous introduirons CuPy à la section C.2.1, le tout peut être installé sous conda avec la commande suivante :

```
$ conda install -c conda-forge cupy cudatoolkit=11.2
```

Un moyen de rapidement suivre l'utilisation effective des GPUs est d'utiliser la commande `nvidia-smi` (une fois connecté sur le noeud, par exemple à l'aide de `srun`) : il faudra alors vérifier si la commande que vous avez exécutée existe parmi les Process du GPU.

C.2 Optimisation de Python

C.2.1 CuPy

Parce que notre post-processing se limite à des statistiques, nous n'avons pas besoin de bibliothèques trop sophistiquées². En l'occurrence, nous utilisons CuPy qui est essentiellement numpy + CUDA. Le module reproduit toutes les méthodes (ou presque) de numpy après les avoir transférées sur un GPU (CuPy a besoin de reconnaître un *device* afin de fonctionner, c'est bel et bien ici que l'option `-gres=gpu :1` dans le job est obligatoire). L'initialisation est aussi simple que `B = cupy.array(A)`; il suffit ensuite de dérouler les fonctions habituelles de numpy emmagasinées sous CuPy. Comme nous le verrons en C.2.2, un GPU est déjà beaucoup plus puissant qu'un CPU afin de manipuler des tableaux; nous ne nous intéressons donc pas ici à des workloads individuels déployés sur plusieurs GPUs³; les exemples que nous donnons dans la présente section exploitent donc un seul device (même si vous en réservez deux, il n'y aura aucun gain; le device reconnu comme 0 par CUDA sera le seul utilisé).

Par-delà la complication de CUDA, l'utilisation de GPUs requiert une gestion de la mémoire plus attentive qu'à l'usuel. Il y a 2 raisons pour cela :

- les GPUs disposent d'une RAM interne qui limite la taille des workloads. Lorsqu'avec un job CPU, vous pouviez allouer autant de RAM qu'un noeud en dispose, vous pouvez toujours le faire avec un job GPU **en prenant garde à ce que le workload invoquant le GPU ne dépasse pas cette limite**. Comme je l'ai expliqué en Section C.1.1, avec un cluster de GPUs, les CPUs sont là pour distribuer les travaux, ils peuvent donc charger un certain nombre d'éléments qui seront transférés sur le GPU. C'est précisément ce transfert qui limite la taille des workloads.
- CuPy dispose d'une gestion de mémoire native qui peut surprendre : la mémoire du GPU n'est pas libérée même quand un objet n'est plus présent sur ce dernier. C'est un comportement **voulu** même si je n'en ai pas très bien compris l'utilité.

Afin de pallier ces limitations, il faut en conséquence modifier le code convenablement.

1. Chaque GPU disposant de sa propre limite de mémoire^a, avant tout job, **il faut effectuer un set de runs-tests afin de mesurer la taille effective des workloads**. Le but est alors de modifier la structure du code afin de faire fitter 1 workload sur le GPU. En pratique, il faut restructurer les données par l'usage de *bags*. La taille du workload est jugée optimale lorsqu'à la fois il remplit la RAM du GPU et qu'un bag ne contient pas *trop* de tableaux. Les deux conditions ne sont pas totalement équivalentes : la deuxième assure que les transferts GPU-CPU ne vont pas transformer le processus en goulot d'étranglement (et donc faire perdre tout l'intérêt de performance).
2. En supposant que les workloads soient déployés séquentiellement (dans une boucle par exemple), on peut améliorer la gestion de mémoire avec les instructions suivantes à la fin d'un processus :

```
del var_1, var_2, ..., var_N  
cupy._default_memory_pool.free_all_blocks()
```

avec {var_i} l'ensemble des variables définies sur le GPU.

Un autre moyen (découvert par hasard) est de spécifier à CuPy d'effectuer des transferts CPU-GPU sans copie : `A = cupy.array(A, copy=False)`; lorsque A sera redéfini dans un nouveau processus, toutes les anciennes variables calculées par CuPy seront à chaque nouveau calcul remplacées.

a. vous pouvez consulter les spécificités de Lab-IA par noeud [ici](#)

Vous pouvez consulter la branche **gpus** de la classe Stats du module [von-Karman-PostProcess](#) et vous verrez bien que les codes de PostProcessing en eux-mêmes sont trivialement modifiés en invoquant simplement cupy à la place de numpy. En ce qui concerne les workloads, pour calculer une densité de probabilité jointe, les runs-tests ont estimé le workload optimal en traitant chaque champ instantané comme bag : on aboutit pour cause à 46 GB avec champ de pénalisation. Seuls les noeuds 1-5 de Lab-IA permettent de déployer ces workloads.

2. la majorité des librairies de machine-learning peuvent reconnaître les GPUs mais veuillez bien lire la documentation associée afin de vous assurer que le GPU est bien exploité

3. le lecteur intéressé pourra se référer à [dask-cuda](#)

C.2.2 Benchmark

Les benchmarks ont été réalisés sur des calculs de norme 3D et d'histogrammes 2D (5000×5000 avec poids statistique) en faisant varier les dimensions des échantillons considérés. Notez qu'on considère uniquement le temps pour retourner les résultats (pas celui pour charger les données qui est naturellement long et peu stable car dépendant du disque). Les figures 43 (a) et 43 (b) affichent un comparatif de performance pour les opérations précédemment mentionnées. En bleu foncé, il s'agit du temps de transfert entre le CPU et le GPU. En bleu cyan visible (autrement dit, en partant de la discontinuité avec le bleu foncé), il s'agit du temps exact du calcul sous GPU. En rouge en partant de l'axe $t = 0$, il s'agit du temps de calcul sous CPU. Le temps de transfert est ici peu pertinent : un GPU nécessite un temps d'échauffement avant d'atteindre son plein potentiel ; **après 2 ou 3 workloads, ce temps de transfert sera réduit d'un facteur allant jusqu'à 5**. Naturellement ici, c'est le temps de transfert primaire que l'on prend en compte. En outre, lorsque l'on fait des calculs sous GPU, le transfert a lieu une seule fois, tous les calculs se font ensuite : en d'autres mots, lorsque l'on enchaîne les calculs (par exemple une norme et un histogramme), le bleu foncé comptera seulement à l'initialisation, tous les calculs suivants se feront en temps rapportés par le bleu cyan.

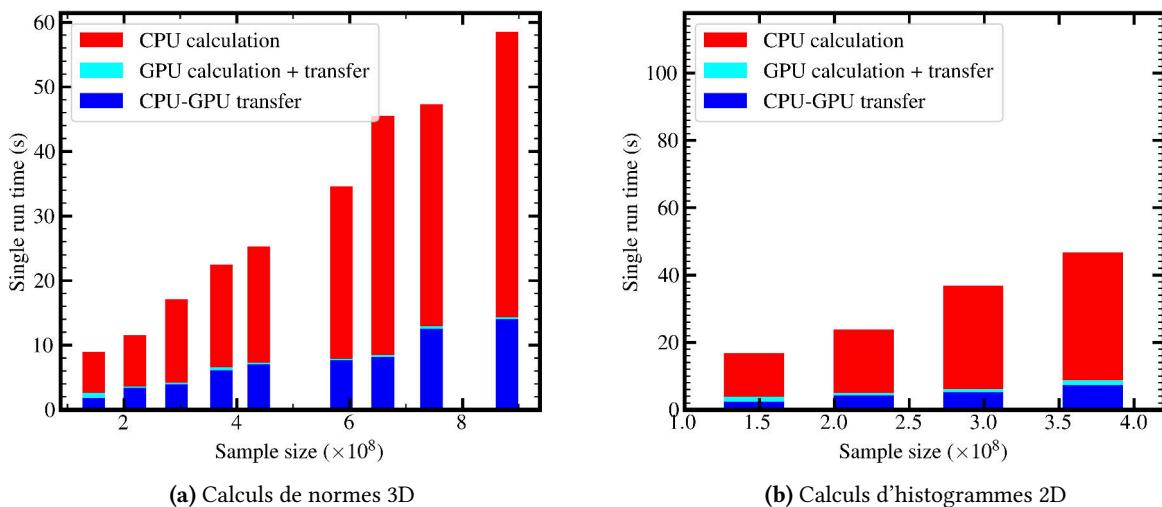


FIG. 43. Benchmarks CPU vs GPU sur Python

D Matériel supplémentaire workloads type probabilité jointe

Étant donné que le code von-karman-PostProcess dispose actuellement d'une branche CPU et d'une autre GPU, le déploiement d'un workload sur Lab-IA pourra être encore plus optimisé en fonction des ressources exploitées. Comme je l'ai mentionné en section §C.2.2, les GPUs accélèrent les calculs statistiques. Toutefois, ces derniers sont limités par leur nombre ainsi que leur mémoire intrinsèque : il sera notamment plus complexe de lancer plusieurs workloads en parallèle sur différents GPUs - ce n'est pas le cas des CPUs dont le nombre et la mémoire allouable sont (très largement) suffisants afin de déployer les calculs en parallèle. On introduit ici les idées clés sous-jacentes au déploiement sur CPU. Des exemples pratiques de déploiement seront rapportés dans la section 2.2 du [tutoriel partagé sur la page git](#).

D.1 Détails techniques workloads GPUs

En section §C.2.1, nous avons indiqué les runs-tests comme primordiales. Nous avons cependant ici un peu triché : le workload maximal a été imposé avec la restructuration des données ; 46 GB (avec champ de pénalisation) car cette structure est facile à générer (slices temporelles). Ces workloads sont uniquement déployables sur les noeuds 1-5 de Lab-IA : on aurait dû normalement adapter la taille selon le GPU réservé (l'inconvénient étant que nous sommes obligés de forcer la réservation sur les cinq premiers noeuds).

On retiendra que, sous GPUs, on déploie un job type probabilités jointes sur les noeuds 1-5 avec 7 coeurs-CPU optimisant la stabilité du chargement des données et 50 GB de RAM CPU⁴.

4. 1 noeud = 8×2 coeurs (128 GB total) + 2 GPUs (48 GB chacun)

D.2 Déploiement parallèle sous CPUs

En toute rigueur, le déploiement séquentiel vu en section §3.2.3 est également possible avec les CPUs : la classe von-Karman-PostProcess a été codée de telle façon à ce que les scripts de PostProcessing soient quasi-indépendants de la branche (il suffit de commenter le bloc 'Transferring to CUDA'). Après avoir retiré l'option `-gres=gpu :1` du script de soumission, nous recommandons également de rajouter un peu plus de mémoire - de l'ordre de 150 GB (les jobs CPUs plantent étrangement avec Slurm, nous suspectons des bugs avec l'instruction `-mem`).

D'autre part, il est également possible de déployer les workloads en parallèle à l'aide de *job arrays* : au lieu de déployer les workloads séquentiellement dans une boucle, on peut déployer 28 workloads en parallèle qui écriront de la même façon les résultats provisoires dans un fichier binaire. Cette approche n'était pas envisageable avec les GPUs : il y a près de 300 coeurs contre 33 GPUs sur Lab-IA, monopoliser 8×6^5 coeurs est en conséquence plus probable que $\min\{33, 8 \times 6\}$ GPUs. Même si cette façon de naivement paralléliser accélère légèrement par rapport aux GPUs, **nous ne recommandons pas de l'utiliser, les bugs de mémoire ont pour cause tendance à augmenter avec les job arrays (il est alors facile de se perdre dans le débogage si les logs ne sont pas rigoureusement rangés)**.

Références

- [1] H. Faller, D. Geneste, T. Chaabo, A. Cheminet, V. Valori, Y. Ostovan, L. Cappanera, C. Cuvier, F. Daviaud, J.-M. Foucaut, and et al., "On the nature of intermittency in a turbulent von kármán flow," *Journal of Fluid Mechanics*, vol. 914, p. A2, 2021.
- [2] B. Dubrulle, "Beyond Kolmogorov cascades," *Journal of Fluid Mechanics*, vol. 867, p. P1, 2019.
- [3] S. Galtier, *Physique de la turbulence*. CNRS éditions (EDP Sciences), 2021.
- [4] A. Kolmogorov, "Dissipation of energy in the locally isotropic turbulence," *Proceedings of the Royal Society of London. Series A : Mathematical and Physical Sciences*, vol. 434, pp. 15–17, July 1991.
- [5] J. Duchon and R. Robert, "Inertial energy dissipation for weak solutions of incompressible euler and navier-stokes equations," *Nonlinearity*, vol. 13, pp. 249–255, Dec. 1999.
- [6] M. Creff, H. Faller, B. Dubrulle, J.-L. Guermond, and C. Nore, "Tracking dynamo mechanisms from local energy transfers : application to the von kármán sodium dynamo," *in preparation*, 2023.
- [7] J.-L. Guermond, "Sfemans documentation," 2021.

5. 6 étant la limite de jobs simultanés par utilisateur