Caractérisation de l'interaction fluide-structure d'un assemblage de coeur de réacteur sous forçage sismique

Romain Rolland

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2022IPPAE005

Thèse de doctorat

ENSTA

IP PARIS

Caractérisation de l'interaction fluide-structure d'un assemblage de cœur de réacteur sous forçage sismique

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'École nationale supérieure de techniques avancées

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Mécanique des fluides et des solides

Thèse présentée et soutenue à Palaiseau, le 21 mars 2022, par

ROMAIN ROLLAND

Composition du Jury :

Elisabeth Lacazedieu Professeur, ENS Paris-Saclay (LMT, UMR 8535)	Président
Benoit Prabel Expert senior, CEA Paris-Saclay	Rapporteur
Guillaume Ricciardi Expert senior, CEA Cadarache	Rapporteur
Morvan Ouisse Professeur des universités, ENSMM	Rapporteur
Philippe Piteau Expert senior, CEA Paris-Saclay	Examinateur
Pierre Moussou Ingénieur senior, EDF (IMSIA, UMR 9219)	Examinateur
Jean-Luc Dion Professeur des universités, ISAE-SUPMECA	Examinateur
Luc Pastur Professeur associé, ENSTA (IMSIA, UMR 9219)	Directeur de thèse

Table des matières

Chapitre 1

Contexte

Le volume de production d'électricité des centrales électriques françaises s'élève à 537.7 TWh pour l'année 2019 [1]. Ce volume est à 70.6% issu de centrales thermiques nucléaires, 11.2% de centrales hydrauliques, 7.9% de centrales thermiques à combustible fossile, 6.3% d'éoliennes, 2.2% de panneaux solaires et enfin 1.8% de bioénergie (biogaz, biomasse, déchets ménagers). Environ 96% de ce volume est issu de moyens de production utilisant la rotation d'un rotor d'alternateur pour produire de l'électricité ; ces moyens de production se fondent sur la conversion d'énergie mécanique en énergie électrique. Ces moyens de production utilisent cependant des méthodes différentes afin d'obtenir le travail mécanique, nécessaire à leur production électrique.

Le travail mécanique des éoliennes provient de l'extraction d'une partie de l'énergie cinétique du vent. L'hélice de l'éolienne, une fois placée dans un courant d'air, est mise en rotation entraînant ainsi directement l'alternateur. Plus la vitesse du vent est importante, plus l'énergie électrique produite est importante.

Le travail mécanique des centrales hydrauliques provient de l'extraction d'une partie de l'énergie potentielle de l'eau. L'eau stockée dans une retenue de hauteur variable suivant les barrages est acheminée vers une roue à augets ou à pâles connectée à un alternateur. Plus la hauteur d'eau est importante, plus l'énergie électrique produite est importante.

Le travail mécanique des centrales thermiques, qu'elles soient nucléaires ou à combustible fossile, provient de l'extraction d'une partie de l'énergie d'un fluide, par exemple de la vapeur d'eau. Le fluide sous pression est acheminé vers un organe de détente qui entraîne la rotation d'un alternateur. Le mécanisme permettant l'extraction de l'énergie du fluide est une machine thermique.

1.1 Machine thermique

Une machine thermique permet de convertir de l'énergie thermique en travail mécanique. Cette conversion est réalisée en appliquant une série cyclique de transformations thermodynamiques à un fluide dit de travail ; la pression et la température du fluide de travail parcourent un cycle thermodynamique. Cette machine est composée *a minima* : d'une source d'énergie thermique, appelée la source chaude ; un dissipateur d'énergie thermique, appelée la source froide ; un extracteur d'énergie mécanique, appelé le moteur. Ce type de machine se rapproche conceptuellement des barrages hydrauliques. De manière analogue à une chute d'eau, pour laquelle plus la hauteur d'eau est importante, plus la quantité de travail mécanique produit est importante ; pour une machine thermique plus la différence de température entre source chaude et source froide est importante, plus la quantité de travail mécanique produit est importante. Dans le contexte de la production d'électricité, le cycle thermodynamique le plus couramment utilisé est le cycle de Rankine ainsi qu'une de ses variantes, le cycle de Hirn. Le fluide de travail le plus couramment utilisé est l'eau.

Lors d'un cycle de Rankine, le fluide de travail subit quatre transformations. La première est une compression sans transfert thermique (souvent réalisée à l'aide d'une pompe) du fluide à l'état liquide. La deuxième est une vaporisation (souvent réalisée à l'aide d'un échangeur de chaleur) permettant de transférer de l'énergie thermique de la source chaude vers le fluide de travail. La troisième est une détente

1. bilan-electrique-2019.rte-france.com/production-totale

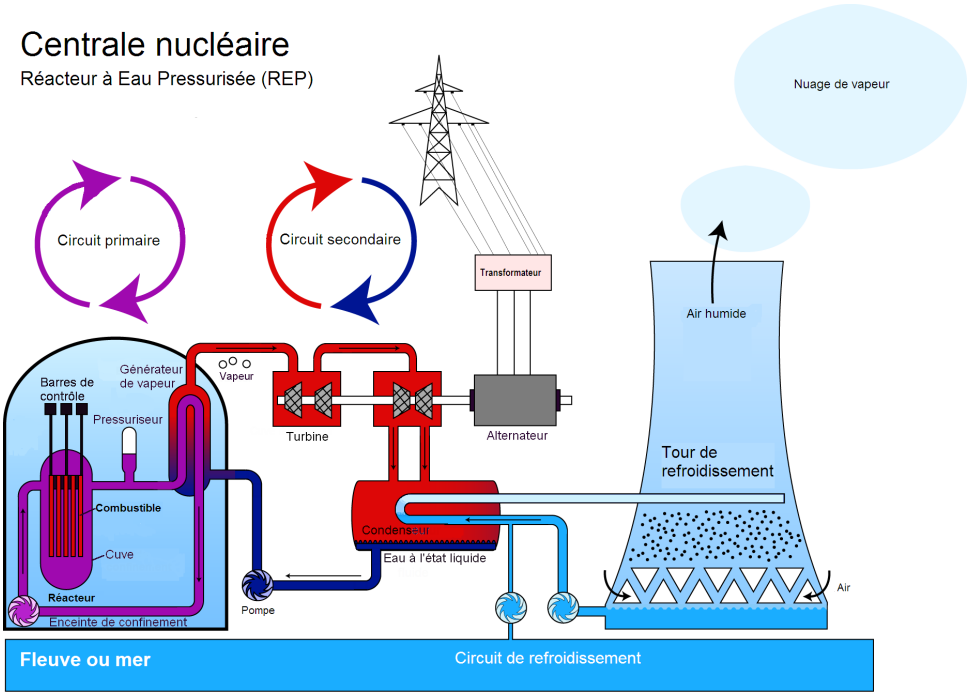


FIGURE 1.1 – Schéma simplifié de l'hydraulique d'un REP [2].

(souvent réalisée à l'aide d'une turbine) du fluide à l'état de vapeur. C'est lors de la transformation de détente que le travail mécanique est produit. La dernière transformation est une liquéfaction (souvent à l'aide d'un échangeur de chaleur) permettant de transférer une part de l'énergie thermique du fluide de travail vers la source froide.

Un cycle de Hirn reprend les transformations du cycle de Rankine en ajoutant une transformation supplémentaire à la fin de la transformation de vaporisation : la surchauffe. Pendant cette transformation, la vapeur nouvellement formée est chauffée une seconde fois. Cette surchauffe augmente l'efficacité de la machine d'une part et évite d'autre part la présence de gouttelettes liquides dans la vapeur. Si le moteur de la machine thermique est une turbine, des gouttelettes contenues dans la vapeur peuvent le détériorer, c'est pourquoi il est indispensable de les éliminer. Les machines permettant de produire un travail mécanique, à travers la production de vapeur d'eau, sont plus communément connues sous le nom de machine à vapeur.

1.2 Machine à vapeur électrogène

Nous proposons d'illustrer le fonctionnement d'une machine à vapeur électrogène moderne par le fonctionnement d'un réacteur à eau pressurisée (REP) présenté en FIGURE 1.1. De manière analogue à une machine thermique générique, trois grands circuits peuvent être distingués pour illustrer son fonctionnement.

Le circuit primaire est un circuit en boucle fermée qui contient la source d'énergie thermique : la cuve contenant le combustible nucléaire. Ce circuit est maintenu sous pression afin que l'eau qu'il contient reste à l'état liquide, donnant son nom à ce type de réacteur.

Le deuxième circuit, également fermé, permet de vaporiser de l'eau liquide en extrayant l'énergie thermique du circuit primaire, par l'intermédiaire du générateur de vapeur. Cette vapeur sous pression est ensuite acheminée, par le même circuit, au travers de turbines à vapeur connectées à un alternateur. La détente de la vapeur dans ces turbines entraîne la rotation de leur rotor et donc la rotation de l'alternateur. La vapeur sortant de la turbine est acheminée au condenseur pour y être liquéfiée, puis réinjectée dans le générateur de vapeur.

2. fr.wikipedia.org/wiki/Centrale_nucleaire

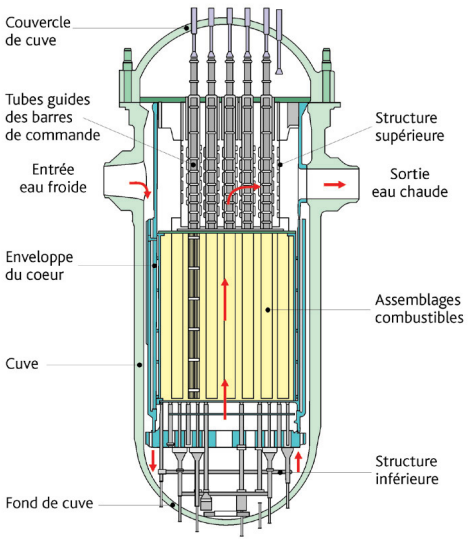


FIGURE 1.2 – Schéma d'une cuve de REP [3].

Le circuit tertiaire sert de source froide et permet de refroidir la vapeur envoyée au condenseur. Ce refroidissement est assuré en pompant de l'eau dans un fleuve ou la mer, suivant la localisation géographique du réacteur. L'eau froide pompée est acheminée au travers du condenseur où elle récupère la chaleur de la vapeur détendue, puis est refroidie dans une tour aéro-réfrigérante. Une tour aéro-réfrigérante permet de refroidir un liquide chaud en le faisant couler en cascade dans un air plus froid. L'échange de chaleur entre l'eau chaude venant du condenseur et l'air froid donne lieu à une pulvérisation et une évaporation de l'eau. Cet aérosol d'eau liquide et de vapeur d'eau est évacué, par convection, par le haut de la tour.

1.3 Assemblages combustibles

En France une cuve de réacteur à eau pressurisée contient environ 200 assemblages (ce nombre varie avec la puissance du réacteur). Les assemblages sont agencés de façon à remplir au mieux le volume cylindrique de la cuve. L'eau primaire entre dans la cuve, descend jusqu'au fond de cuve en empruntant le canal délimité par l'enveloppe de cœur et la cuve, puis remonte dans l'enveloppe de cœur où elle passe dans et autour des assemblages avant de ressortir de la cuve (*cf.* FIGURE 1.2). L'écoulement axial (d'une vitesse de l'ordre de 5 m/s) de l'eau primaire le long des assemblages permet de récupérer une partie de l'énergie thermique produite par la réaction de fission.

L'assemblage combustible (*cf.* FIGURE 1.3) utilisé dans les réacteurs REP français est un objet métallique élancé d'environ 4 m de long et 20 cm de côté. L'assemblage est composé de 264 crayons combustibles rangés sur un réseau carré de 17 par 17 crayons. Chaque crayon est composé d'une gaine en alliage de zirconium dans laquelle sont insérées des pastilles d'uranium. La réaction en chaîne de fission de l'uranium, source de l'énergie thermique nécessaire à la vaporisation de l'eau du circuit secondaire, a lieu dans ces pastilles. Les crayons sont maintenus par plusieurs grilles disposées à différentes altitudes. Le faisceau est complété de 25 tubes guides (insérés à la place de certains crayons) facilitant l'insertion des grappes de commande à l'intérieur de chaque assemblage. Les grappes de commande absorbent une partie des neutrons issus de la fission de l'uranium et ralentissent ainsi la réaction en chaîne. Les grappes de commande permettent de contrôler la réaction en chaîne de fission. Les grappes de commande sont contrôlées à l'aide de l'araignée de maintien des grappes de commande, plus elles sont insérées profondément dans les assemblages, plus la réaction de fission est ralentie.

3. www.irsn.fr/FR/connaissances/Installations_nucleaires/Les-centrales-nucleaires/reacteurs-nucleaires-France
4. www.larousse.fr/encyclopedie/images/Combustible_nucleaire/1001465

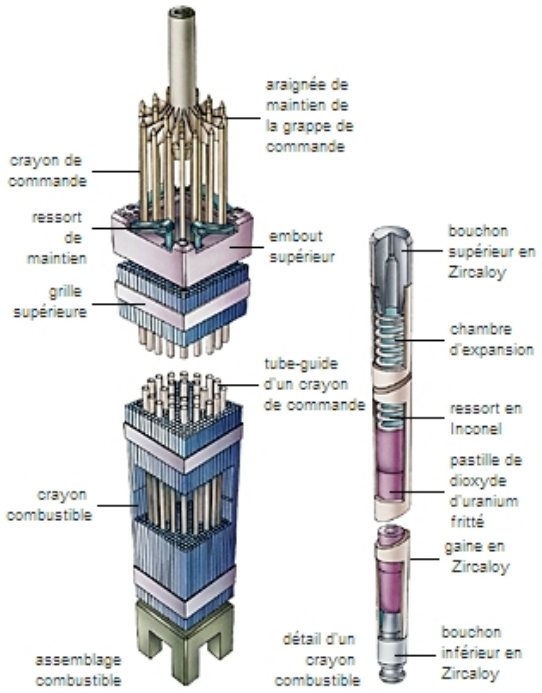


FIGURE 1.3 – Vue éclatée d'un assemblage combustible (à gauche) et d'un crayon combustible (à droite) [4].

1.4 Déformation des assemblages combustibles lors d'un séisme : un enjeu de sûreté

En situation anormale, la priorité de l'exploitant d'un REP est de conserver le contrôle sur la réaction de fission. Dans les situations accidentelles, les grappes de commande sont désaccouplées du dispositif permettant de contrôler leur position, elles tombent alors par gravité jusqu'au fond des assemblages, arrêtant la réaction de fission.

Dans le cas particulier d'un séisme, la secousse peut induire une vibration des assemblages pouvant occasionner des chocs entre assemblages, localisés au niveau des grilles (Viallet *et al.*, 2003). En cas de choc, ces grilles conservent leur géométrie initiale jusqu'à un effort maximal pouvant provoquer une déformation irréversible (Schettino *et al.*, 2013). Déterminer si un chargement dynamique, d'une intensité donnée peut, ou non, mener à des déformations de grilles nécessite, en premier lieu, d'estimer la force nécessaire à la déformation irréversible des grilles et, en second lieu, la force maximale atteinte lors d'un choc entre grilles pour le chargement considéré. La force maximale atteinte est la résultante de plusieurs forces correspondant à différents mécanismes. Parmi ces mécanismes, nous pouvons citer la déformation dynamique des différentes parties de l'assemblage, le contact entre les différentes parties de l'assemblage, mais aussi le transfert d'énergie mécanique entre l'assemblage et le fluide. L'énergie mécanique est transmise de l'assemblage au fluide de sorte que ce transfert d'énergie est un mécanisme de dissipation. Cette dissipation d'énergie absorbe une partie de l'énergie cinétique transmise à l'assemblage par le séisme. Ce mécanisme de dissipation doit donc être quantifié pour évaluer la force maximale atteinte lors d'un choc entre grilles.

1.5 Cadre de l'étude

Dans ce travail de thèse, nous nous intéressons au mécanisme de transfert d'énergie mécanique de l'assemblage au fluide. Nous proposons d'utiliser une approche expérimentale pour étudier l'oscillation, sous écoulement axial, d'un objet qui constitue une épure d'un assemblage combustible. Cet objet doit être constitué d'un faisceau de cylindres rangés sur un réseau carré maintenus par des grilles pour ressembler à l'assemblage réel. En revanche, la géométrie de cet objet doit être la plus simple possible. Nous proposons également de limiter au maximum le nombre de degrés de liberté utilisés pour décrire

le mouvement de la structure, dans le plan orthogonal à l'écoulement axial. Le but de ces simplifications est de se concentrer sur la description du phénomène de dissipation étudié.

1.6 Structure du document

Le chapitre 2 présente un état de l'art sur la description du phénomène de dissipation d'énergie d'un objet élancé oscillant sous écoulement axial. Ce chapitre présente des modèles pour décrire la dynamique d'une structure sous écoulement axial ainsi que les méthodes expérimentales permettant d'obtenir les valeurs des paramètres de ces modèles. Les modèles et méthodes retenus pour l'étude sont discutés.

Le chapitre 3 présente l'objet étudié dans ce travail de thèse ainsi que le banc d'essai utilisé. Les différents systèmes constituant le banc d'essai sont présentés et qualifiés. Les instruments de mesure utilisés sont présentés.

Le chapitre 4 présente les résultats des essais réalisés. Les méthodes présentées au chapitre 2 sont mise en œuvre pour obtenir les valeurs des paramètres des modèles retenus. Une analyse de la capacité des modèles à décrire les essais réalisés, fondée sur les valeurs des paramètres obtenues, est présentée.

Le chapitre 5 présente le modèle numérique utilisé pour compléter l'analyse présentée au chapitre 4. Les choix de modélisations sont discutés. Une analyse fondée sur les résultats des simulations et des essais réalisés est présentée.

Chapitre 2

Amortissement fluide d'une structure élancée sous écoulement axial

Les forces fluides s'exerçant sur des faisceaux de cylindres ont été largement étudiées dans le cas d'écoulements transverses, mais relativement moins pour les écoulements axiaux, qui concernent les assemblages de combustible. Dans la littérature académique, les écoulements axiaux ont essentiellement été étudiés par l'école de M. P. Païdoussis, dans le but de caractériser une famille d'instabilités dont la plus connue est celle du tuyau d'arrosage, et qui se produit dans des gammes de vitesse a priori bien supérieures à celles existant dans un cœur REP : on trouvera dans la référence Païdoussis (2014) une description détaillée de ces mécanismes ainsi qu'une bibliographie fournie. La représentation du fluide est généralement potentielle dans ces études, selon une tradition analytique chère aux mécaniciens : l'expérience montre en effet que des corps raisonnablement bien profilés tels que des cylindres sous écoulement axial, des plaques parallèles à l'écoulement ou des ailes d'avion hors décrochage créent peu de tourbillons, ce qui justifie une représentation simplifiée du champ de vitesse pour prédire les instabilités axiales. Il convient ici de préciser que malgré le « paradoxe de d'Alembert » régulièrement mentionné dans les ouvrages d'hydrodynamique (Lamb, 1932; Guyon *et al.*, 2001), un écoulement potentiel peut parfaitement engendrer une force latérale sur un obstacle par effet Bernoulli, si par exemple une paroi latérale accélère le fluide en le confinant. De tels effets conduisent à l'apparition d'une « raideur fluide » supplémentaire vue de la structure, et dont la valeur est proportionnelle à la pression dynamique ρU^2, ρ étant la masse volumique du fluide et U une vitesse d'écoulement de référence. Il convient de préciser qu'une raideur fluide de ce type est associée à des mécanismes conservatifs au sens de la mécanique rationnelle (Goldstein *et al.*, 2001), et qu'en l'absence de sillage ou de pertes de charges locales, elle ne peut pas engendrer d'amortissement. En présence de décollements et de pertes de charges, des forces fluides supplémentaires apparaissent, qui ne peuvent être décrites dans le formalisme de la mécanique rationnelle (associées à un lagrangien du système par exemple). Il est important d'observer dès maintenant que vues de la structure, de telles forces dissipatives peuvent théoriquement engendrer des variations de raideur fluide, c'est-à-dire d'un terme en apparence associé à une énergie élastique, mais dans la réalité sans lien avec quelque fonction potentielle que ce soit.

2.1 Aspects théoriques de l'amortissement fluide d'une structure élancée

2.1.1 Dissipation fluide lors des oscillations d'un assemblage combustible

Les forces dissipatives ont été mises en évidence de manière indirecte dans le cadre des études menées pour le compte de l'industrie nucléaire dans les années 1980 (Tanaka *et al.*, 1988; Fujita, 1990), et vérifiées par les constructeurs et les exploitants jusqu'à récemment (Flamand *et al.*, 1991; Collard *et al.*, 2003; Viallet et Kestens, 2003; Brenneman *et al.*, 2003; Pisapia *et al.*, 2003; Collard, 2005; Lu et Seel, 2006; Lee, 2013). Ces travaux s'appuient sur des essais réalisés sur des structures qui étaient, soit des assemblages combustibles, soit des maquettes d'assemblage combustible. Ces travaux ont montré, sur

l'objet industriel, que plus la vitesse de l'écoulement est élevée, plus la dissipation est forte. En revanche, ces travaux n'ont pas permis de comprendre finement ce mécanisme de dissipation.

L'objet industriel comporte des grilles qui induisent des difficultés de modélisation supplémentaires. L'écoulement est difficile à modéliser dans le sillage des grilles de mélange. Les grilles de maintien sont des structures complexes dont le comportement mécanique est fortement non linéaire. Nous proposons de considérer une structure la plus proche possible d'un faisceau de cylindres, pour nous affranchir de ces difficultés.

2.1.2 Dynamique d'une structure élancée

Considérons une structure indéformable se déplaçant selon une seule direction. Le mouvement de cette structure est décrite par son déplacement X. Les forces appliquées à cette structure sont séparées en deux contributions : les forces intérieures F_{int}, qui dépendent de la configuration actuelle, et les forces extérieures F_{ext}, qui dépendent seulement de l'instant t. L'équilibre des forces appliquées à cette structure s'écrit :

$$M_s\ddot{X} = F_{\text{int}}(X,t) + F_{\text{ext}}(t),$$

où M_s est la masse de la structure et la notation point désigne la dérivée par rapport au temps. Dans le cas d'une structure immergée dans un fluide, le fluide transmet à la structure des efforts qui s'opposent au mouvement de la structure. Nous séparons les forces intérieures en deux contributions :

$$F_{\text{int}} = F_f + F_s,$$

où F_f représente les efforts transmis à la structure par le fluide et F_s intègre l'ensemble des autres forces internes.

Nous choisissons de représenter la structure par un système masse-ressort-amortisseur. Les forces intérieures sont alors la somme de deux forces opposées au mouvement de la structure : une force d'amortissement et une force de rappel. La force d'amortissement est proportionnelle à la vitesse $\dot{X}$ de la structure. La force de rappel est proportionnelle au déplacement X de la structure. L'expression de F_s est :

$$F_s = -C_s\dot{X} - K_sX,$$

où K_s est une raideur et C_s est un coefficient d'amortissement. K_s et C_s sont des paramètres globaux qui prennent en compte tout ce qui n'est pas modélisé de façon explicite (*e.g.* la déformabilité des différentes pièces qui composent la structure ou le contact-frottant). Le déplacement de la structure est solution de l'équation suivante :

$$M_s\ddot{X} + C_s\dot{X} + K_sX = F_f(X,t) + F_{\text{ext}}(t). \tag{2.1}$$

Pour obtenir un modèle complet, il faut à présent choisir un modèle pour décrire les efforts transmis par le fluide à la structure.

2.1.3 Approche locale : le modèle TLP

Dans la littérature scientifique, le modèle de référence pour la modélisation des efforts fluides s'exerçant sur une structure élancée déformable est le modèle Taylor-Lighthill-Païdoussis (TLP) Païdoussis (2021) propose un article de synthèse sur l'élaboration et l'utilisation du modèle TLP de 1966 à 2021. Ce modèle se fonde sur le modèle proposé par Taylor (1952) pour décrire la nage d'une anguille. Considérons un cylindre faisant partie d'un faisceau de cylindres indéformables oscillant sous écoulement axial. Le faisceau est aligné avec l'écoulement. Les cylindres sont à une distance fixe les uns des autres. Soit x le déplacement selon la direction d'oscillation d'un point du cylindre. Si ce point est suffisamment loin des extrémités, la densité linéique d'efforts f_f^{TLP} exercés par le fluide est (Païdoussis, 1973) :

$$f_f^{\text{TLP}}(x,t) = -\left(C_D\frac{1}{2}\rho D + C_N\frac{1}{2}\rho U D\right)\dot{x} - \chi\rho S\ddot{x}.$$

où D est le diamètre du cylindre, $S = \pi D^2/4$ est la section du cylindre, ρ est la masse volumique du fluide, U est la vitesse de l'écoulement incident, la dérivée par rapport à la variable d'espace est notée prime. χ correspond aux efforts inertiels fluides. C_D et C_N quantifient les forces visqueuses prises en

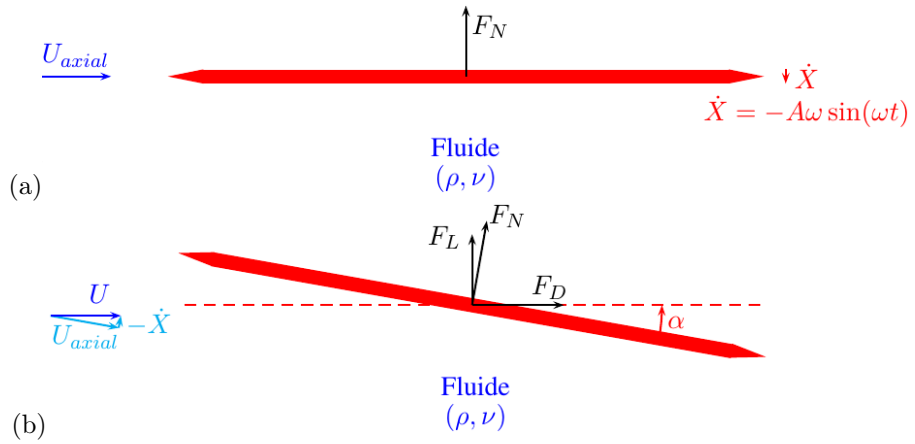


FIGURE 2.1 – Deux configurations équivalentes : (a) cylindre oscillant à la vitesse $\dot{X}$ en écoulement axial et (b) cylindre immobile dans un écoulement oscillant à la vitesse $-\dot{X}$ (d'après Divaret (2014)).

compte par Taylor (1952) pour décrire la dissipation d'énergie induite par l'écoulement ; le coefficient C_N correspond à la portance et le coefficient C_D à la traînée.

En effectuant un changement de référentiel, un cylindre oscillant latéralement à la vitesse $\dot{X}$ est équivalent à un cylindre immobile dans un écoulement oscillant à la vitesse $-\dot{X}$. Par composition des vitesses, le cylindre est donc placé dans un écoulement de vitesse égale à la somme des vitesses axiale (U_{axial}) et latérale ($-\dot{X}$), de norme U, et incliné d'un angle α par rapport à son axe, tel que :

$$\begin{aligned} U_{\text{axial}} &= U\sqrt{1+\left(\frac{\dot{X}}{U}\right)^2}, \\ \tan(\alpha) &= -\frac{\dot{X}}{U}. \end{aligned} \tag{2.2}$$

Si le rapport $\dot{X}/U$ est suffisamment petit, on obtient :

$$\begin{aligned} U_{\text{axial}} &\approx U, \\ \alpha &\approx -\frac{\dot{X}}{U}. \end{aligned} \tag{2.3}$$

Le problème du cylindre oscillant présenté en FIGURE 2.1(a) est donc équivalent au problème d'un cylindre en écoulement oblique présenté en FIGURE 2.1(b), avec un angle d'inclinaison dépendant du temps.

Divaret *et al.* (2014) et Joly *et al.* (2021) ont utilisé cette équivalence pour étudier expérimentalement et numériquement la nature exacte des forces visqueuses prises en compte par Taylor (1952). Ils ont placé dans une soufflerie un cylindre dont l'axe peut former un angle avec la direction horizontale (direction de l'écoulement) et ont utilisé une balance pour mesurer les efforts exercés par le fluide sur le cylindre, selon la direction horizontale et la direction verticale. À partir des forces mesurées, ils ont pu obtenir les valeurs des coefficients C_D et C_N. Divaret *et al.* (2014) a considéré le cas d'un cylindre isolé tandis que Joly *et al.* (2021) a considéré le cas d'un cylindre confiné dans un faisceau de cylindres.

Ils ont pu établir, dans le cas des petits angles ($|\alpha| < 5°$) l'existence d'une force proportionnelle à l'angle α et qui ne se réduit pas à la traînée projetée (αC_D). Cette force proportionnelle à l'angle peut être utilisée pour estimer la force s'exerçant sur un cylindre en écoulement axial et oscillant latéralement. L'estimation de C_N était ainsi sous-évaluée dans les études antérieures, à savoir $C_N = 0.00982$ contre $C_N = 0.11$ chez Divaret *et al.* (2014). Ce nouvel état de l'art a été mentionné par Païdoussis (2021) [1].

Joly *et al.* (2021) a par ailleurs étudié expérimentalement la répartition des forces de pression. Il a mis en évidence l'effet perturbateur des extrémités, où le déplacement de la ligne de décollement peut

1. Païdoussis (2021) exprime cette valeur en fonction du paramètre $c_N = (4/\pi)C_N$; il donne $c_N = 0.0125$.

facilement influer sur la force totale et mettre en défaut la proportionnalité stricte de la force de portance avec l'angle α.

En s'inspirant du modèle TLP, Divaret (2014) a proposé un modèle pour représenter la résultante d'efforts F_f^{TLP} exercés par le fluide sur un faisceau de N cylindres identiques de longueur L :

$$F_f^{\mathrm{TLP}}(X,t) = -C_{\mathrm{amort}}\frac{1}{2}\rho ULND\dot{X} - C_{\mathrm{masse}}\rho SLN\ddot{X}. \tag{2.4}$$

où $C_{\mathrm{amort}} = C_N$ correspond à l'amortissement fluide et $C_{\mathrm{masse}} = \chi$ correspond aux efforts inertiels fluides. En considérant un faisceau de 5×8 avec un rapport pas sur diamètre $P/D = 1.33$, Divaret (2014) a obtenu à partir de mesures de forces $C_{\mathrm{amort}} = 0.18 \pm 0.05$ et $C_{\mathrm{masse}} = 1.45 \pm 0.30$, pour une fréquence de sollicitation f_0, une amplitude d'oscillation A et une vitesse d'écoulement incident U telles que $2\pi f_0/U < \tan(4°) \approx 0.07$. Le modèle proposé par Divaret (2014) ne prédit pas d'effort de raideur fluide sur une structure rigide, compte tenu de l'absence de paroi extérieure ; toutes les positions se valent.

Dans le modèle proposé par Divaret (2014), la résultante des efforts exercés par le fluide, par cylindre est :

$$F_f^{\mathrm{TLP}}(X,t) = -C_{\mathrm{amort}}\frac{1}{2}\rho ULD\dot{X} - C_{\mathrm{masse}}\rho SL\ddot{X}.$$

Nous nous attendons à ce que la densité linéique de force exercée par le fluide en un point d'un cylindre soit :

$$f_f^{\mathrm{TLP}}(x,t) = -C_{\mathrm{amort}}\frac{1}{2}\rho UD\dot{x} - C_{\mathrm{masse}}\rho S\ddot{x}.$$

2.1.4 Approche structurelle en basse fréquence

L'idée consiste à décrire les efforts transmis par le fluide à la structure comme la somme d'une force d'inertie, d'une force d'amortissement et d'une force de rappel. La force d'inertie représente l'inertie du fluide qui entoure la structure. La force d'amortissement représente l'énergie dissipée par frottement à l'interface entre le fluide et la structure. La force de rappel représente une éventuelle « raideur fluide ». Nous définissons ainsi la force F_f^A suivante :

$$F_f^A = -M_f\ddot{X} - C_f\dot{X} - K_fX, \tag{2.5}$$

où M_f est une masse appelée coefficient de masse ajoutée, K_f une raideur appelée coefficient de raideur ajoutée et C_f est un coefficient d'amortissement appelé coefficient d'amortissement ajouté. En choisissant $F_f = F_f^A$, le déplacement de la structure devient solution de l'équation suivante :

$$(M_s + M_f)\ddot{X} + (C_s + C_f)\dot{X} + (K_s + K_f)X = F_{\mathrm{ext}}(t). \tag{2.6}$$

Dans la tradition des Flow-Induced Vibrations (FIV) initiée par Chen (1987), en utilisant cette modélisation, la prise en compte du fluide revient à modifier les paramètres du système masse-ressort-amortisseur qui représente la structure (*i.e.* utiliser M_s+M_f à la place de M_s, C_s+C_f à la place de C_s et K_s+K_f à la place de K_s). Il est important ici de souligner que les coefficients de masse, de raideur et d'amortissement ajoutés condensent un grand nombre de degrés de liberté du fluide, en se fondant sur l'idée qu'en basse fréquence de déplacement (devant les fréquences caractéristiques de l'écoulement), l'écoulement induit par le mouvement de la structure est associé à un champ de déplacement du fluide univoque.

Dans le cas particulier d'un faisceau de N cylindres rigide, le passage du modèle de force locale (2.4) au modèle global (2.6) est immédiat. En effet, en posant :

$$M_f = \rho SLNC_{\mathrm{masse}}, \tag{2.7}$$

$$C_f = \frac{1}{2}\rho ULNDC_{\mathrm{amort}}, \tag{2.8}$$

$$K_f = 0, \tag{2.9}$$

nous obtenons l'égalité entre la résultante F_f^{TLP} des efforts représentés par le modèle TLP et les efforts représentés par la force F_f^A.

2.2 Détermination expérimentale de l'amortissement fluide d'une structure élancée sous écoulement axial

La caractérisation de ces efforts fluides nécessite de mettre en mouvement la structure dans un environnement contrôlé. Deux solutions de mise en mouvement émergent.

2.2.1 Méthode de lâcher libre

La méthode de lâcher consiste à éloigner la structure de sa position d'équilibre et à son lâcher sans autre action (*i.e.* $F_{\text{ext}} = 0$). La décroissance en amplitude de la réponse oscillatoire permet d'estimer l'amortissement. On peut modéliser ce type de configuration par :

$$(M_s + M_f)\ddot{X} + (C_s + C_f)\dot{X} + (K_s + K_f)X = 0. \tag{2.10}$$

On appelle taux d'amortissement le paramètre ξ, défini par :

$$\xi = \frac{C_s + C_f}{2\sqrt{(K_s + K_f)(M_s + M_f)}}.$$

Cette méthode est particulièrement adaptée pour les structures à faible taux d'amortissement $\xi < 1$ tel qu'un cylindre seul comme le montre Divaret (2014) ou dans le cas général, toute configuration permettant l'établissement d'un régime pseudo-périodique. C'est le cas de Stokes et King (1979) avec des essais sur assemblage combustible réel sous écoulement axial inférieur au débit nominal. Dans le cas où l'amortissement est trop élevé pour être mesuré par essais de lâcher, le mouvement de la structure doit être entretenu.

2.2.2 Méthodes d'excitation dynamique

Cette méthode dite d'excitation dynamique consiste à imposer un déplacement, le plus souvent périodique, à la structure et mesurer sa réponse en force. On peut modéliser ce type de configuration par :

$$(M_s + M_f)\ddot{X} + (C_s + C_f)\dot{X} + (K_s + K_f)X = F_{\text{ext}}(t), \tag{2.11}$$

où F_{ext} désigne la force de sollicitation. Il existe plusieurs types de signal d'excitation en usage dans la caractérisation des structures.

Le bruit large bande type bruit blanc, permettant d'exciter tous les modes d'une structure contenus dans la bande en même temps. Ce type de sollicitation permet de faciliter le traitement des signaux de sortie mesurés. Sa qualité est aussi sa principale limite, il est fortement contre-indiqué pour l'étude de systèmes fortement non linéaires, les différents modes étant dans ce cas difficilement discernables et dépendant fortement de l'amplitude de sollicitation. Les efforts fluides étant a priori non linéaires, ce type d'excitation n'est pas exploitable pour notre usage.

Le sinus balayé qui consiste en un sinus dont la fréquence augmente au cours du temps. Il permet d'exciter chaque mode successivement. C'est un très bon moyen d'explorer rapidement une plage fréquentielle, il demande cependant une grande précaution d'usage. L'excitation étant par construction de fréquence variable, cette variation doit se faire suffisamment lentement pour ne pas exciter le système sur plusieurs modes en même temps. Notre retour d'expérience étant faible sur ce type d'excitation, nous préférons l'écarter pour la suite.

L'excitation harmonique au contenu mono-fréquentiel qui à la plus forte probabilité d'exciter un seul mode à la fois. Face à un système peu ou mal connu, ce type d'excitation représente le meilleur parti pour le caractériser.

2.2.3 Estimation des efforts fluides ajoutés

Nous proposons deux méthodes d'estimation des paramètres (M_s, C_s, K_s, M_f, C_f et K_f) du modèle : une méthode d'analyse réalisée dans le domaine temporel et une méthode d'analyse réalisée dans le domaine fréquentiel.

La méthode d'analyse dans le domaine temporel est adaptée au cas où la résultante F_f des efforts transmis par le fluide à la structure peut être isolée. Elle permet de déterminer directement M_f et C_f, en

supposant qu'aucune raideur fluide n'est prise en compte (*i.e.* $K_f = 0$). Elle ne permet pas de déterminer M_s, C_s et K_s. Elle est destinée au dépouillement de résultats de simulations numériques.

La méthode d'analyse dans le domaine fréquentiel ne fait aucune hypothèse *a priori* sur la valeur des coefficients. Elle permet de déterminer les coefficients par paires : $M_s + M_f$, $C_s + C_f$ et $K_s + K_f$. Sous réserve de disposer de jeux de données explorant suffisamment de configurations différentes, elle permet de déterminer tous les coefficients. Elle est destinée aux dépouillements de résultats d'essais.

Méthode d'analyse dans le domaine temporel

Pour un déplacement imposé harmonique de pulsation ω_0 et d'amplitude A, de la forme $X(t) = A\sin(\omega_0 t)$, on a en théorie une force résultante de la forme $F_f(t) = F_0\sin(\omega_0 t + \phi)$. Cette force peut être développée sous la forme :

$$F_f(t) = F_0\cos(\phi)\sin(\omega_0 t) + F_0\sin(\phi)\cos(\omega_0 t). \tag{2.12}$$

Pour ce déplacement X, et sous l'hypothèse $K_f = 0$, les termes de force fluide de (2.5) se développent comme :

$$-M_f\ddot{X} - C_f\dot{X} = M_f A\omega_0^2\sin(\omega_0 t) - C_f A\omega_0\cos(\omega_0 t) \tag{2.13}$$

L'égalité $F_f(t) = -M_f\ddot{X} - C_f\dot{X}$ nous donne :

$$\begin{aligned} C_f A\omega_0 &= -F_0\sin(\phi), \\ M_f A\omega_0^2 &= F_0\cos(\phi). \end{aligned}$$

En utilisant (2.7) et (2.8), on aboutit aux expressions des coefficients d'amortissement C_{amort} et de masse ajoutée C_{masse} suivantes :

$$\begin{aligned} C_{\text{amort}} &= \frac{-F_0\sin(\phi)}{\frac{1}{2}\rho ULNDA\omega_0}, \\ C_{\text{masse}} &= \frac{F_0\cos(\phi)}{\rho SLNA\omega_0^2} \end{aligned}$$

Pour des rapports de vitesse axiale et transverse tel que $A\omega_0/U < \tan(4°) \approx 0.07$, Divaret (2014) a obtenu, pour un assemblage de 5×8 cylindres les valeurs suivantes des coefficients de masse ajoutée et d'amortissement : $C_{\text{masse}} = 1.45 \pm 0.3$ et $C_{\text{amort}} = 0.18 \pm 0.05$.

La connaissance, d'une part, du déphasage temporel ϕ entre le déplacement imposé et la réponse en force de la structure, et d'autre part, de l'amplitude F_0 de la partie fluide de cette réponse en force permet de calculer M_f et C_f. L'obtention de M_f et C_f permet de calculer C_{amort} et C_{masse}.

Le déphasage ϕ est estimé aux instants de passage par 0. Un instant de passage par 0 d'un signal est déterminé par régression linéaire. On cherche la droite passant au plus près d'un groupe de points de mesure, situés dans une bande de largeur égale à 5% de l'amplitude crête à crête du signal, centrée en 0. Le déphasage est obtenu en faisant la différence entre un instant de passage par 0 du signal de déplacement et l'instant de passage par 0 du signal de force qui suit.

Méthode d'analyse dans le domaine fréquentiel

Le même exercice peut être réalisé en se fondant sur la construction de la fonction de transfert, dont l'entrée est la sollicitation en déplacement imposé, et la sortie est la réponse en force complète de la structure. On prend la transformée de Fourier de l'équation (2.11) et on obtient l'équation suivante :

$$[K - M\omega^2 + iC\omega]\widehat{X}(\omega) = \widehat{F}_{\text{ext}}(\omega), \tag{2.14}$$

où $K = K_s + K_f$, $M = M_s + M_f$, $C = C_s + C_f$, ω est une pulsation, $\widehat{X}$ est la transformée de Fourier de X et $\widehat{F}_{\text{ext}}$ est la transformée de Fourier de F_{ext}. La fonction de transfert dont l'entrée est le déplacement imposé et la sortie la réponse en force de la structure est $H = \widehat{F}_{\text{ext}}/\widehat{X}$. D'après (2.14) l'expression de $H(\omega)$ est :

$$H(\omega) = K - M\omega^2 + iC\omega.$$

Les expressions des parties réelle et imaginaire de la fonction de transfert H évaluées en ω sont :

$$[\mathrm{Re}(H)](\omega) = K - M\omega^2, \tag{2.15}$$

$$[\mathrm{Im}(H)](\omega) = C\omega. \tag{2.16}$$

À partir de la transformée de Fourier d'un signal de déplacement imposé et de la transformée de Fourier d'un signal de réponse de la structure, correspondants à une excitation harmonique de pulsation ω_0, on peut construire un point de la courbe représentant l'évolution de $\mathrm{Im}(H)$ en fonction de ω. L'équation (2.16) indique que $\mathrm{Im}(H)$ dépend linéairement de ω. Une régression linéaire permet de construire la droite qui passe au plus près des points ainsi construits. La pente de cette droite est une estimation de C. De manière analogue, on peut construire un point de la courbe représentant l'évolution de $\mathrm{Re}(H)$ en fonction de ω^2. L'équation (2.15) indique que $\mathrm{Re}(H)$ dépend linéairement de ω^2. Une régression linéaire permet de construire la droite qui passe au plus près des points ainsi construits. La pente de cette droite est une estimation de $-M$ et l'ordonnée à l'origine de cette droite est une estimation de K.

Nous envisageons trois campagnes d'essai : une campagne d'essai en air, une campagne d'essai en eau stagnante et une campagne d'essai sous écoulement. Les essais en air fournissent une valeur de référence des coefficients C et M qui représentent alors la dynamique de structure seule ; en d'autres termes, en post-traitant les résultats de ces essais, nous obtenons $C = C_s$ et $M = M_s$. Les essais en eau stagnante permettent de prendre en compte l'augmentation des efforts inertiels induite par l'immersion de la structure ; en d'autres termes, en post-traitant les résultats de ces essais, nous obtenons $M = M_s + M_f$. Les essais sous écoulement permettent de prendre en compte la dissipation d'énergie induite par l'écoulement ; en d'autres termes, en post-traitant les résultats de ces essais, nous obtenons $C = C_s + C_f$.

2.2.4 Décomposition d'un signal périodique en série de Fourier

Nous pouvons voir l'étude de la dynamique de la structure qui nous intéresse comme un système soumis à une sollicitation harmonique. L'entrée de ce système est le signal de déplacement imposé et la sortie de ce système est la réponse en force de la structure. Rien ne prouve que la réponse d'un système à une sollicitation harmonique est un signal harmonique. En conséquence, bien que le signal de déplacement en entrée soit purement harmonique, nous ne pouvons pas nous attendre à ce que le signal de force en sortie soit également purement harmonique. Nous faisons l'hypothèse que ce signal est au moins périodique. Extraire son fondamental est un moyen commode de faciliter son interprétation physique.

Soit s un signal périodique de période T_0 et soit t_0 un passage du signal s par 0 (*i.e.* $s(t_0) = 0$). Le signal s peut se décomposer en série de Fourier comme suit :

$$s(t) = a_0 + \sum_{n=1}^{+\infty} \left[a_n \cos(n\omega_0 t) + b_n \sin(n\omega_0 t) \right],$$

où $\omega_0 = 2\pi/T_0$ est la pulsation fondamentale du signal, le coefficient a_0 est la moyenne du signal sur une période et, pour $n \geq 1$, les coefficients a_n et b_n sont donnés par :

$$a_n = \frac{2}{T_0} \int_{t_0}^{t_0+T_0} s(t) \cos(n\omega_0 t)\ \mathrm{d}t,$$

$$b_n = \frac{2}{T_0} \int_{t_0}^{t_0+T_0} s(t) \sin(n\omega_0 t)\ \mathrm{d}t.$$

On appelle harmonique d'ordre $n \geq 1$ le terme général de la série de Fourier, c'est-à-dire la fonction u_n définie par :

$$u_n(t) = a_n \cos(n\omega_0 t) + b_n \sin(n\omega_0 t).$$

L'harmonique u_n peut se mettre sous la forme suivante :

$$u_n(t) = A_n \sin(n\omega_0 t + \varphi_n),$$

où A_n et φ_n sont respectivement l'amplitude et la phase à l'origine de l'harmonique. L'amplitude A_n et la phase φ_n sont liées aux coefficients a_n et b_n par les relations suivantes :

$$a_n = A_n \sin(\varphi_n),$$

$$b_n = A_n \cos(\varphi_n),$$

ou encore :

$$A_n = \sqrt{a_n^2 + b_n^2},$$
$$\varphi_n = \arctan\left(\frac{a_n}{b_n}\right).$$

On appelle fondamental du signal la première harmonique, c'est-à-dire la fonction u_1 définie par :

$$u_1(t) = a_1 \cos(\omega_0 t) + b_1 \sin(\omega_0 t) = A_1 \sin(\omega_0 t + \varphi_1).$$

Si le signal s est de carré intégrable, l'égalité de Parseval permet d'écrire :

$$\frac{1}{T_0}\int_{t_0}^{t_0+T_0} [s(t)]^2 \, \mathrm{d}t = a_0^2 + \frac{1}{2}\sum_{n=1}^{+\infty}\left[a_n^2 + b_n^2\right] = a_0^2 + \frac{1}{2}\sum_{n=1}^{+\infty} A_n^2.$$

Ce résultat montre que la série de terme général A_n^2 converge. Par conséquent, la suite des A_n^2 est décroissante et tend vers 0. On en déduit que l'amplitude A_1 du fondamental u_1 est plus grande que les amplitudes des harmoniques suivantes. En particulier, si le signal s est harmonique, il coïncide avec son fondamental (*i.e.* $s(t) = u_1(t)$).

La transformée de Fourier de l'harmonique d'ordre $n \geq 1$, notée $\widehat{u}_n$, évaluée pour une pulsation ω s'écrit :

$$\widehat{u}_n(\omega) = \begin{cases} -\dfrac{A_n}{2i}\exp(-i\varphi_n), \text{ si } \omega = -n\omega_0 \\ +\dfrac{A_n}{2i}\exp(+i\varphi_n), \text{ si } \omega = +n\omega_0 \\ 0, \text{sinon} \end{cases}.$$

La transformée de Fourier du signal périodique s, notée $\widehat{s}$, évaluée pour la pulsation ω s'écrit ainsi :

$$\widehat{s}(\omega) = \begin{cases} -\dfrac{A_k}{2i}\exp(-i\varphi_k), \text{ si } \omega = -k\omega_0 \\ a_0, \text{ si } \omega = 0 \\ +\dfrac{A_k}{2i}\exp(+i\varphi_k), \text{ si } \omega = +k\omega_0 \\ 0, \text{ sinon} \end{cases},$$

où k est un entier supérieur ou égal à 1. En particulier, la valeur de la transformée de Fourier du signal en ω_0 est :

$$\widehat{s}(\omega_0) = \frac{A_1}{2i}\exp(+i\varphi_1).$$

2.3 Conclusion

Nous avons présenté un état de l'art des essais réalisés sur des assemblages combustibles réels et des modèles réduits de géométrie similaire. Ces essais ont mis en évidence un effet d'amortissement induit par l'écoulement axial. En revanche, ces essais réalisés sur des géométries complexes n'ont pas permis de proposer un modèle pour décrire cet effet d'amortissement.

Nous avons également présenté le modèle TLP qui décrit la dynamique d'un cylindre sous écoulement axial. Ce modèle décrit la résultante des efforts exercés par le fluide sur un point du cylindre situé loin des extrémités. Divaret (2014) a proposé une extension de ce modèle pour représenter la résultante des efforts exercés par le fluide sur un faisceau de cylindres. Les essais de Divaret (2014) ont été réalisés sur une structure conçue de sorte que l'effet des entretoises entre les cylindres soit le plus faible possible.

Nous proposons de réaliser de nouveaux essais sur un objet de complexité intermédiaire entre l'objet industriel et la structure étudiée par Divaret (2014). Cet objet possède des grilles qui ont une influence sur l'écoulement du fluide mais pas sur la dynamique de la structure. Nous proposons un modèle masse-ressort-amortisseur pour décrire la dynamique de cet objet. Nous proposons un protocole d'évaluation de la capacité du modèle proposé par Divaret (2014) à représenter la dynamique de l'objet sous écoulement

axial. Ce protocole est fondé sur le calcul de coefficients sans dimension à partir des paramètres du modèle masse-ressort-amortisseur.

Nous présentons dans le chapitre suivant la géométrie de la structure étudiée ainsi que le banc d'essai utilisé. Ce chapitre présente la métrologie mais aussi un système innovant de mise en mouvement de la structure conçu pour réaliser une excitation harmonique de la structure.

Chapitre 3

Banc d'essai ELLEZ

3.1 Présentation de l'existant

Le dispositif expérimental utilisé pour les essais dynamiques, présenté dans la partie suivante, est une version modifiée d'une boucle hydraulique déjà existante dont la description et les objectifs détaillés peuvent être trouvés dans (Adjiman, 2016). Cette boucle destinée initialement à l'étude du départ en instabilité statique de faisceaux de cylindres flexibles sous écoulement axial a accueilli les études préliminaires aux travaux dynamiques, notamment des essais de caractérisation de l'écoulement et un essai avec une maquette d'assemblage statique flexible composé de crayons en PMMA non instrumentée. La photo 3.1 présente une vue globale de la boucle. On peut y voir le bâti aluminium supportant le réservoir supérieur, la tuyauterie en PVC gris et l'escalier à droite permettant aux opérateurs de monter sur la passerelle.

3.2 Présentation générale et principe de fonctionnement

Le banc d'essai est composé de 4 grands systèmes qui sont :

— **La veine hydraulique** permettant d'assurer l'écoulement axial autour de la maquette d'assemblage.
— **La maquette d'assemblage** qui sert d'objet d'étude.
— **Le système de forçage** qui permet d'imposer le déplacement de la maquette d'assemblage à tout instant. Pour apporter une certaine modularité dans les mouvements pouvant être imposés à la maquette, le système de forçage est composé de deux lignes d'axe pouvant être synchronisées ou pilotées indépendamment l'une de l'autre.

FIGURE 3.1 – Photo de la zone d'installation de la boucle hydraulique.

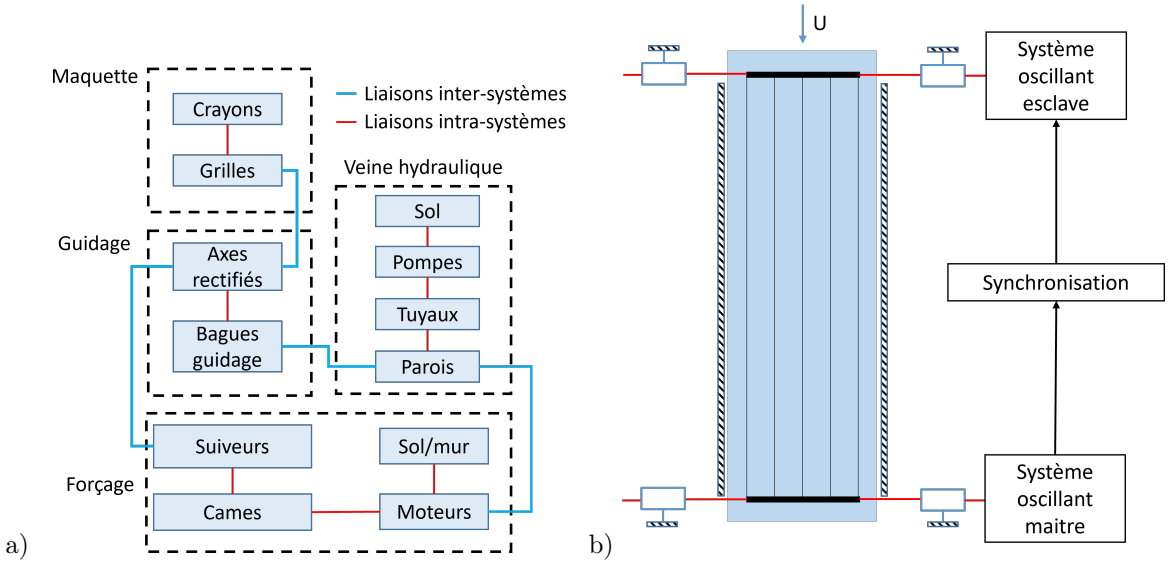


FIGURE 3.2 – Synthèse des systèmes et schéma de principe du banc d'essai.

— **Le système de guidage** qui permet de supporter le poids de la maquette et la guider dans son mouvement linéaire au niveau de ses extrémités.

La FIGURE 3.2 présente dans sa partie gauche une synthèse des différents systèmes explicitant leurs composantes principales ainsi que leurs liaisons, dans sa partie droite un schéma de principe du banc d'essai dynamique tel qu'envisagé.

3.3 Présentation détaillée des systèmes

3.3.1 Circuit hydraulique

Présentation globale

La boucle hydraulique fait partie des canaux hydrauliques dits gravitaires, le moteur de l'écoulement est la gravité. La FIGURE 3.3 présente le schéma hydraulique de l'installation. La boucle est composée d'une veine d'essai de hauteur $H = 2$ m, de section carrée de côté $l = 0.15$ m, présentée en détail dans la section suivante. La boucle dispose de deux réservoirs ouverts à pression atmosphérique. Le réservoir bas a une contenance de 2.2 m^3, le réservoir haut situé à 4 m de hauteur est d'un volume de 0.5 m^3. Le remplissage de ce réservoir est assuré par deux pompes TP100-120 d'une puissance électrique de 2.2 kW et d'une hauteur d'eau et débit nominal de respectivement 7.8 m et 72.8 m^3/h. Le retour par gravité de l'eau peut se faire par 3 circuits distincts :

— la veine d'essai,
— le circuit de fuite,
— le circuit de trop-plein.

La veine d'essai est pourvue à son extrémité d'un débitmètre et d'une vanne réglable manuellement permettant de réguler le débit et d'atteindre la vitesse d'essai souhaitée. Le circuit de fuite relie directement les deux réservoirs et permet de réguler plus finement le niveau d'eau du réservoir supérieur à l'aide d'une vanne manuelle. Ce circuit permet de garder un niveau d'eau constant dans le réservoir supérieur et limiter les dérives dans le temps de la vitesse dans la veine, cette vitesse étant directement pilotée par le niveau d'eau. Le circuit de trop plein sert de sécurité contre le risque de débordement.

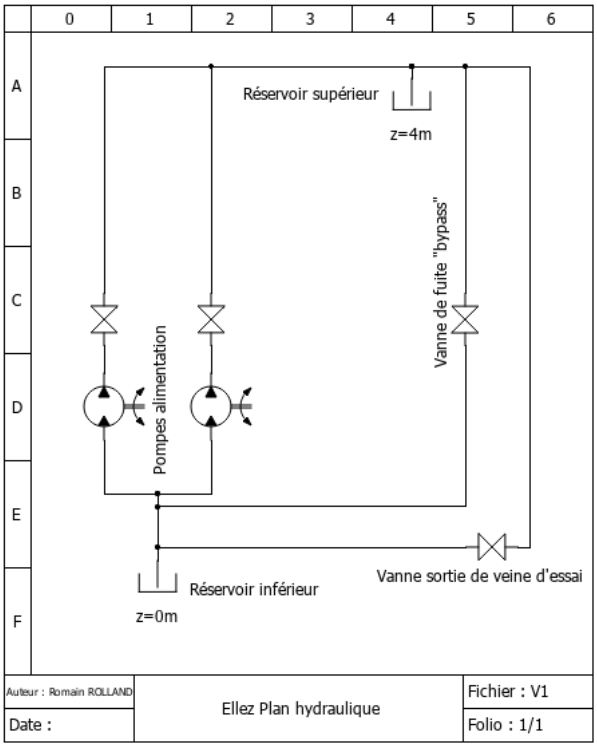


FIGURE 3.3 – Plan du circuit hydraulique.

Veine d'essai

On présente une vue de détail de la veine d'essai [1] en FIGURE 3.4. Cette veine a été réalisée en plusieurs parties vissées entre elles. Ce choix est motivé par des exigences de conception différentes pour chacune de ces parties qui sont détaillées ci-après.

La veine haute fait la liaison entre le réservoir supérieur et la veine d'essai. Elle a également pour fonction de servir de support à un nid d'abeille permettant de casser les structures de l'écoulement entrant dans la veine. Cette pièce doit permettre un accès optique complet afin de repérer un éventuel entraînement de bulle d'air ou une obstruction du nid d'abeille. Elle doit également présenter une certaine flexibilité du fait de sa fonction de pièce de liaison. Elle a été réalisée en plexiglas.

Les manchons de guidage font la liaison entre la veine d'essai et les systèmes de forçage. Ils doivent avoir une rigidité importante pour supporter les contraintes mécaniques qui leur sont appliquées. Ils sont réalisés en aluminium. Une présentation plus complète de ces pièces se trouve dans la partie dédiée au système de forçage.

Le corps de veine permet la circulation de l'écoulement autour de la maquette d'assemblage. Il doit permettre un accès optique sous tous ses angles. Il a été réalisé à partir de plaques de plexiglas usinées puis collées entre elles.

Les hublots présents sur deux faces opposées de la veine permettent un montage et un accès facilité à la maquette d'assemblage. Ils sont réalisés dans la même matière que le corps de veine. Ces hublots sont montés vissés sur le corps de veine à l'aide de goujons. L'étanchéité est réalisée par un joint torique en gorge courant sur tout le pourtour des hublots.

La veine basse fait le lien entre la veine et la ligne de retour au réservoir inférieur. Des pieds, fixés sur les côtés de la veine basse et prenant appui sur le sol, lui permettent de reprendre le poids de la veine. C'est une pièce qui doit être légère, rigide et compatible avec un milieu humide. Elle a été réalisée en aluminium.

L'étanchéité entre chaque partie est assurée par un joint torique comprimé dans une gorge. Les gorges sont respectivement portées par des gorges sur la veine haute, le corps de veine et la veine basse. La

1. Nous remercions P. Jain, T. Ait-Chaite et L. Desprez, techniciens à EDF et T. Pichon, ingénieur d'étude et fabrication à ENSTA Paris pour leurs contributions à la phase de conception de la veine d'essai.

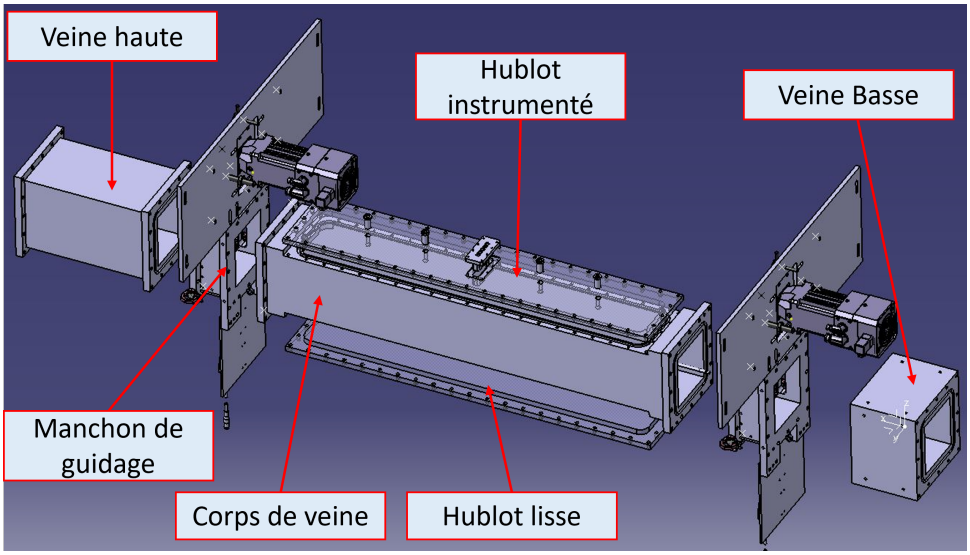


FIGURE 3.4 – Vue en éclaté de la veine d'essai.

compression du joint entre chaque pièce est réalisée par serrage de 18 boulons également répartis sur la portée du contact. Le serrage sur ce type de pièce en plexiglas est une opération délicate du fait de la fragilité du matériau et de la faible reprise d'effort permise par un assemblage collé de plexiglas. Si les dimensions des joints suivent les prescriptions de la norme ISO, le critère utilisé pour la compression du joint en dévie. On choisit ici un serrage empirique tel que le méplat lié à la compression du joint fasse au minimum 5 mm. L'utilisation de ce critère est rendu possible par la faiblesse des différences de pressions mises en jeu sur ce type de boucle hydraulique, ici 0.4 bar au plus haut, et l'accès optique sur l'ensemble des pièces plexiglas.

Nous définissons un repère de l'espace $(\mathrm{O}, \mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z)$ comme suit. L'origine du repère O est le barycentre du corps de veine. La direction $\mathrm{O}z$ correspond à la direction de l'écoulement. La direction $\mathrm{O}x$ correspond à la direction de sollicitation. Le triplet $(\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z)$ est une base orthonormée directe. La base $(\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z)$ est illustrée FIGURE 3.14.

3.3.2 Maquette d'assemblage

La maquette complète est présentée en FIGURE 3.5. Afin de mesurer correctement les efforts fluides, elle doit être légère et indéformable. Elle est composée de deux grilles identiques, servant de point de liaison avec le système de forçage. Ces deux grilles sont placées aux extrémités d'un réseau carré de 5×5 tubes identiques représentant des crayons combustibles. P représente la distance entre deux centres de tubes sur la même ligne et D le diamètre d'un tube. L'écart entre tubes est choisi de sorte que le rapport P/D soit égal à 1.34. Ce rapport est choisi en cohérence avec les travaux précédents, notamment ceux de Divaret (2014), Joly (2018). Ce rapport est proche de celui utilisé sur assemblage réel qui est de 1.326. Des surgrilles sont montées vissées sur les grilles afin de servir de cache-vis ainsi que d'élément "profilant" hydrodynamique. On fait ici le choix de demi-sphères, cette forme étant un compromis entre la minimisation de la surface opposée à l'écoulement transverse et l'adoucissement de la surface opposée à l'écoulement axial.

Grilles

Les grilles réalisées doivent, d'une part, répondre au besoin de rigidité et de légèreté de la maquette et, d'autre part, opposer la plus petite surface possible aux écoulements axial et transverse. Ce sont également elles qui règlent l'écart entre deux tubes, elles doivent donc être produites avec précision. On montre en FIGURE 3.6 une vue isométrique de leur géométrie ainsi qu'une vue de dessus et d'une coupe. Ces contraintes de conception nous amènent à écarter toute solution assemblée et à nous orienter vers des grilles d'un seul tenant. On peut ainsi les produire en aluminium usiné dans le plan sur fraiseuse 3

a)

b)

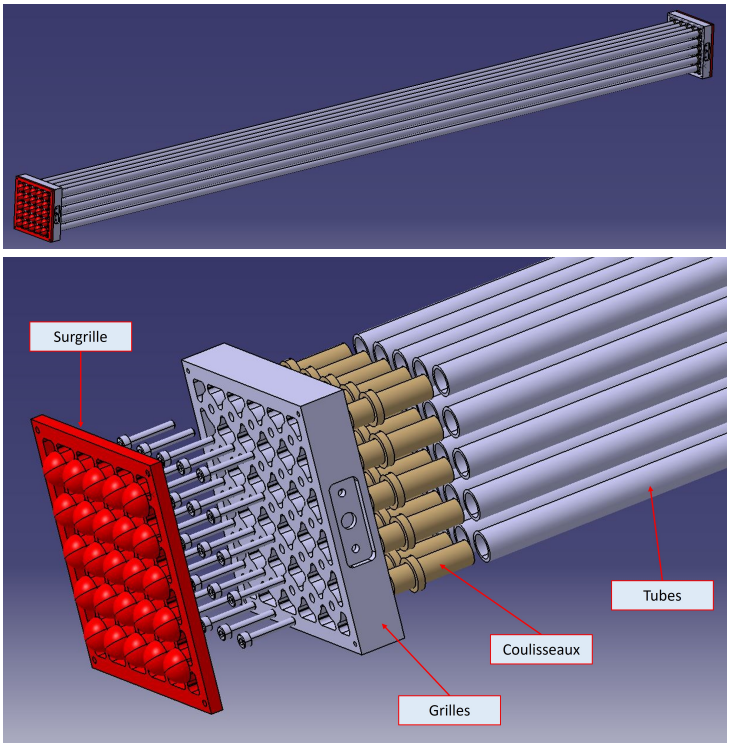


FIGURE 3.5 – Vue globale de la maquette d'assemblage en (a) et vue de détail d'une extrémité en (b).

Milieu	M1 (Hz)	M2 (Hz)	M3 (Hz)	M4 (Hz)
Air	44.0	121.9	239.0	395.0
Eau stagnante	37.8	104.1	204.2	337.5
Eau stagnante avec confinement	36.0	99.1	194.4	321.4

TABLE 3.1 – Estimation des fréquences propres d'un tube utilisé pour la réalisation de la maquette.

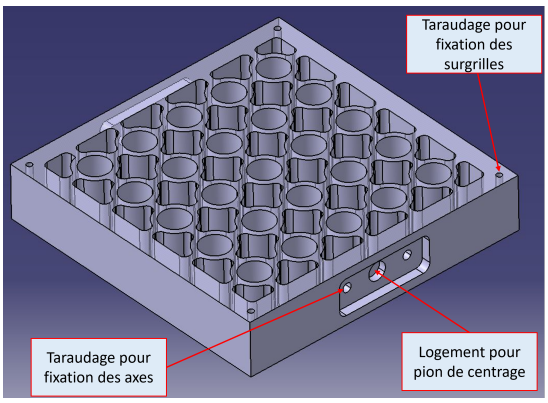
axes, garantissant ainsi une grande précision dans leurs dimensions. La liaison au système de forçage se fait par assemblage vissé. Un logement pour pion de centrage est visible sur la vue CAO, sa fonction sera détaillée dans la partie dédiée au système de forçage.

Tubes

L'amortissement fluide d'un faisceau de cylindres est proportionnel à la longueur des cylindres (*cf.* section 2.2.3). On choisit ici de maximiser cet effort en utilisant la plus grande longueur de tube permise par les dimensions de la veine d'essai, L=1.3 m. On décide également d'utiliser des tubes de diamètre extérieur D=10 mm dans un souci de proximité avec les crayons réels. Enfin, pour satisfaire à la contrainte de rigidité de la maquette tout en minimisant sa masse, nous choisissons des tubes en aluminium de diamètre intérieur d=6 mm.

Afin de mesurer correctement les efforts fluides, il faut s'assurer que la fréquence d'excitation de la maquette ne permettra pas l'entrée en résonance des tubes. Le comportement d'un tube diffère selon que l'essai est réalisé en air ou en eau stagnante. Dans le cas d'un essai en eau stagnante, le comportement d'un tube diffère suivant la position du tube dans l'assemblage. Le comportement d'un tube de l'assemblage est situé entre deux cas extrêmes : le comportement d'un tube seul et le comportement d'un tube confiné par 8 tubes voisins. Dans le cas d'un essai en air, l'effet du confinement est négligé.

La table 3.1 regroupe les fréquences propres théoriques des 4 premiers modes de flexion d'un tube utilisé pour la réalisation de la maquette, dans différentes conditions : un tube vibrant dans l'air, un tube seul vibrant dans l'eau stagnante et un tube confiné par 8 tubes voisins vibrant dans l'eau stagnante.

a) 

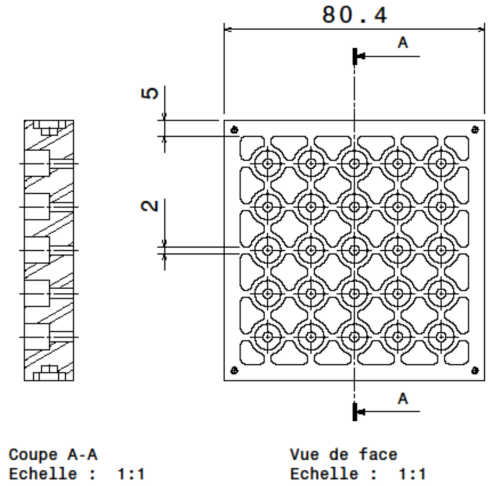
b) 

FIGURE 3.6 – Vue (a) et plan (b) des grilles de la maquette d'assemblage.

Propriété	**Choix**	**Motivation technique**
Matériau	Aluminium	Rapport masse/raideur optimal. $E = 70$ Gpa $\rho = 2700$ kg/m^3
Diamètre extérieur	$D = 10$ mm	Diamètre proche des crayons combustibles REP
Diamètre intérieur	$d = 6$ mm	Compromis entre allègement de la structure et rigidité
Longueur	$L = 1.3$ m	Les plus longs possibles dans la limite des dimensions de la veine d'essai
Pas	$P/D = 1.34$	Pas proche d'un assemblage réel
Nombre	5×5	Complexité jugée suffisante.

TABLE 3.2 – Synthèse des choix de conception pour la maquette d'assemblage.

Ces fréquences propres ont été calculées en modélisant le tube comme une poutre encastrée à chaque extrémité. L'effet de l'air est pris en compte en additionnant la masse volumique de l'air (1.2 kg/m^3) à celle du tube (2700 kg/m^3) dans le calcul des fréquences propres. L'effet de l'eau stagnante est pris en compte en additionnant la masse volumique de l'eau (1000 kg/m^3) à celle du tube dans le calcul des fréquences propres. L'effet du confinement est pris en compte en utilisant une masse volumique de l'eau modifiée. Cette modification consiste à multiplier la masse volumique de l'eau par un coefficient de masse ajoutée. Moretti et Lowery (1976) ont déduit de leurs mesures un coefficient de masse ajoutée lié au confinement égal à 1.381 pour un rapport $P/D = 1.33$. Nous avons fait l'hypothèse que ce coefficient est aussi valable dans notre cas ($P/D = 1.34$).

La table 3.1 montre que la gamme de fréquences ($[0; 5\text{Hz}]$) que nous souhaitons utiliser pour exciter la maquette est éloignée des fréquences propres de chaque tube pris indépendamment. Nous en déduisons que l'excitation n'est pas susceptible de faire entrer la maquette en résonance.

Liaison tube/grille

La liaison entre grille et tube est réalisée à l'aide de coulisseaux en aluminium vissés à la grille d'un côté et glissant dans les tubes de l'autre afin de reprendre les jeux d'assemblage liés à la veine d'essai. Cette solution permet également de garantir des conditions d'encastrement à l'extrémité de chaque tube.

Synthèse de la géométrie et caractéristiques mécaniques

Les choix de conception de la maquette d'assemblage sont résumés en table 3.2.

On donne en table 3.3 une estimation des efforts d'amortissement et inertiels maximaux à l'aide des termes de l'équation 2.4. Cette estimation est fondée sur une sollicitation harmonique de fréquence de

Efforts	1Hz	2Hz	3Hz	4Hz	5Hz
Inertie structure (N)	0.8	2.0	4.4	7.9	12.3
Effort de masse ajoutée (N)	0.2	0.8	1.7	3.0	4.8
Amortissement fluide (N)	0.4	0.9	1.4	1.8	2.3

TABLE 3.3 – Estimation des efforts dynamiques pour un forçage harmonique d'amplitude absolue A=2.5mm.

$f_0 = 5$ Hz et d'amplitude $A = 2.5$ mm ainsi qu'une vitesse d'écoulement axiale $U = 0.9$ m/s. La vitesse U est calculée comme suit :

$$U = U_{\mathrm{q}} \frac{S_{\mathrm{veine}}}{S_{\mathrm{veine}} - NS},$$

où S est la section d'un tube de l'assemblage et $N = 25$. Cette vitesse est jugée plus représentative de l'influence de l'écoulement axial sur la maquette. Divaret (2014) a montré que le coefficient d'amortissement fluide a pour valeur $C_{\mathrm{amort}} = 0.18$ et le coefficient de masse ajoutée a pour valeur $C_{\mathrm{masse}} = 1.45$, pour un pas $P/D = 1.33$ et pour un rapport $2\pi f_0 A/U < \tan(4°) \approx 0.07$. Dans notre cas, le pas est $P/D = 1.34$ et la plus grande valeur du rapport $2\pi f_0 A/U$ est 0.09. Nous faisons l'hypothèse que les valeurs des coefficients proposés par Divaret (2014) sont encore valables pour cette valeur du rapport $2\pi f_0 A/U$. On estime la masse à sec de la maquette d'assemblage à environ 5 kg et la masse de fluide déplacée à 2.7 kg.

3.3.3 Système de forçage et système de guidage

Transformation, prescription et transmission du mouvement linéaire

Il existe dans le monde industriel de nombreux mécanismes permettant d'imposer un mouvement linéaire oscillant à une structure. Nous proposons ici d'exposer ceux utilisés sur des problématiques similaires ainsi que d'autres alternatives susceptibles de répondre au besoin. Nous listons ci-après ces mécanismes en commençant par les systèmes de transformation de mouvement linéaire-linéaire et en finissant par les systèmes rotation-linéaire.

Les vérins hydrauliques composés d'un piston et d'une à deux chambres dont on fait varier la pression, permettent d'imposer de grands efforts et possèdent une bonne fiabilité en fonctionnement. Néanmoins, leur pilotage en déplacement nécessite un asservissement en boucle fermée faisant le lien entre la pression de commande et la position souhaitée. Ce type d'asservissement est complexe à mettre en place puisqu'il doit intégrer dans sa loi de commande la dynamique de l'objet déplacé, notamment les efforts inertiels. Cette solution de déplacement pose également une difficulté logistique, l'organe de puissance de ce type de système étant usuellement une centrale hydraulique pouvant développer plusieurs centaines de bars de pression. Sa mise en œuvre dans le cadre de cette thèse a été jugée trop contraignante.

Les vérins pneumatiques partagent la même complexité que leurs équivalents hydrauliques quant à leur pilotage en position. Cette difficulté est aggravée par la compressibilité du fluide utilisé, en effet celle-ci peut limiter le temps de réaction d'un tel système et mener au départ en instabilité de la boucle de rétroaction. Il est également plus complexe de compenser des efforts inertiels importants. Ce système ne répond pas au besoin de précision du déplacement imposé.

Les systèmes vis/écrous aussi connus sous l'appellation vérins électriques. Ce type de système est constitué d'une vis, accouplée à un moteur, et sur laquelle est montée une douille à bille précontrainte dont la rotation est bloquée. Ce système est très répandu dans les machines-outils industrielles. Il peut transmettre des efforts importants et permet des positionnements précis au prix d'une vitesse de translation réduite. Cependant, imposer un mouvement harmonique à l'aide de ce type de système implique d'asservir à la fois la vitesse de rotation du moteur et sa direction de rotation. De plus, il nécessite de déterminer empiriquement l'accélération et l'impulsion maximales à donner pour se rapprocher d'un signal harmonique. Il en résulte un pilotage moteur complexe demandant un ajustement à chaque modification, même mineure, du système et donc une potentielle variabilité dans les déplacements imposés. Pour cette raison, ce système a été écarté.

Les systèmes bielle/manivelle Ce système est constitué d'un bras articulé à ses deux extrémités appelé bielle fixé sur un point excentré d'une pièce tournante appelée manivelle. Ce type de système

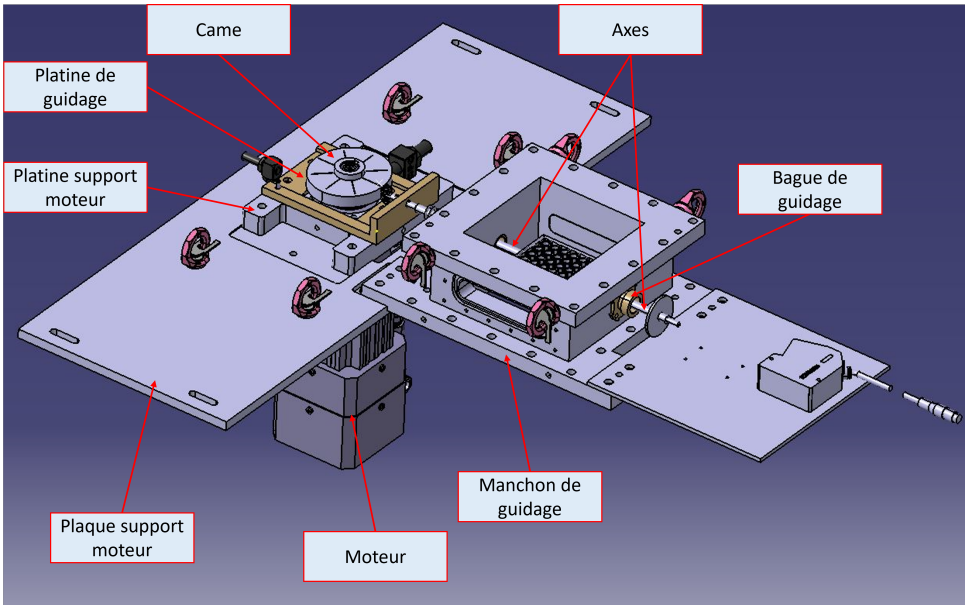


FIGURE 3.7 – Vue globale du système de forçage.

est commun dans les moteurs automobiles à piston monté sur vilebrequin et les pompes médicales dites "volumétriques". Il permet de prescrire, à l'aide d'un asservissement en vitesse du moteur actionnant la manivelle, le déplacement d'une charge ou d'une structure, et ce, dans un intervalle borné. Ceci implique que, contrairement aux systèmes précédemment listés, celui-ci ne nécessite pas de butée mécanique supplémentaire, le mécanisme ne pouvant pas dépasser sa consigne en déplacement, contrairement à un vérin. Ce système comporte par construction plusieurs pivots glissants ou rotules pour permettre la bonne rotation des différentes pièces. Or nous souhaitons ici mesurer un effort de forçage lié à une sollicitation harmonique. Ces liaisons pivots peuvent occasionner des chocs lors du changement de sens de déplacement de la structure, parasitant ainsi la force mesurée. Ce dernier point nous a amené à écarter ce système.

Les systèmes à came sont un autre type de système permettant de transformer un mouvement rotatif en un mouvement de translation. On les trouve dans la plupart des moteurs thermiques au niveau de la commande des soupapes et dans certaines presses industrielles n'utilisant pas de vérin. Ce système est composé au minimum de deux pièces. La première est une pièce tournante, souvent de forme elliptique, appelée came. La seconde est une pièce guidée en translation monodirectionnelle et en contact permanent avec la came appelé suiveur. Ce contact est assuré soit par gravité, soit à l'aide d'un ressort de rappel. Le principe de fonctionnement de ce type de système est simple. La came ayant un rayon non constant et le suiveur ne pouvant translater que dans un sens, la rotation de la came entraîne la poussée du suiveur lorsque ce rayon augmente et la force de rappel entraîne sa traction lorsque ce rayon diminue. Ce système permet d'imposer très précisément un déplacement à des fréquences d'oscillation modestes. Il a, par son principe de fonctionnement, l'avantage de fournir un effort de forçage ne changeant pas de signe contrairement au système bielle/manivelle. Il est enfin très simple de mise en œuvre en termes de pilotage, un asservissement à vitesse constante du moteur entraînant la came est suffisant. Ce type d'asservissement est maîtrisé depuis des décennies sur moteur électrique, il est à la fois précis et robuste. Ce système demande cependant des précautions de conception importantes, notamment liées aux alignements et aux phénomènes d'usure au contact entre came et suiveur. Nous choisissons pour la suite de composer avec ces contraintes et de retenir ce mécanisme pour la conception du système de forçage.

Présentation des choix de conception retenus

Le choix d'un mécanisme à base de came étant fait, on présente dans cette partie les décisions prises pour passer du choix technologique à sa mise en œuvre technique. La FIGURE 3.7 présente les principaux composants d'un système de forçage. L'organe de puissance du système de forçage pour chaque ligne est

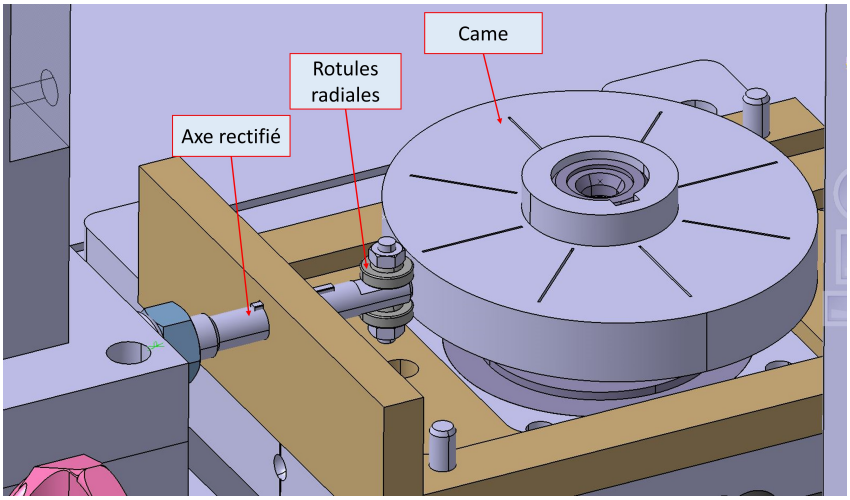


FIGURE 3.8 – Vue du suiveur roulant et de la came.

un servomoteur synchrone Stober EZ404. Ce moteur originellement destiné aux machines-outils robotisées offre une précision angulaire de 0.08 °(donnée constructeur). Ce moteur permet des mises et maintiens à une position angulaire donnée. Une came est montée sur l'axe moteur en prise directe. Le moteur est monté sur une platine support, elle-même montée dans une gorge sur une plaque vissée au manchon de guidage et goupillée. La platine en gorge permet un réglage latéral du système. Une platine de guidage vissée sur la platine moteur permet un guidage de l'axe au plus proche du point de contact came/suiveur, limitant ainsi une éventuelle flexion de l'axe.

Cames et suiveurs

La FIGURE 3.8 présente un gros plan de la came ainsi que de son suiveur. Les cames utilisées ici ont été conçues afin de limiter l'introduction de balourd en rotation dans le système et imposent de ce fait un déplacement sous forme de deux périodes de la fonction sinus pour un tour de came. Les cames ont été conçues en utilisant les règles de conception proposées par Martin (2004). Elles sont réalisées en fonte EN-GJS-500-7 (norme AFNOR) afin d'augmenter leur tenue à l'usure. On fait ici le choix d'un suiveur roulant, faisant le contact entre la came et l'axe. Le contact roulant permet de limiter l'usure au point de contact. On utilise comme suiveur des roulements sans bille appelés rotules radiales. Ces roulements ont pour propriété de tolérer de fortes charges et d'avoir une grande capacité d'auto-alignement. Un ressort de rappel permet de plaquer ce suiveur sur la came et d'assurer le contact entre la came et le suiveur à tout instant.

Axes rectifiés

Les axes utilisés pour la transmission du mouvement linéaire et le guidage de la maquette sont réalisés en acier inoxydable 316L (norme AISI) et rectifiés. Ils sont assemblés vissés de chaque côté des grilles de la maquette. Tous sont pourvus de pions de centrages venant s'encastrer au moment du serrage dans le logement prévu sur les grilles. Ces pions, illustrés FIGURE 3.9, permettent d'aligner toute la ligne d'axe.

Guidage

On utilise pour chaque pivot glissant, servant à guider les axes liés à l'assemblage, des bagues lisses. Ces bagues présentées en FIGURE 3.10 sont réalisées en cupro-aluminium pour sa résistance à la corrosion, notamment galvanique, et à l'abrasion. Elles sont pourvues des logements nécessaires pour porter un joint torique faisant l'étanchéité entre la bague et la paroi de la veine, ainsi que deux joints racleurs en caoutchouc à simple lèvre pour assurer l'étanchéité dynamique des axes et leur lubrification dans le temps. Les joints à lèvre ont été préalablement enduits d'un revêtement plastique (PTFE) afin de limiter

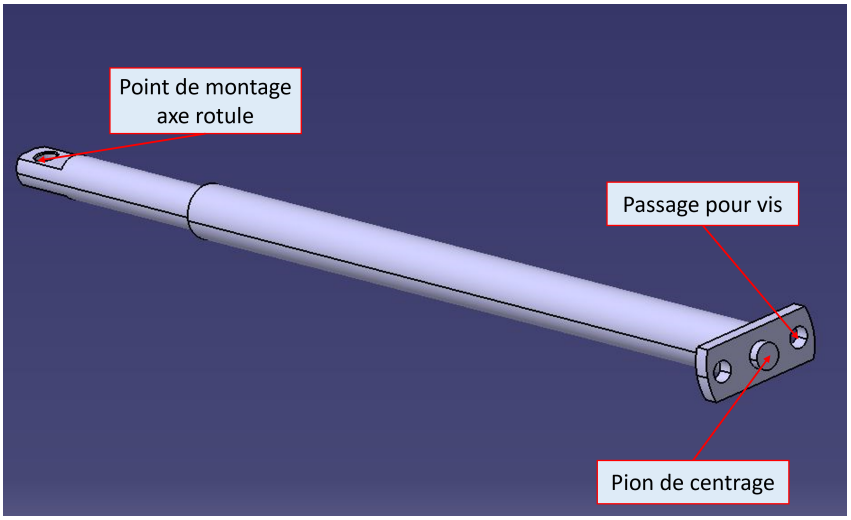


FIGURE 3.9 – Vue d'un axe transmettant le mouvement linéaire.

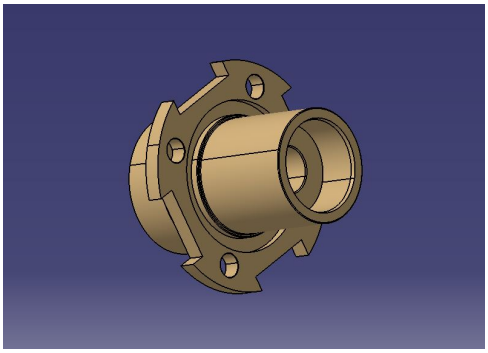
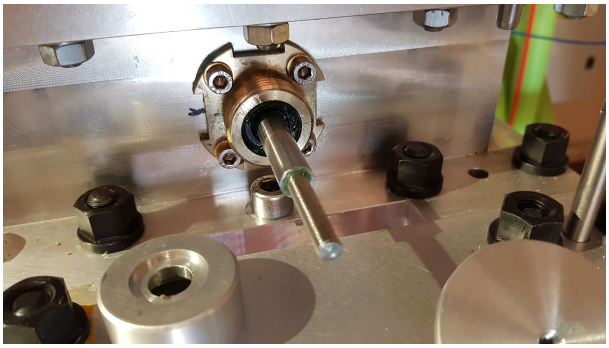
a)  b) 

FIGURE 3.10 – Vue d'une bague de guidage (a) et photo de la pièce installée (b).

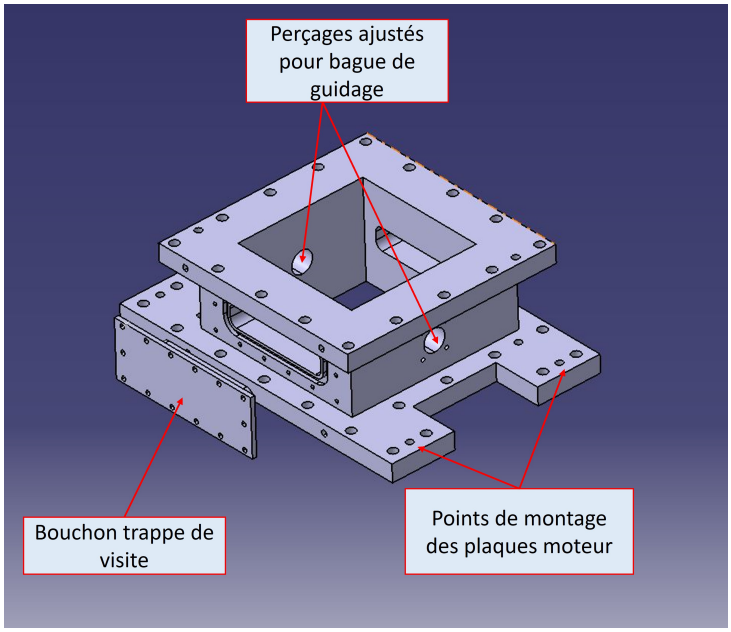


FIGURE 3.11 – Vue d'un manchon de guidage.

le phénomène de *stick-slip*[2]. En effet, le caoutchouc non traité peut adhérer aux surfaces métalliques à faible vitesse et provoquer des alternances de blocages et relâchements brusques.

L'alternative des douilles à billes linéaires a été considérée puis écartée. Cette solution technologique représente l'avantage d'être le moyen le plus sûr de minimiser les frottements, mais présente l'inconvénient d'autoriser de trop grands débattements impropres à assurer le guidage nécessaire aux axes pour un mouvement oscillant répétable.

Manchon de guidage

Les manchons de guidage ont pour fonction de supporter les guidages de la maquette et faire le lien entre la partie hydraulique du banc d'essais et le système de forçage. Ils doivent ainsi être résistants à l'eau et permettre le montage des bagues de guidage sans altérer l'alignement des axes. Pour ces raisons, ces manchons sont réalisés en aluminium. Le choix d'une matière métallique permet une précision plus importante sur la position des perçages accueillant les bagues de guidages. Ces perçages présentés en FIGURE 3.11 ont ainsi pu être réalisés en une seule opération d'usinage, garantissant un désalignement inférieur aux jeux fonctionnels des axes et des bagues de guidage.

Le perçage carré permettant d'adapter la section de la pièce à celle du corps de veine afin de ne pas gêner l'écoulement a été réalisé par découpe au fil. La découpe au fil est un procédé d'électroérosion fondé sur l'utilisation d'un fil tendu. Chaque manchon est pourvu de deux trappes de visite étanches, l'une est fermée par un bouchon en plexiglas afin d'avoir un accès optique aux grilles, l'autre par un bouchon en aluminium. La trappe fermée par un bouchon en aluminium était destinée à être percée pour permettre le passage de câbles électriques. Elle ne sera pas utilisée ici, la maquette n'étant pas instrumentée. L'assemblage des manchons au corps de veine se fait à l'aide de boulons. Chaque manchon est de plus goupillé à l'aide de deux goupilles disposées selon une diagonale passant par le centre du plan de joint. Ce positionnement, de type centreur/*locating*, permet de limiter le désalignement entre les deux lignes d'axe et donc la torsion de l'assemblage lors de la mise en mouvement. Ce désalignement, ou défaut de parallélisme, est ainsi limité à $\pm 0.03°$.

2. Mouvement saccadé parfois observé lors du glissement relatif de deux objets.

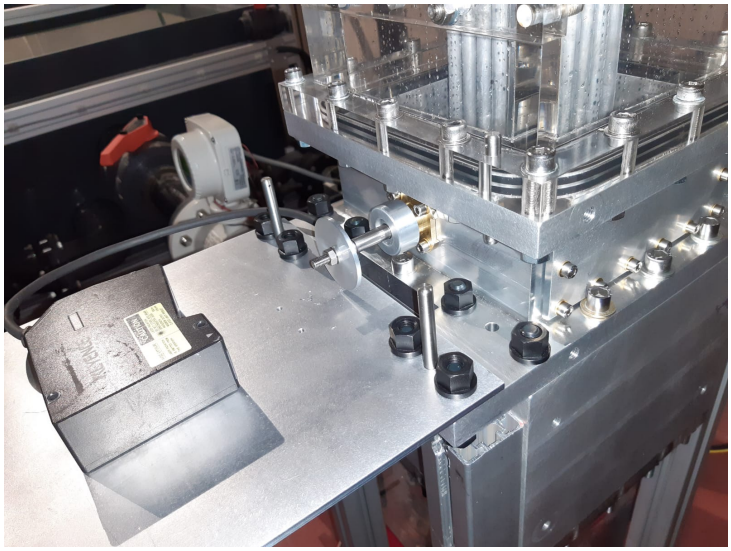


FIGURE 3.12 – Capteur de déplacement installé sur sa platine goupillée.

3.4 Présentation de la métrologie

3.4.1 Mesure de déplacement

La mesure du déplacement réel de chaque grille doit être précise, car elle sert de référence à toutes les analyses réalisées par la suite. On utilise pour cela des capteurs de déplacement laser Keyence LK-G407. Ce type de capteurs, repose sur une méthode de mesure optique, est peu sensible aux vibrations et n'introduit pas de frottement supplémentaire dans le système. De plus, de précédents travaux ont montré leur robustesse à basse fréquence (*cf.* Adjiman (2016)).

Ces capteurs, installés à l'opposé des moteurs, pointent sur des cibles montées sur les axes permettant de guider et supporter la maquette d'assemblage. Afin de limiter au maximum les différences de montage entre la ligne de la grille haute et celle de la grille basse, les capteurs sont montés vissés sur des platines, elles-mêmes vissées et goupillées à la veine. De cette façon, les distances et angles de mesure sont prescrits par les tolérances d'usinages des pièces assemblées. La tolérance maximale permise par l'assemblage goupillé est de 0.08 mm en translation, ce qui conduit à une variation angulaire de $\pm 0.04°$ entre l'axe de mesure du laser et le plan de la cible. Ce faible écart permet de considérer comme négligeable les différences de mesure entre les deux capteurs de déplacement.

3.4.2 Mesure de force

La mesure de l'effort nécessaire au déplacement de la maquette est également essentielle à la caractérisation de l'amortissement. On utilise pour cela deux cellules de force piézoélectriques annulaires Dytran 1210C2. Ces capteurs sont montés et précontraints sur les axes de commande des systèmes de forçage (*cf.* FIGURE 3.13). Ces capteurs sont adaptés aux mesures statiques et basses fréquences. Ces cellules sont utilisées avec des amplificateurs de charge Kistler 5011. D'autres types de capteurs ont été considérés tels que les capteurs vissés ou des capteurs galette. Ceux-ci ont été écartés, leur intégration au système de forçage posant un problème d'alignement. Les assemblages vissés de ces capteurs ne permettent pas une mise en position suffisamment précise pour notre besoin.

3.4.3 Mesure de pression et d'accélération

Pressions

Neuf capteurs de pression 113B28SN de marque PCB sont montés sur le hublot. Ces capteurs mesurent la fluctuation de pression résultant du mouvement de la maquette d'assemblage (*cf.* FIGURE 3.14). Si les

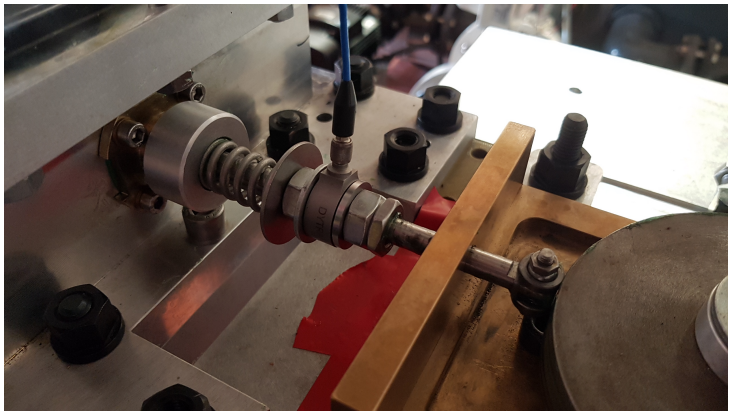


FIGURE 3.13 – Photo d'une cellule de force installée sur son axe.

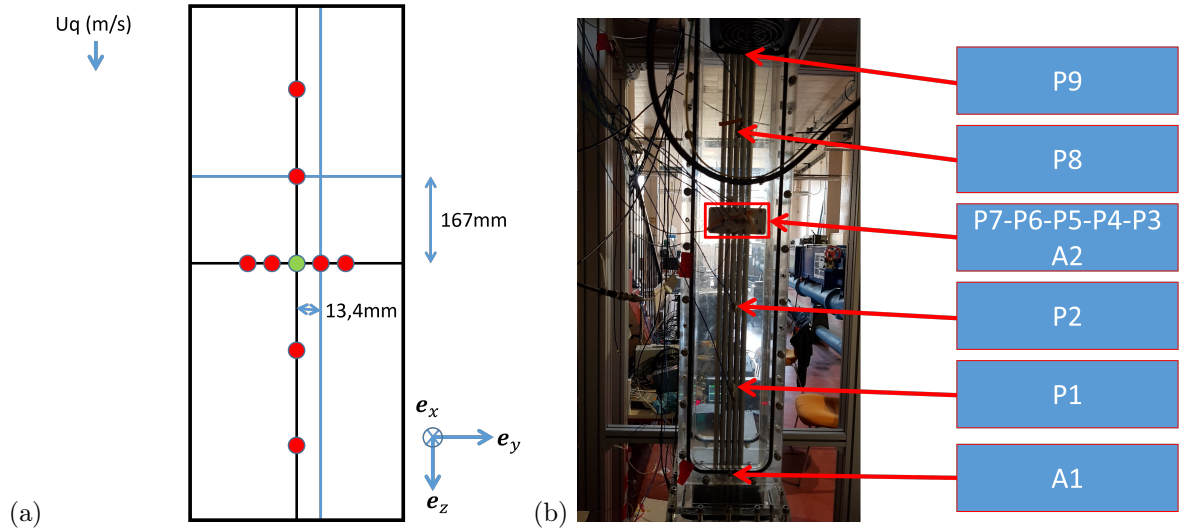


FIGURE 3.14 – Position des capteurs sur le hublot de la veine d'essai : (a) écartement entre les capteurs de pression, (b) position des capteurs de pression P1 à P9 et des accéléromètres A1 et A2. Le capteur de pression P5 est représenté en vert et coïncide avec le centre du hublot. Les autres capteurs de pression sont représentés en rouge.

essais réalisés permettent de mettre en évidence le phénomène d'amortissement fluide, les fluctuations de pression mesurées seront susceptibles d'apporter une information supplémentaire pour le décrire.

Accélérations

On utilise des accéléromètres monoaxiaux afin de surveiller le comportement vibratoire de la veine d'essai. Cette surveillance sert deux objectifs, le premier est de s'assurer que la rigidité de la veine est suffisante pour éviter l'apparition de modes de structure parasites. Le second est de s'assurer que les capteurs de pressions montés en paroi captent les différences de pression dues au mouvement de l'assemblage et non celles dues aux vibrations et à la déformation de la paroi. Ces accéléromètres sont collés à la cire (*cf.* FIGURE 3.15). La FIGURE 3.14 indique la position de chaque accéléromètre.

3.4.4 Mesure de vitesse débitante

La mesure de la vitesse d'entrée de l'écoulement axial dans la veine est un paramètre indispensable à la construction d'un coefficient d'amortissement. On utilise pour cette mesure un débitmètre électromagnétique watermaster de marque ABB. Il est placé entre la sortie de la veine d'essai et le réservoir inférieur.

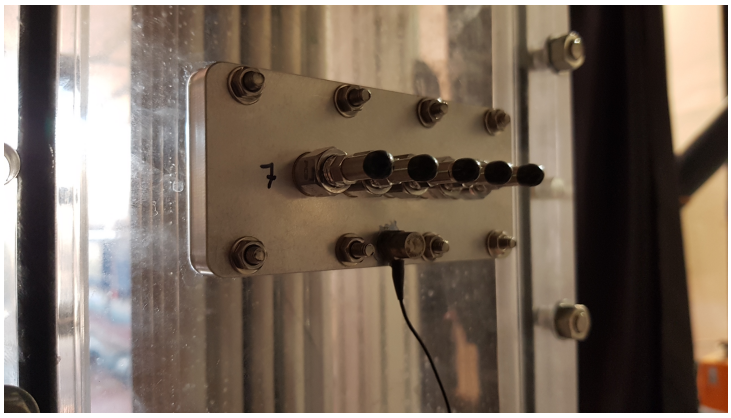


FIGURE 3.15 – Photo des capteurs placés à mi-hauteur de la veine d'essai. Les cinq capteurs alignés sont les capteurs de pression P3 à P7 (de droite à gauche). Le capteur isolé est l'accéléromètre A2.

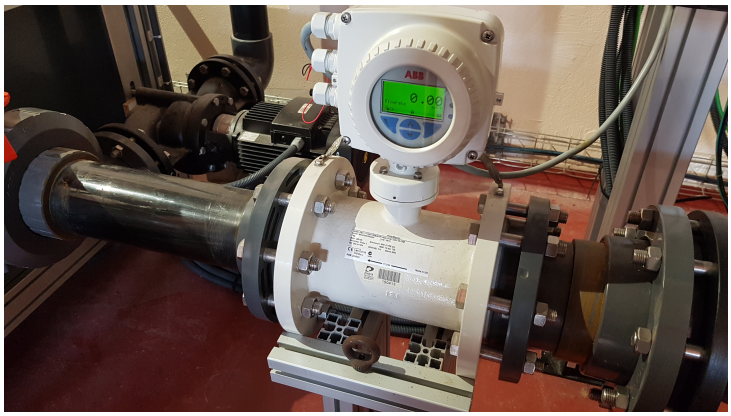


FIGURE 3.16 – Photo du débitmètre installé sur la ligne de retour au réservoir inférieur.

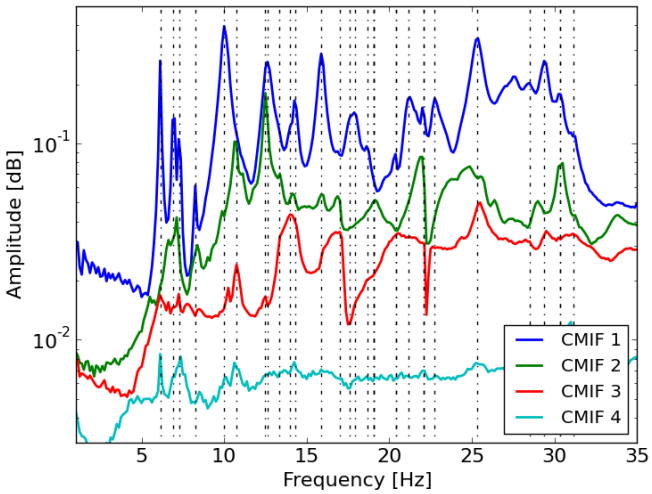


FIGURE 3.17 – Réponse en fréquence de différents composants de la structure, en absence d'écoulement : le bâti aluminium (en bleu), la veine d'essai (en vert), la passerelle (en rouge) et la tuyauterie (en cyan).

3.5 Essais de qualification

3.5.1 Objectifs des essais de qualification

Nous nous proposons dans cette partie de présenter les essais réalisés en vue de caractériser au mieux de nos capacités les déviations par rapport à la configuration idéale. Nous nous intéressons dans la suite à la présence éventuelle de structures d'écoulement préexistantes dans la veine d'essai. Ces structures pourraient biaiser les mesures de forces et de pressions. Nous nous intéressons aussi à la stabilité de l'écoulement dans le temps, afin de nous assurer que nos analyses se fondent sur un régime hydraulique établi. Une partie de ces essais est consacrée à l'étude des performances du système de forçage. L'objectif de cette étude est de caractériser le comportement du système de forçage sur sa gamme d'utilisation.

3.5.2 Analyse modale de la structure

Un essai sur maquette immobile en écoulement axial réalisé dans la veine d'origine, bien que non instrumenté, a révélé la présence d'oscillations d'ensemble des cylindres. On avance l'hypothèse que ces vibrations parasites sont dues à une déformation non contrôlée de la veine d'essai. Une analyse [3] modale du banc d'essai complet est proposée pour vérifier cette hypothèse. Cette analyse comporte deux parties : une analyse modale à l'arrêt et une analyse modale en fonctionnement.

Analyse modale à l'arrêt

La première partie est une analyse modale sur la structure complète avec pompe à l'arrêt pour identifier expérimentalement les modes de vibrations de la veine et de sa structure portante. Cette analyse est réalisée en eau stagnante afin d'avoir une distribution de masse proche des conditions d'essais. La source d'excitation de la structure est un marteau de choc pourvu d'une cellule de force. La réponse en accélération de la structure est mesurée en différents points.

La FIGURE 3.17 présente la réponse en fréquence de différents composants de la structure, en absence d'écoulement. Chaque pic d'amplitude peut correspondre à une fréquence propre de la structure associée à un mode propre de déformation d'un composant. Les pics apparaissant dans la bande $[0, 10\text{Hz}]$ correspondent aux fréquences 6.1 Hz, 6.9 Hz, 7.3 Hz, 8.2 Hz et 10.0 Hz. Pour que les déformées associées à ces pics puissent être considérés comme des modes propres, ils doivent être décorrélés entre eux.

La FIGURE 3.18 représente la matrice des coefficients de corrélation entre les déformées associées aux pics identifiés. Chaque coefficient de la matrice est compris entre 0 et 1. La valeur 0 indique une absence

3. Nous remercions M. Corus, expert à EDF, d'avoir dirigé cette analyse.

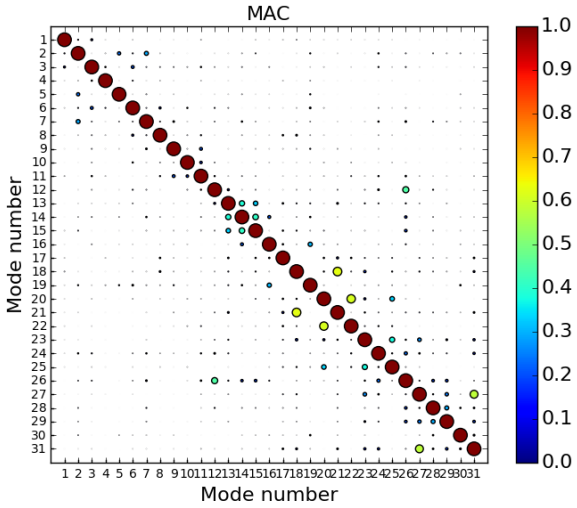


FIGURE 3.18 – Représentation graphique de la matrice des coefficients de corrélation entre les déformées associées aux pics identifiés.

de corrélation tandis que la valeur 1 indique une corrélation parfaite. On remarque que la cohérence entre deux déformées différentes est faible. On en déduit que les fréquences identifiées correspondent à des modes propres. La FIGURE 3.19 présente la déformée modale pour le premier mode. Ce mode est un mode de ballotement, la structure entière décrit un mouvement avant/arrière. Il s'agit d'un type de mode à même de forcer l'oscillation de toute maquette insérée dans la veine, indépendamment du régime d'écoulement. Il est à noter que l'amplitude de déformation de la tuyauterie est plus importante que l'amplitude de déformation des autres composants.

Analyse modale en fonctionnement

La deuxième partie est une analyse modale en fonctionnement destinée à déterminer les modes de la base modale précédemment identifiée constituant la réponse de la structure à la vibration des pompes et à l'écoulement. Cette analyse est réalisée à une vitesse débitante de 0.8 m/s. La FIGURE 3.20 présente la réponse en fréquence de différents composants de la structure. Les pics d'amplitude apparaissant dans la bande $[0, 10\text{Hz}]$ correspondent aux fréquences 6.0 Hz, 7.8 Hz, 8.6 Hz et 9.4 Hz. La FIGURE 3.21 présente la déformée modale pour le premier mode. C'est un mode de vibration pendulaire à même de forcer l'oscillation d'une maquette insérée dans la veine d'essai. Il est à noter qu'en fonctionnement, l'amplitude de déformation de la tuyauterie est plus importante que l'amplitude de déformation des autres composants.

Les résultats des analyses modales permettent d'expliquer les oscillations parasites observées. La vibration de la tuyauterie entraîne la vibration du réservoir supérieur, qui entraîne la vibration de la veine d'essai et de la maquette.

Les pompes permettant la circulation de l'eau sont fixées au bâti aluminium supportant le réservoir inférieur. Elles n'ont aucune liaison avec le sol. Elles se comportent donc comme des balourds et amplifient les vibrations de la tuyauterie en fonctionnement.

Modifications consécutives à l'analyse modale

Les pompes du banc d'essai agissent comme des excitateurs sur une structure laissée libre à de nombreux endroits. Contraindre complètement la structure peut mener à d'autres complications et demanderait des travaux conséquents. On choisit ici un compromis entre la contrainte complète et l'état initial. Nous avons agi directement sur l'excitateur en supportant les pompes par le sol afin de bloquer leur mouvement de balancier. La solution technique pour ce blocage consiste à visser une platine à la pompe et mettre en compression l'ensemble par appui sur le sol à l'aide de pieds réglables (*cf.* FIGURE 3.22).

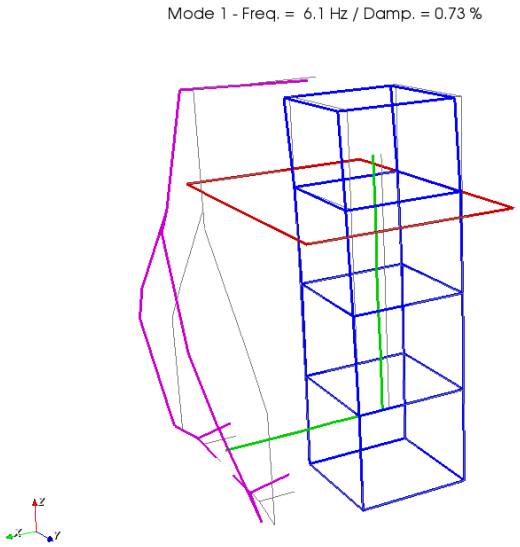


FIGURE 3.19 – Déformée modale pour le 1er mode, à l'arrêt (6.1 Hz). Le bâti aluminium est représenté en bleu, la veine d'essai et la ligne de retour sont représentés en vert, la tuyauterie et les pompes sont représenté en violet et la passerelle d'accès au réservoir haut est représentée en rouge. La configuration non déformée est représentée en filigrane noir

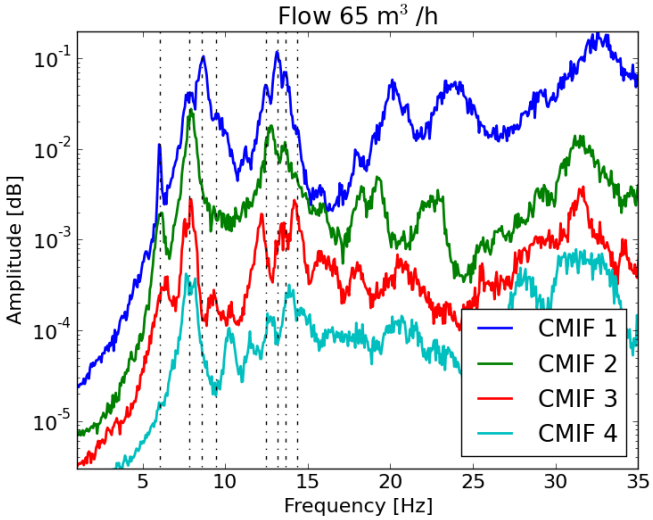


FIGURE 3.20 – Réponse en fréquence de différents composants de la structure, pour une vitesse d'écoulement $U_{\text{deb}} = 0.8\text{m/s}$: le bâti aluminium (en bleu), la veine d'essai (en vert), la passerelle (en rouge) et la tuyauterie (en cyan).

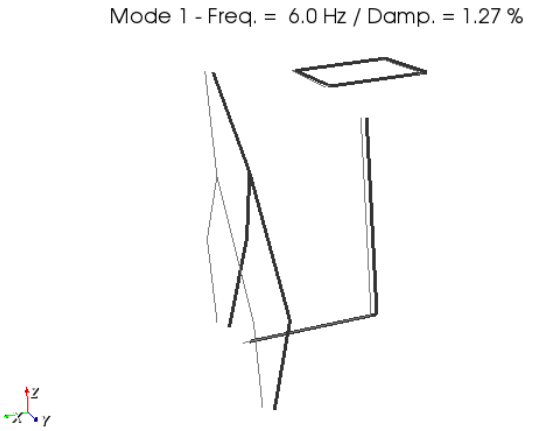


FIGURE 3.21 – Déformée modale pour le 1er mode, en fonctionnement (6.0 Hz). La configuration déformée apparaît en trait noir épais et la configuration initiale en trait noir fin.

FIGURE 3.22 – Une pompe et son support. Le support permet de bloquer le mouvement de balourd de la pompe.

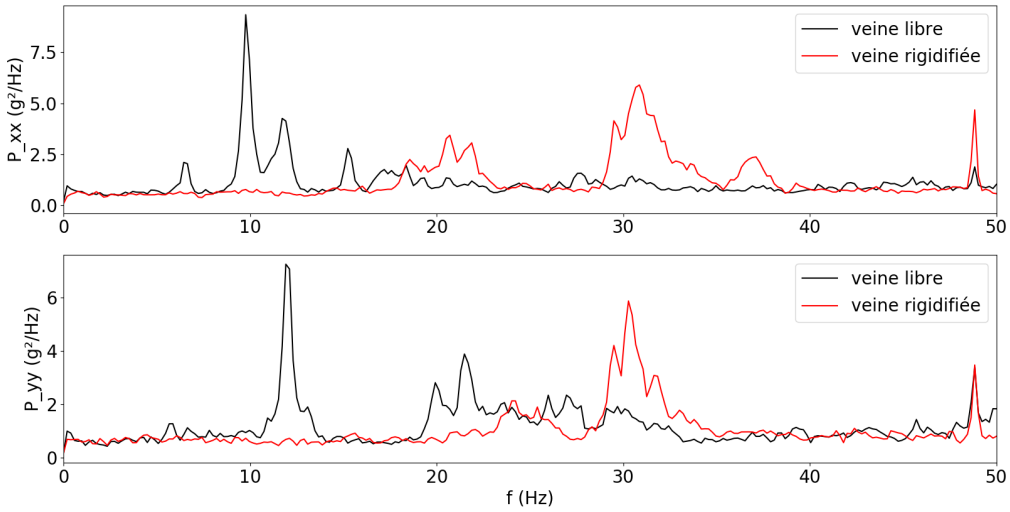


FIGURE 3.23 – Densité spectrale de puissance de l'accélération mesurée en fonctionnement sur deux parois perpendiculaires de la veine. Chaque graphe correspond à une paroi.

On a laissé libre le reste de la structure à l'exception de la veine que l'on a rigidifié. On présente en FIGURE 3.23 un exemple de mesure d'accélération en fonctionnement dans un plan orthogonal à l'écoulement à mi-hauteur de la veine d'essai. Deux mesures distinctes sont représentées, correspondant chacune à une configuration différente. La configuration appelée "veine libre" correspond à la veine avant modification. La configuration appelée "veine rigidifiée" correspond à une veine mise en appui sur les murs adjacents à l'aide de barres rigides. On constate que, sans support, la bande d'intérêt pour les essais dynamiques [0-10Hz] est polluée par les modes de structures identifiés par l'analyse modale. La courbe représentative de la densité spectrale de puissance de la veine rigidifiée ne présente pas de pics dans la bande [0-10Hz]. Nous en déduisons qu'aucun mode propre de la structure rigidifiée n'est associée à une fréquence de sollicitation qui sera utilisée dans les essais. Nous en concluons que les résultats de nos essais ne seront pas pollués par l'entrée en résonance d'un composant de la structure.

3.5.3 Qualité de l'écoulement

Pour quantifier l'homogénéité et la stationnarité de l'écoulement, nous avons eu recours à la PIV (*Particle Image Velocimetry*). Nous avons réalisé cette analyse pour différents débits. Pour chaque débit Q étudié, la Table 3.4 indique la vitesse débitante $U_q = Q/l^2$ et la valeur du nombre de Reynolds [4] Re correspondante. La fréquence d'acquisition est 10 Hz, la durée d'acquisition 60 s et le pas de temps entre deux images constituants un doublet est 1 ms. Chaque série est donc constituée de 600 champs de vitesse $\mathbf{u}(\mathbf{r}, t)$ (1 200 images de particules), où $\mathbf{r} = x\mathbf{e}_x + y\mathbf{e}_y + z\mathbf{e}_z$ est le vecteur position. Trois plans parallèles sont explorés, respectivement d'équation $y/l = -1/4$, $y/l = 0$, et $y/l = +1/4$. Dans chacun de ses plans, le champ de vitesse se décompose comme suit : $\mathbf{u}(\mathbf{r}, t) = u_x(\mathbf{r}, t)\mathbf{e}_x + u_z(\mathbf{r}, t)\mathbf{e}_z$.

Deux moyennes du champ de vitesse $\mathbf{u}$ sont introduites dans la suite. Une moyenne temporelle, réalisée en chacun des points du champ d'observation :

$$\langle \mathbf{u} \rangle_t (\mathbf{r}) = \frac{1}{T} \int_0^T \mathbf{u}(\mathbf{r}, t)\, \mathrm{d}t,$$

où T est la durée d'une acquisition. La grandeur résultante $\langle \mathbf{u} \rangle_t (\mathbf{r})$ est un *champ* de vitesse moyen (vis-à-vis du temps). Une moyenne spatiale, réalisée sur un domaine $\mathcal{D}$ inclus dans un plan parallèle au plan $x\mathrm{O}z$:

$$\langle \mathbf{u} \rangle_{\mathcal{D}} (t) = \frac{1}{\mathcal{A}} \iint_{\mathcal{D}} \mathbf{u}(\mathbf{r}, t)\, \mathrm{d}x\mathrm{d}z,$$

4. Le nombre de Reynolds est donné par : $Re = U_q l/\nu$, où $\nu = 10^{-6}$ m^2/s est la viscosité cinématique de l'eau.

Q (m^3/h)	U_q (m/s)	Re (-)
28.6	0.35	52 500
40.5	0.50	75 000
51.6	0.65	96 000
60.0	0.74	110 000

TABLE 3.4 – Caractéristiques des écoulements étudiés : débits volumiques, vitesses débitantes et valeurs du nombre de Reynolds.

où $\mathcal{A} = \iint_{\mathcal{D}} \mathrm{d}x\mathrm{d}z$ est l'aire du domaine. La grandeur $\langle \mathbf{u} \rangle_{\mathcal{D}}(t)$ résultante est la série temporelle du *vecteur* vitesse instantané moyen (sur le domaine).

Homogénéité de l'écoulement

On se donne pour la suite de l'analyse des données deux variables représentatives de la qualité de l'écoulement. Nous prenons pour référence le vecteur résultant de la moyenne spatiale du champ moyen $\overline{\mathbf{u}} = \langle \langle \mathbf{u} \rangle_t \rangle_{\mathcal{D}} = \overline{u}_x \mathbf{e}_x + \overline{u}_z \mathbf{e}_z$ dans le plan $y = 0$, à l'exception de quatre bandes excluant les artefacts liés aux parois de la veine (gauche et droite) ainsi qu'aux défauts d'illumination laser (en haut et en bas de l'image). Le premier champ est la déviation, exprimée en degrés d'angle, de chaque vecteur du champ moyen par rapport au vecteur de référence :

$$\alpha(\mathbf{r}) = \arccos\left(\frac{\langle \mathbf{u} \rangle_t (\mathbf{r}) \cdot \overline{\mathbf{u}}}{\| \langle \mathbf{u} \rangle_t (\mathbf{r}) \| \| \overline{\mathbf{u}} \|} \right).$$

Le second champ est l'écart relatif de la composante de vitesse verticale, en tout point du champ moyen, avec la composante verticale du vecteur de référence :

$$u'_z(\mathbf{r}) = \frac{\langle u_z \rangle_t (\mathbf{r}) - \overline{u}_z}{\overline{u}_z}.$$

Pour un écoulement parfaitement vertical, d'une part, la moyenne spatiale du champ moyen $\overline{\mathbf{u}}$ est portée par l'axe Oz et, d'autre part, les champs α et u'_z sont nuls. Chacun de ces deux champs est calculé pour 4 débits Q croissants, dans chacun des trois plans considérés, respectivement en $y/l = -1/4$, $y/l = 0$, $y/l = +1/4$. La figure FIGURE 3.24 illustre les champs obtenus pour le plus grand débit ($Q = 60$ m^3/h). Loin des parois, les écarts relatifs de la composante de vitesse verticale sont au plus de 5% (*cf.* FIGURE 3.24 (a) (b) et (c)). Nous en concluons qu'aucune structure spécifique n'apparaît dans l'écoulement.

Les deux bandes verticales jaunes à gauche et à droite de chaque graphe, indiquant un écart relatif plus élevé, sont liées pour partie à l'effet des parois. Les taches jaunes en bas à gauche de chaque plan sont liées à des défauts d'illumination laser et à des réflexions parasites de la veine.

Stationnarité de l'écoulement

On présente dans cette partie l'évolution de la vitesse moyenne de l'écoulement $\langle u_z \rangle_{\mathcal{D}}(t)$ en fonction du temps pour chaque débit testé. On définit le taux de stationnarité comme le rapport entre l'écart type et la moyenne de $\langle u_z \rangle_{\mathcal{D}}(t)$. Pour un écoulement stationnaire, le taux de stationnarité est nul. À bas débit, $Q = 28.6\,\mathrm{m}^3/\mathrm{h}$, le taux de stationnarité de l'écoulement est inférieur à 3%. Pour les débits plus importants considérés dans cette étude, le taux de stationnarité est inférieur à 5%. Nous en concluons que l'écoulement est stationnaire. Les résultats pour les quatre débits considérés sont regroupés dans la Table 3.5.

3.5.4 Pilotage hydraulique

Le débit dans la veine d'essai est piloté par l'ouverture ou la fermeture de deux vannes : la vanne manuelle située à la sortie de la veine d'essai et la vanne manuelle du débit de fuite. Pour un essai donné, seuls quelques couples de positions de ces deux vannes permettent d'assurer un régime de fonctionnement stable, c'est-à-dire un régime pour lequel le niveau d'eau dans le réservoir supérieur reste constant. Ces

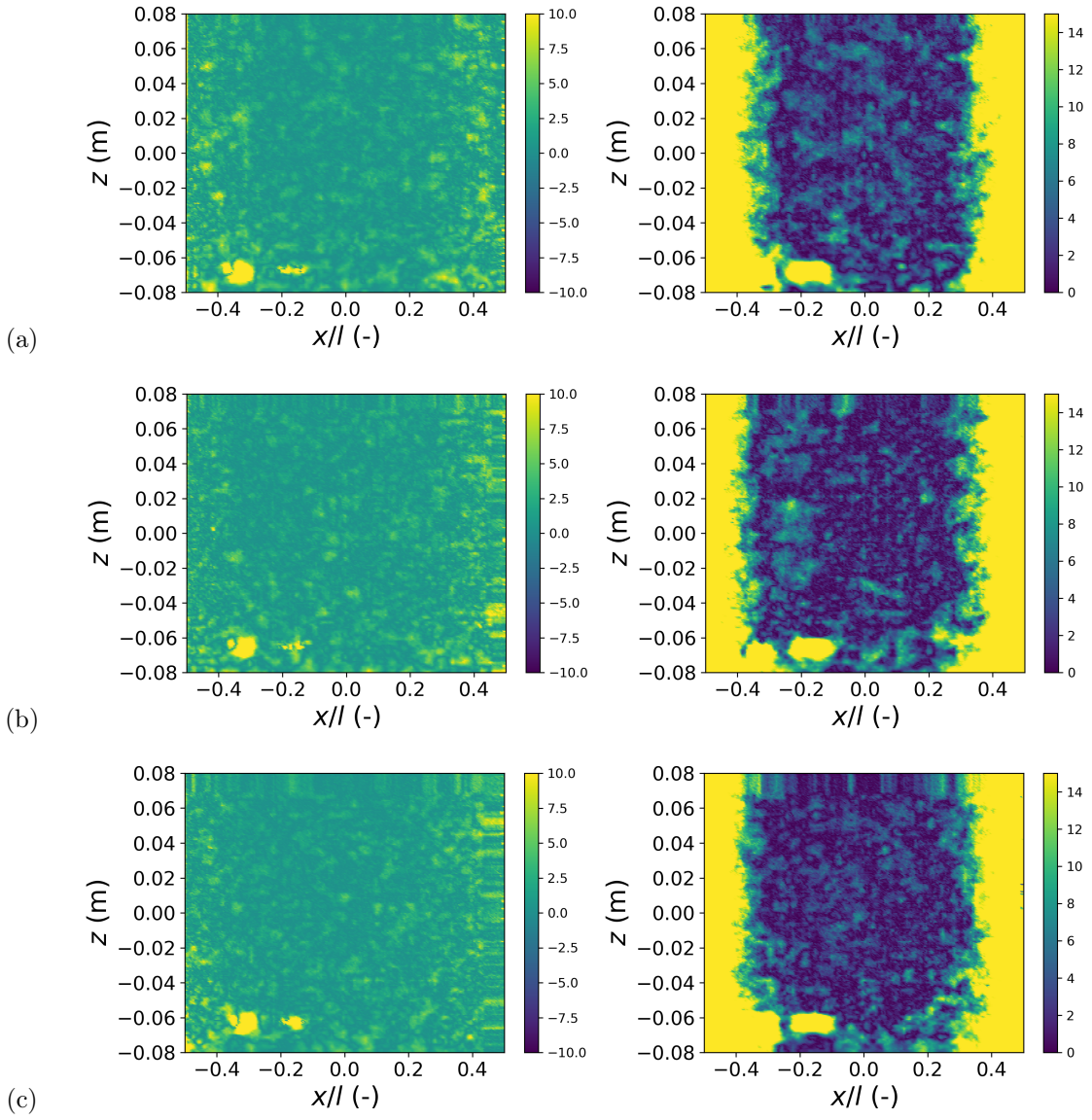


FIGURE 3.24 – Déviation α (à gauche) et écart relatif $u'_z(\mathbf{r})$ (à droite), pour un débit de $Q = 60$ m^3/h, dans le plan $y/l = -0.25$ (a), $y/l = 0$ (b), $y/l = +0.25$ (c).

Q (m^3/h)	$y/l = -0.25$	$y/l = 0$	$y/l = +0.25$
28.6	2.4	2.0	2.1
40.5	5.0	4.3	5.3
51.6	4.2	3.4	3.8
60.0	4.1	4.8	4.9

TABLE 3.5 – Taux de stationnarité de l'écoulement (exprimé en %) pour chaque débit étudié et pour chaque plan considéré.

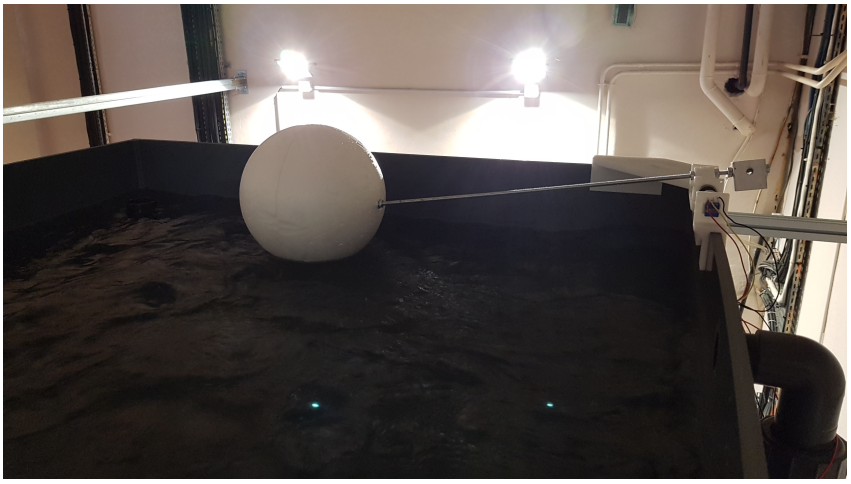


FIGURE 3.25 – Photo du flotteur servant à mesurer le niveau d'eau dans le réservoir supérieur.

couples sont inconnus *a priori* et ils sont déterminés empiriquement. La mesure du débit à la sortie de la veine d'essai permet de déterminer quand le régime stable a été atteint. Cette mise au point doit être faite à nouveau à chaque fois que les conditions d'écoulement sont modifiées. Durant cette mise au point, nous pouvons rencontrer deux types de régimes instables. Le premier régime correspond à une vidange du réservoir supérieur et *in fine* de la veine. Le second régime correspond à un remplissage du réservoir supérieur qui peut mener à un débordement. Un tel débordement pose des problèmes de sécurité majeur, notamment en cas d'aspersion des servomoteurs qui sont des matériels électriques de forte puissance.

Afin de limiter au maximum le risque de débordement, nous avons développé un système de régulation automatique du niveau d'eau fondé sur l'utilisation d'un PLC (*Programmable Logic Controller*). La mesure du niveau d'eau dans le réservoir supérieur est réalisée par l'intermédiaire d'un flotteur (illustré FIGURE 3.25) sur pivot accouplé à un potentiomètre linéaire. La tension mesurée aux bornes de ce potentiomètre correspond à la hauteur d'eau. Le PLC prend en entrée la tension mesurée et renvoie en sortie une autre tension qui sert à commander une servo-vanne pilotée (illustrée FIGURE 3.26). Cette servo-vanne remplace la vanne manuelle du circuit de fuite. Le PLC met en oeuvre un régulateur PID (Proportionnel Intégral Dérivé) qui compare la tension mesurée à une tension de référence correspondant à un niveau d'eau cible. Le PID ajuste la tension envoyée à la servo-vanne. Si la tension mesurée est plus faible que la tension de référence, alors le niveau d'eau est inférieur au niveau cible et le PID envoie une tension ordonnant à la servo-vanne de se fermer. Si la tension mesurée est plus forte que la tension de référence, alors le niveau d'eau est supérieur au niveau cible et le PID envoie une tension ordonnant à la servo-vanne de s'ouvrir.

3.5.5 Pilotage et qualité du forçage

Nous nous intéressons dans cette partie à la qualité du forçage, c'est-à-dire la capacité du système de forçage à imposer le déplacement souhaité à la maquette. Le système de forçage impose le déplacement de chacune des grilles situées aux extrémités de la maquette. Le déplacement imposé à chaque grille est réalisé par un mécanisme de came. Les deux mécanismes sont mécaniquement indépendants. Les moteurs entraînant les cames sont synchronisés électroniquement selon une configuration maître/esclave. Le moteur inférieur commande le moteur supérieur. Pour juger de la qualité du système de forçage, deux indicateurs sont considérés. Le premier est l'écart entre le déplacement imposé et le déplacement mesuré au niveau de chaque grille ; cet écart quantifie la capacité du système à imposer un déplacement harmonique. Le second est l'écart entre les deux déplacements mesurés ; cet écart quantifie la performance de la synchronisation électronique. Ces écarts sont calculés à partir de données d'essais réalisés en air et un déplacement d'amplitude $A = 2.5$ mm et pour les fréquences de sollicitation suivantes : 1 Hz, 2 Hz, 3 Hz, 4 Hz et 5 Hz.

FIGURE 3.26 – Photo de la vanne pilotée installée sur la ligne de fuite entre les réservoirs supérieur et inférieur.

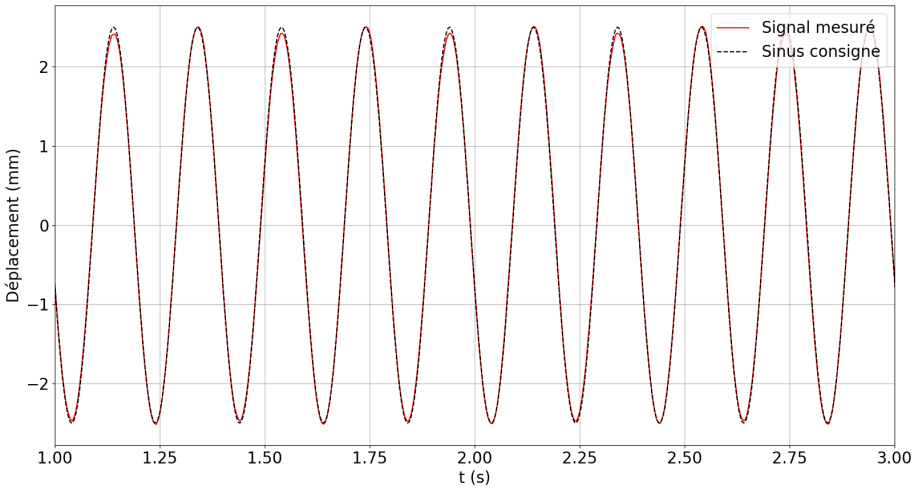


FIGURE 3.27 – Déplacement imposé par la came inférieure et déplacement de la grille inférieure mesuré.

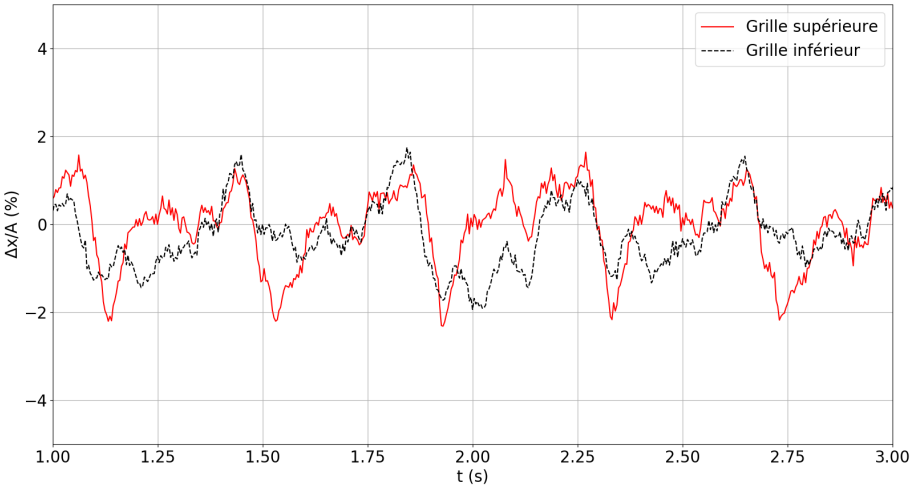


FIGURE 3.28 – Écart relatif entre le déplacement imposé et le déplacement mesuré, pour les grilles inférieure et supérieure.

Fréquence (Hz)	Déphasage (ms)	Décalage (mm)	Écart max (mm)
1	3.10	0.39	0.54
2	6.10	0.39	0.68
3	7.13	0.39	0.83
4	8.10	0.38	0.97
5	8.24	0.38	1.10

TABLE 3.6 – Quantification de la différence entre les mesures du déplacement de la grille inférieure et du déplacement de la grille supérieure.

Écart à la consigne de déplacement

La FIGURE 3.27 représente le déplacement imposé par la came inférieure et le déplacement de la grille inférieure mesuré, pour une fréquence de sollicitation de 5 Hz. Les courbes représentant le déplacement mesuré et le déplacement imposé sont quasiment superposées. Les *extrema* des deux courbes sont atteints aux mêmes instants. Nous en déduisons que le signal de déplacement mesuré est en phase avec le déplacement imposé. Le maximum de l'écart entre le déplacement mesuré et le déplacement imposé se localise aux *extrema*. On remarque des *maxima* du déplacement mesuré en retrait par rapport aux *maxima* du déplacement imposé toutes les deux périodes. Ce retrait peut s'expliquer par un désaxement de la came sur l'axe moteur, la came portant sur son profil deux périodes du déplacement imposé.

La figure 3.28 représente l'écart entre le déplacement imposé et le déplacement mesuré, rapporté à l'amplitude crête à crête du déplacement imposé, pour les grilles inférieure et supérieure. L'écart relatif ne dépasse pas 2.5% de l'amplitude. Le maximum de l'écart absolu est d'environ 0.12 mm.

Synchronisation entre les axes de forçage

La FIGURE 3.29 représente les déplacements mesurés, pour les grilles inférieure et supérieure, dans le cas d'une fréquence de sollicitation de 5 Hz. Les *extrema* des deux courbes ne sont pas atteints aux mêmes instants. Nous en déduisons que les deux signaux de déplacement sont déphasés.

Nous introduisons trois indicateurs pour quantifier la différence entre les mesures du déplacement de la grille inférieure et du déplacement de la grille supérieure. Le premier indicateur est la moyenne du déphasage entre les deux signaux. Le second indicateur est l'écart entre les moyennes des deux signaux. Le dernier indicateur est le maximum de l'écart entre les deux signaux. La table 3.6 rassemble les valeurs

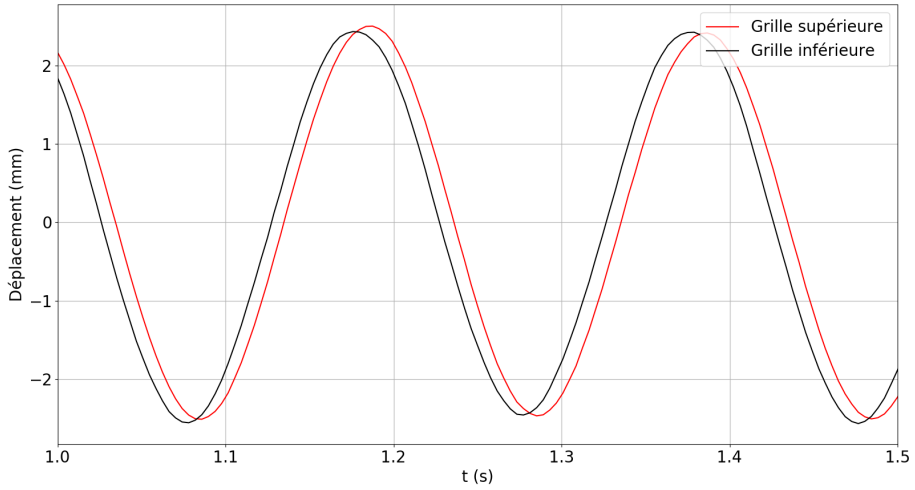


FIGURE 3.29 – Déplacements de chaque grille mesurés pour une fréquence de sollicitation de 5 Hz.

de ces indicateurs pour chaque fréquence de sollicitation étudiée.

Nous déduisons de ces essais que le système de forçage permet de transmettre à la maquette d'assemblage une sollicitation sinusoïdale de fréquence et d'amplitude fixes.

3.6 Conclusion

Nous avons présenté dans ce chapitre l'objet d'étude et le banc d'essai utilisé. L'objet d'étude est constitué d'un faisceau de 25 cylindres maintenus ensemble par deux grilles. Cette structure peut être décrite comme un objet indéformable. Le banc d'essai a été conçu à partir d'une boucle hydraulique préexistante. Le circuit hydraulique est composée d'une veine d'essai verticale reliée à deux réservoirs. Deux pompes assurent la circulation du fluide dans la boucle. Nous avons étudié les propriétés de l'écoulement dans la veine. Nous avons conclu qu'aucune structure fluide spécifique n'apparaît dans l'écoulement et que l'écoulement est stationnaire.

Nous avons proposé un système innovant de mise en mouvement de la structure. Ce système réalise une excitation harmonique de la structure. Il est composé de deux mécanismes à came synchronisés. Chaque mécanisme déplace une grille de l'objet d'étude. Nous avons étudié la capacité de ce système à reproduire un signal harmonique. Nous avons montré que ce système reproduit fidèlement un signal harmonique. Le déplacement mesuré de chaque grille est quasiment une sinusoïde. Les déplacements des deux grilles sont légèrement déphasés.

Nous avons présenté la métrologie. Un capteur laser mesure le déplacement de chaque grille. Une cellule de force mesure la résultante des efforts exercés sur chaque grille. Des capteurs de pression et des accéléromètres sont positionnés sur un des hublots de la veine d'essai.

Le chapitre suivant présente les résultats des essais réalisés. Nous présentons une analyse de la résultante des efforts exercés sur l'objet d'étude fondée sur le modèle masse-ressort-amortisseur présenté au chapitre 2. Cette analyse permet de déterminer la capacité du modèle de Divaret (2014) à décrire la dynamique de l'objet d'étude.

Chapitre 4

Résultats expérimentaux

4.1 Objectifs des essais

Nous présentons dans cette partie les essais réalisés dans le banc d'essai ELLEZ (*cf.* section 3.1). L'objectif principal de ces essais est, d'une part, de montrer l'existence d'un amortissement fluide dans le cas où la structure oscillante est la maquette d'assemblage proposée et, d'autre part, de le quantifier. Nous avons montré en section 3.5.5 que le système de forçage permet de transmettre à la maquette d'assemblage une sollicitation sinusoïdale de fréquence et d'amplitude fixes. Ces essais ont également pour objectif de déterminer si les efforts transmis à la maquette d'assemblage, en réaction à un déplacement imposé sinusoïdal de fréquence et d'amplitude fixes, sont sinusoïdaux de fréquence et d'amplitude fixes.

4.2 Traitement des mesures issues des essais

Afin d'obtenir une estimation de ces coefficients, plusieurs types d'essais sont réalisés pour une fréquence de sollicitation f_0 de 1, 2, 3, 4 et 5 Hz. L'amplitude du déplacement imposé est de 5 mm crête à crête, soit un demi-diamètre de tube.

Nous avons réalisé trois types d'essais : des essais en air, des essais en eau stagnante et des essais en écoulement. Le post-traitement de ces essais permet de déterminer les paramètres du modèle masse-ressort-amortisseur qui représente la dynamique de la maquette d'assemblage combustible. Les essais en air donnent accès aux efforts de structure uniquement. Le post-traitement de ces essais (les scripts utilisés sont présentés en Annexe B) permet de déterminer les paramètres M_s, C_s et K_s. Les essais en eau stagnante permettent une estimation de la masse ajoutée de l'assemblage. Le post-traitement de ces essais permet de déterminer $M_s + M_f$ et une valeur de $C_s + C_f$ valable en eau stagnante. Les essais en écoulement permettent de mettre en évidence l'amortissement sous écoulement axial. Le post-traitement de ces essais permet de déterminer une valeur de $C_s + C_f$ valable en écoulement. La comparaison entre les valeurs de $C_s + C_f$ obtenues en eau stagnante et en écoulement permet de quantifier l'amortissement sous l'écoulement axial.

4.3 Mise en évidence d'un mécanisme de dissipation

Nous souhaitons montrer que la structure oscillante se comporte comme un système linéaire dans la gamme de sollicitation explorée. Nous imposons une sollicitation périodique et sinusoïdale à la structure. Si la réponse de la structure est périodique et de même période que la sollicitation, alors le comportement de la structure peut être considéré comme linéaire.

Nous proposons la méthode graphique suivante pour déterminer si la réponse de la structure est périodique et de même période que la sollicitation. Nous découpons l'axe du temps en intervalles d'une longueur égale à la période de la sollicitation. Le point de départ de cette suite d'intervalles est un passage par zéro de la sollicitation. Si nous superposons les courbes représentatives de l'évolution de la sollicitation sur chacun de ces intervalles sur un même graphique, nous observerons que toutes les courbes se superposent, car la sollicitation étant périodique, elle évolue de façon identique dans chaque

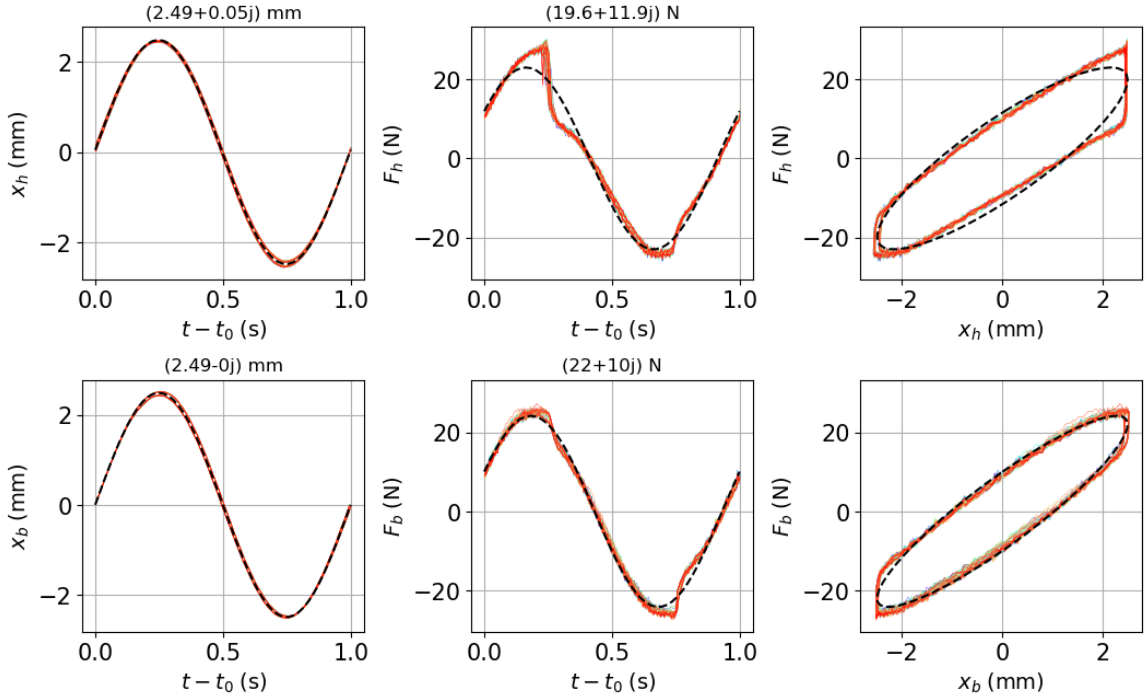


FIGURE 4.1 – Signaux obtenus en air à 1Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

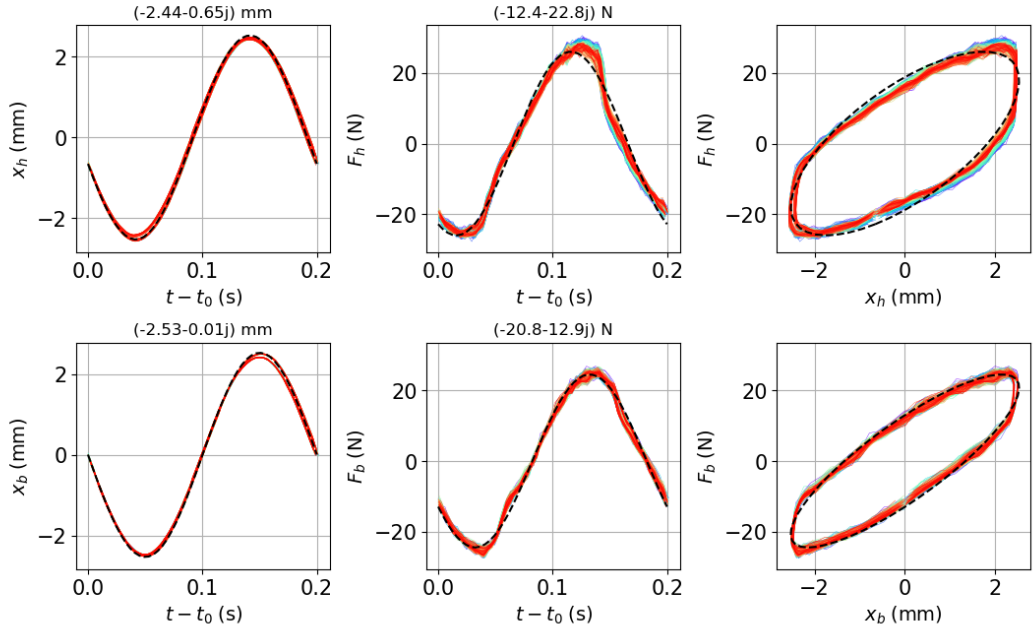


FIGURE 4.2 – Signaux obtenus en air à 5Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

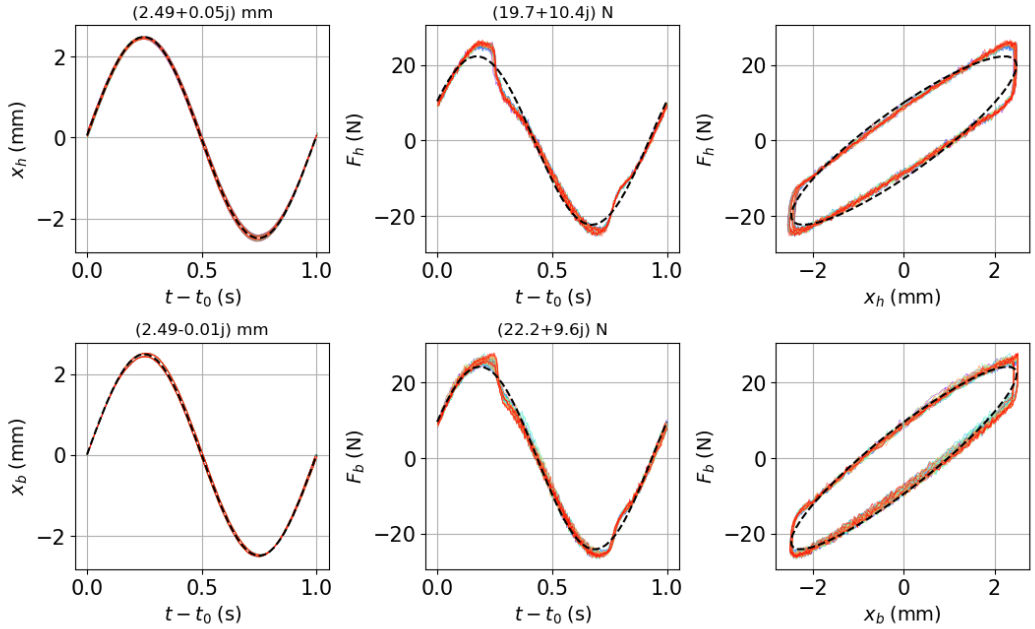


FIGURE 4.3 – Signaux obtenus en eau stagnante à 1Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

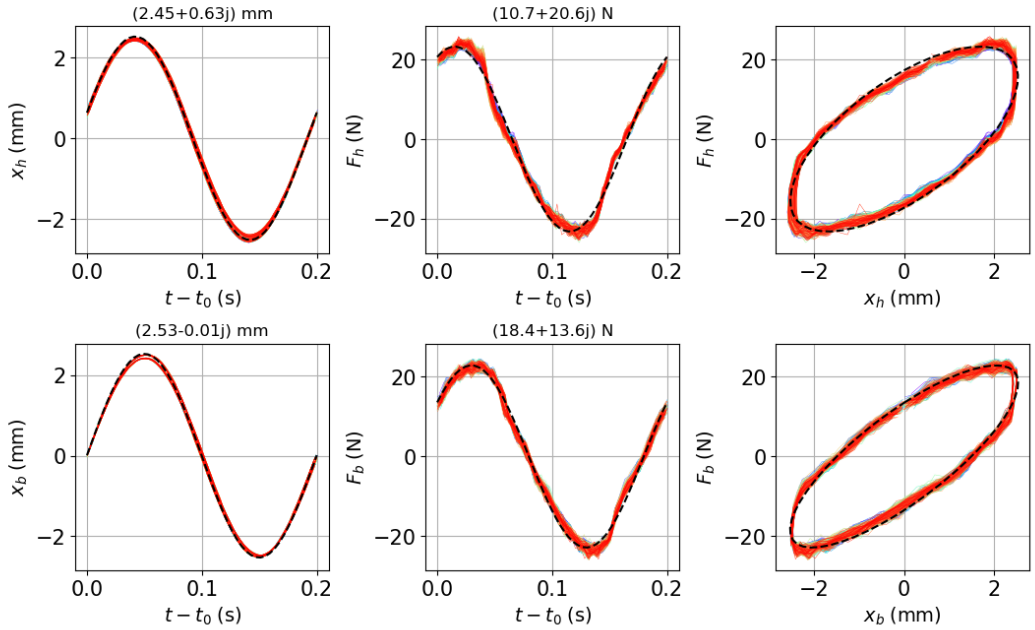


FIGURE 4.4 – Signaux obtenus en eau stagnante à 5Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

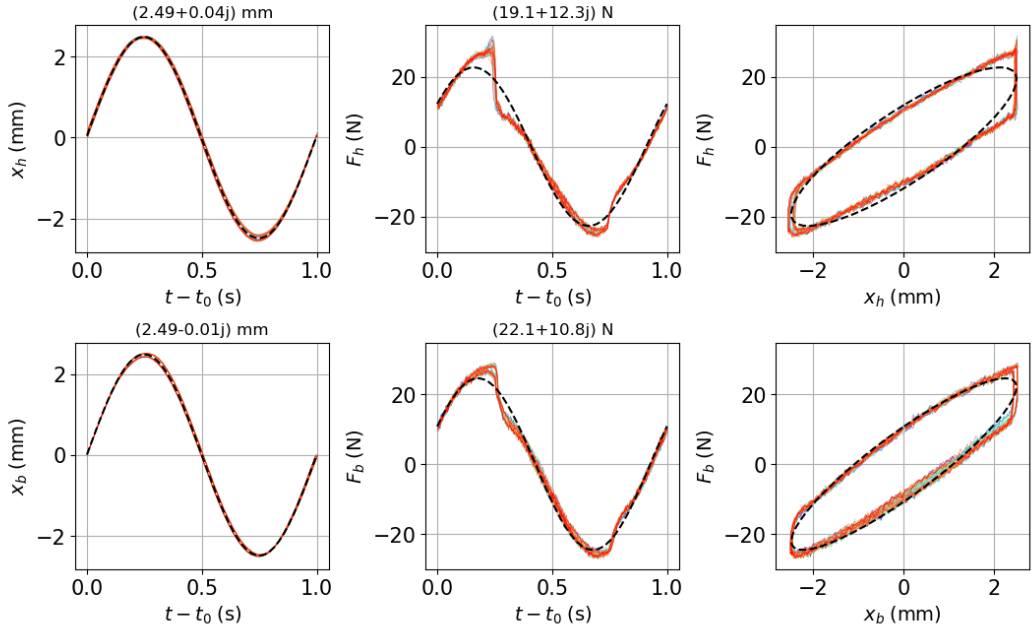


FIGURE 4.5 – Signaux obtenus en écoulement à $U = 0.9$ m/s à 1Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

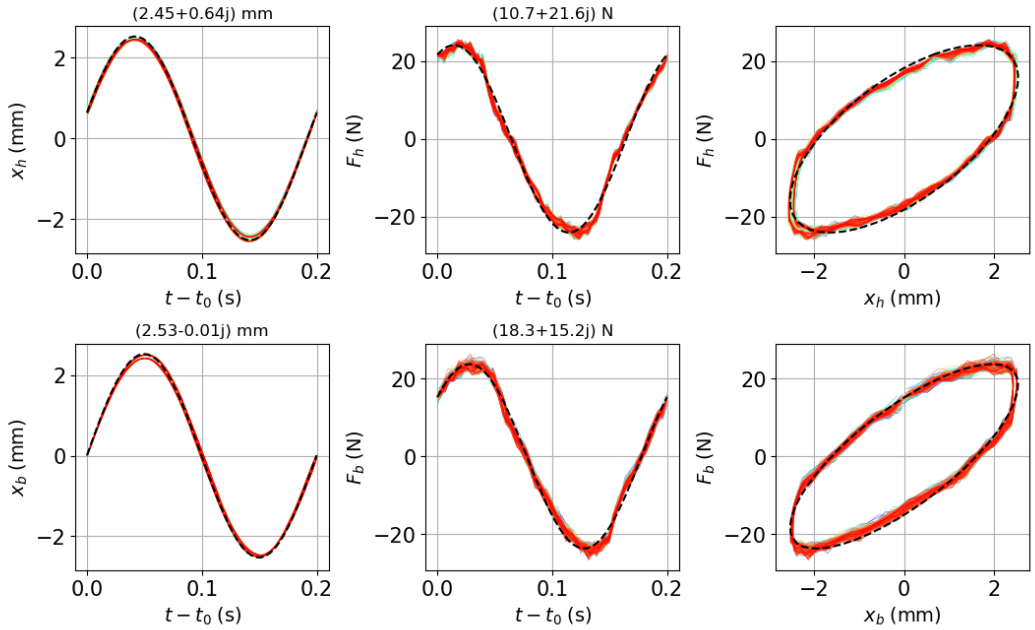


FIGURE 4.6 – Signaux obtenus en écoulement à $U = 0.9$ m/s à 5Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

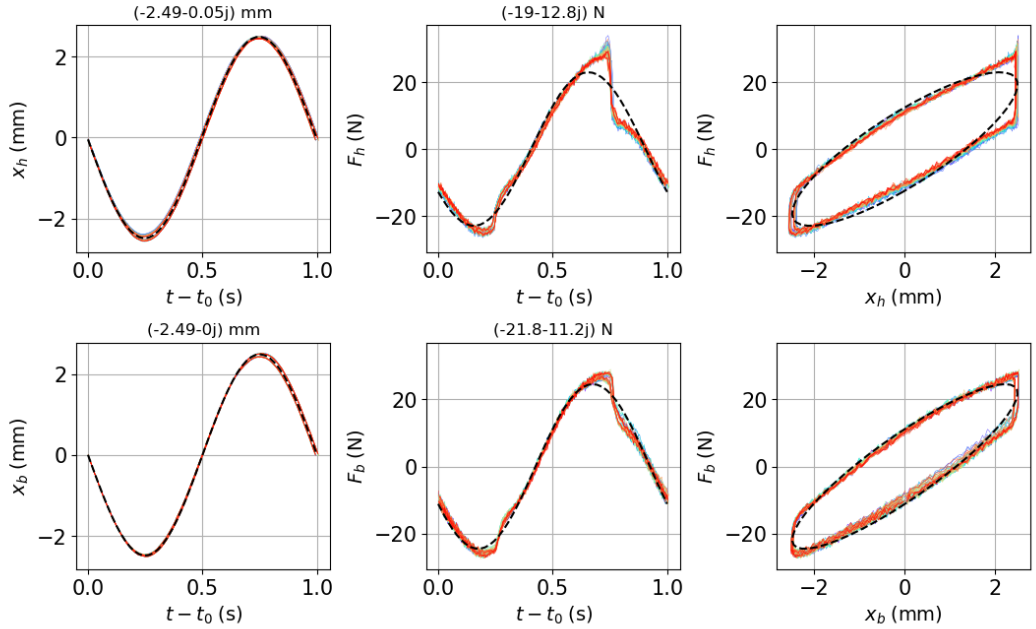


FIGURE 4.7 – Signaux obtenus en écoulement à $U = 1.0$ m/s à 1Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

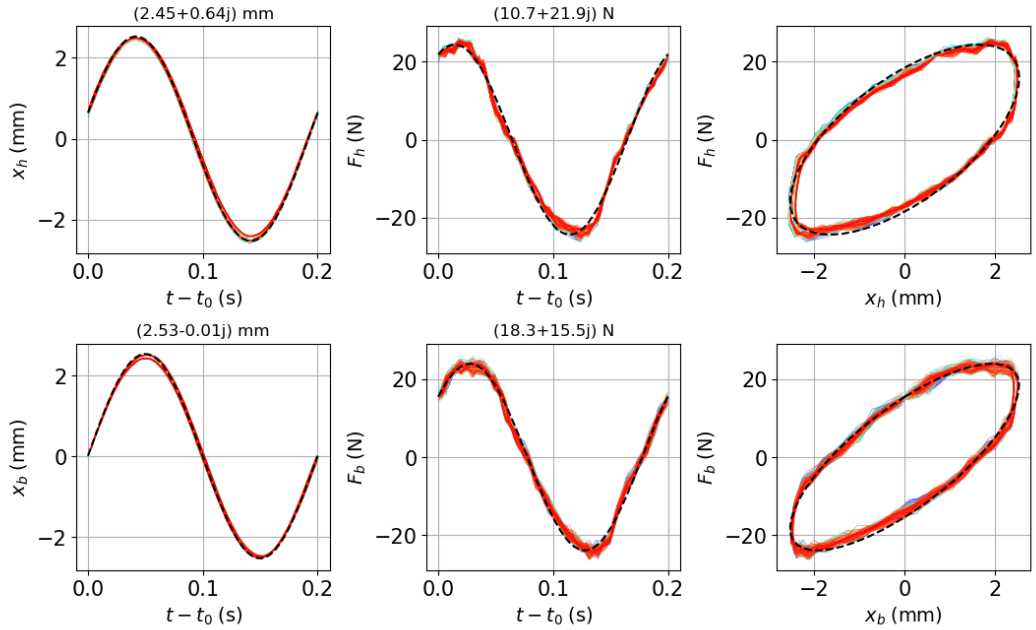


FIGURE 4.8 – Signaux obtenus en écoulement à $U = 1.0$ m/s à 5Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

intervalle. Nous proposons donc de superposer les courbes représentatives de l'évolution de la réponse en force sur chacun de ces intervalles sur un même graphique. Si les courbes se confondent, nous montrerons que la réponse en force évolue de façon identique dans chaque intervalle. Nous pourrons alors en déduire que la réponse en force est périodique. La longueur de chaque intervalle étant égale à la période de la sollicitation, nous pourrons en déduire que la réponse en force est périodique et de même période que la sollicitation.

Deux signaux peuvent représenter la sollicitation : le déplacement de la grille du haut et le déplacement de la grille du bas. Compte tenu de l'architecture du système de forçage, le déplacement de la grille du haut est piloté par le déplacement de la grille du bas. Nous choisissons le déplacement de la grille du bas comme représentation de la sollicitation. Chaque période de la sollicitation est donc délimitée par deux passages par zéro du signal de déplacement de la grille du bas. Les passages par zéro du signal de déplacement sont identifiés en appliquant la méthode présentée section 2.2.3. L'instant du premier passage par zéro du signal de déplacement de la grille du bas est noté t_0.

Les figures 4.1 à 4.8 illustrent les résultats obtenus dans différentes conditions d'essai, pour des fréquences de sollicitation de 1 Hz et 5 Hz (les résultats obtenus pour les fréquences 2, 3 et 4 Hz sont présentés en Annexe A). Chaque figure est composée de 6 graphiques. Les 3 graphiques de la ligne du bas correspondent aux mesures acquises sur la grille inférieure de la maquette d'assemblage et les 3 graphiques de la ligne du haut correspondent aux mesures acquises sur la grille supérieure. Sur chaque ligne, le graphique de gauche représente le signal temporel de déplacement, le graphique du milieu représente le signal temporel de force et le graphique de droite représente des courbes de Lissajous. Sur chaque graphique représentant des courbes de Lissajous, les abscisses sont des déplacements et les ordonnées sont des forces. Les figures 4.1 et 4.2 illustrent les résultats obtenus en air pour une fréquence de sollicitation de respectivement 1 Hz et 5 Hz. Les figures 4.3 et 4.4 illustrent les résultats obtenus en eau stagnante pour une fréquence de sollicitation de respectivement 1 Hz et 5 Hz. Les figures 4.5 et 4.6 illustrent les résultats obtenus en écoulement pour une vitesse $U = 0.9$ m/s et pour une fréquence de sollicitation de respectivement 1 Hz et 5 Hz. Les figures 4.7 et 4.8 illustrent les résultats obtenus en écoulement pour une vitesse $U = 1.0$ m/s et pour une fréquence de sollicitation de respectivement 1 Hz et 5 Hz. Pour chaque graphique représentant un signal temporel, le signal (de déplacement ou de force) est représenté sous la forme d'une superposition de périodes ; 60 périodes sont superposées pour les essais à 1 Hz et 300 pour les essais à 5 Hz.

Sur chaque graphique représentant un signal de force, pour une grille donnée et des conditions d'essais données, les courbes sont quasi confondues. Une légère dispersion se localise au niveau des extrema dans certains cas (*e.g.* essai réalisé en air à 5 Hz FIGURE 4.2). Nous attribuons cette dispersion au bruit de mesure.

La réponse en force obtenue lors des essais à 5 Hz semble quasi harmonique. Si la réponse en force est harmonique, alors elle se confond avec son fondamental (*cf.* section 2.2.4). Nous proposons de tracer, pour chaque signal temporel, la courbe représentative du fondamental du signal. L'amplitude A_1 et la phase φ_1 du fondamental sont approchées par les moyennes de l'amplitude et de la phase calculées pour chaque période du signal. Pour chaque graphique représentant un signal temporel, la courbe en pointillé représente la fonction u_1 définie par : $u_1(t) = A_1 \sin(2\pi f_0 t + \varphi_1)$. Le nombre complexe reporté au-dessus de chaque graphique représentant un signal temporel s'écrit sous la forme : $A_1 \cos(\varphi_1) + iA_1 \sin(\varphi_1)$. Nous en concluons que la réponse en force est périodique, de même période que la sollicitation, et donc que le système se comporte comme un système linéaire dans la gamme de sollicitation explorée.

Dans le cas des essais à 5 Hz, le fondamental de la réponse en force obtenue pour chaque essai se confond avec les différentes périodes du signal représentées. Dans le cas des essais à 1 Hz, le fondamental de la réponse en force obtenue pour chaque essai approche les différentes périodes du signal représentées ; l'écart entre le signal et son fondamental se concentre aux extrema.

Nous souhaitons à présent savoir si nos essais permettent de mettre en évidence un mécanisme de dissipation. Si le système est dissipatif, la réponse en force est déphasée par rapport à la sollicitation. La représentation de Lissajous permet de déterminer si deux signaux de même fréquence sont en phase ou non. Si la courbe prend la forme d'une droite, alors les deux signaux comparés sont en phase. Si la courbe prend la forme d'une ellipse, alors les deux signaux sont déphasés.

Nous proposons de représenter, pour chaque essai et pour chaque grille, le signal de déplacement et le signal de force de la façon suivante. À chaque instant où une mesure est effectuée, nous associons le point du plan dont l'abscisse est le déplacement mesuré et l'ordonnée est la force mesurée. Pour une période donnée, le lieu des points obtenu est une courbe fermée. Les courbes obtenues à partir de chaque période

se confondent. Ces courbes sont des courbes de Lissajous dans le cas où les signaux de déplacement et de force sont sinusoïdaux, c'est-à-dire dans le cas des essais réalisés à 5 Hz. Nous appliquons la même représentation au fondamental du signal de déplacement et au fondamental du signal de force. Le fondamental d'un signal étant une fonction sinusoïdale, la courbe obtenue représentée en pointillée est une courbe de Lissajous dans tous les cas. La courbe de Lissajous construite à partir du fondamental d'un signal de déplacement et d'un signal de force est un outil qui permet d'interpréter les essais pour lesquels les courbes construites à partir des mesures ne sont pas des courbes de Lissajous, en l'occurrence les essais réalisés à 1 Hz.

Dans tous les cas, les courbes de Lissajous sont des ellipses. Nous en déduisons que, dans tous les cas, il existe un déphasage entre la sollicitation en déplacement et la réponse en force. Nous en concluons que nos essais permettent de mettre en évidence un mécanisme de dissipation.

4.4 Mesure de force globale

Nous avons montré dans la section précédente que, dans la gamme de sollicitation explorée, le système se comporte comme un système linéaire dont l'entrée est une sollicitation en déplacement et la sortie est une réponse en force. Le déplacement de la structure est représenté par le déplacement de la grille du haut, noté X_h, et le déplacement de la grille du bas, noté X_b. Nous choisissons de décrire l'entrée du système par la moyenne de ces deux déplacements. La réponse en force de la structure est la somme de la réponse en force au déplacement de la grille du haut, notée F_h, et de la réponse en force au déplacement de la grille du bas, notée F_b. La fonction de transfert H, dont l'entrée est le déplacement imposé et la sortie la réponse en force de la structure, s'écrit :

$$H = 2\frac{\widehat{F}_h + \widehat{F}_b}{\widehat{X}_h + \widehat{X}_b}.$$

Nous souhaitons à présent savoir si le comportement du système peut être représenté par le modèle masse-ressort-amortisseur introduit en section 2.2.3. Si tel est le cas, la fonction de transfert H peut s'écrire sous la forme :

$$H(\omega) = K - M\omega^2 + iC\omega,$$

où ω est une pulsation, $K = K_s + K_f$, $M = M_s + M_f$ et $C = C_s + C_f$.

Nous proposons de vérifier cette hypothèse par la méthode graphique suivante. Pour chaque essai, nous évaluons la valeur de la fonction de transfert $H(\omega_0)$, où $\omega_0 = 2\pi f_0$ est la pulsation de la sollicitation. Chaque essai permet ainsi de construire un point de la courbe représentative de l'évolution de la partie réelle de H en fonction de ω^2 et un point de la courbe représentative de l'évolution de la partie imaginaire de H en fonction de ω. Si ces deux courbes sont des droites, alors l'hypothèse est vérifiée.

Pour calculer $H(\omega_0)$, il faut calculer les valeurs en ω_0 des transformées de Fourier des signaux suivant : F_h, F_b, X_h et X_b. Chacun de ces signaux est une fonction périodique. La valeur en ω_0 de la transformée de Fourier $\widehat{s}$ d'un signal périodique s, de pulsation ω_0 vérifie :

$$2i\widehat{s}(\omega_0) = A_1 \exp(i\varphi_1) = A_1 \cos(\varphi_1) + iA_1 \sin(\varphi_1).$$

Chaque nombre complexe représentant l'amplitude de Fourier du fondamental d'un signal mesuré, calculé dans la section précédente, est donc égal à la valeur de la transformée de Fourier du signal, évaluée en ω_0, multipliée par $2i$. En pratique, nous utilisons ces nombres complexes pour calculer $H(\omega_0)$ en appliquant la formule suivante :

$$H(\omega_0) = 2\frac{2i\widehat{F}_h(\omega_0) + 2i\widehat{F}_b(\omega_0)}{2i\widehat{X}_h(\omega_0) + 2i\widehat{X}_b(\omega_0)}.$$

4.4.1 Estimation du coefficient d'amortissement

La FIGURE 4.9 représente l'évolution de la partie imaginaire de la fonction de transfert H en fonction de la pulsation ω. Pour chaque type d'essai, les points semblent s'aligner sur une droite, à l'exception des points correspondants aux essais à 1 Hz. L'analyse des signaux temporels a montré que, contrairement aux autres signaux de réponse en force, les signaux de réponse en force obtenus lors des essais à 1 Hz

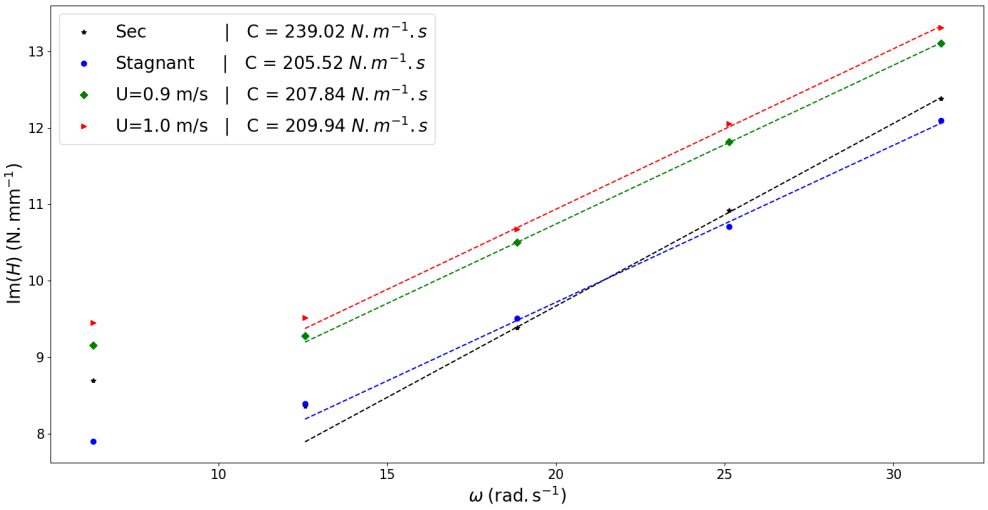


FIGURE 4.9 – Évolution de la partie imaginaire de la fonction de transfert en fonction de la pulsation.

ne sont pas complètement représentés par leur fondamental. Il apparaît donc raisonnable d'exclure les résultats des essais à 1 Hz de la suite de cette analyse.

Pour chaque type d'essai, la droite qui passe au plus près des points correspondant aux mesures réalisées de 2 Hz à 5 Hz est obtenue par régression linéaire. Chacune de ces droites est représentée en trait pointillé sur la FIGURE 4.9. Pour chaque type d'essai, la courbe obtenue approche les points correspondant aux mesures. Nous en déduisons que la partie imaginaire de la fonction de transfert H dépend linéairement de la pulsation. En conséquence, nous assimilons la pente de chaque droite à la valeur du paramètre C correspondant au type d'essais réalisés. Pour chaque type d'essai, la valeur du paramètre C est reportée dans la légende du graphique.

Pour une vitesse d'écoulement donné, la valeur de C_f est calculée en retirant la valeur de C obtenue dans le cas stagnant à la valeur de C obtenue dans le cas sous écoulement. $C_f = 2.32$ N.m^{-1}.s pour une vitesse incidente $U = 0.9$ m.s^{-1} et $C_f = 4.42$ N.m^{-1}.s pour une vitesse incidente $U = 1.0$ m.s^{-1}. Le coefficient C_f augmente avec la vitesse incidente U. Ce résultat est cohérent avec les résultats des études réalisées sur des assemblages combustibles réels (Stokes et King, 1979).

Nous souhaitons déterminer si la représentation de l'assemblage comme un faisceau de cylindres, c'est-à-dire sans prise en compte des grilles dans le modèle, proposée section 2.1.3 est cohérente avec nos résultats d'essai. Pour ce faire, nous calculons, pour chaque vitesse incidente considérée, la valeur du coefficient d'amortissement C_{amort}, en appliquant l'adimensionnement proposé en 2.2.3. Nous obtenons, $C_{\text{amort}} = 0.32$ pour une vitesse incidente $U = 0.9$ m.s^{-1} et $C_{\text{amort}} = 0.54$ pour une vitesse incidente $U = 1.0$ m.s^{-1}. Le coefficient C_{amort} a deux valeurs différentes pour deux vitesses incidentes différentes. Nous en concluons que l'adimensionnement proposé n'est pas adapté. En revanche, à ce stade, nous ne pouvons pas dire si le problème vient de l'adimensionnement lui-même ou du modèle sous-jacent.

Le modèle TLP représente uniquement les efforts exercés par le fluide sur les cylindres. Les forces mesurées ne distinguent pas les contributions des grilles et les contributions des cylindres. Il n'est pas possible de séparer ces contributions *a posteriori*. Au chapitre 5, la simulation numérique nous permettra d'isoler les efforts exercés par le fluide sur les cylindres. Nous serons alors en capacité d'évaluer la pertinence du modèle TLP.

4.4.2 Estimation du coefficient de masse ajoutée

La FIGURE 4.10 représente l'évolution de la partie réelle de la fonction de transfert H en fonction du carré de la pulsation ω. Pour chaque type d'essai, les points semblent s'aligner sur une droite, à l'exception des points correspondants aux essais à 1 Hz, qui ont été exclus de l'analyse.

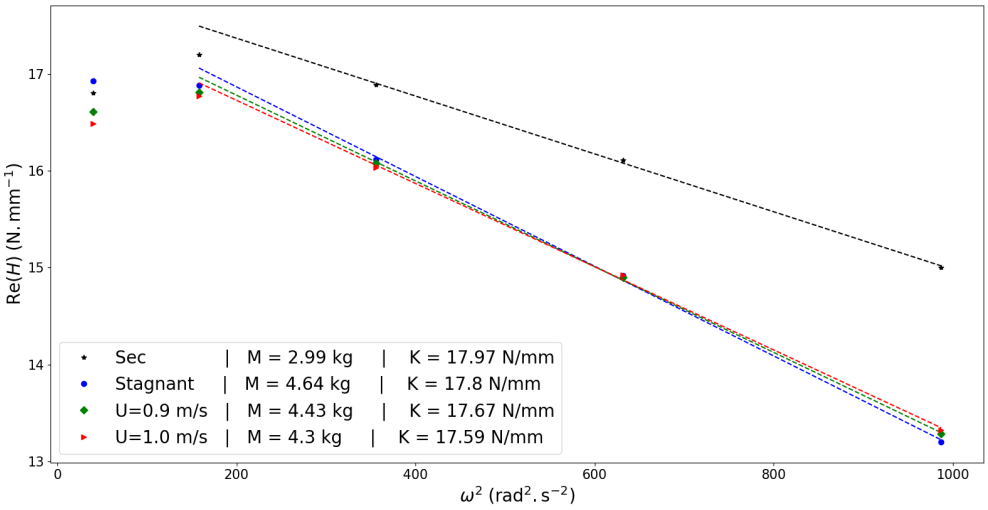


FIGURE 4.10 – Évolution de la partie réelle de la fonction de transfert en fonction du carré de la pulsation.

Pour chaque type d'essai, la droite qui passe au plus près des points correspondant aux mesures réalisées de 3 Hz à 5 Hz est obtenue par régression linéaire. Chacune de ces droites est représentée en trait pointillé sur la FIGURE 4.10. Pour chaque type d'essai, la courbe obtenue approche les points correspondant aux mesures. Nous en déduisons que la partie réelle de la fonction de transfert H dépend linéairement du carré de la pulsation. En conséquence, nous assimilons la pente de chaque droite à l'opposé du paramètre M et son ordonnée à l'origine au paramètre K, correspondant au type d'essai réalisé. Pour chaque type d'essai, les valeurs des paramètres M et K sont reportées dans la légende du graphique.

Les essais à sec se distinguent des essais en eau. La pente de la droite obtenue pour les essais à sec est inférieure aux pentes obtenues pour les essais en eau. Cette différence met en évidence un effet de masse ajoutée. Les pentes des droites obtenues pour les essais en eau sont quasi identiques, ce qui est cohérent avec l'idée que l'effet de masse ajoutée ne dépend pas des conditions d'écoulement. Compte tenu des dimensions de la veine, des dimensions de l'assemblage et de l'amplitude du déplacement imposé, nous ne nous attendons pas à observer un effet de raideur fluide lié à la proximité des parois. Les ordonnées à l'origine des droites correspondant à chaque type d'essai sont quasi identiques. Ce résultat est cohérent avec l'absence d'un effet de raideur fluide. Nous considérons donc que l'hypothèse $K_f = 0$ est vérifiée et en conformité avec les travaux précédents (Joly *et al.* (2021), Divaret (2014)).

La valeur du paramètre K obtenue à partir des essais à sec est $K_s = 18$ N/mm. La caractérisation des ressorts effectuée en statique a fourni une raideur de 18.2 ± 2 N/mm. Ces deux valeurs sont proches. La valeur du paramètre M correspondant aux essais à sec est $M_s = 3$ kg, ce qui est inférieur à la masse totale de la structure qui est de 5 kg. Une première hypothèse pour expliquer cet écart serait l'existence d'un défaut dans la chaîne d'acquisition. La chaîne d'acquisition utilisée pour mener les essais en dynamique a également été utilisée pour caractériser les ressorts en statique. Il nous semble peu probable qu'une chaîne d'acquisition défectueuse ait pu fournir des raideurs quasi identiques en statique et en dynamique. Nous en concluons qu'un défaut de la chaîne d'acquisition ne peut pas expliquer l'écart observé entre M_s et la masse de la structure. L'hypothèse retenue pour expliquer ces écarts est que nous essayons ici d'estimer, avec des essais de très basse fréquence, une grandeur variant avec le carré de la fréquence. On peut ainsi relever dans la FIGURE 4.10 que la partie réelle de la fonction de transfert varie de moins de 20% sur la plage de fréquences testée. Pour vérifier cette hypothèse, il faudrait tester une plus large gamme de fréquences pour capter une grandeur variant avec le carré de la fréquence. De plus, utiliser des fréquences éloignées les unes des autres rendrait l'estimation de la grandeur d'intérêt plus robuste au bruit de mesure.

L'écart important entre la masse M_s et la masse de la structure nous convainc de ne pas tenter

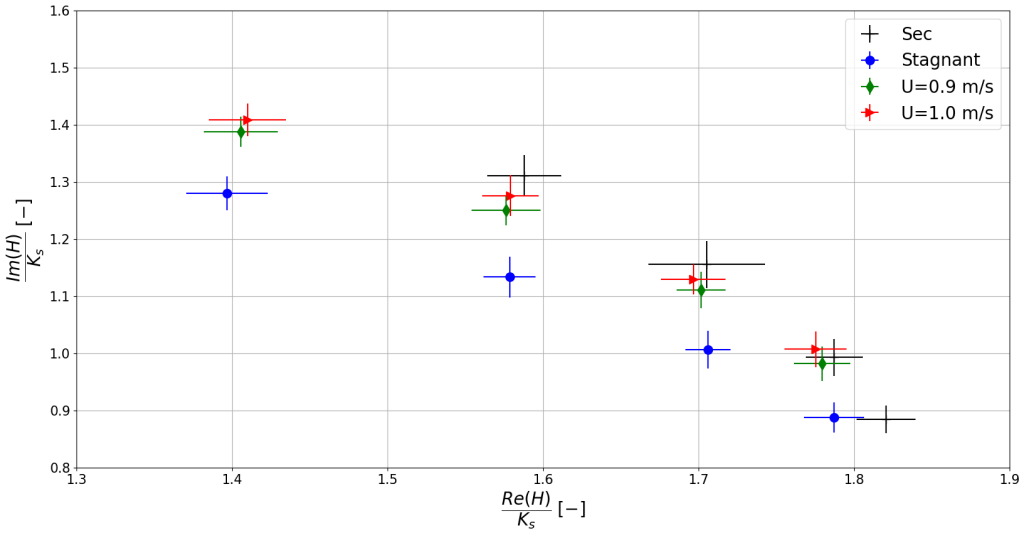


FIGURE 4.11 – Représentation graphique des valeurs de la fonction de transfert H obtenues à partir des essais réalisés à 2 Hz, 3 Hz, 4 Hz et 5 Hz.

d'estimer la masse ajoutée M_f. Estimer la masse ajoutée à partir des résultats de nos essais n'est pas nécessaire, car cette estimation peut être effectuée par d'autres moyens. Au chapitre 5, nous utiliserons des résultats de simulation numériques pour estimer la masse ajoutée.

4.4.3 Synthèse des mesures de force

La FIGURE 4.11 représente les valeurs de la fonction de transfert H obtenues à partir des essais réalisés à 2 Hz, 3 Hz, 4 Hz et 5 Hz. Pour chaque point représenté, l'abscisse correspond à la partie réelle de la fonction de transfert et l'ordonnée correspond à la partie imaginaire de la fonction de transfert (diagramme d'Argand). La partie réelle et la partie imaginaire de la fonction de transfert sont adimensionnées par le paramètre de raideur K_s. Les barres d'erreur représentent l'incertitude liée aux mesures. La partie réelle de la fonction de transfert H diminue quand la fréquence augmente, de sorte que, pour un type d'essai donné, parcourir les fréquences par ordre croissant de 2 Hz revient à parcourir les points de droite à gauche.

Pour expliquer d'où proviennent les barres d'erreur, il faut revenir à la manière dont les valeurs de la fonction de transfert ont été calculées. Chaque nombre complexe représenté a été calculé à partir des transformées de Fourier des signaux de déplacement et des signaux de forces correspondants à la grille du haut et à la grille du bas. Chaque signal étant périodique, la valeur de chacune de ces transformées de Fourier a été obtenue à partir de paramètres représentant le fondamental d'un de ces signaux. Ces paramètres ont été calculés comme la moyenne des paramètres obtenus à partir de chaque période d'un signal. Nous avons estimé l'écart type de chacun de ces paramètres. Nous avons alors pu calculer, pour chaque signal, deux nouvelles valeurs de chaque paramètre : la différence entre la moyenne et l'écart-type du paramètre, d'une part, la somme de la moyenne et de l'écart-type du paramètre, d'autre part. À partir de ces deux nouvelles valeurs de chaque paramètre, nous avons calculé, pour chaque essai, deux nouvelles valeurs de la fonction de transfert H. Chaque point correspond à la valeur de la fonction de transfert obtenue à partir des moyennes des paramètres. L'extrémité de chacune des barres d'erreur correspond à l'une des deux nouvelles valeurs de la fonction de transfert calculées à partir de la moyenne et de l'écart-type des paramètres. Cette représentation montre, sous une autre forme, la dispersion des efforts de masse et d'amortissement en fonction des conditions d'écoulement et de la fréquence d'excitation de la structure.

Pour une fréquence donnée, la partie réelle de la fonction de transfert H est plus petite pour un essai réalisé en eau que pour un essai réalisé à sec. Cela traduit l'effet de masse ajoutée, car cela signifie que le paramètre de masse du modèle masse-ressort-amortisseur est plus grand pour les essais en eau que

pour les essais à sec. Pour une fréquence donnée, les barres d'erreur sur la partie réelle de la fonction de transfert H, correspondant aux essais réalisés en eau, s'intersectent. Ce résultat tend à montrer que l'effort de masse ajoutée ne varie pas suivant les conditions d'écoulement.

Pour une fréquence donnée, la partie imaginaire de la fonction de transfert H est plus importante pour un essai en écoulement que pour l'essai en eau stagnante. Pour une fréquence donnée, la barre d'erreur sur la partie imaginaire de la fonction de transfert H, correspondant à l'essai en eau stagnante, n'intersecte pas les barres d'erreurs correspondant aux essais réalisés sous écoulement. Cela confirme que l'effet d'amortissement est plus élevé sous écoulement qu'en eau stagnante.

4.5 Mesures de pression

Une analyse fondée sur un modèle masse-ressort-amortisseur du comportement dynamique de la structure a permis de mettre en évidence l'effet d'amortissement induit par le fluide. Ce modèle décrit l'amortissement par un paramètre scalaire. L'analyse des forces mesurées lors de nos essais a permis de déterminer la valeur de ce paramètre pour un type d'essai donné (*i.e.* en eau stagnante ou sous écoulement) et pour des conditions d'essai données (*e.g.* une vitesse incidente donnée). L'effet d'amortissement observé résulte d'un transfert de quantité de mouvement entre le fluide et la structure. La variation de pression aux parois résulte également d'un transfert de quantité de mouvement entre le fluide et les parois. Nous formulons l'hypothèse suivante : le signal de fluctuation de pression à la paroi contient l'information de la quantité de mouvement transmise à la structure. Pour vérifier cette hypothèse, nous proposons donc de réaliser des mesures de pressions à la paroi de la veine. Si une réponse en pression à la sollicitation de la structure peut être définie, notre hypothèse sera vérifiée. Nous pourrons alors déterminer si cette réponse en pression contient une information sur l'amortissement.

Les capteurs de pression mesurent aussi bien les fluctuations de pression liées à la turbulence ainsi qu'à l'effet de l'oscillation de l'assemblage, que des fluctuations de pression dues à la vibration de la veine. Afin de nous assurer que les fluctuations de pressions mesurées ne sont pas toutes dues à la vibration de la veine, nous analysons conjointement un signal de pression mesuré par un capteur et un signal d'accélération mesuré par un accéléromètre placé au plus près du capteur de pression. En l'occurrence, l'accéléromètre A1 est au plus près du capteur de pression P1 et l'accéléromètre A2 est au plus près de P6.

La FIGURE 4.12 présente une partie du signal temporel de pression mesuré par le capteur P6 pendant un essai réalisé à 5 Hz. Les fluctuations de pression turbulente masquent la variation de pression synchrone avec l'oscillation de l'assemblage. La méthode d'analyse utilisée pour le traitement des mesures de force ne peut donc pas être employée pour analyser les mesures de pression. Nous allons donc utiliser une autre approche. Nous proposons d'étudier le contenu fréquentiel des signaux mesurés en calculant la densité spectrale de puissance (DSP) de chaque signal, c'est-à-dire la transformée de Fourier discrète de l'auto-corrélation de chaque signal. La densité spectrale de puissance est une fonction à valeurs réelles de la fréquence.

La FIGURE 4.13 présente la densité spectrale de puissance du signal de déplacement de la grille du bas, mesurée pendant un essai réalisé à 5 Hz. Le pic de plus forte amplitude (repéré par la ligne en pointillé rouge) correspond à la fréquence 5 Hz, c'est-à-dire la fréquence de sollicitation. Le second pic de plus forte amplitude correspond à la fréquence 2.5 Hz, c'est-à-dire la fréquence d'apparition des retraits des *maxima* du déplacement observés toutes les deux périodes (*cf.* 3.5.5). Nous remarquons que l'amplitude du pic à 2.5 Hz est trois ordres de grandeurs plus petite que l'amplitude du pic à 5 Hz. Cela signifie que l'énergie (au sens du traitement du signal) associée à la fréquence 2.5 Hz est négligeable devant l'énergie associée à la fréquence de sollicitation. Nous en déduisons que l'énergie associée à chaque autre pic est négligeable devant l'énergie associée à la fréquence de sollicitation. Nous en concluons que seule la contribution à 5 Hz constitue le signal utile tandis que toutes les autres contributions peuvent être assimilées au bruit de mesure.

La FIGURE 4.14 présente les densités spectrales de puissance des signaux d'accélération mesurés par les accéléromètres A1 et A2 et densités spectrales de puissance des signaux de pression mesurés par les capteurs P1 et P6 pendant un essai à sec réalisé à 5 Hz. Ces mesures représentent le bruit de fond de notre chaîne d'acquisition. Plusieurs fréquences correspondent à la fois à un pic d'accélération et à un pic de pression (*e.g.* à 22.5 Hz, 25 Hz, 27.5 Hz et 35 Hz), ce qui montre que les capteurs de pression sont suffisamment sensibles pour capter la vibration de la veine d'essai. Le pic de pression qui apparaît à 5 Hz

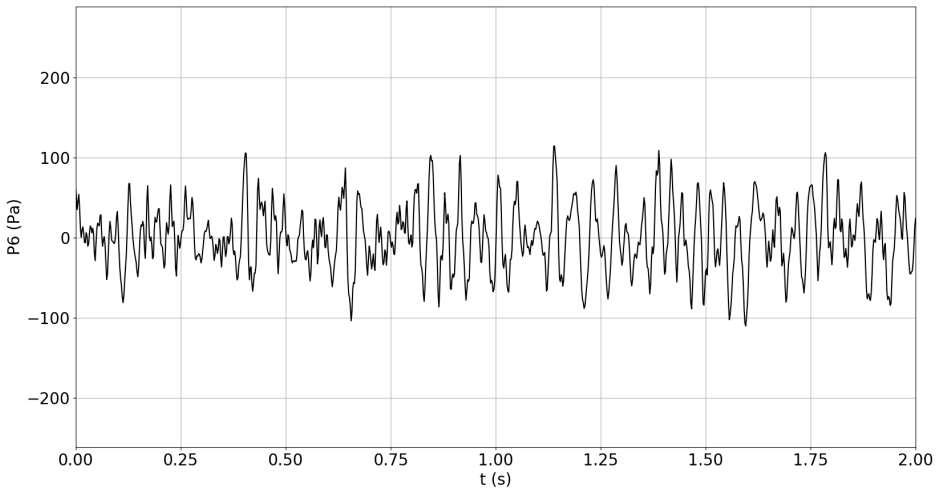


FIGURE 4.12 – Signal temporel de pression mesuré par le capteur P6 sur 10 périodes, pendant un essai réalisé à 5 Hz, pour une vitesse incidente U=0.9m/s.

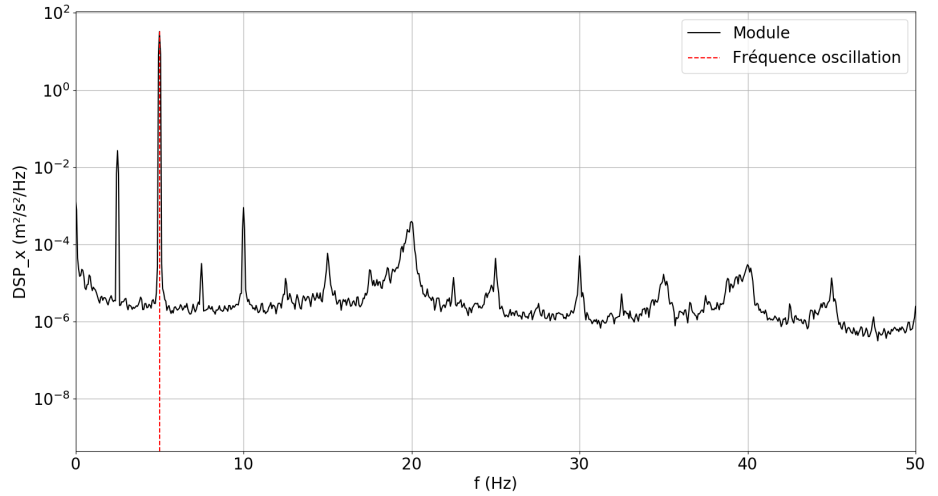


FIGURE 4.13 – Densité spectrale de puissance du déplacement de la grille du bas, mesuré pendant un essai réalisé à 5 Hz, pour une vitesse incidente U=0.9 m/s.

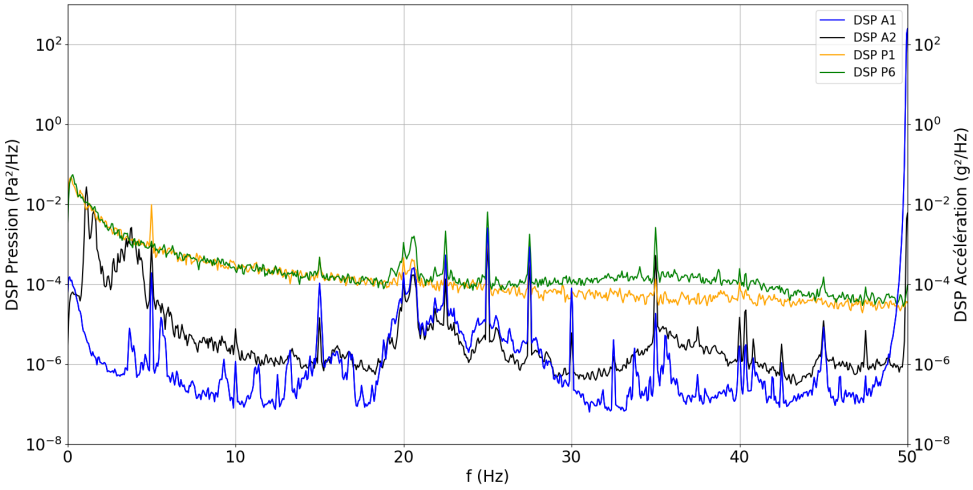


FIGURE 4.14 – Densités spectrales de puissance des signaux d'accélération mesurés par les accéléromètres A1 et A2 et densités spectrales de puissance des signaux de pression mesurés par les capteurs P1 et P6 pendant un essai à sec réalisé à 5 Hz.

pour le capteur P1 correspond avec un pic d'accélération pour l'accéléromètre A1 et pour l'accéléromètre A2. Nous en déduisons que ce pic correspond à une vibration de la paroi de la veine induite par le mouvement imprimé à l'assemblage. Nous en concluons que ce pic de pression correspond aussi au bruit de fond des essais, cette fois en liaison avec le mouvement de l'assemblage.

La FIGURE 4.15 présente les densités spectrales de puissance des signaux d'accélération mesurés par les accéléromètres A1 et A2 et densités spectrales de puissance des signaux de pression mesurés par les capteurs P1 et P6 pendant un essai en eau stagnante réalisé à 5 Hz. Sur la FIGURE 4.15, pour le signal de pression enregistré par le capteur P6, le pic de plus forte amplitude correspond à la fréquence 5 Hz alors que sur la FIGURE 4.14, pour le même signal de pression, aucun pic n'est visible à cette fréquence. Nous en déduisons que le pic à 5 Hz qui apparaît sur la FIGURE 4.15, pour le signal de pression enregistré par le capteur P6, décrit le comportement du fluide. Nous remarquons que le bruit large bande des signaux de pression mesurés par les capteurs P1 et P6 a gagné un ordre de grandeur entre 10 Hz et 50 Hz, par rapport au cas sec. Nous supposons que cette augmentation du bruit de fond s'explique de la façon suivante : l'eau stagnante transmet les vibrations des moteurs de leurs points de liaison avec la veine aux points de montage des capteurs de pression.

La FIGURE 4.16 présente les densités spectrales de puissance des signaux d'accélération mesurés par les accéléromètres A1 et A2 et densités spectrales de puissance des signaux de pression mesurés par les capteurs P1 et P6 pendant un essai sous écoulement réalisé à 5 Hz, pour une vitesse incidente à $U = 0.9$ m/s. La puissance du signal de pression mesuré par un capteur de pression lors d'un essai sous écoulement est plus élevée que la puissance du signal de pression mesuré par le même capteur lors d'un essai en eau stagnante. Cette augmentation de la puissance du signal s'explique par la turbulence de l'écoulement, au sens où des tourbillons apparaissent et disparaissent constamment, à toutes les échelles. Le pic à 5 Hz observé sur la FIGURE 4.15 est également présent sur la FIGURE 4.16, avec une amplitude similaire. Nous en déduisons que ce pic à 5 Hz décrit aussi le comportement du fluide sous écoulement, bien que l'effet de la turbulence le rende moins discernable.

L'analyse des densités spectrales de puissance a montré que pour des essais en eau stagnante et sous écoulement, un pic de pression significatif correspond à la fréquence de sollicitation (ici 5 Hz) tandis que, pour les essais à sec, aucun pic de pression significatif ne correspond à cette fréquence. Nous en concluons que la réponse en pression à la sollicitation en déplacement de la structure émerge du bruit de fond.

Afin de caractériser la réponse en pression à la sollicitation en déplacement de la structure, nous calculons la densité spectrale croisée (DSC) entre le signal de déplacement de la grille du bas et le signal de pression mesuré par le capteur. La densité spectrale croisée entre deux signaux est calculée comme la

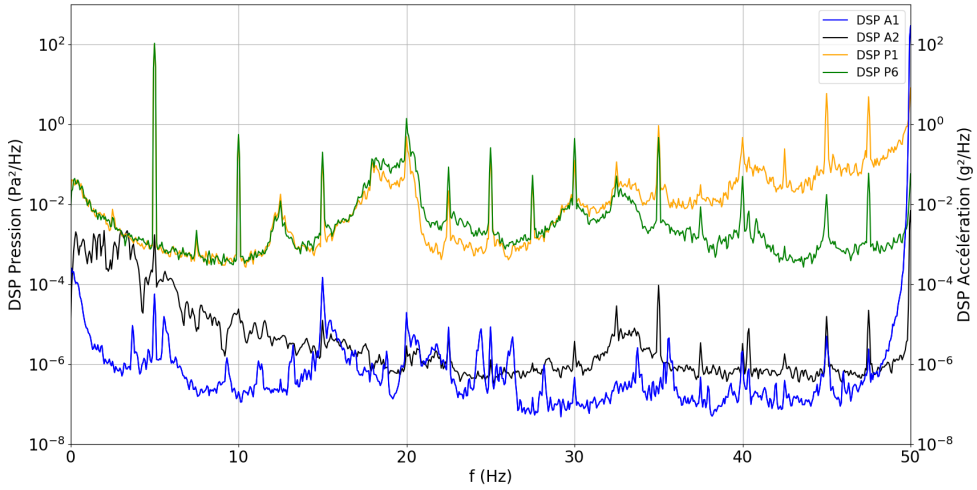


FIGURE 4.15 – Densités spectrales de puissance des signaux d'accélération mesurés par les accéléromètres A1 et A2 et densités spectrales de puissance des signaux de pression mesurés par les capteurs P1 et P6 pendant un essai en eau stagnante réalisé à 5 Hz.

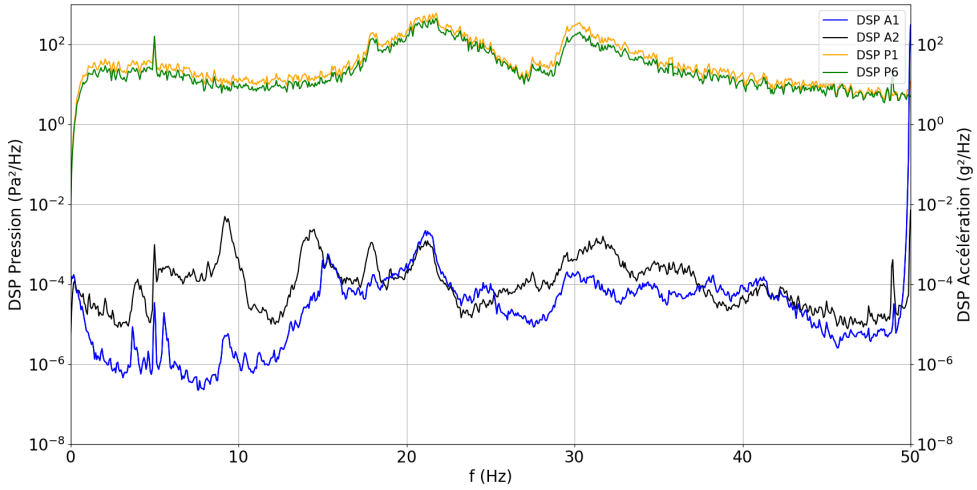


FIGURE 4.16 – Densités spectrales de puissance des signaux d'accélération mesurés par les accéléromètres A1 et A2 et densités spectrales de puissance des signaux de pression mesurés par les capteurs P1 et P6 pendant un essai sous écoulement réalisé à 5 Hz, pour une vitesse incidente de $U = 0.9$ m/s.

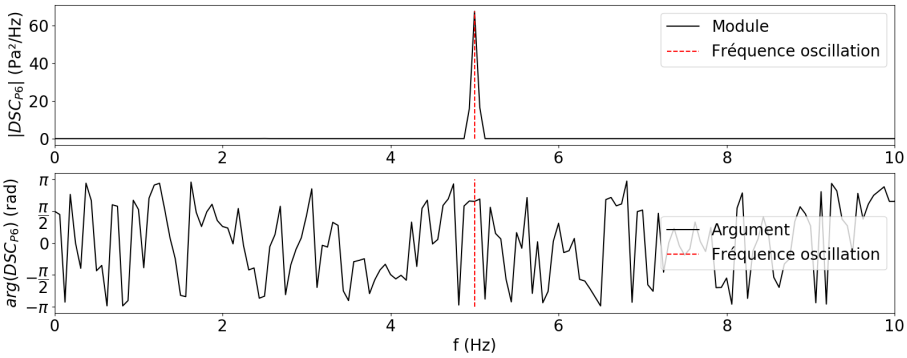


FIGURE 4.17 – Module (en haut) et argument (en bas) de la densité spectrale croisée entre le signal de pression mesurée par le capteur P6 et le signal de déplacement de la grille du bas obtenus, pendant un essai sous écoulement réalisé à 5 Hz, pour une vitesse incidente de $U = 0.9$ m/s.

Fréquence (Hz)	Amplitude (Pa)	Déphasage (rad)
1	1.99 ± 0.28	0.09 ± 0.08
2	1.94 ± 0.19	1.52 ± 0.21
3	2.14 ± 0.12	1.9 ± 0.15
4	2.32 ± 0.14	1.43 ± 0.03
5	3.61 ± 0.1	2.07 ± 0.03

TABLE 4.1 – Amplitude moyenne de la réponse en pression et déphasage moyen entre la réponse en pression et la sollicitation, obtenus pour chaque fréquence de sollicitation, à partir des essais réalisés en écoulement, pour une vitesse incidente de $U = 0.9$ m/s.

transformée de Fourier discrète de la corrélation croisée entre deux signaux ; c'est une fonction à valeurs complexes de la fréquence. Son module est non nul pour les fréquences où les signaux sont corrélés. Son argument est égal au déphasage entre les deux signaux. La FIGURE 4.18 présente le module et la phase de la densité spectrale croisée entre le signal de pression mesurée par le capteur P6 et le déplacement de la grille du bas obtenus, pendant un essai sous écoulement réalisé à 5 Hz, pour une vitesse incidente de $U = 0.9$ m/s. Le module de densité spectrale croisée est non nul pour une seule fréquence : la fréquence de sollicitation.

À partir des essais sous écoulement, réalisés pour une vitesse incidente de $U = 0.9$ m/s, nous avons calculé, pour chaque capteur de pression, le déphasage entre le signal de pression et le signal de déplacement, à partir de la densité spectrale croisée entre le signal de pression et le signal de déplacement. La FIGURE 4.18 présente, pour chaque capteur, l'évolution du déphasage obtenu en fonction de la fréquence de sollicitation. Nous avons également calculé, pour chaque fréquence de sollicitation, la moyenne et l'écart type du déphasage. Le déphasage moyen et l'écart-type du déphasage correspondant à chaque fréquence de sollicitation sont reportés dans la table 4.1.

À partir des mêmes essais, pour un capteur de pression donné et pour une fréquence de sollicitation donnée, nous calculons la densité spectrale de puissance du signal de pression mesuré. Nous isolons le pic de pression associé à la fréquence de sollicitation et nous calculons son intégrale. Cette intégrale correspond à la moitié du carré de l'amplitude de la réponse temporelle en pression à la sollicitation. Nous en déduisons l'amplitude de la réponse temporelle en pression à la sollicitation. Pour une fréquence de sollicitation donnée, nous calculons la moyenne et l'écart type des amplitudes de réponse temporelle calculées pour chaque capteur. L'amplitude moyenne et l'écart-type de l'amplitude correspondant à chaque fréquence de sollicitation sont reportés dans la table 4.1.

Pour chaque fréquence considérée, le déphasage entre la réponse en pression et la sollicitation en déplacement est non nul et varie peu d'un capteur à l'autre. Pour chaque fréquence considérée, le déphasage moyen entre la réponse en pression et la sollicitation en déplacement est significatif. Nous en déduisons que le signal de pression contient une information sur un phénomène de dissipation. Ces résultats d'essais

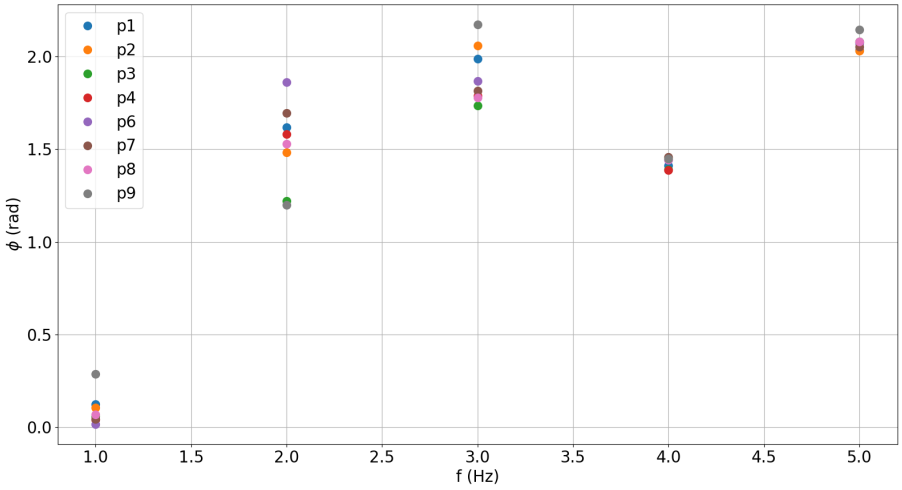


FIGURE 4.18 – Déphasage entre le déplacement imposé à la structure et la pression mesurée en paroi, en fonction de la fréquence, pour les capteurs de pression P1 à P9, obtenus à partir d'essais sous écoulement, réalisés pour une vitesse incidente de $U = 0.9$ m/s.

pourront servir de point de comparaison avec d'autres résultats d'essais et des résultats de simulations numériques.

4.6 Conclusion

Nous avons présenté les résultats des essais réalisés. Nous avons analysé la résultante des efforts exercés sur l'objet d'étude. Cette analyse a montré que le modèle masse-ressort-amortisseur présenté au chapitre 2 décrit fidèlement la dynamique de l'objet d'étude pendant chaque essai. Le calcul des paramètres de ce modèle pour chaque essai a permis de mettre en évidence un effet d'amortissement induit par l'écoulement.

Nous avons montré que le paramètre d'amortissement augmente quand la vitesse d'écoulement augmente. Le coefficient d'amortissement ne dépend pas linéairement de la vitesse d'écoulement. Ce résultat n'est pas en accord avec le modèle proposé par Divaret (2014). Nous émettons l'hypothèse que cet écart provient de la présence des grilles.

Nous avons montré que le paramètre de raideur fluide est nul dans nos essais. Les paramètres de masse obtenus à partir de nos essais nous semblent incorrects. En particulier, la valeur du paramètre obtenue à partir des essais sans eau sous-estime la masse de l'objet d'étude.

Nous avons analysé les signaux de pression mesurés. Nous avons montré que l'utilisation d'accéléromètres permet de distinguer les réponses en pression au mouvement de l'objet d'étude des réponses en pression à la vibration des parois de la veine d'essai. Les premiers permettent une comparaison avec les résultats d'autres essais ou des résultats de simulation numérique. Nous avons montré que le signal de pression n'est pas en phase avec le déplacement. En revanche, nous n'avons pas identifié le lien entre ce déphasage et l'amortissement.

Afin de compléter l'analyse des résultats d'essais, nous présentons dans le chapitre suivant un modèle numérique de la veine d'essai et les résultats des simulations réalisées. La simulation numérique nous permet de distinguer les contributions du faisceau de cylindre et des grilles à la résultante des efforts exercés par le fluide sur l'objet d'étude.

Chapitre 5

Modélisation numérique du banc d'essais

5.1 Objectifs de la modélisation numérique

Notre analyse des résultats d'essai présentée au chapitre 4 a permis de montrer que le modèle masse-ressort-amortisseur proposé représente la dynamique de l'assemblage soumis à un déplacement imposé le long d'une direction perpendiculaire à l'écoulement axial. Nous avons déterminé la valeur du paramètre de ce modèle qui quantifie l'amortissement induit par le fluide pour chacun des essais réalisés. En revanche, cette analyse n'a pas permis de déterminer si le modèle TLP représente les efforts exercés par le fluide sur les cylindres qui composent l'assemblage. En effet, les efforts mesurés ne distinguent pas la contribution des grilles de celle du faisceau de cylindres. Nous proposons dans ce chapitre de recourir à la simulation numérique pour accéder à la distribution des efforts exercés par le fluide à la paroi de l'assemblage. Nous pourrons alors extraire les efforts exercés par le fluide sur les cylindres et déterminer s'ils sont représentés par le modèle TLP.

5.2 Description de la modélisation

On présente dans un premier temps la construction de ce modèle ainsi que les hypothèses utilisées. On expose, dans un second temps, notre stratégie de validation fondée à la fois sur des données théoriques et expérimentales et, dans un dernier temps, notre analyse concernant l'hypothèse d'homogénéité (les programmes utilisés sont présentés en Annexe C).

5.2.1 Géométrie

Les simulations numériques ont pour but de représenter l'écoulement du fluide dans la veine d'essai. Le domaine occupé par le fluide est délimité tout d'abord par les parois verticales de la veine d'essai et la paroi de l'assemblage. Le domaine occupé par le fluide ne représente pas l'intégralité du circuit hydraulique, de sorte que la frontière du domaine occupé par le fluide est fermée par deux faces correspondant respectivement à l'entrée et à la sortie du domaine occupé par le fluide.

Nous choisissons de nous conformer aux standards de la simulation numérique en mécanique des fluides. Notre approche consiste donc à résoudre les équations de Navier-Stokes numériquement en utilisant la Méthode des Volumes Finis. Cette méthode s'appuie sur une discrétisation du domaine occupé par le fluide, c'est-à-dire une partition du domaine en cellules. Dans la plus pure tradition de la simulation numérique en mécanique des fluides, nous utilisons une discrétisation du domaine occupé par le fluide composée d'hexaèdres. Compte tenu que l'assemblage oscille, la position de l'assemblage évolue au cours du temps, de sorte que la géométrie du domaine occupé par le fluide évolue au cours du temps. La discrétisation du domaine occupé par le fluide doit suivre cette évolution. Nous choisissons d'utiliser une approche ALE (*Arbitrary Lagrangian-Eulerian*) afin que la position de chaque point à l'intérieur du domaine discrétisé soit considéré comme une inconnue du système d'équations à résoudre à chaque

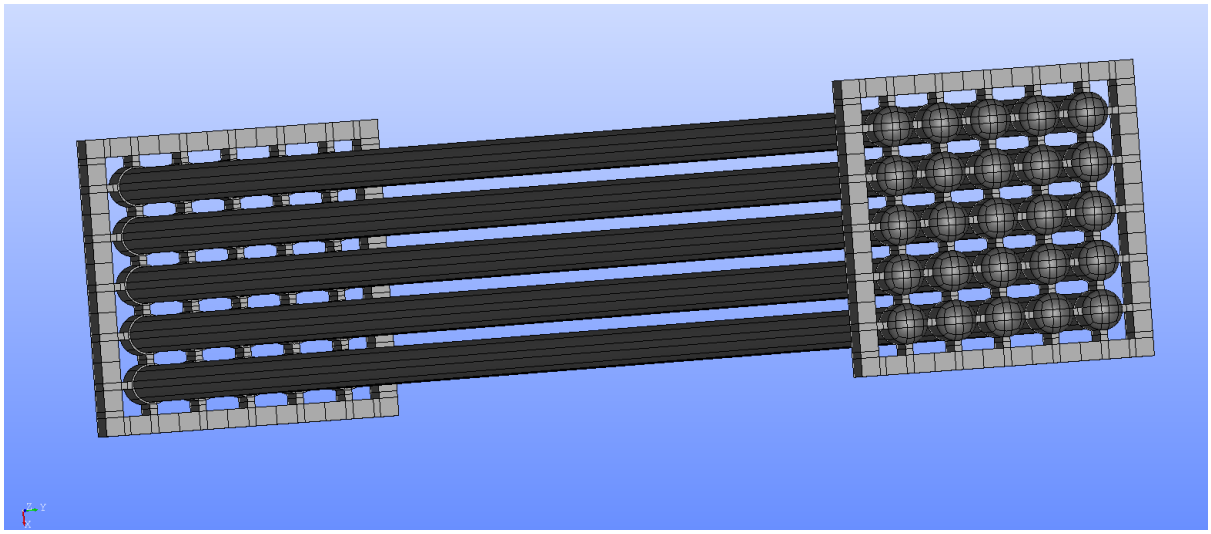


FIGURE 5.1 – Géométrie de l'assemblage utilisée dans les simulations numériques.

instant. De cette façon, nous avons juste besoin de décrire la géométrie du domaine occupé par le fluide lorsque l'assemblage est dans sa configuration initiale.

L'approche la plus simple pour construire la géométrie du domaine occupé par le fluide consiste à dessiner un parallélépipède et lui retirer le domaine occupé par l'assemblage, représenté FIGURE 3.5 (a). Le domaine occupé par l'assemblage présente des détails fins (*e.g.* les congés de raccordement). La prise en compte de ces détails géométriques nécessite un grand nombre de cellules de petite taille dans une zone où nous n'avons pas besoin que l'écoulement soit représenté avec précision.

Nous avons choisi une approche fondée sur la description d'une épure de l'assemblage. L'assemblage est décomposé en primitives géométriques simples : des hémisphères, des cylindres et des parallélépipèdes rectangles. Une partition du parallélépipède représentant le fluide contenu dans la veine est obtenue en considérant les boîtes englobantes de chaque élément constituant l'assemblage. Chacune de ces boîtes englobantes est fermée par six plans qui partitionnent l'espace. L'ensemble de tous ces plans définit la partition du parallélépipède représentant le fluide contenu dans la veine. Chaque élément de cette partition est un parallélépipède rectangle. Ces parallélépipèdes sont séparés en deux ensembles : les parallélépipèdes qui ont une intersection vide avec l'assemblage et les parallélépipèdes qui ont une intersection non vide avec l'assemblage. Chaque parallélépipède de la partition ayant une intersection non vide avec l'assemblage a une intersection non vide avec une primitive constituant l'assemblage. Si cette intersection est le parallélépipède de la partition lui-même (*e.g.* si la primitive est un parallélépipède ou si le parallélépipède de la partition est strictement inclus dans un cylindre), ce parallélépipède est inclus dans le domaine occupé par l'assemblage et il est retiré de la partition. Dans le cas contraire, le parallélépipède de la partition a une intersection avec un hémisphère ou un cylindre. Ce parallélépipède est remplacé par une partition du domaine constitué du parallélépipède privé de l'hémisphère ou du cylindre. La partition du domaine constitué du parallélépipède privé de l'hémisphère ou du cylindre est choisie de sorte que chaque volume la composant a huit sommets, douze arêtes et six faces. Le résultat de l'application de cette méthode est une partition du domaine occupé par le fluide en volumes possédant chacun huit sommets, douze arêtes et six faces. Chaque volume constituant cette partition peut être vu comme l'image par une transformation géométrique continue d'un hexaèdre de référence. À partir de cette partition, un logiciel de génération automatique de maillages peut construire la discrétisation en hexaèdres du domaine occupé par le fluide. Nous avons mis en œuvre [1] cette méthode de construction du domaine occupé par le fluide en utilisant le logiciel de CAO (Conception Assistée par Ordinateur) fourni par la plate-forme Salomé [2] développée par un consortium comprenant notamment EDF et le CEA.

La FIGURE 5.1 représente la partie de la frontière du domaine occupé par le fluide correspondant à la surface de l'assemblage. La surface représentée est partitionnée en faces ayant chacune quatre sommets

1. Nous remercions A. Martin, ingénieur de recherche CNRS, qui a élaboré le script Python de génération et discrétisation de la géométrie du domaine occupé par le fluide.
2. https ://www.salome-platform.org/

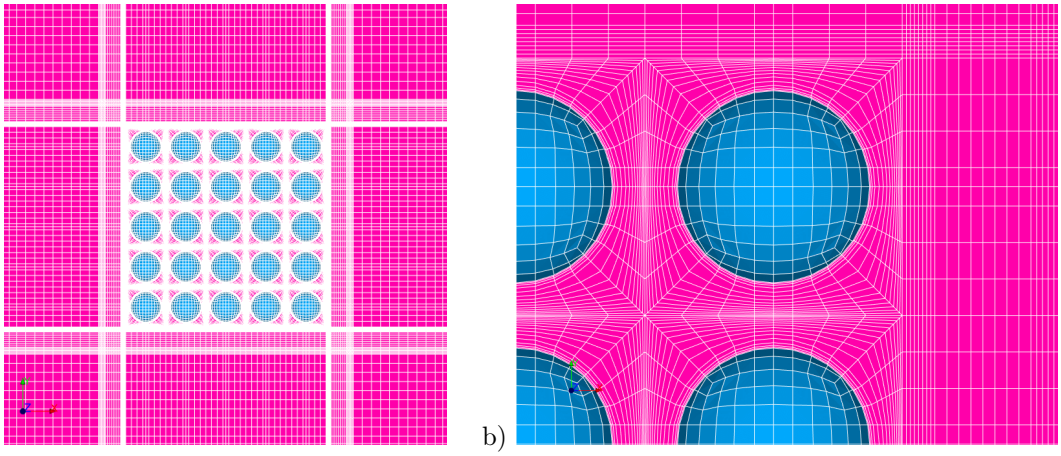


FIGURE 5.2 – Coupe dans un plan parallèle au plan $(x0z)$ de la discrétisation du domaine occupé par le fluide (a). Vue de détail de la discrétisation autour d'un cylindre (b).

et quatre arêtes. Chacune de ces faces appartient à un volume composant la partition du domaine occupé par le fluide. Représenter cette surface permet d'apprécier la complexité de cette partition. La FIGURE 5.1 illustre également les simplifications de la géométrie de l'assemblage qui ont été réalisées pour construire cette partition. Chaque arête est vive ; il n'y a ni congé, ni arrondi. Nous avons fait le choix de ne pas représenter les axes de forçage, afin de nous affranchir de la définition de la partition de la part du domaine occupé par le fluide entourant la jointure entre une grille et un axe de forçage. En revanche, les entretoises (*i.e.* une pièce qui relie deux cylindres ou une pièce qui relie un cylindre à la grille) sont représentées, car le sillage de l'écoulement derrière chacune des entretoises fait partie de l'influence des grilles sur les efforts ajoutés par l'écoulement axial.

5.2.2 Modèle de turbulence

Nous nous intéressons à des écoulements turbulents d'un fluide incompressible. Les équations RANS (Reynolds-*Averaged* Navier-Stokes) décrivent l'écoulement turbulent. Le système d'équations à résoudre dépend du modèle de turbulence considéré. Nous choisissons le modèle k-ω SST (*Shear-Stress Transport*). Le k-ω SST combine deux modèles classiques de la littérature : le modèle k-ω et le modèle k-ε. Le modèle k-ω est utilisé en proche paroi. Le modèle k-ε est utilisé loin des parois. Ce choix permet de profiter des avantages et d'éviter les inconvénients de chacun des deux modèles. Ce modèle de turbulence a été utilisé par (Joly *et al.*, 2021) pour montrer que le modèle TLP représente les efforts induits par le fluide sur un réseau de cylindre, dans le cadre d'essais réalisés en statique.

5.2.3 Construction du maillage

Le modèle de turbulence seul ne permet pas de représenter le comportement du fluide dans la première couche de cellules, rencontrée en partant d'une paroi, qui contient la couche limite. Afin de représenter correctement le comportement du fluide dans la couche limite, nous recourons à une loi de paroi. Le choix de la loi de paroi à appliquer dépend de la distance du centre de la première cellule à la paroi.

Le critère permettant de choisir la loi de paroi est écrit en termes de la valeur adimensionnelle de la distance du centre de la cellule à la paroi. La distance adimensionnelle y^+ s'obtient en divisant la distance y par l'équation suivante :

$$y^+ = \frac{yU^*}{\nu},$$

où ν est la viscosité cinématique du fluide et U^* est la vitesse de frottement à la paroi. Dans le cas où y^+ est inférieur à 5, la loi de paroi est décrite par le modèle de sous-couche visqueuse. Dans le cas où y^+ est supérieur à 30, la loi de paroi est décrite par le modèle de sous-couche inertielle.

La dimension de la première cellule à la paroi, dans la direction normale à la paroi, est appelée l'épaisseur de la cellule à la paroi. L'épaisseur de la première cellule à la paroi correspond au double de

la distance du centre de cette cellule à la paroi. Nous choisissons l'épaisseur de la première cellule à la paroi des cylindres et des grilles de sorte que le critère $y^+ < 5$ soit vérifié. Compte tenu de la méthode de construction du domaine occupé par le fluide, ce choix conditionne une dimension pour de nombreuses cellules de la discrétisation. Les dimensions des cellules qui n'ont pas été fixées par la règle précédente sont choisies *a priori* en appliquant des critères subjectifs. Nous choisissons, par exemple, les deux autres dimensions des cellules à la paroi des hémisphères de sorte à disposer d'une représentation raisonnable de la courbure des hémisphères (*cf.* FIGURE 5.2). L'épaisseur de la première cellule à la paroi de la veine est telle que la loi de paroi est décrite par le modèle de sous-couche inertielle. Nous avons utilisé le logiciel de génération automatique de maillages fourni par la plate-forme Salomé pour réaliser la discrétisation du domaine occupé. Une analyse de la convergence des quantités d'intérêt permet de vérifier, *a posteriori*, la qualité de la discrétisation.

La vitesse de frottement à la paroi n'est pas connue *a priori*. Afin de pouvoir définir un critère permettant de choisir les dimensions des cellules à la paroi *a priori*, nous utilisons l'approximation suivante :

$$U^* = U\sqrt{\frac{\lambda}{8}},$$

où U est la vitesse d'écoulement incidente et λ est le coefficient de frottement. Le coefficient de frottement λ est estimé par la formule empirique de Haaland :

$$\frac{1}{\sqrt{\lambda}} = -1.8\log_{10}\left(\frac{6.9}{Re_h}\right),$$

où Re_h est le nombre de Reynolds calculé à partir du diamètre hydraulique. Pour un conduit, le diamètre hydraulique est défini par :

$$D_h = 4\frac{\mathcal{A}}{\mathcal{P}},$$

où $\mathcal{A}$ est l'aire de la section mouillée et $\mathcal{P}$ est le périmètre mouillé. Dans notre cas, $\mathcal{A}$ est la section de la veine retranchée de la section des 25 cylindres de l'assemblage et $\mathcal{P}$ est le périmètre de la section de veine auquel on ajoute le périmètre de chacun des 25 cylindres.

5.2.4 Configuration commune à toutes les simulations numériques réalisées

Nous avons utilisé code_saturne [3], le logiciel de simulation numérique en mécanique des fluides développé par EDF R&D. Toutes les simulations réalisées partagent des caractéristiques communes listées ci-après.

Domaine de calcul

Le domaine occupé par le fluide est inclus dans un parallélépipède fermé par quatre faces représentant les parois de la veine, une face représentant l'entrée du domaine et une face représentant la sortie du domaine. Le domaine occupé par le fluide peut être décomposé en trois parties : une partie en amont de l'assemblage, une partie qui contient l'assemblage et une partie en aval de l'assemblage. Pour fixer la taille du domaine occupé par le fluide, il faut choisir la longueur de la partie en amont et la longueur de la partie en aval de l'assemblage. Nous choisissons pour ces deux parties une longueur égale à 30 fois le diamètre D d'un cylindre. Ce choix permet de garantir que dans la première couche de cellules après l'entrée du domaine et que dans la dernière couche de cellules avant la sortie du domaine, le gradient de pression soit prescrit par les conditions limites et ne dépende pas de l'écoulement à l'intérieur du domaine.

Conditions limites

L'usage et le retour d'expérience sur ces modèles nous conduisent à simuler l'écoulement en flux imposé, c'est-à-dire à choisir une condition de vitesse imposée en entrée et de pression imposée en sortie. La section de la veine d'essais modélisée est vue comme un canal infini. Ainsi, en condition d'entrée du domaine, une condition d'entrée en vitesse uniforme telle que la vitesse imposée en entrée est U_q, la

3. https ://www.code-saturne.org/cms/web/

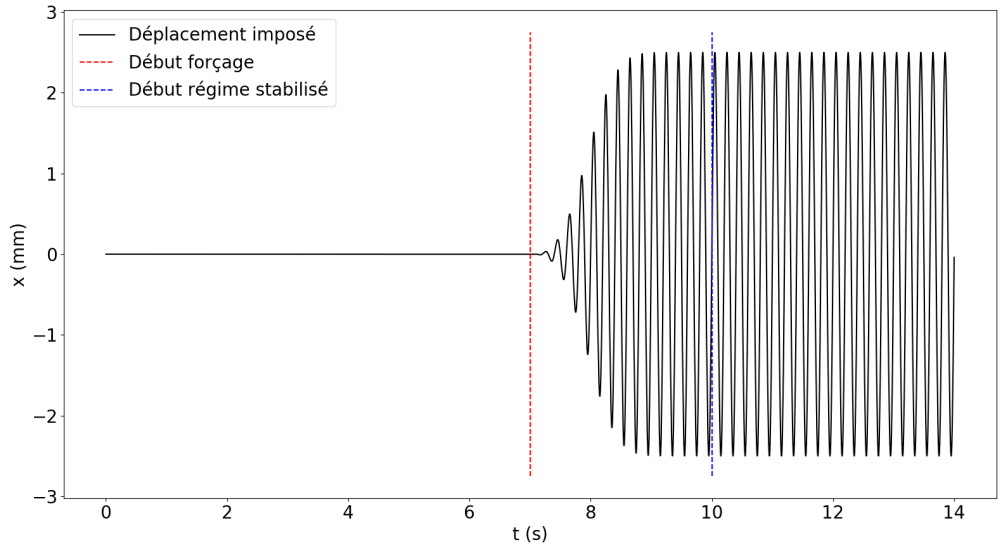


FIGURE 5.3 – Évolution, au cours du temps, de l'amplitude du déplacement imposé à l'assemblage dans les simulations numériques, pour une sollicitation harmonique d'amplitude $A = 2.5$ mm et une fréquence de sollicitation $f_0 = 5$ Hz.

vitesse débitante utilisée expérimentalement. Toutes les simulations présentées dans ce chapitre ont été réalisées pour $U_q = 0.8$ m/s, soit une vitesse incidente $U = 0.9$ m/s. On utilise, en condition de sortie, une condition à pression imposée. Dans nos essais, la pression imposée est la pression atmosphérique. Cette sortie modélisant la sortie de la veine d'essais vers le réservoir inférieur à surface libre, ce choix nous paraît approprié. Les conditions aux limites appliquées aux autres parties du bord du domaine sont des conditions de mur sans glissement classiques.

Paramètres numériques

L'intervalle de temps simulé est divisé en pas de temps de longueur fixe. Dans chaque cellule, à chaque instant de la simulation, le nombre de Courant est défini comme le produit de la vitesse et de la longueur du pas de temps, divisé par la taille de la cellule. Afin de garantir la stabilité du schéma numérique, la longueur du pas de temps doit être choisie de façon à satisfaire la condition CFL (Courant-Friedrichs-Lewy) qui stipule que le nombre de Courant doit être inférieur à 1 à chaque instant et dans chaque cellule.

Nous utilisons un pas de temps de longueur $\Delta t = 2.5.10^{-4}$ s. Ce choix assure que dans chaque cellule et à chaque instant de calcul, le nombre de Courant est inférieur à 0.5.

Éléments sur la machine de calcul utilisée

La discrétisation du domaine occupé par le fluide comporte environ 60 millions de cellules. Les simulations sont réalisées sur le cluster de calcul scientifique Gaïa [4] de EDF R&D et mobilisent 80 nœuds de calcul. Chaque nœud de calcul a deux processeurs (Xeon Gold 6140) comportant chacun 18 cœurs. La charge de calcul est donc d'environ 20 000 cellules par cœur. Chaque résultat de calcul représente la simulation de 14 s d'écoulement, soit 7 fois le temps nécessaire à une particule pour parcourir le domaine de calcul. Le résultat d'une telle simulation est obtenue en 72 h.

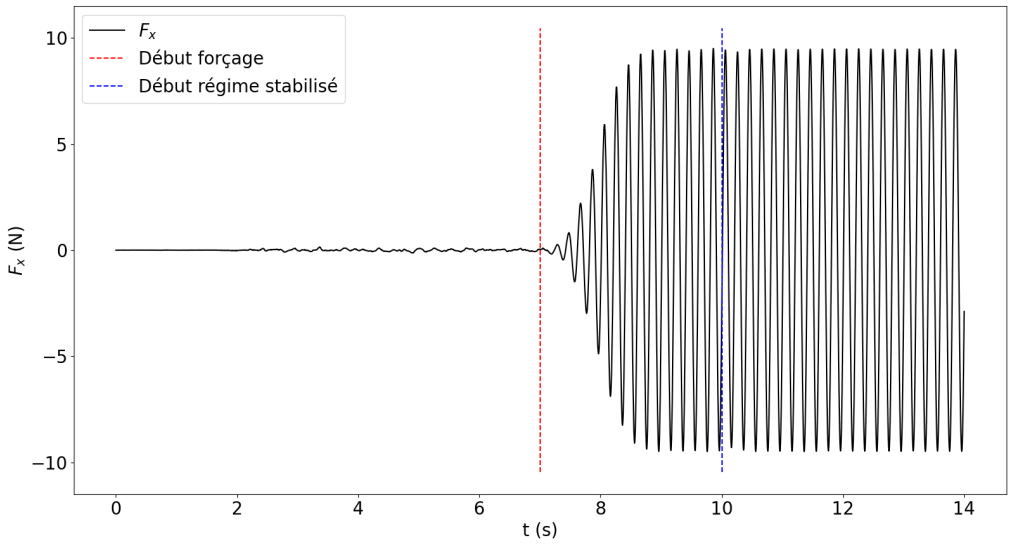


FIGURE 5.4 – Évolution de la composante F_x de la portance, obtenue à partir d'une simulation réalisée pour une fréquence de sollicitation $f_0 = 5$ Hz.

Déplacement imposé

Dans les essais, le déplacement de l'assemblage est imposé dans la direction (Ox). Le déplacement imposé est de la forme : $X(t) = A\sin(\omega_0 t)$, où ω_0 est la pulsation de la fréquence de sollicitation. Pour imposer le déplacement de l'assemblage dans les simulations numériques, nous imposons le déplacement $\mathbf{u}(t) = X(t)\mathbf{e}_x$, à chaque face située sur la partie de la frontière du domaine fluide correspondant à l'assemblage, à chaque instant. Compte tenu de la nouvelle position de ces faces, le module ALE calcule une nouvelle position pour chaque point à l'intérieur du domaine occupé par le fluide. Nous ne pouvons pas utiliser directement la fonction X pour décrire le déplacement imposé à chaque instant, car nous devons attendre que l'écoulement axial soit établi avant de mettre l'assemblage en mouvement.

Nous souhaitons que le déplacement imposé soit nul pendant une certaine période au début de la simulation puis devienne le déplacement X après un certain temps. La transition de 0 à la fonction X ne peut pas être instantanée, car le déplacement imposé doit être une fonction suffisamment régulière pour que l'accélération de l'assemblage soit une fonction continue du temps.

Nous avons choisi d'imposer le déplacement $\mathbf{u}(t) = \widetilde{X}(t)\mathbf{e}_x$, où la fonction $\widetilde{X}$ est définie par :

$$\widetilde{X}(t) = \begin{cases} 0, \text{ si } 0 \leq t < 7 \\ A\sin\left[\omega_0(t-7)\right]\left(1 - \exp\left[-0.8(t-7)^3\right]\right), \text{ si } t \geq 7 \end{cases} .$$

L'assemblage reste immobile pendant les 7 premières secondes de la simulation. À l'instant $t = 7$ s, l'assemblage se met en mouvement. Une phase, pendant laquelle l'amplitude du déplacement augmente progressivement, débute. Après l'instant $t = 10$ s, l'assemblage suit la fonction sinusoïdale d'amplitude A utilisée dans les essais. La FIGURE 5.3 représente l'évolution de la fonction $\widetilde{X}$ au cours du temps. La section 5.2.5 propose une étude qualitative des forces appliquées par le fluide à l'assemblage. Le résultat de cette analyse montre, *a posteriori*, que la fonction choisie pour imposer le déplacement de l'assemblage a les propriétés désirées.

5.2.5 Étude qualitative des efforts appliqués par le fluide sur l'assemblage

Nous proposons une analyse préliminaire des forces appliquées par le fluide à l'assemblage, issues d'une simulation numérique réalisée à partir d'une discrétisation raisonnable du domaine occupé par le fluide. Cette discrétisation du domaine ne sera pas nécessairement utilisée pour mener l'analyse quantitative des forces. En revanche, l'analyse qualitative proposée dans cette section permet de définir les quantités d'intérêts à étudier, pour déterminer les dimensions des cellules acceptables pour construire la discrétisation du domaine, qui sera utilisée pour réaliser l'analyse quantitative des forces. Cette analyse va également montrer que la fonction choisie pour imposer le déplacement de l'assemblage a les propriétés souhaitées.

La force appliquée par le fluide sur l'assemblage se décompose sous la forme : $F_x\mathbf{e}_x + F_y\mathbf{e}_y + F_z\mathbf{e}_z$. Nous appelons F_z la force de traînée. Nous appelons $F_x\mathbf{e}_x + F_y\mathbf{e}_y$ la force de portance. La force de portance se décompose en deux composantes : F_x et F_y. La force F_f introduite au chapitre 2.1.2 est la projection sur l'axe (Ox) des efforts exercés par le fluide sur la structure. La composante F_x de la portance correspond donc à la force F_f. Analyser la force exercée par le fluide sur la structure, en réponse à la sollicitation en déplacement imposée, consiste donc à considérer, dans le régime oscillant établi (*i.e.* pour $t \geq 10$), le système dont l'entrée est le déplacement imposé $X = \widetilde{X}$ et la sortie est la force $F_f = F_x$.

À chaque instant de la simulation, code_saturne donne à l'utilisateur accès à la résultante des forces appliquées par le fluide sur chaque face de la discrétisation appartenant à la frontière du domaine occupé par le fluide. Nous avons écrit une fonction en Fortran 90 pour calculer la résultante des forces exercées par le fluide sur l'assemblage. À chaque instant de la simulation, cette fonction calcule la somme des résultantes des forces appliquées par le fluide sur chaque face de la discrétisation située à la surface de l'assemblage. Il est à noter que les forces appliquées par le fluide sur chaque face de la discrétisation appartenant à la frontière du domaine occupé par le fluide ne font pas partie des données enregistrées par code_saturne. Il n'est donc pas possible de calculer la résultante des forces appliquées par le fluide sur l'assemblage en post-traitement.

La FIGURE 5.4 représente l'évolution au cours du temps de la composante F_x de la portance, obtenue à partir d'une simulation réalisée pour une fréquence de sollicitation $f_0 = 5$ Hz. Dans le régime oscillant établi, la composante F_x de la portance semble être une fonction sinusoïdale du temps. Pour montrer que la composante F_x de la portance est une fonction sinusoïdale du temps, nous analysons les passages par zéro de F_x. Considérons trois passages par zéro successifs de la composante F_x de la portance. L'intervalle de temps qui commence au premier passage par zéro et finit au troisième passage par zéro a une longueur égale à la période de la sollicitation en déplacement. Le deuxième passage par zéro se situe à la moitié de cet intervalle de temps. Nous en déduisons que si la fonction F_x est périodique, sa période est la période de la sollicitation en déplacement. Nous analysons ensuite l'amplitude absolue de la composante F_x de la portance. Cette amplitude absolue ne varie pas au cours du temps. Nous en déduisons que la composante F_x de la portance est une fonction sinusoïdale du temps, dont la fréquence est identique à la fréquence de sollicitation en déplacement. Nous en concluons que $F_f = F_x$ a la forme $F_f(t) = F_0 \sin(\omega_0 t + \phi)$ nécessaire pour pouvoir appliquer la méthode d'analyse dans le domaine temporel présentée en section 2.2.3. Les quantités d'intérêt à étudier, pour déterminer les dimensions des cellules acceptables pour construire la discrétisation du domaine, qui sera utilisée pour réaliser l'analyse quantitative des forces, sont : l'amplitude absolue F_0 et le déphasage ϕ.

La FIGURE 5.5 représente l'évolution au cours du temps de la traînée F_z, obtenue à partir d'une simulation réalisée pour une fréquence de sollicitation $f_0 = 5$ Hz. La traînée F_z a une évolution importante jusqu'à l'instant $t = 4$ s où elle fluctue autour d'une valeur moyenne d'environ 5.5 N. Nous attribuons cette fluctuation à la turbulence de l'écoulement. Nous considérons donc qu'à l'instant $t = 7$ s, lorsque l'assemblage entre en mouvement, l'écoulement axial est établi.

La FIGURE 5.6 représente l'évolution au cours du temps de la composante F_y de la portance, obtenue à partir d'une simulation réalisée pour une fréquence de sollicitation $f_0 = 5$ Hz. La composante F_y fluctue autour de 0. Nous attribuons ces fluctuations à la turbulence de l'écoulement. Ce résultat était attendu compte tenu que l'assemblage est sollicité dans la seule direction (Ox).

Aucune des trois composantes de la force exercées par le fluide sur la structure présente une discontinuité. Nous en concluons que le choix de la fonction $\widetilde{X}$ pour imposer le déplacement permet que l'accélération de l'assemblage soit une fonction continue du temps. Nous avons vu que la mise en mouve-

4. Page consacrée à Gaïa dans le top des 500 clusters les plus puissants au monde : https ://www.top500.org/system/179569/

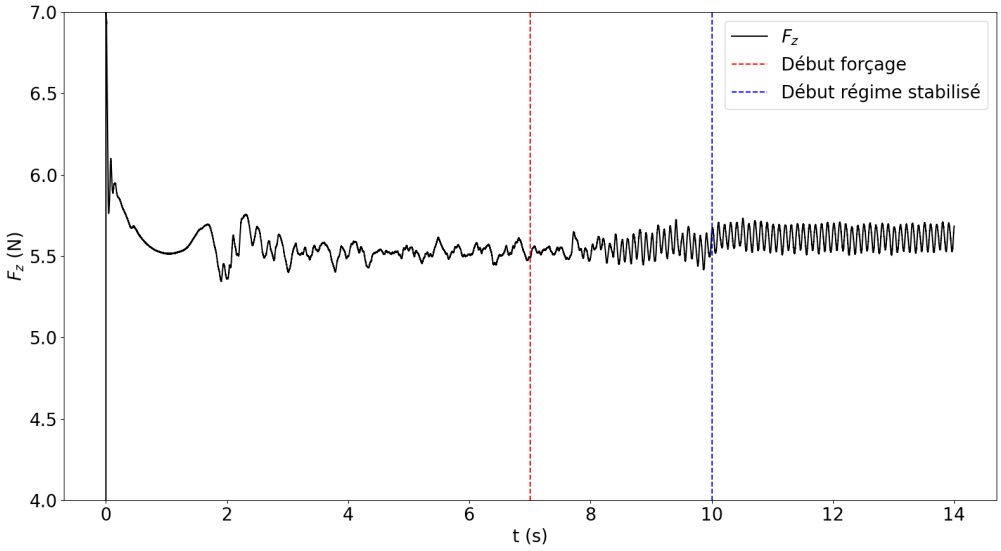


FIGURE 5.5 – Évolution de la traînée F_z, obtenue à partir d'une simulation réalisée pour une fréquence de sollicitation $f_0 = 5$ Hz.

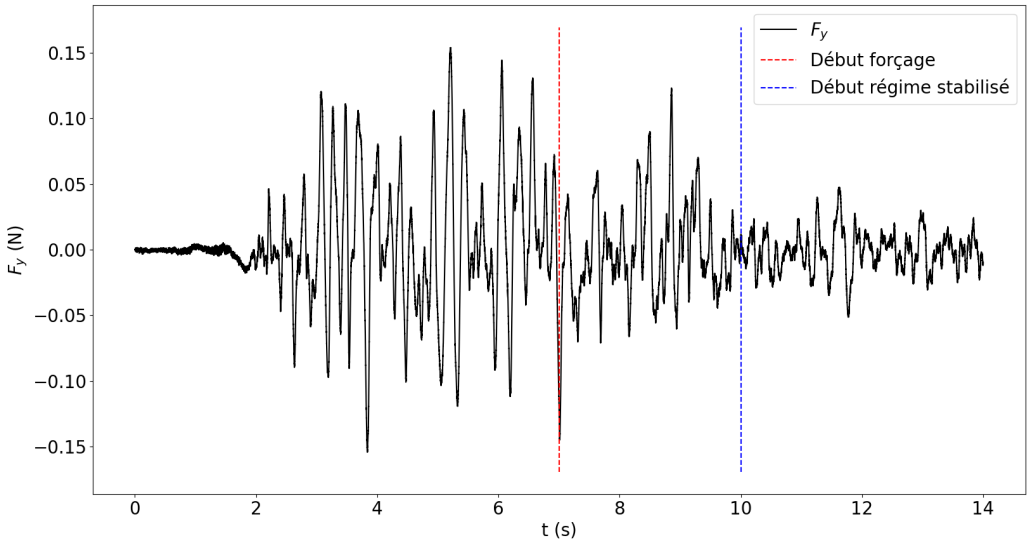


FIGURE 5.6 – Évolution de la composante F_y de la portance, obtenue à partir d'une simulation réalisée pour une fréquence de sollicitation $f_0 = 5$ Hz.

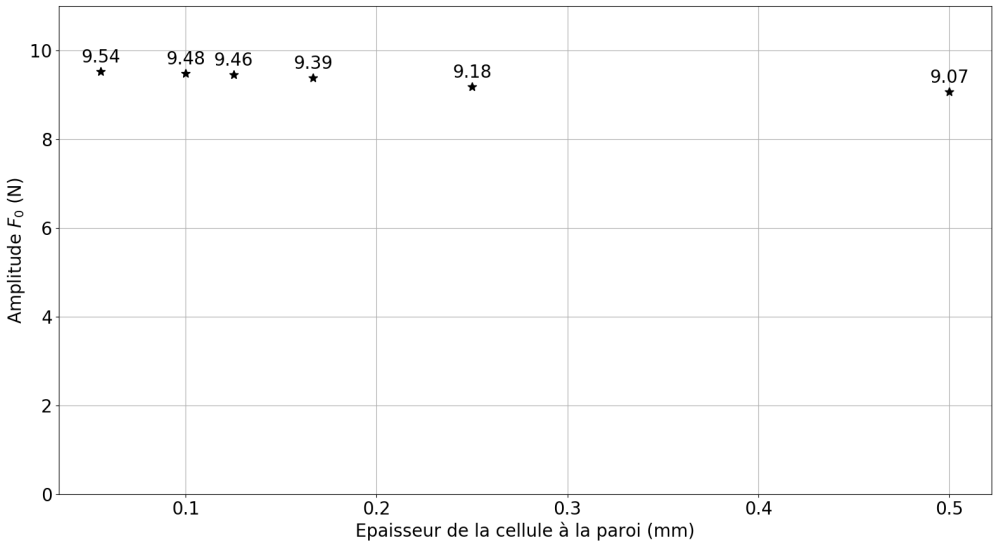


FIGURE 5.7 – Évolution de l'amplitude F_0 en fonction de l'épaisseur de la cellule à la paroi des cylindres, obtenue à partir de simulations réalisées pour une fréquence de sollicitation $f_0 = 5$ Hz.

ment de l'assemblage commence lorsque l'écoulement axial est établi. Nous en concluons que la fonction $\widetilde{X}$ a les propriétés souhaitées.

5.2.6 Choix de la finesse de la discrétisation en espace

L'analyse quantitative des forces appliquées par le fluide sur la structure est fondée sur le calcul de l'amplitude F_0 et du déphasage ϕ qui caractérisent la force F_f. Les propriétés de convergence de la méthode des Volumes Finis mise en œuvre dans code_saturne assurent que si la discrétisation (en espace et en temps) est suffisamment fine, les quantités d'intérêt F_0 et ϕ n'évoluent quasiment plus avec la finesse de la discrétisation en espace. Nous proposons d'étudier comment ces quantités d'intérêt évoluent quand la finesse de la discrétisation en espace évolue.

Chaque dimension de chaque cellule est fixée par la subdivision d'une arête de la géométrie en un nombre de segments donné. La définition de la subdivision de quelques arêtes bien choisies suffit à décrire toute la discrétisation de l'espace occupé par le fluide. Ces règles de subdivision sont liées entre elles de sorte que la finesse de la discrétisation est fixée quand l'épaisseur de la cellule à la paroi des cylindres est fixée. Étudier comment les quantités d'intérêt évoluent quand la finesse de la discrétisation en espace évolue revient donc à étudier comment les quantités d'intérêt évoluent quand l'épaisseur de la cellule à la paroi des cylindres évolue. La discrétisation utilisée dans l'étude qualitative présentée dans la section 5.2.5 correspond à la plus petite épaisseur de la cellule à la paroi des cylindres considérée.

La FIGURE 5.7 présente l'évolution de l'amplitude F_0 en fonction de l'épaisseur de la cellule à la paroi des cylindres, obtenue à partir de simulations réalisées pour une fréquence de sollicitation $f_0 = 5$ Hz. La moyenne des amplitudes F_0 obtenues est 9.35 N et l'écart-type à la moyenne des amplitudes correspondant à deux épaisseurs de la cellule à paroi des cylindres différentes est 0.17 N, soit 2% de la moyenne des amplitudes.

La FIGURE 5.8 présente l'évolution du déphasage ϕ en fonction de l'épaisseur de la cellule à la paroi des cylindres, obtenue à partir de simulations réalisées pour une fréquence de sollicitation $f_0 = 5$ Hz. La moyenne des déphasages ϕ obtenus est 1.27 rad et l'écart-type à la moyenne des déphasages correspondant à deux épaisseurs de la cellule à paroi des cylindres différentes est 0.02 rad, soit 1.5% de la moyenne des déphasages.

Les deux quantités d'intérêt varient d'au plus 2% sur l'intervalle d'épaisseur de la cellule à la paroi des cylindres considéré. Nous en concluons que les quantités d'intérêt n'évoluent quasiment pas avec la finesse

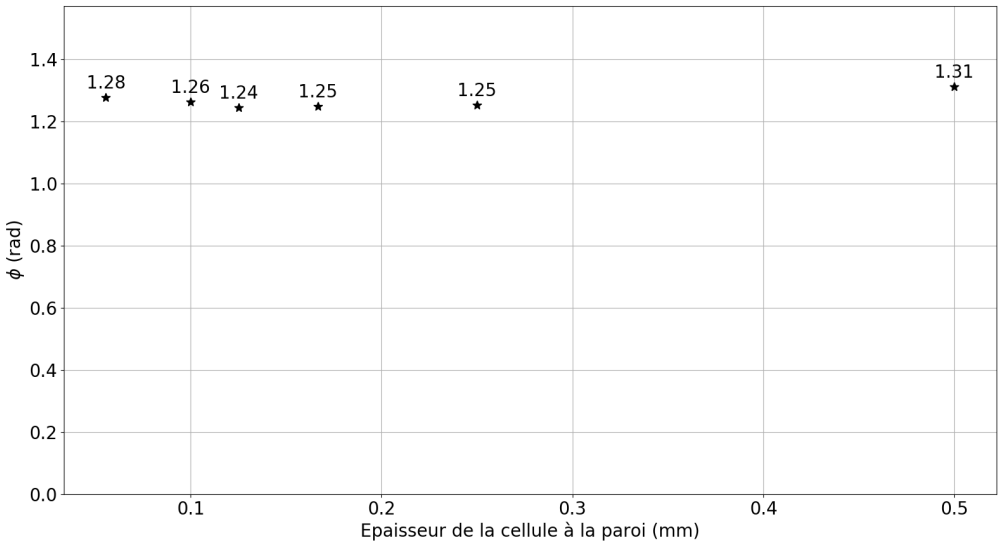


FIGURE 5.8 – Évolution du déphasage ϕ en fonction de l'épaisseur de la cellule à la paroi des cylindres, obtenue à partir de simulations réalisées pour une fréquence de sollicitation $f_0 = 5$ Hz.

de la discrétisation en espace. Nous choisissons de conserver la discrétisation utilisée lors de l'analyse qualitative pour réaliser l'analyse quantitative des forces appliquées par le fluide sur l'assemblage.

5.3 Étude quantitative des efforts appliqués par le fluide sur l'assemblage

5.3.1 Analyse de la résultante des efforts appliqués par le fluide sur le faisceau de cylindres

Nous avons confronté les signaux de force mesurés lors de nos essais à un modèle qui représente l'assemblage comme un faisceau de cylindres, sans prendre en compte les grilles (*cf.* 4.4.1). Ce modèle fait intervenir deux paramètres sans dimensions : le coefficient d'amortissement C_{amort} et le coefficient de masse C_{masse}. Nous avons proposé une méthode pour calculer la valeur de ces coefficients à partir de nos résultats d'essai. Si le modèle représente fidèlement nos essais et que la méthode de calcul des coefficients est correcte, les coefficients obtenus sont intrinsèques. La valeur du coefficient C_{amort} calculée à partir de nos résultats d'essai varie avec la vitesse incidente U. Nous avons conclu que soit le modèle n'est pas adapté, soit la méthode de calcul des coefficients n'est pas adaptée.

Notre hypothèse est que les grilles ont une contribution non négligeable à l'amortissement. Pour vérifier notre hypothèse, nous proposons d'isoler le faisceau de cylindres en calculant, à chaque instant de calcul de chaque simulation numérique réalisée, la résultante des forces appliquées par le fluide sur les cylindres. Pour chaque fréquence de sollicitation étudiée, nous appliquons la méthode d'analyse dans le domaine temporel présentée en section 2.2.3, au système dont l'entrée est le déplacement imposé $X = \tilde{X}$ et la sortie est la composante suivant l'axe (Ox) de la résultante des forces appliquées par le fluide sur les cylindres, pour obtenir une valeur du coefficient C_{amort} et une valeur du coefficient C_{masse}. Si l'analyse de l'évolution avec la fréquence des deux coefficients montre qu'ils sont indépendants de la fréquence, nous en conclurons que le modèle proposé représente fidèlement le faisceau de cylindres et que la méthode de calcul des coefficients est adaptée.

Nous appliquons la méthode d'analyse dans le domaine temporel présentée en section 2.2.3, au système dont l'entrée est le déplacement imposé $X = \tilde{X}$ et la sortie est la composante suivant l'axe (Ox) de la résultante des forces appliquées par le fluide sur l'intégralité de l'assemblage. Pour chaque fréquence de

Fréquence (Hz)	$C_{\text{amort}}(/)$	$C_{\text{amort}}(/)$ **sans grilles**
1	0.19	0.17
2	0.17	0.16
3	0.20	0.17
4	0.22	0.19
5	0.24	0.19

TABLE 5.1 – Coefficient d'amortissement calculé pour l'assemblage entier et pour le faisceau de cylindre seul.

Fréquence (Hz)	$C_{\text{masse}}(/)$	$C_{\text{masse}}(/)$ **sans grilles**
1	1.23	1.26
2	1.43	1.29
3	1.47	1.31
4	1.46	1.32
5	1.44	1.32

TABLE 5.2 – Coefficient de masse ajoutée calculé pour l'assemblage entier et pour le faisceau de cylindre seul.

sollicitation étudiée, nous obtenons une valeur du coefficient C_{amort} et une valeur du coefficient C_{masse}. Nous ne disposons pas d'un modèle capable de représenter le comportement des grilles. Dans le but d'alimenter une réflexion sur la proposition d'un modèle prenant en compte les grilles, nous présentons l'évolution avec la fréquence des coefficients C_{amort} et C_{masse} obtenus.

Les valeurs du coefficient C_{amort} obtenues en analysant les efforts exercés par le fluide sur l'intégralité de l'assemblage ainsi que les valeurs du coefficient C_{amort} obtenues en analysant les efforts exercés par le fluide sur le faisceau de cylindres sont reportés dans la table 5.1. Les valeurs du coefficient C_{masse} obtenues en analysant les efforts exercés par le fluide sur l'intégralité de l'assemblage ainsi que les valeurs du coefficient C_{masse} obtenues en analysant les efforts exercés par le fluide sur le faisceau de cylindres sont reportés dans la table 5.2.

Coefficient d'amortissement

La FIGURE 5.9 présente les évolutions du coefficient C_{amort} en fonction de la fréquence de sollicitation, obtenues dans différentes conditions (les scripts utilisés pour produire cette figure sont présentés en Annexe D). L'évolution du coefficient C_{amort}, obtenue à partir des résultats de simulation numérique, en prenant en compte l'intégralité de l'assemblage, est représentée par des étoiles. L'évolution du coefficient C_{amort}, obtenue à partir des résultats de simulation numérique, en ne prenant pas en compte les grilles, est représentée par des points. L'évolution du coefficient C_{amort}, obtenue par Divaret (2014), à partir de résultats d'essais, est représentée par une ligne pointillée bleue. L'évolution du coefficient C_{amort}, obtenue à partir de nos résultats d'essais, est représentée par une ligne pointillée rouge. Il est à noter que nous avons calculé la valeur du coefficient C_{amort}, à partir de nos résultats d'essais, en appliquant un adimensionnement au coefficient d'amortissement C_f obtenu en appliquant la méthode d'analyse dans le domaine fréquentiel (*cf.* section 2.2.3). La méthode de calcul du coefficient d'amortissement C_f, proposé par la méthode d'analyse dans le domaine fréquentiel, est fondée sur l'analyse de l'évolution d'une quantité d'intérêt avec la fréquence. C'est pourquoi le coefficient C_{amort} obtenu à partir des essais ne dépend pas de la fréquence.

Les points décrivant l'évolution du coefficient C_{amort}, obtenue à partir des résultats de simulation numérique et en ne prenant pas en compte l'intégralité de l'assemblage, semblent s'aligner sur une droite, à l'exception du point correspondant à la simulation réalisée à 1 Hz. La FIGURE 5.10 représente l'évolution au cours du temps de la composante F_x de la portance (calculée en ne prenant pas en compte l'intégralité de l'assemblage), obtenue à partir d'une simulation réalisée pour une fréquence de sollicitation $f_0 = 1$ Hz. Dans le régime oscillant établi, la composante F_x de la portance n'est pas une fonction sinusoïdale de

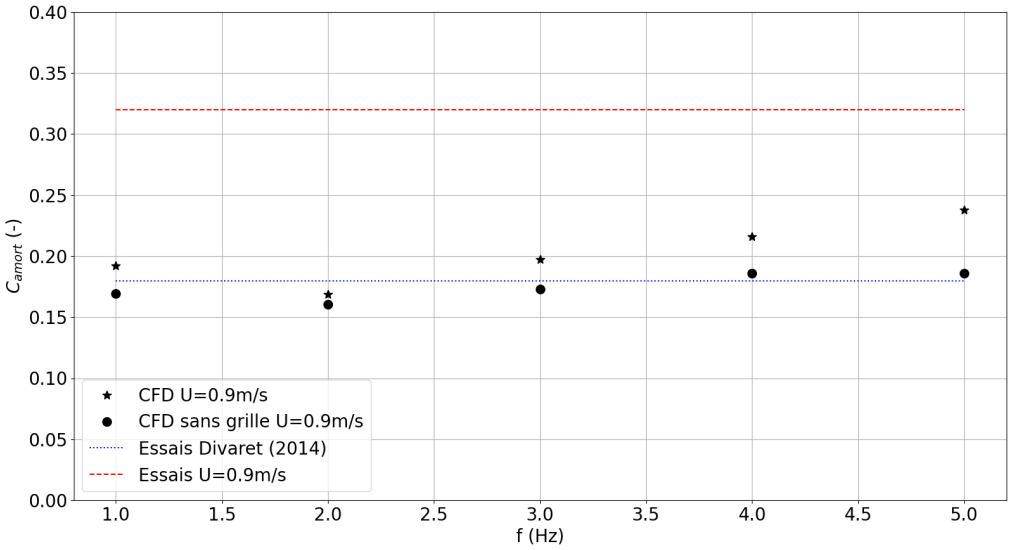


FIGURE 5.9 – Évolutions du coefficient d'amortissement C_{amort} en fonction de la fréquence de sollicitation, obtenues dans différentes conditions.

sorte que les coefficients C_{amort} et C_{masse} calculés ne sont pas représentatifs. Avant que l'assemblage ne soit mis en mouvement, durant une phase où seule la turbulence de l'écoulement peut faire évoluer la force F_x, le maximum de la force est d'environ 0.2 N tandis que, dans le régime oscillant établi, le maximum de la force F_x est d'environ 0.5 N. La contribution de la turbulence n'est pas négligeable devant le maximum de la force F_x, dans le régime oscillant établi. Il apparaît donc raisonnable d'exclure les résultats des simulations numériques correspondant à $f_0 = 1$ Hz de la suite de cette analyse.

Pour les fréquences entre 2 Hz et 5 Hz, la moyenne des valeurs du coefficient C_{amort}, obtenue à partir des résultats de simulation numérique et en ne prenant pas en compte les grilles, est 0.18 et l'écart-type à la moyenne est 0.01, soit 5% de la moyenne. Nous en déduisons que le coefficient C_{amort} ne dépend pas de la fréquence. La valeur du coefficient obtenue est $C_{\text{amort}} = 0.18 \pm 0.01$; cette valeur est celle obtenue par Divaret (2014), *i.e.* $C_{\text{amort}} = 0.18 \pm 0.05$. Ce résultat va dans le sens de montrer que le modèle proposé représente les efforts exercés par le fluide sur le faisceau de cylindres de l'assemblage. Toutefois, nous ne pouvons pas conclure avant d'avoir analysé l'évolution du coefficient C_{masse}.

Pour les fréquences entre 2 Hz et 5 Hz, les points décrivant l'évolution du coefficient C_{amort}, obtenue à partir des résultats de simulation numérique et prenant en compte l'intégralité de l'assemblage, semblent s'aligner sur une droite de pente non nulle. Nous avons montré que le coefficient C_{amort}, obtenu en ne prenant pas en compte les grilles, ne dépend pas de la fréquence. Nous en déduisons que la dépendance à la fréquence du coefficient C_{amort}, obtenu en prenant en compte l'intégralité de l'assemblage, est due aux grilles. La contribution des grilles à l'amortissement semble donc varier linéairement avec la fréquence. Des simulations numériques supplémentaires sont nécessaires pour confirmer cette tendance.

Nous remarquons que les valeurs du coefficient C_{amort}, obtenues à partir des résultats de simulation numérique et prenant en compte l'intégralité de l'assemblage, sont inférieures à la valeur obtenue à partir de nos essais. Dans nos simulations, le profil de vitesse à l'entrée du domaine occupé par le fluide est uniforme. Or, dans les essais, le profil de vitesse à l'entrée de la veine n'est pas uniforme car l'écoulement a traversé un nid d'abeille installé dans la veine haute, à la sortie du réservoir supérieur. Notre hypothèse est que la différence entre ces deux profils de vitesse explique que l'effet d'amortissement de l'écoulement simulé est différent de l'effet d'amortissement de l'écoulement réel. Notre hypothèse est compatible avec les travaux de De Ridder *et al.* (2015) qui a montré que dans certaines conditions, certains coefficients du modèle TLP peuvent dépendre de caractéristiques de l'écoulement, en l'occurrence le niveau de turbulence de l'écoulement. Il nous faut toutefois rappeler que la géométrie de l'assemblage utilisée dans les simulations ne correspond pas exactement à la géométrie de l'assemblage utilisée dans les essais, et

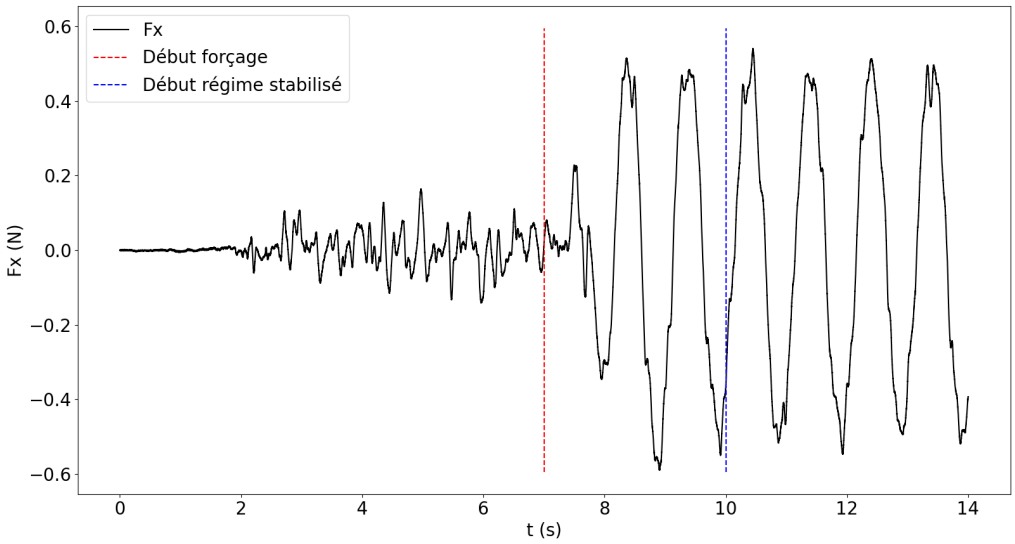


FIGURE 5.10 – Évolution de la composante F_x de la portance, obtenue à partir d'une simulation réalisée pour une fréquence de sollicitation $f_0 = 1$ Hz.

que nous ne connaissons pas *a priori* les conséquences de cette simplification de la géométrie sur la valeur du coefficient C_{amort}.

Coefficient de masse ajoutée

La FIGURE 5.11 présente les évolutions du coefficient C_{masse} en fonction de la fréquence de sollicitation, obtenues dans différentes conditions (les scripts utilisés pour produire cette figure sont présentés en Annexe D). L'évolution du coefficient C_{masse}, obtenue à partir des résultats de simulation numérique, en prenant en compte l'intégralité de l'assemblage, est représentée par des étoiles. L'évolution du coefficient C_{masse}, obtenue à partir des résultats de simulation numérique, en ne prenant pas en compte les grilles, est représentée par des points. L'évolution du coefficient C_{masse}, obtenue par Divaret (2014), à partir de résultats d'essais, est représentée par une ligne pointillée bleue.

Afin de disposer d'une valeur de référence pour le paramètre de masse C_{masse}, nous avons réalisé une simulation numérique fondée sur un modèle simplifié. Nous considérons un faisceau de 25 cylindres infiniment long répartis comme les cylindres de l'assemblage. L'écoulement axial entre ces cylindres ne dépend pas de la coordonnée z. Nous pouvons donc utiliser un modèle bidimensionnel pour représenter cet écoulement dans le plan xOy. En faisant l'hypothèse d'un fluide parfait, la méthode proposée par Chen (1987) permet d'estimer le coefficient de masse ajoutée du faisceau, qui constitue une borne inférieure pour le coefficient C_{masse}. Ce coefficient ne dépend pas de la fréquence. Il est représenté par la ligne pointillée verte sur la FIGURE 5.11.

Pour les fréquences entre 2 Hz et 5 Hz, la moyenne des valeurs du coefficient C_{masse}, obtenue à partir des résultats de simulation numérique et en ne prenant pas en compte les grilles, est 1.31 et l'écart-type à la moyenne est 0.01, soit 1% de la moyenne. Nous en déduisons que le coefficient C_{masse} ne dépend pas de la fréquence. Compte tenu que nous avons déjà montré que le coefficient C_{amort} ne dépend pas de la fréquence, nous en concluons que le modèle proposé représente les efforts exercés par le fluide sur le faisceau de cylindres de l'assemblage. La valeur du coefficient obtenue est $C_{\text{masse}} = 1.31 \pm 0.01$; cette valeur est inférieure à la valeur obtenue par Divaret (2014), *i.e.* $C_{\text{masse}} = 1.45 \pm 0.3$. Cet écart peut s'expliquer par le fait que la résultante des efforts étudiée par Divaret (2014) s'exerce sur un système composé d'un faisceau de cylindres et d'entretoises et pas d'un faisceau de cylindres seul.

Pour les fréquences entre 2 Hz et 5 Hz, les points décrivant l'évolution du coefficient C_{masse}, obtenue à partir des résultats de simulation numérique et prenant en compte l'intégralité de l'assemblage, semblent s'aligner sur une droite horizontale. Nous en déduisons que, pour le modèle représentant les efforts exercés

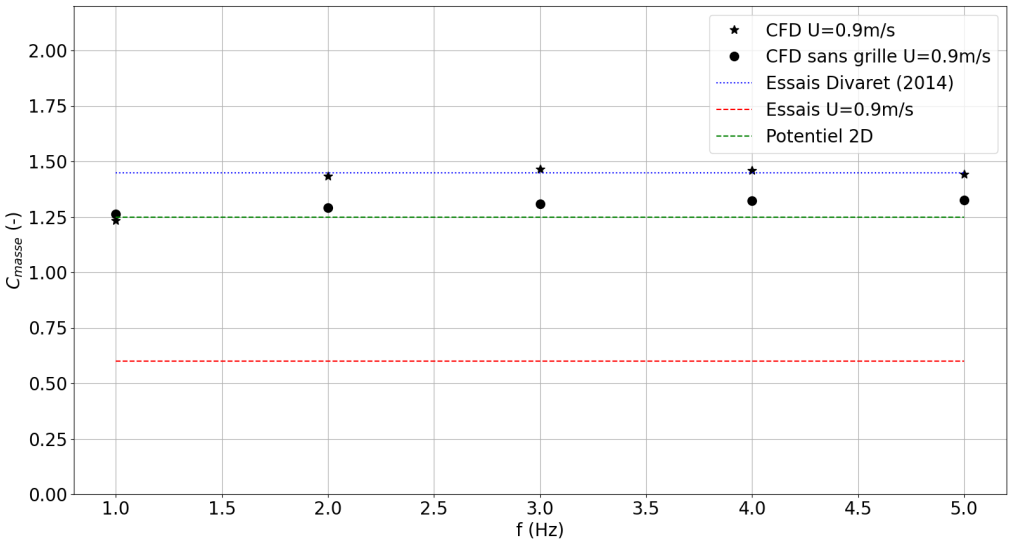


FIGURE 5.11 – Évolutions du coefficient de masse C_{masse} en fonction de la fréquence de sollicitation, obtenues dans différentes conditions.

par le fluide sur l'intégralité de l'assemblage, le coefficient C_{masse} ne dépend pas de la fréquence. La valeur du coefficient obtenue à partir des résultats de simulation numérique et prenant en compte l'intégralité de l'assemblage est $C_{\text{masse}} = 1.45 \pm 0.015$; cette valeur correspond à la valeur obtenue prévue par Divaret (2014). Ce résultat semble indiquer que, en ce qui concerne la représentation de la masse ajoutée, le modèle et la méthode d'adimensionnement proposée peuvent être étendus à l'intégralité de l'assemblage. La prise en compte des grilles se traduit alors par une augmentation de la valeur du coefficient C_{masse}, par rapport à la valeur du coefficient C_{masse} obtenue sans prendre en compte les grilles.

À présent que nous disposons d'une valeur de référence pour le coefficient C_{masse} nous pouvons calculer la valeur du coefficient C_{masse} correspondant aux essais réalisés pour $U = 0.9$ m/s. Pour ces essais, nous avons $M_f = 1.63$ kg (*cf.* FIGURE 4.10). En appliquant l'adimensionnement proposé en section 2.2.3, nous obtenons $C_{\text{masse}} = 0.6$. Ce coefficient ne dépend pas de la fréquence. Il est représenté par la ligne pointillée rouge sur la FIGURE 5.11. Contrairement aux valeurs obtenues à partir de nos résultats de simulation numérique et à la valeur obtenue par Divaret (2014), la valeur du coefficient C_{masse} obtenue à partir de nos essais est inférieure à la valeur fournie par le modèle bidimensionnel, qui constitue une estimation basse. Cela confirme que des essais supplémentaires, permettant de couvrir une plus grande gamme de fréquence, sont nécessaires pour obtenir une estimation correcte de la masse ajoutée à partir des résultats d'essais.

5.3.2 Analyse de la résultante des efforts appliqués par le fluide sur chaque cylindre du faisceau

Nous avons montré que le modèle proposé par Divaret (2014) représente fidèlement la résultante des efforts exercés par le fluide sur le faisceau de cylindres de l'assemblage. Ce modèle fait intervenir un paramètre d'amortissement C_{amort} et un paramètre de masse C_{masse} qui sont intrinsèques au faisceau de cylindres. Divaret (2014) propose d'utiliser les mêmes paramètres pour décrire les efforts exercés sur chaque cylindre du faisceau. Cette proposition est fondée sur l'hypothèse suivante : les résultantes des efforts exercés par le fluide sur chacun des N cylindres qui composent le faisceau sont identiques. Nous souhaitons à présent déterminer si cette hypothèse est vérifiée.

Fréquence (Hz)	$C_{\text{amort}}(/)$	$C_{\text{masse}}(/)$
1	0.19 ± 0.09	1.37 ± 0.63
2	0.17 ± 0.05	1.35 ± 0.13
3	0.18 ± 0.05	1.33 ± 0.18
4	0.19 ± 0.05	1.33 ± 0.13
5	0.19 ± 0.07	1.33 ± 0.19

TABLE 5.3 – Moyenne et écart-type à la moyenne, des coefficients d'amortissement C_{amort} et de masse C_{masse}, obtenus pour chaque cylindre.

Homogénéité des efforts appliqués sur le faisceau de cylindres

Nous avons calculé les valeurs des paramètres d'amortissement C_{amort} et un paramètre de masse C_{masse} du modèle qui représente la résultante des efforts exercés par le fluide sur le faisceau de cylindres de l'assemblage. Pour ce faire, nous avons appliqué la méthode d'analyse dans le domaine temporel au système, dont l'entrée est le déplacement imposé $X = \widetilde{X}$ et la sortie est la composante suivant l'axe (Ox) de la résultante des forces appliquées par le fluide sur le faisceau de $N = 25$ cylindres. Nous proposons dans cette section d'appliquer la méthode d'analyse dans le domaine temporel au système, dont l'entrée est le déplacement imposé $X = \widetilde{X}$ et la sortie est la composante suivant l'axe (Ox) de la résultante des forces appliquées par le fluide sur chacun des cylindres du faisceau, pour obtenir une valeur du coefficient C_{amort} et une valeur du coefficient C_{masse} pour chaque cylindre du faisceau. Si l'hypothèse est vérifiée, nous obtiendrons que les valeurs des coefficients C_{amort} et C_{masse} sont identiques pour chaque cylindre et égales, respectivement, aux valeurs des C_{amort} et C_{masse} correspondant au faisceau de cylindres.

La table 5.3 présente la moyenne et l'écart-type à la moyenne des coefficients d'amortissement C_{amort} et de masse C_{masse} obtenus pour chaque cylindre. Pour chaque fréquence entre 2 Hz et 5 Hz, l'écart-type à la moyenne représente environ 30% de la moyenne des valeurs de C_{amort} tandis que l'écart-type à la moyenne représente environ 12% de la moyenne des valeurs de C_{masse}. Nous en concluons que les valeurs des coefficients C_{amort} et C_{masse} varient d'un cylindre à l'autre et donc que l'hypothèse n'est pas vérifiée.

Répartition des efforts dans le faisceau de cylindres

À présent que nous avons montré que les valeurs des coefficients C_{amort} et C_{masse} varient d'un cylindre à l'autre, nous souhaitons décrire cette variation. Les cylindres sont disposés sur un réseau carré de cinq cylindres de côté. L'axe de chaque cylindre est repéré dans le plan (xOy) par ses coordonnées x et y. Nous associons à chaque cylindre du faisceau un couple de coordonnées sans dimension $i_x = x/P$ et $i_y = y/P$. La FIGURE 5.12 présente la disposition des $N = 25$ cylindres du faisceau.

Les figures 5.13 à 5.16 présentent la répartition des coefficients d'amortissement C_{amort} et de masse C_{masse}. Chaque figure correspond à une fréquence de sollicitation entre 2 Hz et 5 Hz. Chaque figure comporte deux graphes : à gauche, la répartition du coefficient C_{amort} et à droite, la répartition du coefficient C_{masse}. Sur chaque graphe, chaque courbe correspond à une rangée de cylindres partageant la même coordonnée i_x du cylindre et représente l'évolution du coefficient en fonction de la coordonnée i_y du cylindre.

Pour chaque fréquence, les courbes représentatives de l'évolution du coefficient d'amortissement C_{amort} se répartissent en trois groupes : 1- les courbes correspondant à $i_x = -2$ et $i_x = +2$, 2- les courbes correspondant à $i_x = -1$ et $i_x = +1$ et 3- la courbe correspondant à $i_x = 0$. Dans chaque groupe, les courbes sont quasiment superposées. Les courbes du groupe 1 correspondent à une valeur du coefficient C_{amort} beaucoup plus élevée que la valeur correspondant aux courbes du groupe 2. Les courbes du groupe 2 correspondent à une valeur du coefficient C_{amort} légèrement plus élevée que la valeur correspondant à la courbe du groupe 3. Le groupe 1 correspond aux rangées de cylindres situées aux extrémités du réseau selon l'axe de sollicitation. Le fait que les cylindres de ce groupe possèdent la valeur la plus élevée du coefficient C_{amort} traduit que les cylindres situés aux extrémités du réseau selon l'axe de sollicitation ont la plus forte contribution à l'amortissement. Ce résultat est cohérent puisque l'espace entre ces cylindres constitue l'entrée et la sortie des canaux empruntés par l'écoulement transverse. L'écoulement transverse a perdu une partie de son énergie quand il atteint la rangée de cylindres

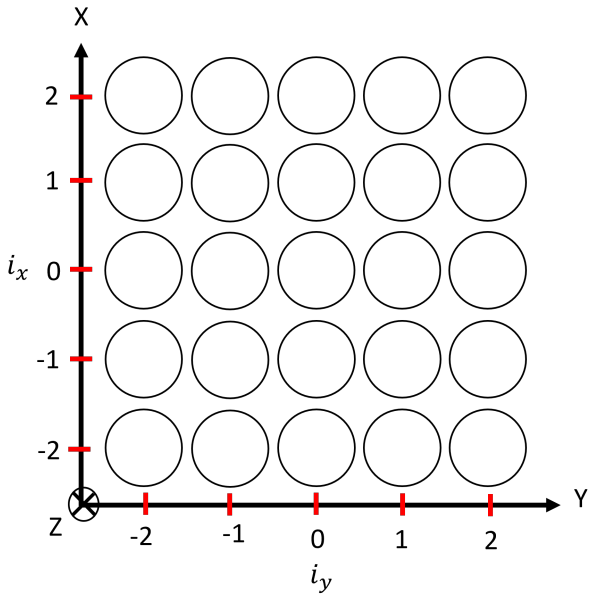


FIGURE 5.12 – Disposition des cylindres du faisceau.

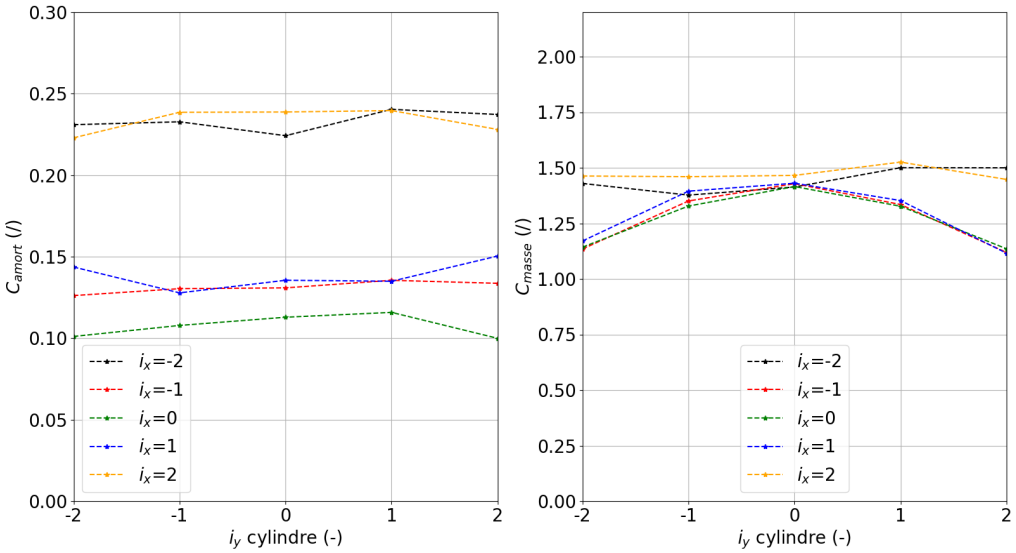


FIGURE 5.13 – Coefficient d'amortissement C_{amort} d'un cylindre en fonction de la coordonnée i_y du cylindre (à gauche). Coefficient d'amortissement C_{masse} d'un cylindre en fonction de la coordonnée i_y du cylindre (à droite). Sur chaque graphe, chaque courbe correspond à une rangée de cylindres partageant la même coordonnée i_x. Valeurs obtenues pour une fréquence de sollicitation de 2 Hz.

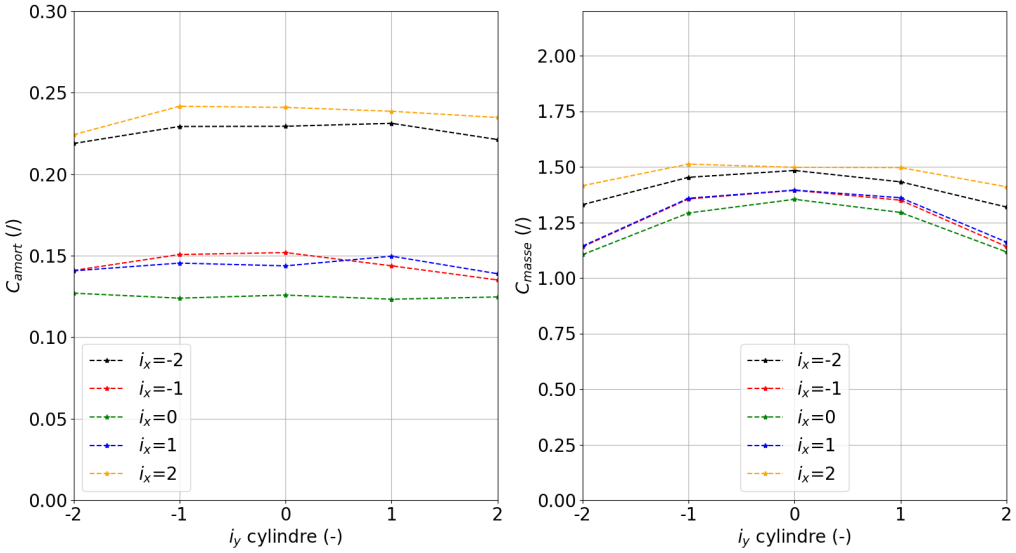


FIGURE 5.14 – Coefficient d'amortissement C_{amort} d'un cylindre en fonction de la coordonnée i_y du cylindre (à gauche). Coefficient d'amortissement C_{masse} d'un cylindre en fonction de la coordonnée i_y du cylindre (à droite). Sur chaque graphe, chaque courbe correspond à une rangée de cylindres partageant la même coordonnée i_x. Valeurs obtenues pour une fréquence de sollicitation de 3 Hz.

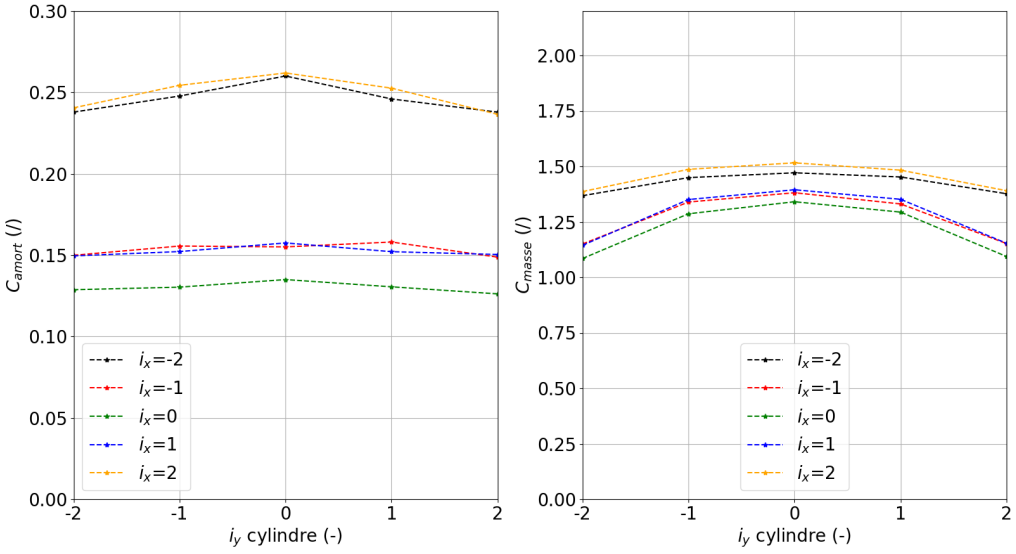


FIGURE 5.15 – Coefficient d'amortissement C_{amort} d'un cylindre en fonction de la coordonnée i_y du cylindre (à gauche). Coefficient d'amortissement C_{masse} d'un cylindre en fonction de la coordonnée i_y du cylindre (à droite). Sur chaque graphe, chaque courbe correspond à une rangée de cylindres partageant la même coordonnée i_x. Valeurs obtenues pour une fréquence de sollicitation de 4 Hz.

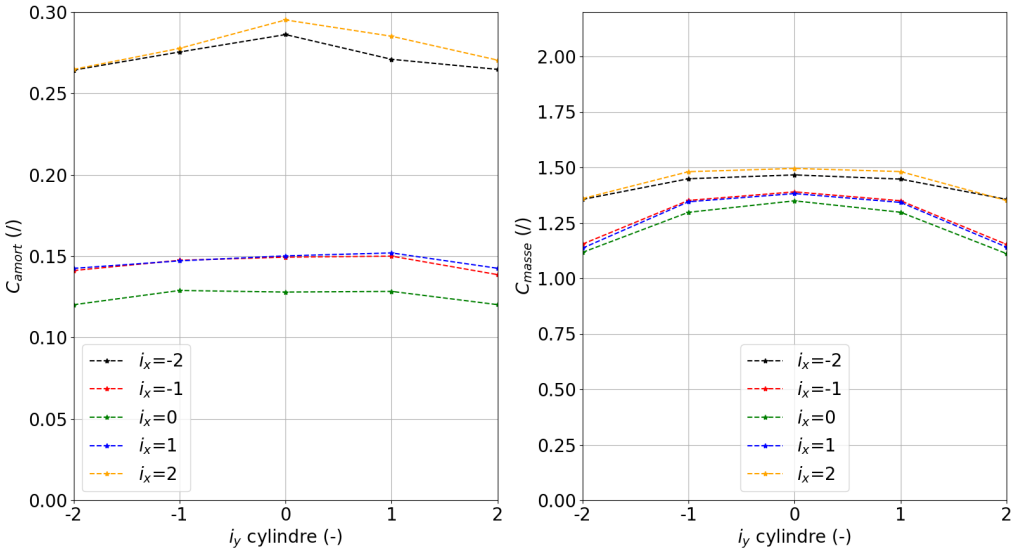


FIGURE 5.16 – Coefficient d'amortissement C_{amort} d'un cylindre en fonction de la coordonnée i_y du cylindre (à gauche). Coefficient d'amortissement C_{masse} d'un cylindre en fonction de la coordonnée i_y du cylindre (à droite). Sur chaque graphe, chaque courbe correspond à une rangée de cylindres partageant la même coordonnée i_x. Valeurs obtenues pour une fréquence de sollicitation de 5 Hz.

suivante, constituée des cylindres du groupe 2. L'écoulement transverse a perdu encore une partie de son énergie quand il atteint la rangée de cylindre centrale, constituée par le groupe 3.

Dans chaque groupe, les courbes semblent être des droites, de sorte que le coefficient C_{amort} semble ne pas dépendre de la coordonnée i_y. Ce résultat est cohérent car il indique que la contribution d'un cylindre à l'amortissement dépend peu de la position de ce cylindre sur l'axe orthogonal à la direction de sollicitation.

Les courbes du groupe 1 correspondent à une valeur du coefficient C_{masse} légèrement plus élevée que la valeur correspondant aux courbes du groupe 2. Les courbes du groupe 2 correspondent à une valeur du coefficient C_{masse} légèrement plus élevée que la valeur correspondant à la courbe du groupe 3. Dans chaque groupe, le coefficient C_{masse} dépend de la coordonnée i_y : la valeur du coefficient décroît lorsqu'on s'éloigne du cylindre au centre de la rangée ($i_y = 0$).

Répartition des efforts sur un cylindre

Le modèle qui représente la résultante des efforts exercés par le fluide sur un cylindre a été obtenu à partir d'un modèle qui représente la densité linéique des efforts exercés par le fluide le long du cylindre. Le passage du modèle qui représente la densité linéique d'efforts le long du cylindre au modèle qui représente la résultante des efforts sur le cylindre est fondé sur l'hypothèse suivante : la densité linéique d'efforts en un point ne dépend pas de la position du point le long de l'axe du cylindre. Pour vérifier cette hypothèse, nous avons extrait [5], à chaque instant de calcul, la densité linéique des efforts exercés par le fluide le long de chaque cylindre.

La FIGURE 5.17 présente la composante suivant l'axe (Ox) de la force exercée par le fluide sur plusieurs cylindres en fonction de la coordonnée z. Chaque courbe correspond à un cylindre. Les efforts ont été extraits à un instant correspondant à un maximum de la composante F_x de la portance, pour une fréquence de sollicitation $f_0 = 5$ Hz. Nous considérons trois zones. À chaque extrémité d'un cylindre, nous définissons une zone où la composante suivant l'axe (Ox) de la force exercée par le fluide varie brutalement. Les efforts exercés par le fluide sur chaque cylindre dans ces zones dépendent fortement des grilles. Entre ces deux zones, nous identifions une zone qui est moins soumise à l'influence des grilles.

5. Nous remercions S. Benhamadouche, expert senior EDF, de nous avoir fourni la fonction Fortran nécessaire.

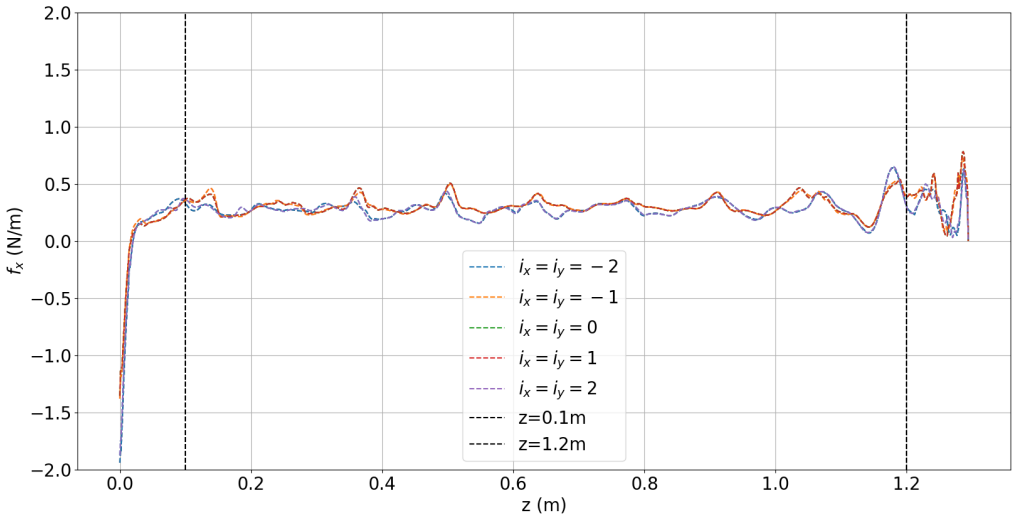


FIGURE 5.17 – Composante suivant l'axe (Ox) de la force exercée par le fluide sur plusieurs cylindres en fonction la coordonnée z. Chaque courbe correspond à un cylindre. Les efforts ont été extrait à un instant correspondant à un maximum de la composante F_x de la portance, pour une fréquence de sollicitation $f_0 = 5$ Hz.

Dans cette zone, la force semble varier autour d'une valeur constante. Nous attribuons ces variations à la turbulence. Nous en concluons que, dans cette zone, la densité linéique de force est constante et l'hypothèse est vérifiée.

5.4 Conclusion

Nous avons présenté dans ce chapitre le modèle numérique de la veine d'essai. Ce modèle se fonde sur une discrétisation en hexaèdres du domaine occupé par le fluide. La réalisation de cette discrétisation s'appuie sur une simplification de la géométrie de l'objet d'étude : les détails les plus fins sont omis et les axes de forçage ne sont pas représentés. La simulation numérique calcule des champs décrivant l'écoulement turbulent du fluide autour de la structure mobile.

Les efforts exercés par le fluide sur l'objet d'étude sont enregistrés à chaque instant de calcul. Les contributions du faisceau de cylindres et des grilles à la résultante de ces efforts sont séparées. L'analyse conjointe de la résultante des forces exercées sur le faisceau de cylindres et sur l'objet d'étude complet permet de montrer que le modèle proposé par Divaret (2014) décrit la force d'amortissement exercée sur le faisceau de cylindre et la force de masse ajoutée exercée sur l'objet d'étude complet. La valeur du coefficient d'amortissement obtenue à partir des efforts exercés sur le faisceau de cylindres est égale à la valeur obtenue par Divaret (2014). La valeur du coefficient de masse ajoutée obtenue à partir des efforts exercés sur l'objet d'étude complet correspond à la valeur obtenue par Divaret (2014).

Nous avons également analysé la répartition des efforts exercés par le fluide sur chacun des cylindres du faisceau. Nous avons observé que dans une rangée de cylindres alignée avec la direction orthogonale à la direction de sollicitation, les efforts sont répartis de façon homogène sur les cylindres de la rangée. Les efforts exercés sur une rangée sont symétriques par rapport à la rangée centrale. Plus la rangée est éloignée de la rangée centrale, plus les efforts exercés sur la rangée sont importants.

Nous avons enfin analysé la densité linéique des forces exercées par le fluide le long de plusieurs cylindres. Nous avons montré que les densités linéiques de forces sont homogènes loin des extrémités, comme le modèle TLP le prévoit.

Le chapitre suivant clos ce manuscrit. Il présente un bilan de l'ensemble du travail réalisé et liste quelques pistes d'approfondissement.

Chapitre 6

Bilan et perspectives

6.1 Bilan

Dans ce travail de thèse, nous avons étudié la dynamique d'une maquette d'assemblage oscillant sous écoulement axial, dans le but de décrire le mécanisme de transfert d'énergie mécanique de la maquette au fluide. La maquette d'assemblage est constituée d'un faisceau de 25 cylindres maintenus ensemble par deux grilles. Cette structure peut être décrite comme un objet indéformable. La maquette d'assemblage est placée dans une veine d'essai verticale et à section carrée. L'eau s'écoule de haut en bas dans la veine d'essai. Deux parois de la veine en vis-à-vis sont munies de hublots. Durant un essai, la maquette subit une translation, dans le plan horizontal, suivant une direction perpendiculaire à ces parois. Le déplacement de la maquette est une fonction sinusoïdale du temps.

Nous avons conçu, mis en oeuvre et qualifié un système innovant pour imposer le déplacement de la maquette. Ce système est composé de deux mécanismes à came synchronisés. Chaque mécanisme déplace une grille de l'objet d'étude. Nous avons étudié la capacité de ce système à reproduire un signal harmonique. Nous avons montré que ce système reproduit fidèlement un signal harmonique. Le déplacement mesuré de chaque grille est quasiment une sinusoïde. Les déplacements des deux grilles sont légèrement déphasés.

Un capteur laser mesure le déplacement de chaque grille. Une cellule de force mesure la résultante des efforts exercés sur chaque grille. Nous avons proposé un modèle masse-ressort-amortisseur pour décrire la dynamique de la maquette d'assemblage. Les fréquences de sollicitation étudiées dans les essais sont $f_0 = 1, 2, 3, 4$ et 5 Hz. Pour chaque fréquence, nous avons réalisé des essais en air, en eau stagnante et sous écoulement axial. Les vitesses d'écoulement étudiées sont $U = 0.9$ et 1 m/s. L'analyse des résultats de chaque essai a montré que le système, dont l'entrée est le signal de déplacement imposé et la sortie est la réponse en force de la maquette d'assemblage, est linéaire. La dynamique de ce système est décrite par le modèle masse-ressort-amortisseur. Nous avons calculé, à partir des résultats de chaque essai, les valeurs des paramètres du modèle masse-ressort-amortisseur. Nous avons montré que la valeur du paramètre d'amortissement augmente avec la vitesse d'écoulement, ce qui met en évidence un effet d'amortissement induit par l'écoulement.

Nous avons présenté le modèle TLP qui décrit la dynamique d'un cylindre sous écoulement axial. Ce modèle décrit la résultante des efforts exercés par le fluide sur un point du cylindre situé loin des extrémités. Le modèle TLP appliqué à notre maquette d'assemblage décrit les forces appliquées par le fluide sur la maquette comme la somme de deux contributions : une force de masse ajoutée et une force d'amortissement qui décrit le transfert d'énergie mécanique de la maquette au fluide. La force de masse ajoutée est décrite par le coefficient sans dimension C_{masse} et la force d'amortissement par le coefficient sans dimension C_{amort}.

Nous avons montré que le paramètre de raideur fluide est nul dans nos essais. L'absence de raideur fluide est en accord avec le modèle proposé par Divaret (2014). Le coefficient d'amortissement C_{amort} obtenu à partir de nos essais dépend légèrement de la vitesse d'écoulement. Ce résultat n'est pas en accord avec le modèle proposé par Divaret (2014). Nous avons émis l'hypothèse que cet écart provient de la présence des grilles.

Afin de compléter l'analyse des résultats d'essais, nous avons proposé un modèle numérique de la

veine d'essai fondé sur une discrétisation en hexaèdres du domaine occupé par le fluide. La réalisation de cette discrétisation s'appuie sur une simplification de la géométrie de l'objet d'étude : les détails les plus fins sont omis et les axes de forçage ne sont pas représentés. La simulation numérique calcule des champs décrivant l'écoulement turbulent du fluide autour de la structure mobile. La vitesse d'écoulement étudiée dans les simulations est $U = 0.9$ m/s.

Les efforts exercés par le fluide sur la maquette d'assemblage sont enregistrés à chaque instant de calcul. Les contributions du faisceau de cylindres et des grilles à la résultante de ces efforts sont séparées. L'analyse conjointe de la résultante des forces exercées sur le faisceau de cylindres et sur la maquette d'assemblage complète permet de montrer que le modèle proposé par Divaret (2014) décrit la force d'amortissement exercée sur le faisceau de cylindre et la force de masse ajoutée exercée sur la maquette d'assemblage complète. La valeur du coefficient d'amortissement obtenue à partir des efforts exercés sur le faisceau de cylindres est $C_{\text{amort}} = 0.18 \pm 0.01$; cette valeur est égale à la valeur obtenue par Divaret (2014). La valeur du coefficient de masse ajoutée obtenue à partir des efforts exercés sur la maquette d'assemblage complète est $C_{\text{masse}} = 1.45 \pm 0.015$; cette valeur correspond à la valeur obtenue par Divaret (2014).

Nous avons également analysé la répartition des efforts exercés par le fluide sur chacun des cylindres du faisceau. Nous avons observé que dans une rangée de cylindres alignée avec la direction orthogonale à la direction de sollicitation, les efforts sont répartis de façon homogène sur les cylindres de la rangée. Les efforts exercés sur une rangée sont symétriques par rapport à la rangée centrale. Plus la rangée est éloignée de la rangée centrale, plus les efforts exercés sur la rangée sont importants.

6.2 Perspectives

La suite immédiate de ce travail nous semble être la réalisation d'essais et de simulations numériques supplémentaires pour consolider les résultats obtenus et vérifier les hypothèses encore ouvertes. Les valeurs du paramètre de masse du modèle masse-ressort-amortisseur obtenus à partir de nos essais sont sous-estimées. La valeur du paramètre de masse obtenue à partir des essais en air est inférieure à la masse de la maquette d'assemblage. La valeur du paramètre de masse obtenue à partir des essais en eau stagnante est inférieure à la valeur du paramètre de masse ajouté prédit par un modèle d'écoulement potentiel. Réaliser des essais qui explorent une gamme de fréquences plus élevées devrait permettre d'obtenir des valeurs plus précises du paramètre de masse.

Nous avons commencé par étudier la vitesse d'écoulement $U = 0.9$ m/s, car elle correspond au régime dans lequel le niveau dans le réservoir supérieur reste stable. Nous avons choisi de ne pas étudier des vitesses d'écoulement inférieures à 0.9 m/s. En effet, la force à mesurer diminue avec la vitesse d'écoulement, de sorte que plus la vitesse d'écoulement est faible, plus l'intensité de la force est faible et moins la mesure de la force est précise. Notre étude des vitesses supérieures à 0.9 m/s s'est limitée à $U = 1$ m/s, car il n'est pas possible d'atteindre des vitesses supérieures à 1 m/s sans déclencher la vidange de la veine d'essai. Réaliser des essais en considérant des vitesses d'écoulement plus élevées devrait permettre d'identifier la façon dont le paramètre d'amortissement dépend de la vitesse d'écoulement. Toutefois, réaliser des essais à des vitesses plus élevées nécessite des modifications de la boucle hydraulique.

Nos simulations numériques nous ont permis d'extraire la résultante des efforts exercés par le fluide sur le faisceau de cylindres en parallèle de la résultante des efforts exercés par le fluide sur la maquette d'assemblage entière. La valeur du coefficient d'amortissement obtenue à partir de la résultante des efforts exercés par le fluide sur le faisceau de cylindres est égal à 0.18 ± 0.01 et ne dépend pas de la fréquence. En revanche, la valeur du coefficient d'amortissement obtenue à partir de la résultante des efforts exercés par le fluide sur la maquette d'assemblage entière semble dépendre du rapport entre la vitesse de sollicitation, définie comme le produit de la pulsation et de l'amplitude de sollicitation, et la vitesse d'écoulement : la valeur du coefficient passe de 0.17 à 0.24 quand la fréquence de sollicitation passe de 2 à 5 Hz, pour une valeur donnée du nombre de Reynolds. La différence entre ces deux résultantes d'effort est la contribution conjointe des extrémités des cylindres et des grilles, placées aux extrémités. La question qui se pose est la suivante. Quelle est la contribution des extrémités et quelle est la contribution des grilles au coefficient d'amortissement ?

Pour répondre à cette question, nous proposons tout d'abord d'explorer, expérimentalement et numériquement, d'autres vitesses d'écoulement. Les essais seront réalisés à des vitesses supérieures, après modification de la boucle hydraulique. Les simulations seront réalisées aux mêmes vitesses, en utilisant le

modèle de turbulence LES (*Large Eddy Simulation*) qui constitue un intermédiaire entre la modélisation de la turbulence proposée par le modèle $k-\omega$ SST et la résolution directe des équations de Navier-Stokes. Nous proposons également d'explorer expérimentalement et numériquement d'autres géométries : des cylindres de longueurs supérieures et des formes d'extrémités autres que des hémisphères. Le modèle TLP décrit un point du cylindre suffisamment loin des extrémités, car les extrémités du cylindre modifient localement les forces exercées par l'écoulement sur le cylindre. Utiliser des cylindres plus longs permet de garantir que la portion de la paroi du cylindre, sur laquelle les forces exercées par l'écoulement sont influencées par les extrémités, est négligeable par rapport à la surface de la paroi latérale du cylindre. Faire varier la forme des extrémités doit permettre d'identifier la contribution des extrémités au coefficient d'amortissement. Joly *et al.* (2021) a montré qu'une modification mineure des extrémités peut avoir une influence majeure sur la valeur et l'indépendance vis-à-vis des paramètres d'un coefficient du modèle TLP.

Dans nos simulations, le profil de vitesse à l'entrée du domaine occupé par le fluide est uniforme. Or, dans les essais, le profil de vitesse à l'entrée de la veine n'est pas uniforme car l'écoulement a traversé un nid d'abeille installé dans la veine haute, à la sortie du réservoir supérieur. Nous avons émis l'hypothèse que la différence entre ces deux profils de vitesse explique que l'effet d'amortissement de l'écoulement simulé est différent de l'effet d'amortissement de l'écoulement réel. Pour vérifier cette hypothèse, il nous semblerait pertinent de mesurer le profil de vitesse, dans un plan orthogonal à la direction d'écoulement, en aval de la maquette d'assemblage afin de l'appliquer comme condition d'entrée dans de nouvelles simulations numériques. Pour réaliser cette mesure de vitesse, nous proposons de mettre en œuvre la technique LDV (*Laser Doppler Velocimetry*).

Nous avons équipé de capteurs de pression le hublot de la veine d'essai. L'analyse des pressions mesurées nous a convaincu que le signal de pression contient une information sur l'amortissement induit par l'écoulement. En revanche, cette analyse ne nous a pas permis d'obtenir des résultats quantitatifs. L'étape suivante serait de réaliser également des mesures de pression à la paroi des cylindres de la maquette d'assemblage. L'analyse du profil de pression à la paroi des cylindres permettrait d'accéder expérimentalement à la répartition des efforts exercés par le fluide sur les cylindres de la maquette d'assemblage. Instrumenter les cylindres pour réaliser des mesures de pression à la paroi nécessite de percer les cylindres et de placer un capillaire reliant chaque perçage à un capteur de pression. Le diamètre intérieur des cylindres de la maquette nous semble trop petit pour intégrer ces capillaires sans les pincer, ce qui altérerait la mesure. Il faudrait utiliser des cylindres d'un diamètre intérieur supérieur pour réaliser une intégration des capillaires qui n'altère pas la mesure. Cependant, utiliser des cylindres d'un diamètre intérieur supérieur nécessite une révision majeure de la conception de la maquette d'assemblage et du banc d'essai.

La conception du banc d'essai a constitué le cœur de ce travail de thèse. L'objectif de se doter d'un système de forçage maîtrisé, qui impose avec une grande précision un signal de déplacement harmonique à une structure sous écoulement axial, a été atteint. La réalisation de ce banc d'essai démontre la faisabilité d'un banc d'essai conçu pour étudier l'effet d'amortissement d'un écoulement axial sur un assemblage combustible réel soumis à une excitation harmonique. Les règles de conception utilisées pour ce banc d'essai pourrait être appliquée à l'adaptation d'un moyen d'essai existant, dans lequel un assemblage combustible réel est soumis à un écoulement axial dont la vitesse correspond à la vitesse nominale en cœur (environ 5 m/s). Un tel banc d'essai fournirait des mesures de références aux acteurs de l'industrie nucléaire.

Annexe A

Résultats expérimentaux complémentaires

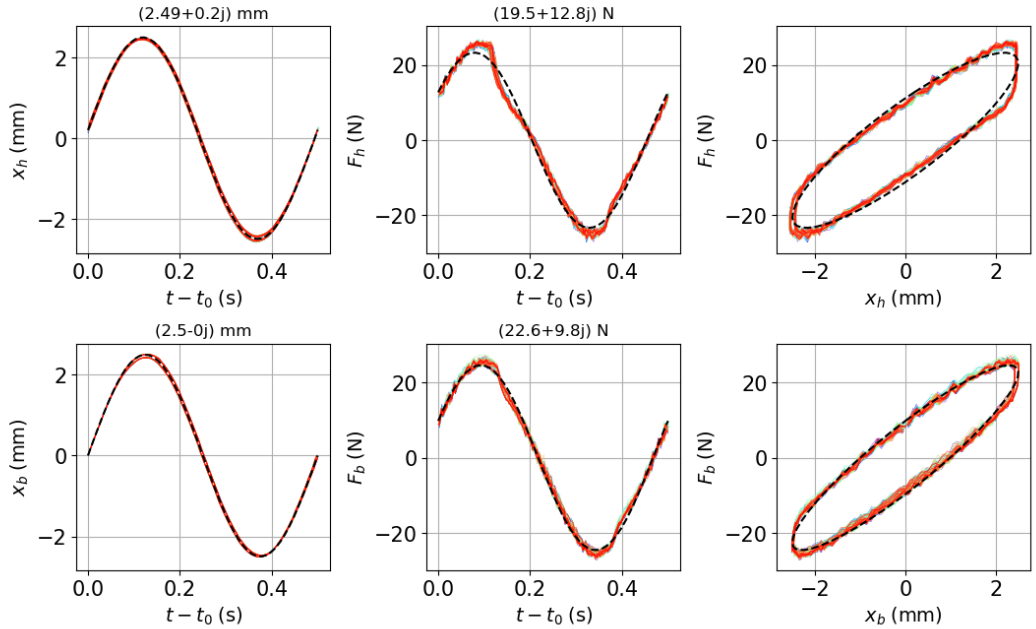


FIGURE A.1 – Signaux obtenus en air à 2Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

Les figures A.1 à A.12 illustrent les résultats obtenus, dans les différentes configurations (en air, en eau stagnante et pour les vitesses d'écoulement $U = 0.9$ et 1 m/s), pour les fréquences de sollicitations $f_0 = 2, 3$ et 4 Hz. La structure de chaque figure est identique à celle présentée en section 4.3.

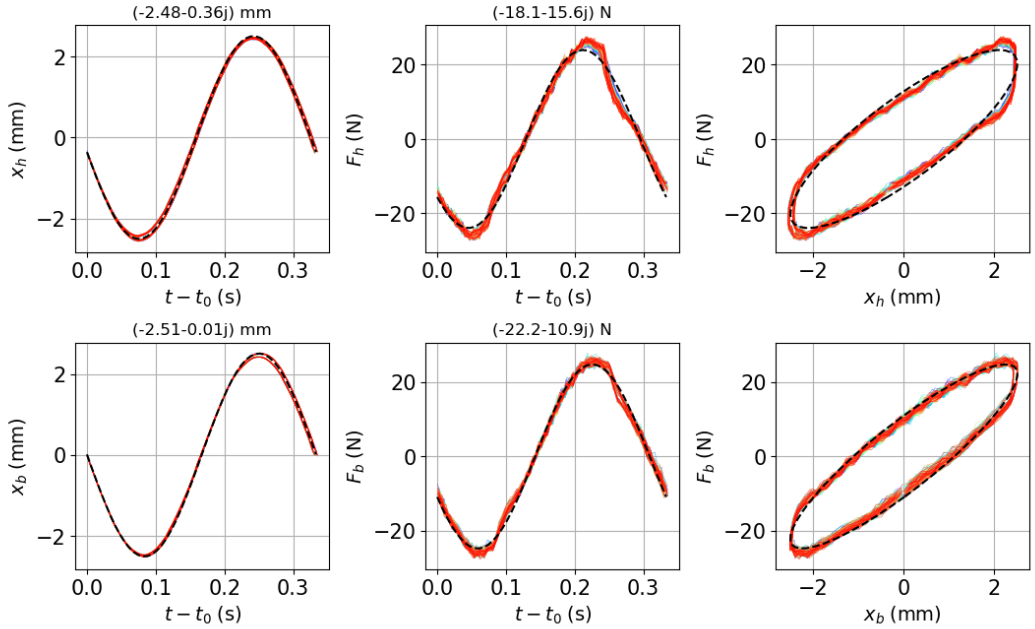


FIGURE A.2 – Signaux obtenus en air à 3Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

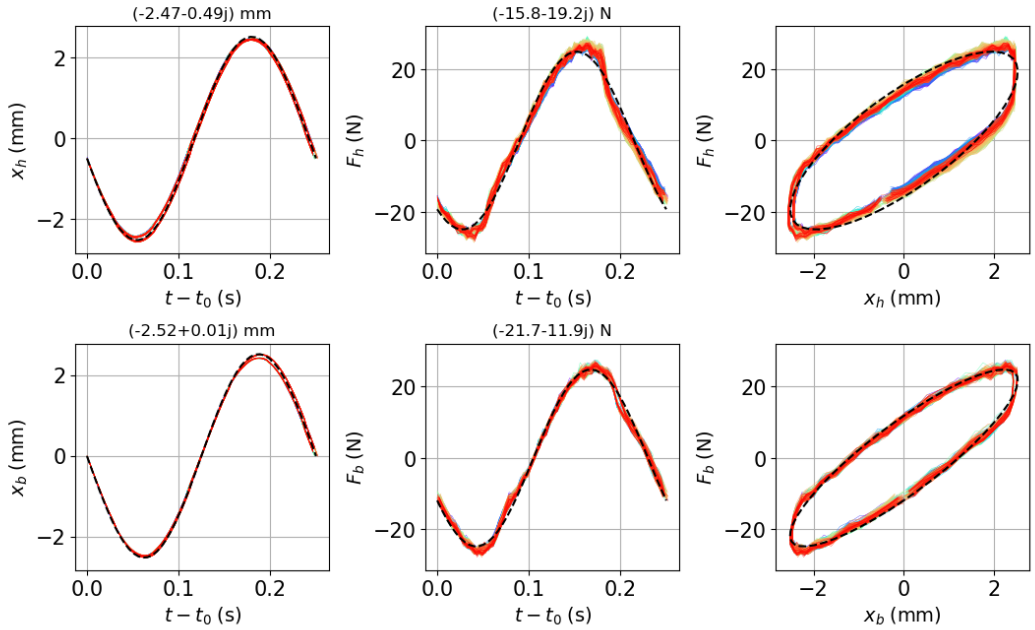


FIGURE A.3 – Signaux obtenus en air à 4Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

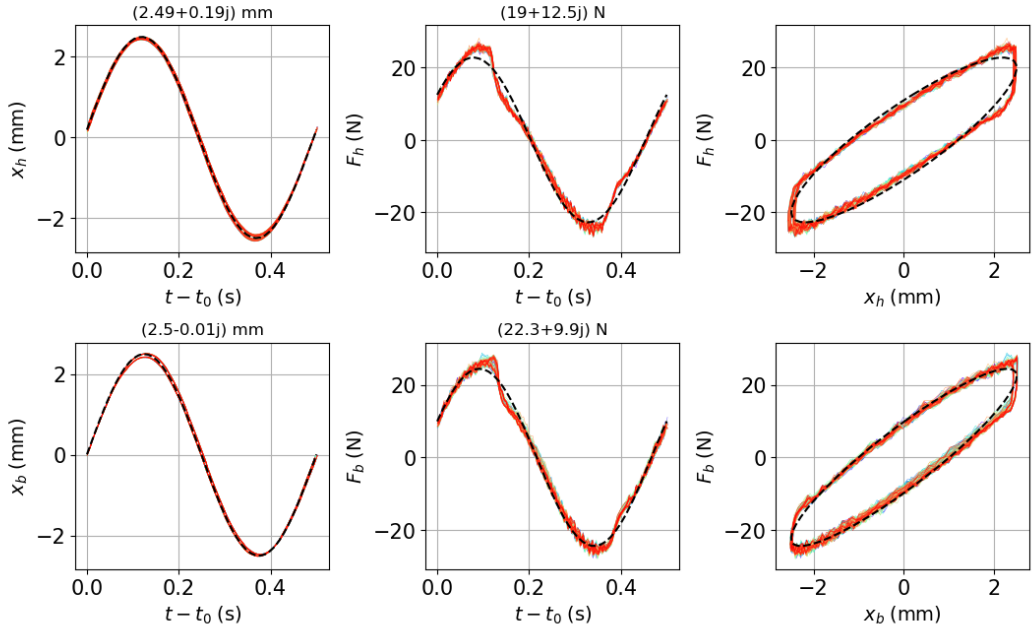


FIGURE A.4 – Signaux obtenus en eau stagnante à 2Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

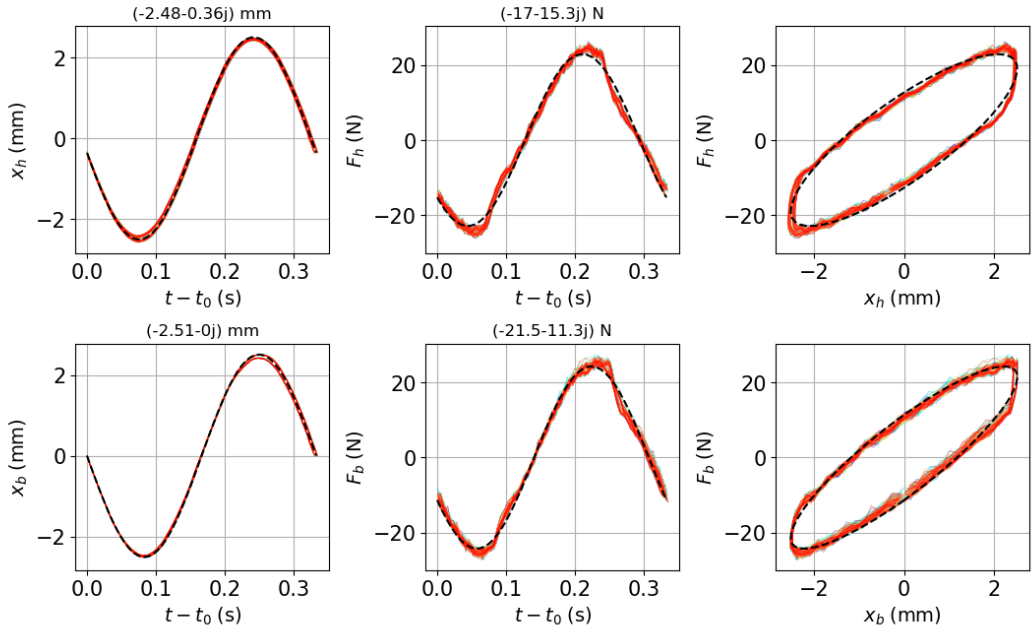


FIGURE A.5 – Signaux obtenus en eau stagnante à 3Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

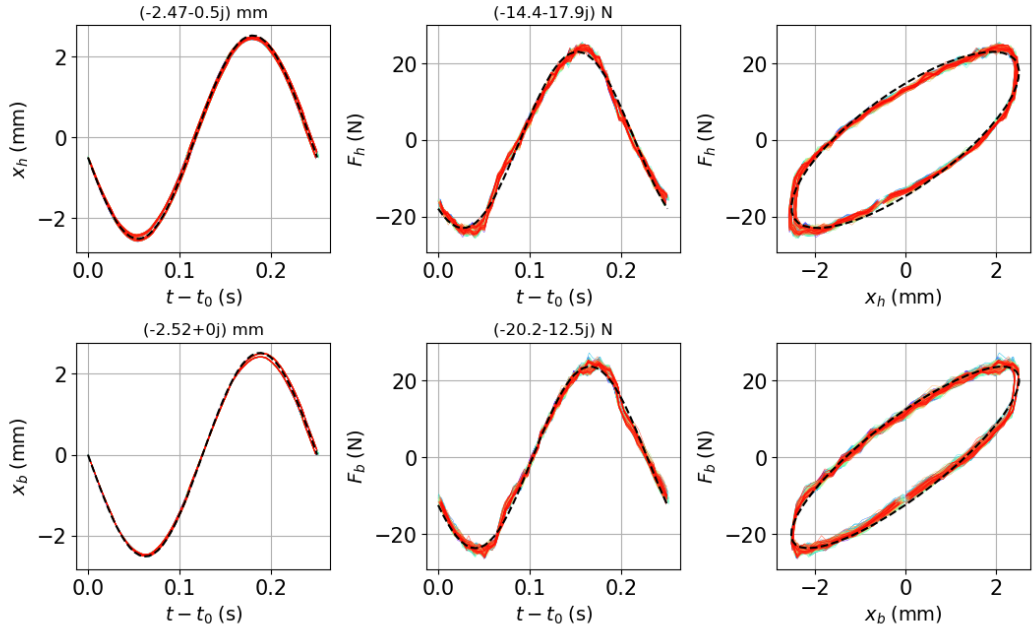


FIGURE A.6 – Signaux obtenus en eau stagnante à 4Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

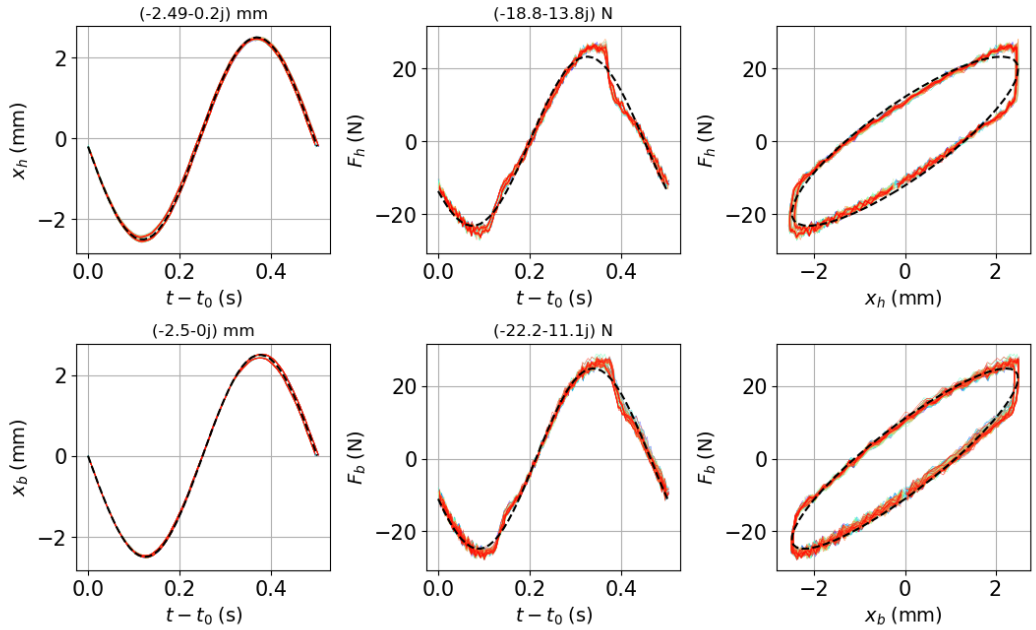


FIGURE A.7 – Signaux obtenus en écoulement à $U = 0.9$ m/s à 2Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

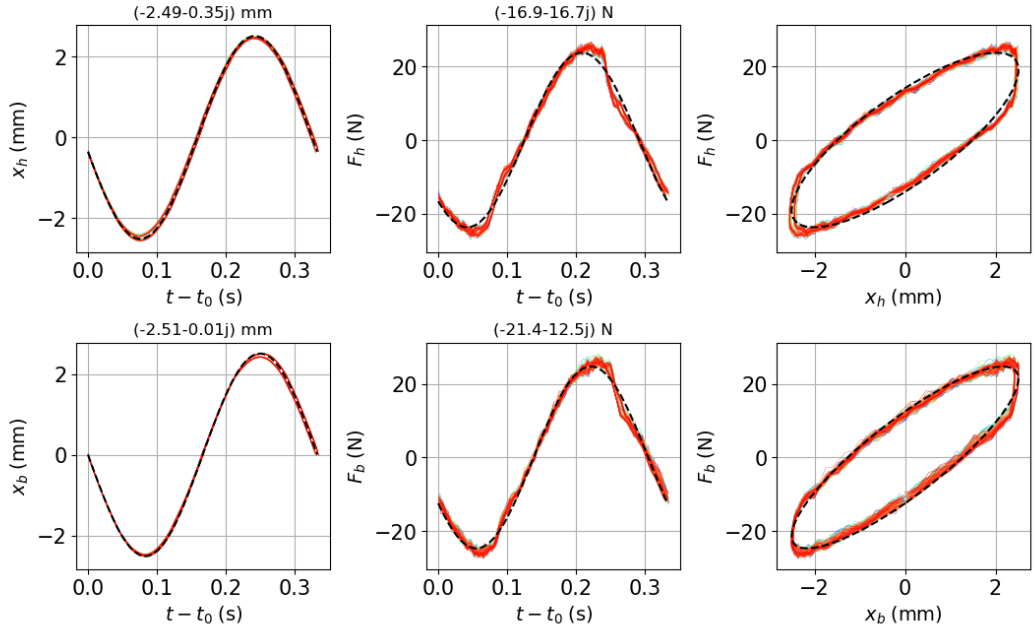


FIGURE A.8 – Signaux obtenus en écoulement à $U = 0.9$ m/s à 3Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

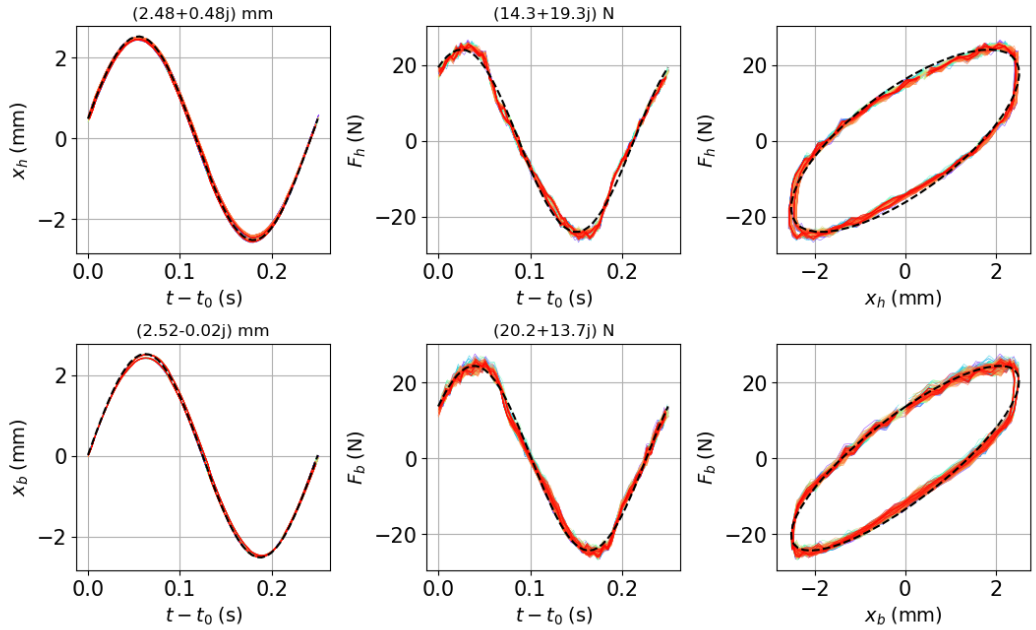


FIGURE A.9 – Signaux obtenus en écoulement à $U = 0.9$ m/s à 4Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

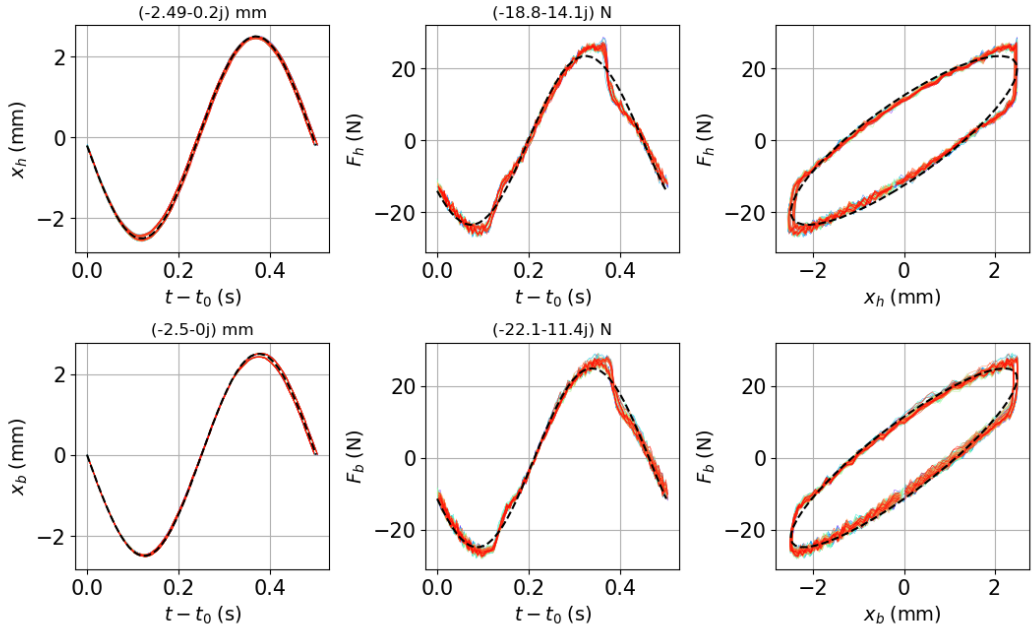


FIGURE A.10 – Signaux obtenus en écoulement à $U = 1.0$ m/s à 2Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

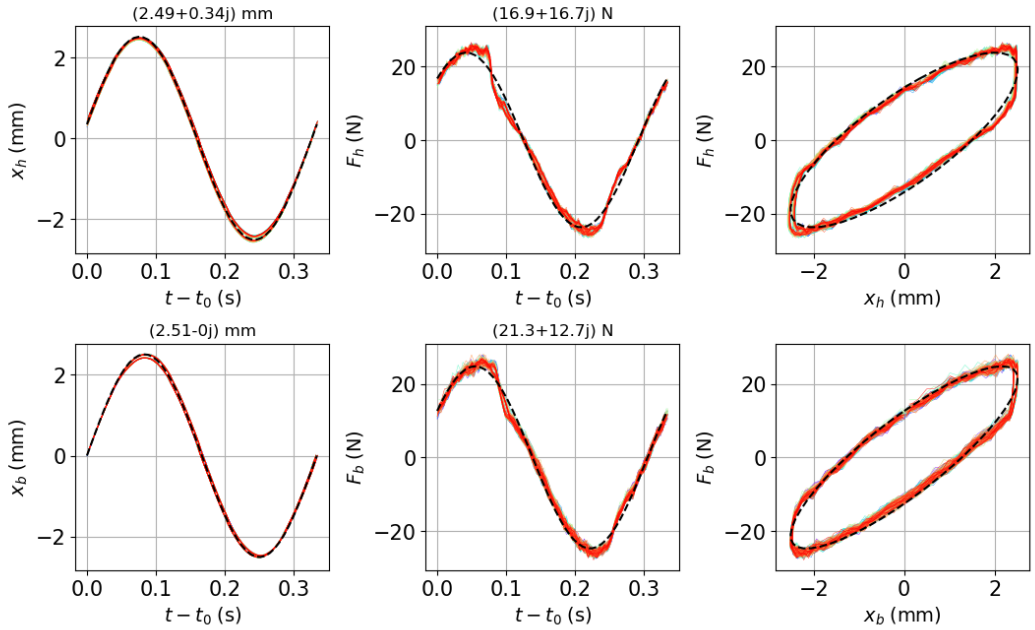


FIGURE A.11 – Signaux obtenus en écoulement à $U = 1.0$ m/s à 3Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

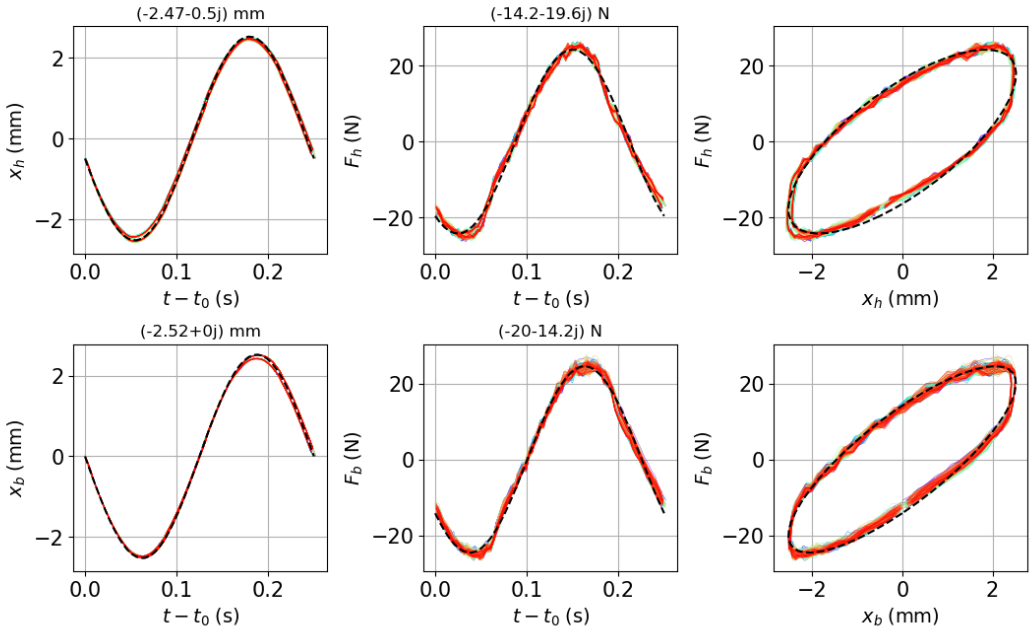


FIGURE A.12 – Signaux obtenus en écoulement à $U = 1.0$ m/s à 4Hz. En haut : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du haut. En bas : signal de déplacement (gauche), signal de force (milieu) et courbe de Lissajous (droite), pour la grille du bas.

Annexe B

Scripts de post-traitement des essais

B.1 Script de calcul du fondamental d'un signal

Convertisseur.py

— Pré-conditions : Pour chaque *frek* dans {1, 2, 3, 4, 5}, les fichiers suivants existent dans le répertoire courant : "X_et_F_*frek*Hz_U_1_m_s.npy", "X_et_F_*frek*Hz_U_0p8_m_s.npy", "X_et_F_*frek*Hz_sec.npy" et "X_et_F_*frek*Hz_stagnant.npy". Chaque fichier "*fichier*.npy" est obtenu en appliquant la méthode "numpy.save" au fichier texte "*fichier*.txt".
— Exécution : Exécuter la commande "python3 Convertisseur.py" dans un terminal.
— Post-conditions : Un fichier PNG "*fichier*.png" a été écrit à partir de chaque fichier "*fichier*.npy".

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# La ligne qui precede est indispensable pour que Python ne s'etouffe pas
# en lisant un caractere accentue, meme dans un commentaire.


# &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
# &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
#
#           LECTURE ET PRE-TRAITEMENT DES ESSAIS ELLEZ HARMONIQUES
#
# &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
# &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&


# Appel aux librairies de calcul scientifique
import numpy as np
import os as nonosse
import matplotlib.pyplot as plt
import matplotlib.cm as centimetre

# Reglage des axes
##plt.rcParams['axes.labelsize'] = 20
##plt.rcParams['axes.titlesize'] = 20
##plt.rcParams['xtick.labelsize'] = 20
##plt.rcParams['ytick.labelsize'] = 20
glob_police_1 = 12
glob_nb_grad_x = 4
glob_nb_grad_y = 4




# ***********************************************************************
def arbo_donnees() :
# ***********************************************************************
    # A changer selon l'ordinateur utilise !
    return nonosse.getcwd()
```

```
# ************************************************************************
def temps_zero(temps = None, signal = None, seuil = 0.25) :
    """Estime les instants de passage par zero du signal"""
# ************************************************************************
    # Seuil pour eliminer les passages par zero dus au bruit
    seuil_absolu = seuil * np.std(signal)

    # Extraction des instants ou abs(signal) depasse le seuil
    indices, = np.where(abs(signal) > seuil_absolu) # Un tuple !
    temps_red = temps[indices]
    signal_red = signal[indices]

    # Creation de deux signaux decales d'un instant
    temps_1 = temps_red[:-1]
    temps_2 = temps_red[1:]
    signal_1 = signal_red[:-1]
    signal_2 = signal_red[1:]

    # Estimation affine des instants ou le signal change de signe
    indices_zeros, = np.where((signal_1 * signal_2) < 0) # Tuple !
    t1 = temps_1[indices_zeros]
    t2 = temps_2[indices_zeros]
    s1 = signal_1[indices_zeros]
    s2 = signal_2[indices_zeros]
    pente = (t2 - t1)/(s2 - s1)

    return t1 - pente*s1



# ************************************************************************
class Donnees :
    """Stockage d'un essai complet"""
# ************************************************************************
    # ------------------------------------------------------------------
    def __init__(self, arbo = arbo_donnees(), fichier =  "20210618
    _force_dep_pression_fe256_5Hz_Q66.npy") :
        """ Lecture des donnees brutes """
    # ------------------------------------------------------------------
        # Colonnes des mesures : t, Q, f2, x2, f1, x1

        # ......... Nom et arborescence du fichier de donnees ..........
        self.arborescence = arbo
        self.fichier = fichier

        #  ................ Lecture du fichier ascii .................
        nom_complet = nonosse.path.join(self.arborescence,self.fichier)
        donnees = np.load(nom_complet)
        # donnees = np.loadtxt(nom_complet, dtype = float, delimiter = ',')

        self.temps = donnees[:,0]           # s
        self.freq_echantillonnage = (len(self.temps) - 1) / (self.temps[-1] - self.temps
    [0])

        coeff_debit = 72.83                     # Sensibilite du debitmetre
        offset_debit = -62.534
        self.debit = coeff_debit*donnees[:,1] + offset_debit              #debit en m3/h
    ,
        surface_fluide = 0.15*0.15 - 25*(np.pi*0.25*0.01**2)     # surface fluide dans
    une section du faisceau
        self.U = np.mean(self.debit) / surface_fluide / 3600.

        # On elimine d'office les offsets pour les forces et deplacements
        self.force_haut = donnees[:, 2]  - np.mean(donnees[:, 2])            # N
        self.force_bas = donnees[:, 4] - np.mean(donnees[:, 4])              # N
```

```
        self.deplacement_haut = donnees[:,3] - np.mean(donnees[:, 3])       # m
        self.deplacement_bas = donnees[:,5]  - np.mean(donnees[:, 5])      # m

        # ..................... Messages de lecture .....................
        print (u"Les donnees de l'essai " + self.fichier + u" ont ete lues.")
        print(u"La vitesse d'ecoulement est de " + str(round(self.U, 2)) + " m/s.")



# ------------------------------------------------------------------
def decoupe_en_cycles(self, seuil = 0.05, t_deb = 5, t_fin = 50, nb_harmoniques = 3)
 :
    """ Extrait une plage temporelle, cree une phase et decoupe le signal en cycles
"""
# ------------------------------------------------------------------
    # Indices de debut et de fin de plage temporelle
    indices, = np.where((self.temps >= t_deb) & (self.temps <= t_fin))
    self.temps_extrait = self.temps[indices]

    # Voie de reference et phase associee sur les instants extraits
    ref = self.deplacement_bas[indices]
    ref = ref - np.mean(ref)
    t_zeros = temps_zero(temps = self.temps_extrait, signal = ref, seuil = 0.25)

    # On repere les indices correspondant a des demi cycles et on en conserve
    # un sur deux. Attention a bien travailler sur les indices du signal entier
    # et non du signal extrait
    indy = list(np.searchsorted(self.temps, t_zeros))
    ind_deb, ind_fin = indy[:-1:2], indy[2: : 2]
    self.intervalles = [range(ideb, ifin) for ideb, ifin in zip(ind_deb, ind_fin)]
    t0_debut = t_zeros[0:-2:2]
    t0_fin = t_zeros[2::2]

    # Fabrication de phases synchronisees selon xref ~ cos phi
    self.phases = [np.interp(self.temps[inter], [td, tf], [-np.pi/2., 3./2.*np.pi])
                               for inter, td, tf in zip(self.intervalles, t0_debut,
t0_fin)]
    self.temps_red = [self.temps[inter] - td for inter, td in zip(self.intervalles,
t0_debut)]

    duree_totale = self.temps[self.intervalles[-1][-1] - self.intervalles[0][0]]
    self.frequence = len(self.intervalles) / duree_totale

     #  ......................... Message de lecture ..................
    print ("L'extrait de l'essai " + self.fichier + " est a " + str(np.round(self.
frequence, 1)) + " Hz. Il comporte " + str(len(self.intervalles)) + " cycles
successifs complets.")
    print

    # --- Calcul des fondamentaux des signaux par projection, intervalle par
intervalle
    # Principe du calcul harmonique : un signal de frequence harmonique w s'ecrit
    #           s(t) = Sr cos wt   -   Sim sin wt
    #
    # ce qui resulte de la definition de l'amplitude complexe Sr + i Sim :
    #           s(t) = Re { (Sr + i Sim) exp iwt   }
    #
    # Pour trouver l'amplitude harmonique d'un signal par projection, il faut ecrire
 sur une periode T :
    #           Sr + i Sim = 2/T integrale( exp -iwt  s(t) dt)
    coprods = [np.exp(-1j*phase) * 2./(len(phase) - 1) for phase in self.phases]
    har_Xb = [np.dot(coprod, self.deplacement_bas[interv]) for coprod, interv in zip
(coprods, self.intervalles)]
    har_Xh = [np.dot(coprod, self.deplacement_haut[interv]) for coprod, interv in
zip(coprods, self.intervalles)]
    har_Fb = [np.dot(coprod, self.force_bas[interv]) for coprod, interv in zip(
coprods, self.intervalles)]
    har_Fh = [np.dot(coprod, self.force_haut[interv]) for coprod, interv in zip(
```

```
    coprods, self.intervalles)]

        self.Xb_m = np.mean(np.array(har_Xb))
        self.Xh_m = np.mean(np.array(har_Xh))
        self.Fb_m = np.mean(np.array(har_Fb))
        self.Fh_m = np.mean(np.array(har_Fh))




# ------------------------------------------------------------------
def visu_repetabilite(fichier = "X_et_F_5Hz_U_1_m_s.npy", t_deb = 1, t_fin = 360,
    nb_harmoniques = 1, save_fig = False) :
# ------------------------------------------------------------------
    # Ouverture du fichier d'essai
    essai = Donnees(fichier = fichier)

    # Decoupe en cycles
    essai.decoupe_en_cycles(seuil = 0.07, t_deb = t_deb, t_fin = t_fin, nb_harmoniques =
     nb_harmoniques)

    # Preparation des graphes
    cm_per_inch = 2.54
    fifi = plt.figure(figsize = (28./cm_per_inch, 18./cm_per_inch))
    fifi.subplots_adjust(left = 0.1, right = 0.98, top = 0.90, bottom = 0.12, wspace =
    0.38, hspace = 0.4)

    graf_Liss_h = fifi.add_subplot(231)
    graf_Liss_b = fifi.add_subplot(234, sharex = graf_Liss_h, sharey = graf_Liss_h)
    graf_Liss_h.set_xlabel(r"$ x_h \mathrm{\; (mm)} $", rotation = 'horizontal', size =
    glob_police_1)
    graf_Liss_b.set_xlabel(r"$ x_b \mathrm{\; (mm)} $", rotation = 'horizontal', size =
    glob_police_1)
    graf_Liss_h.set_ylabel(r"$ F_h \; \mathrm{(N)} $", rotation = 'vertical', size =
    glob_police_1)
    graf_Liss_b.set_ylabel(r"$ F_b \; \mathrm{(N)} $", rotation = 'vertical', size =
    glob_police_1)

    graf_X_h = fifi.add_subplot(232)
    graf_X_b = fifi.add_subplot(235, sharex = graf_X_h, sharey = graf_X_h)
    for graf in graf_X_h, graf_X_b :
        graf.set_xlabel(r"$ t - t_0 \; \mathrm{(s)} $", rotation = 'horizontal', size =
    glob_police_1)
        graf.locator_params(axis = 'x', nbins = glob_nb_grad_x)
        graf.locator_params(axis = 'y', nbins = glob_nb_grad_y)
    graf_X_h.set_ylabel(r"$ x_h \mathrm{\; (mm)} $", rotation = 'vertical', size =
    glob_police_1)
    graf_X_b.set_ylabel(r"$ x_b \mathrm{\; (mm)} $", rotation = 'vertical', size =
    glob_police_1)

    graf_F_h = fifi.add_subplot(233)
    graf_F_b = fifi.add_subplot(236, sharex = graf_F_h, sharey = graf_F_h)
    for graf in graf_F_h, graf_F_b :
        graf.set_xlabel(r"$ t - t_0 \; \mathrm{(s)} $", rotation = 'horizontal', size =
    glob_police_1)
        graf.locator_params(axis = 'x', nbins = glob_nb_grad_x)
        graf.locator_params(axis = 'y', nbins = glob_nb_grad_y)
    graf_F_h.set_ylabel(r"$ F_h \mathrm{\; (N)} $", rotation = 'vertical', size =
    glob_police_1)
    graf_F_b.set_ylabel(r"$ F_b \mathrm{\; (N)} $", rotation = 'vertical', size =
    glob_police_1)

    couleurs = centimetre.rainbow(np.linspace(0, 1, len(essai.intervalles)))
    for couleur, indy, temps_red  in zip(couleurs, essai.intervalles, essai.temps_red) :
        graf_Liss_h.plot(essai.deplacement_haut[indy], essai.force_haut[indy], '-',
    color = couleur, linewidth = 0.3)
        graf_Liss_b.plot(essai.deplacement_bas[indy], essai.force_bas[indy], '-', color
    = couleur, linewidth = 0.3)
```

```
        graf_X_b.plot(temps_red, essai.deplacement_bas[indy],  '-', color = couleur,
    linewidth = 0.3)
        graf_X_h.plot(temps_red, essai.deplacement_haut[indy],  '-', color = couleur,
    linewidth = 0.3)
        graf_F_h.plot(temps_red, essai.force_haut[indy],  '-', color = couleur,
    linewidth = 0.3)
        graf_F_b.plot(temps_red, essai.force_bas[indy],  '-', color = couleur, linewidth
     = 0.3)

    fifi.suptitle(fichier[:-4])
    Xbm, Xhm, Fbm, Fhm = essai.Xb_m, essai.Xh_m, essai.Fb_m, essai.Fh_m

    # Pour verifier les amplitudes, on trace les fondamentaux resynthetises
    phase_synthetique = np.linspace(-np.pi/2., -np.pi/2. + 2.*np.pi, 50)
    temps_synthetique = np.linspace(0., 1./essai.frequence, 50)
    cosi = np.cos(phase_synthetique)
    sinu = np.sin(phase_synthetique)

    Xb_s = np.real(Xbm) * cosi + np.imag(Xbm) * sinu
    Xh_s = np.real(Xhm) * cosi - np.imag(Xhm) * sinu
    Fb_s = np.real(Fbm) * cosi - np.imag(Fbm) * sinu
    Fh_s = np.real(Fhm) * cosi - np.imag(Fhm) * sinu

    graf_X_b.plot(temps_synthetique, Xb_s,  '-', color = 'green', linewidth = 1.5)
    graf_X_h.plot(temps_synthetique, Xh_s,  '-', color = 'green', linewidth = 1.5)
    graf_F_b.plot(temps_synthetique, Fb_s,  '-', color = 'green', linewidth = 1.5)
    graf_F_h.plot(temps_synthetique, Fh_s,  '-', color = 'green', linewidth = 1.5)

    graf_Liss_h.plot(Xh_s, Fh_s, '-', color = 'green', linewidth = 1.5)
    graf_Liss_b.plot(Xb_s, Fb_s, '-', color = 'green', linewidth = 1.5)


    graf_X_b.set_title(str(np.round(Xbm, 2)) + " mm", fontsize = 9)
    graf_X_h.set_title(str(np.round(Xhm, 2)) + " mm", fontsize = 9)
    graf_F_b.set_title(str(np.round(Fbm, 1)) + " N", fontsize = 9)
    graf_F_h.set_title(str(np.round(Fhm, 1)) + " N", fontsize = 9)

    if save_fig:
        fifi.savefig(fichier[0:-4]+'.png')
        plt.close('all')




# ************************************************************************
#               Programme principal
# ************************************************************************

if True:
    for frek in [1,2,3,4,5]:
        visu_repetabilite(fichier = "X_et_F_" + str(frek) + "Hz_U_1_m_s.npy", t_deb = 5,
     t_fin = 60)
    for frek in [1,2,3,4,5]:
        visu_repetabilite(fichier = "X_et_F_" + str(frek) + "Hz_U_0p8_m_s.npy", t_deb =
    5, t_fin = 60)
    for frek in [1,2,3,4,5]:
        visu_repetabilite(fichier = "X_et_F_" + str(frek) + "Hz_sec.npy", t_deb = 5,
    t_fin = 60)
    for frek in [1,2,3,4,5]:
        visu_repetabilite(fichier = "X_et_F_" + str(frek) + "Hz_stagnant.npy", t_deb =
    5, t_fin = 60)


##for frek in [1, 2, 3, 4, 5]:
##    visu_repetabilite(fichier = "X_et_F_" + str(frek) + "Hz_U_0p8_m_s.npy", t_deb =
    0., t_fin = 165)
```

```
plt.show()
```

B.2 Script de calcul des coefficients ajoutés

Traitement.py

— Pré-conditions : Aucune, les données utilisées sont les nombres complexes calculés par l'exécution du script "Convertisseur.py". Un nombre complexe est associé à chaque signal, pour chaque essai.
— Exécution : Exécuter la commande "python3 Traitement.py" dans un terminal.
— Post-conditions : Trois graphes sont affichés :

1. Les courbes représentatives de Im(H) en fonction de ω, pour chaque essai (*cf.* FIGURE 4.9).
2. Les courbes représentatives de Re(H) en fonction de ω^2, pour chaque essai (*cf.* FIGURE 4.10).
3. Le diagramme d'Argand (*cf.* FIGURE 4.11).

```
# -*- coding: utf-8 -*-
# La ligne qui precede est indispensable pour que Python ne s'etouffe pas
# en lisant un caractere accentue, meme dans un commentaire.


# &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
# &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
#
#           TRAITEMENT DES ESSAIS ELLEZ HARMONIQUES
#
# &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
# &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&


# Appel aux librairies de calcul scientifique
import numpy as np
import os as nonosse
import matplotlib.pyplot as plt
import matplotlib.cm as centimetre
from collections import OrderedDict

# Reglage des axes
plt.rcParams['axes.labelsize'] = 20
plt.rcParams['axes.titlesize'] = 20
plt.rcParams['xtick.labelsize'] = 15
plt.rcParams['ytick.labelsize'] = 15
glob_police_1 = 12
glob_nb_grad_x = 4
glob_nb_grad_y = 4

def k_ressort_haut() :
# **********************************************************************
    return (9.7+7.8)/2 # N/mm
# **********************************************************************
def k_ressort_bas() :
# **********************************************************************
    return k_ressort_bas()+k_ressort_haut() # N/mm

def k_tot() :
# **********************************************************************
    return (10.7+8.2)/2 # N/mm

def freqs() :   # Frequences associees aux essais (Hz)
    return np.array([1., 2., 3., 4., 5. ])

def U() :       # Vitesses d'ecoulement en section courante des cylindres (m/s)
    return [None, 0., 0.8, 1.2]

def Xh() :  #  Deplacement harmonique de la grille superieure en mm : lignes = U
    croissant, colonne = f croissante
```

```
    liste = [
                [(2.4850+0.0472j), (2.4893+0.1880j), (-2.4826+-0.3422j), (-2.4676+-0.4944
    j), (-2.4409+-0.6549j) ],
                [ (2.4850+0.0480j), (2.4857+0.2055j), (-2.4822+-0.3479j),
    (-2.4674+-0.4991j), (2.4446+0.6356j) ],
                [(2.4900+0.0452j), (-2.4922+-0.2043j), (-2.4866+-0.3489j), (2.4711+0.5077
    j), (2.4503+0.6400j) ],
                [(-2.4893+-0.0479j), (-2.4914+-0.2030j), (2.4882+0.3296j),
    (-2.4664+-0.5235j), (2.4495+0.6389j) ] ]
    return np.array(liste)

def Xb() :
    liste = [
                [ (2.4872+-0.0028j), (2.4983+-0.0120j), (-2.5083+0.0074j),
    (-2.5192+0.0043j), (-2.5293+-0.0057j) ],
                [  (2.4883+-0.0038j), (2.4987+0.0028j), (-2.5087+0.0048j),
    (-2.5196+0.0030j), (2.5300+-0.0090j) ],
                [ (2.4872+-0.0022j), (-2.4985+-0.0072j), (-2.5084+-0.0013j),
    (2.5201+0.0106j), (2.5298+-0.0073j) ],
                [ (-2.4878+-0.0029j), (-2.4980+-0.0055j), (2.5080+-0.0187j),
    (-2.5189+-0.0266j), (2.5296+-0.0082j) ] ]
    return np.array(liste)

def Fh() :
    liste = [
             [ (19.634+11.943j), (19.556+12.674j), (-18.256+-15.493j), (-15.798+-19.249j
    ), (-12.447+-22.791j) ],
             [ (19.714+10.449j), (18.935+12.584j), (-17.088+-15.235j), (-14.388+-17.937j
    ), (10.651+20.622j) ],
             [ (19.094+12.283j), (-18.769+-13.800j), (-16.934+-16.628j), (14.095+19.486j
    ), (10.664+21.654j)],
             [(-18.967+-12.778j), (-18.769+-14.075j), (16.996+16.624j), (-13.994+-19.766
    j), (10.709+21.874j) ] ]
    return np.array(liste)

def Fb() :
    liste = [ [(21.964+10.041j), (22.612+9.690j), (-22.321+-10.744j), (-21.703+-11.924j)
    , (-20.752+-12.942j) ],
              [ (22.207+9.568j), (22.278+10.084j), (-21.512+-11.265j), (-20.149+-12.467j
    ), (18.392+13.594j) ] ,
             [ (22.047+10.858j), (-22.208+-11.136j), (-21.387+-12.409j), (20.018+13.857j
    ), (18.266+15.195j) ],
             [ (-21.823+-11.159j), (-22.092+-11.418j), (21.399+12.527j),
    (-19.889+-14.392j), (18.264+15.463j) ] ]
    return np.array(liste)

def dev_Fh() :
    liste = [
             [ (0.28+0.18j), (0.26+0.22j), (0.38+0.17j), (0.49+0.73j), (0.41+0.39j) ],
             [ (0.28+0.19j), (0.26+0.21j), (0.39+0.11j), (0.44+0.18j), (0.24+0.40j) ],
             [ (0.27+0.22j), (0.34+0.10j), (0.38+0.11j), (0.18+0.29j), (0.18+0.31j)],
             [(0.33+0.27j), (0.34+0.12j), (0.24+0.23j), (0.41+0.20j), (0.20+0.36j) ] ]
    return np.array(liste)

def dev_Fb() :
    liste = [ [(0.35+0.22j), (0.30+0.25j), (0.35+0.31j), (0.40+0.25j), (0.36+0.28j) ],
              [ (0.33+0.27j), (0.33+0.27j), (0.37+0.29j), (0.37+0.30j), (0.38+0.30j) ] ,
             [ (0.35+0.32j), (0.35+0.36j), (0.34+0.31j), (0.39+0.30j), (0.36+0.33j) ],
             [ (0.38+0.44j), (0.38+0.38j), (0.35+0.30j), (0.38+0.32j), (0.40+0.31j) ] ]
    return np.array(liste)



Ftotale = Fh() + Fb()
Xmoy = (Xh() + Xb())/2.

FT_moy = Ftotale/Xmoy
ohm = (2. * np.pi * freqs())
om2 = (2. * np.pi * freqs())**2
```

```
fifi = plt.figure(figsize = (19, 10))
fifi.subplots_adjust(left = 0.18, right = 0.95, top = 0.95, bottom = 0.12, wspace =
    0.28, hspace = 0.25)

graf = fifi.add_subplot(211)
grif = fifi.add_subplot(212)
pente_cc=[]
pente_mm=[]
for raw, kolor, marque in zip(Ftotale/Xmoy, ['black', 'lightgrey', 'darkgrey', 'white'],
     ["*", "o", "D", ">", "<"] ):
    mm, kk = np.polyfit(om2[2:], np.real(1000*raw[2:]), 1)
    cc, c0 = np.polyfit(ohm[2:], np.imag(1000*raw[2:]), 1)
    pente_cc.append(cc)
    pente_mm.append(-mm)
    etiqu = "m = " + str(np.around(-mm, 2)) + " kg"
    graf.plot(om2, np.real(raw),  linestyle = "", markerfacecolor = kolor, marker =
    marque, label = etiqu)
    grif.plot(ohm, np.imag(raw), linestyle = "", markerfacecolor = kolor, marker =
    marque)

    # print ("m = ", -1000.*mm, " kg      k  = ", kk, "  N/mm")
    # print ("c = ", np.around(cc,2), "       c0  = ", np.around(c0,2), "  ")


graf.set_ylim([0, None])
graf.set_xlabel(r"$ \omega^2 \mathrm{\; (rad^2.s^{-2})} $", rotation = 'horizontal',
    size = 13)
graf.set_ylabel(r"$ \mathrm{Re} \left({ 2\frac{F_h + F_b}{X_h + X_b} }\right) \mathrm{\;
     (N.mm^{-1})} $", rotation = 'vertical', size = 13)
grif.set_xlabel(r"$ \omega \mathrm{\; (rad/s)} $", rotation = 'horizontal', size = 13)
grif.set_ylabel(r"$ \mathrm{Im} \left({ 2\frac{F_h + F_b}{X_h + X_b} }\right) \mathrm{\;
     (N.mm^{-1})} $", rotation = 'vertical', size = 13)

graf.legend(loc = 'lower left', fontsize = 9)

plt.show()

A=0.0025
rho=1000.0
U=0.9
D=0.01
L=1.3

u09=np.imag(FT_moy[2,:])
u0=np.imag(FT_moy[1,:])
CN=1*(u09-u0)/(0.5*rho*U*D*A*ohm*25*L)
print(np.around(CN,2))
# ki=/(rho*(25*L*np.pi*0.25*D**2)*A*om2)


pente_cc=np.array(pente_cc)

c_amort_U09=(pente_cc[2]-pente_cc[1])/(0.5*rho*0.9*D*A*ohm*25*L)
c_amort_U1=(pente_cc[3]-pente_cc[1])/(0.5*rho*1.04*D*A*ohm*25*L)

# c_masse_sec=
c_masse_u0=(pente_mm[1]-pente_mm[0])/(rho*(25*L*np.pi*0.25*D**2)*A*om2)

plt.figure(3,figsize=(19,10))
plt.plot(freqs(),c_amort_U09,color='k',marker='*',ls='--',ms=10)
plt.plot(freqs(),c_amort_U1,color='r',marker='o',ls='--',ms=10)
for x,y,z in zip(freqs(),c_amort_U09,c_amort_U1):
    label=str(np.around(y,2))
    plt.annotate(label, (x,y),textcoords="offset points",xytext=(-0,-30),ha='center',
    fontsize=20)
    label=str(np.around(z,2))
    plt.annotate(label, (x,z),textcoords="offset points",xytext=(25,25),ha='center',
    fontsize=20)
plt.xlabel('F (Hz)',size=20)
plt.ylabel('$C_{amort}$ (-)',size=20)
```

```
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.grid('on')
plt.ylim([0,2])
plt.legend(['U = 0.9 m/s','U = 1.0 m/s'],fontsize=20)
plt.savefig("coeff_amort.png")

graphe=plt.figure(figsize = (19, 10))
graf_masse=graphe.add_subplot(111)
# graf_masse.set_xlabel(r"$ \omega^2 \mathrm{\; (rad^2.s^{-2})} $", rotation = '
    horizontal', size = 20)
# graf_masse.set_ylabel(r"$ \mathrm{Re} \left({ 2\frac{F_h + F_b}{X_h + X_b} }\right) \
    mathrm{\; (N.mm^{-1})} $", rotation = 'vertical', size = 20)
graf_masse.set_xlabel(r"$ \omega^2 \mathrm{\; (rad^2.s^{-2})} $", rotation = 'horizontal
    ', size = 20)
graf_masse.set_ylabel(r"$ \mathrm{Re} \left({ H }\right) \mathrm{\; (N.mm^{-1})} $",
    rotation = 'vertical', size = 20)
vitesses=['Sec           ','Stagnant  ','U=0.9 m/s','U=1.0 m/s']
for raw, kolor, marque, vitesse in zip(Ftotale/Xmoy, ['black', 'blue', 'green', 'red'],
    ["*", "o", "D", ">", "<"], vitesses ):
    mm, kk = np.polyfit(om2[2:], np.real(raw[2:]), 1)
    etiqu = "M = " + str(np.around(-1000*mm, 2)) + " kg     |    " + "K = " + str(np.
    around(kk, 2)) + " N/mm"
    graf_masse.plot(om2, np.real(raw),  linestyle = "", color = kolor, marker = marque,
    label = vitesse+"   |   "+etiqu)
    graf_masse.plot(om2[1:], mm*om2[1:]+kk,  linestyle = "--", color = kolor)
    # print ("m = ", -1000.*mm, " kg     k  = ", kk, "  N/mm")
graf_masse.legend(loc = 'lower left', fontsize = 20)

# graphe=plt.figure(figsize = (19, 10))
# graf_masse=graphe.add_subplot(111)
# # graf_masse.set_xlabel(r"$ \omega^2 \mathrm{\; (rad^2.s^{-2})} $", rotation = '
    horizontal', size = 20)
# # graf_masse.set_ylabel(r"$ \mathrm{Re} \left({ 2\frac{F_h + F_b}{X_h + X_b} }\right)
    \mathrm{\; (N.mm^{-1})} $", rotation = 'vertical', size = 20)
# graf_masse.set_xlabel(r"$ \omega^2 \mathrm{\; (rad^2.s^{-2})} $", rotation = '
    horizontal', size = 20)
# graf_masse.set_ylabel(r"$ \mathrm{Re} \left({ H }\right) \mathrm{\; (N.mm^{-1})} $",
    rotation = 'vertical', size = 20)

# vitesses=['Sec           ','Stagnant  ','U=0.9 m/s','U=1.0 m/s']
# for raw, kolor, marque, vitesse in zip(Ftotale/Xmoy, ['black', 'blue', 'green', 'red
    '], ["*", "o", "D", ">", "<"], vitesses ):
#     mm, kk = np.polyfit(om2[2:], np.real(raw[2:]), 1)
#     etiqu = "m = " + str(np.around(-1000*mm, 2)) + " kg     |    " + "k = " + str(np.
    around(kk, 2)) + " N/mm"
#     graf_masse.plot(ohm, np.real(raw),  linestyle = "", color = kolor, marker = marque
    , label = vitesse+"   |   "+etiqu)
#     graf_masse.plot(ohm[1:], mm*ohm[1:]**2+kk,  linestyle = "--", color = kolor)
#     # print ("m = ", -1000.*mm, " kg     k  = ", kk, "  N/mm")
# graf_masse.legend(loc = 'lower left', fontsize = 20)

# graphe=plt.figure(figsize = (19, 10))
# graf_masse=graphe.add_subplot(111)
# graf_masse.set_xlabel(r"$ \omega^2 \mathrm{\; (rad^2.s^{-2})} $", rotation = '
    horizontal', size = 20)
# graf_masse.set_ylabel(r"$ \mathrm{Re} \left({ 2\frac{F_h + F_b}{X_h + X_b} }\right) \
    mathrm{\; (N.mm^{-1})} $", rotation = 'vertical', size = 20)
# vitesses=['Sec           ','Stagnant  ','U=0.9 m/s','U=1.0 m/s']
# for raw, kolor, marque, vitesse in zip(Ftotale/Xmoy, ['black', 'blue', 'green', 'red
    '], ["*", "o", "D", ">", "<"], vitesses ):
#     mm, kk = np.polyfit(np.log(ohm[2:]), np.log(abs(np.real(raw[2:])-17)), 1)
#     etiqu = "m = " + str(np.around(-1000*mm, 2)) + " kg     |    " + "k = " + str(np.
    around(kk, 2)) + " N/mm"
#     graf_masse.loglog(np.log(ohm), np.log(abs(np.real(raw))),  linestyle = "", color =
     kolor, marker = marque, label = vitesse+"   |   "+etiqu)
#     graf_masse.plot(np.log(ohm[1:]), np.log(mm*ohm[1:]+kk),  linestyle = "--", color =
     kolor)
#     print ("m = ", -1000.*mm, " kg     k  = ", kk, "  N/mm")
# graf_masse.legend(loc = 'lower left', fontsize = 20)
```

```
graphe2=plt.figure(figsize = (19, 10))
graf_amort=graphe2.add_subplot(111)
# graf_amort.set_xlabel(r"$ \omega \mathrm{\; (rad.s^{-1})} $", rotation = 'horizontal',
     size = 20)
# graf_amort.set_ylabel(r"$ \mathrm{Im} \left({ 2\frac{F_h + F_b}{X_h + X_b} }\right) \
    mathrm{\; (N.mm^{-1})} $", rotation = 'vertical', size = 20)
graf_amort.set_xlabel(r"$ \omega \mathrm{\; (rad.s^{-1})} $", rotation = 'horizontal',
    size = 20)
graf_amort.set_ylabel(r"$ \mathrm{Im} \left({ H }\right) \mathrm{\; (N.mm^{-1})} $",
    rotation = 'vertical', size = 20)
vitesses=['Sec           ','Stagnant  ','U=0.9 m/s','U=1.0 m/s']
for raw, kolor, marque, vitesse in zip(Ftotale/Xmoy, ['black', 'blue', 'green', 'red'],
    ["*", "o", "D", ">", "<"], vitesses ):
    cc, c0 = np.polyfit(ohm[2:], np.imag(raw[2:]), 1)
    etiqu = "C = " + str(np.around(1000*cc, 2)) + " $N.m^{-1}.s$"
    graf_amort.plot(ohm, np.imag(raw),  linestyle = "", color = kolor, marker = marque,
    label = vitesse+"   |   "+etiqu)
    graf_amort.plot(ohm[1:], cc*ohm[1:]+c0,  linestyle = "--", color = kolor)
    # print ("m = ", -1000.*mm, " kg     k  = ", kk, "  N/mm")
graf_amort.legend(loc = 'upper left', fontsize = 20)


argand=plt.figure(figsize = (19, 10))
graf_argand=argand.add_subplot(111)
# graf_argand.set_xlabel(r"$ Re \frac{F_X}{KX}  \mathrm{\; [-]} $", rotation = '
    horizontal', size = 20)
# graf_argand.set_ylabel(r"$ Im \frac{F_X}{KX}  \mathrm{\; [-]} $ ", rotation = '
    vertical', size = 20)
graf_argand.set_xlabel(r"$ \dfrac{Re \left(H \right)} {K_s} \mathrm{\; [-]} $", rotation
     = 'horizontal', size = 20)
graf_argand.set_ylabel(r"$ \dfrac{Im \left(H \right)} {K_s} \mathrm{\; [-]} $ ",
    rotation = 'vertical', size = 20)
graf_argand.set_xlim([1.3,1.9])
graf_argand.set_ylim([0.8,1.6])
graf_argand.grid('on')
partie_reelle=np.real(FT_moy/k_tot())
partie_imag=np.imag(FT_moy/k_tot())
dev_real=np.real((dev_Fb()+dev_Fh())/(Xmoy*k_tot()))
dev_imag=np.imag((dev_Fb()+dev_Fh())/(Xmoy*k_tot()))
markers = "+", "o", "d", ">", "*", "<", "^"
couleurs="black","blue","green","red"
labels="Sec","Stagnant","U=0.9 m/s","U=1.0 m/s"
for hihi in [0,1,2,3]:
    graf_argand.errorbar(partie_reelle[hihi,1:],partie_imag[hihi,1:], dev_real[hihi,1:],
     dev_imag[hihi,1:], marker = markers[hihi], ms=10, linestyle=' ', color = couleurs[
    hihi], ecolor = couleurs[hihi],label=labels[hihi], barsabove=True)
handles, labels = graf_argand.get_legend_handles_labels()
by_label = OrderedDict(zip(labels, handles))
graf_argand.legend(by_label.values(), by_label.keys(), fontsize = 20)
```

Annexe C

Programmes utilisés pour les simulations numériques

C.1 Script de construction du maillage

make_geometry.py

- Pré-conditions : Salomé 9.6.0 installé.
- Exécution : Exécuter la commande "salome -t make_geometry.py" dans un terminal.
- Post-conditions : Un fichier MED contenant le maillage est écrit dans le répertoire "/scratch/g42679/ellez_ALE". Le nom du fichier est "Ellez_r*raffinage_r*adapt_z*raffinage_z*_dth2_f4.med", où *raffinage_r* et *raffinage_z* sont des variables du script remplacées par leurs valeurs respectives.

```
#!/usr/bin/env python

import time
import copy

# homebrew timer class
class timer:
    # constructor
    def __init__(self):
        self.__start = None
        self.__elapsed = 0.

    # start the timer
    def start(self):
        if self.__start is not None:
            raise RuntimeError("Timer is already running. Use .stop() to stop it.")
        self.__start = time.time()

    # stop the timer
    def stop(self):
        if self.__start is None:
            raise RuntimeError("Timer is not running. Use .start() to start it.")
        self.__elapsed += time.time() - self.__start
        self.__start = None

    # get the elapsed time
    def elapsed(self):
        return self.__elapsed

# define some timers:
#    * to time comnsummed to load the modules of Salome
module_loading_timer = timer()
#    * to compute the total elapsed time (excpet the time consummed to load the modules)
global_timer = timer()
#    * to compute the time elapsed in the building of the fluid domain
```

```
geom_building_timer = timer()
#    * to compute the time elapsed in the definition of the groups, in the geometry
geom_def_groups_timer = timer()
#    * to compute the time elapsed in the definition of the mesh
#        - total time elapsed in the definition of the mesh
mesh_definition_timer = timer()
#        - time elapsed in the GEOM module
mesh_def_GEOM_timer = timer()
#        - time elapsed in the SMESH module
mesh_def_SMESH_timer = timer()
#    * to compute the time elapsed in the computation of the mesh
mesh_computation_timer = timer()
#    * to compute the time elapsed in the definition of the groups, in the mesh
mesh_def_groups_timer = timer()
#    * to compute the time consummed to write the mesh in a MED file
mesh_write_timer = timer()

# start the timer to compute the time consummed to load the modules
module_loading_timer.start()

###
### This file is generated automatically by SALOME v9.3.0 with dump python functionality
###

import sys
import salome

salome.salome_init()
import salome_notebook
theStudy = salome.myStudy
notebook = salome_notebook.NoteBook(theStudy)

###
### GEOM component
###

import GEOM
from salome.geom import geomBuilder
import math
import numpy as np
import SALOMEDS

geompy = geomBuilder.New(theStudy)

###
### SMESH component
###

import  SMESH, SALOMEDS
from salome.smesh import smeshBuilder

smesh = smeshBuilder.New(theStudy)
#smesh.SetEnablePublish( False ) # Set to False to avoid publish in study if not needed
    or in some particular situations:
                                # multiples meshes built in parallel, complex and
    numerous mesh edition (performance)

O = geompy.MakeVertex(0, 0, 0)
OX = geompy.MakeVectorDXDYDZ(1, 0, 0)
OY = geompy.MakeVectorDXDYDZ(0, 1, 0)
OZ = geompy.MakeVectorDXDYDZ(0, 0, 1)

# stop the timer to compute the time consumed to load the modules
module_loading_timer.stop()

# print the time consummed to load the modules
print("+--> Time consummed to load the modules: ", module_loading_timer.elapsed() )

# start the global timer
global_timer.start()
```

```
print("-> Start the global timer")

####
#
# Create the pattern : cylinder + two grids
#
####

# N.B. flow section: 150 mm x 150 mm

####
#
# parameters
#
####

# set numbers of cylinders in a row, the assembly is NxN cylinders
N = 5

# unit: milimeter
mm = 1.e-3

#    * d, diameter of a cylinder wrt the grid
#    * P, pitch
d = 11.*mm
P = 13.4*mm

#    * e, thickness of a spacerband
#    * h, height of a spacerband
e = 2.*mm
h = 15.*mm

#    * e_frame, thickness of the frame
e_frame = 5.*mm

#    * L_cyl, length of the cylinder
#    * d_cyl, diameter of the cylinder
L_cyl = 1295.*mm # <- is this length the length from one grid to the other? Ask Romain
    ^^' RR20200304-Yep !
d_cyl = 10.*mm

#    * w_tunnel, width of the water tunnel
w_tunnel = 150.*mm

#    * A, maximal amplitude of the assembly displacement
A = d_cyl

# compute the radius of the sphere wrt the grid
r = 0.5*d
# compute the radius of the cylinder
r_cyl = 0.5*d_cyl

# We want the grid of a NxN assembly to lie in a square of length (N + 1)*P.
# Each cylinder lies in a square of length P. Let L be the distance from a clylinder
    pattern to the frame.
# The length of a side of the squrare in which the grid lies reads:
#     N*P + 2*L + 2*e_frame
# We thus have:
#     N*P + 2*L + 2*e_frame = (N + 1)*P

# compute L
L = 0.5*P - e_frame

# compute the minimal layer of fluid, between the sphere and the boundary of the
    clyinder pattern.
e_layer = 2.0*(0.5*P - r) #modif largeur x2

# am: I recall e_layer is used to define a fluid box surrounding the assembly, in which
    the cells will not be changed by the ALE method.
```

```
#     The fluid domain does not reach the water tunnel borders yet ^^'!
#     The following parameter w_tunnel would be useful to define this missing part of
    the fluid domain.

# The size of the grid is: (N + 1)*P.
# The size of the fluid box surrounding the assembly is then: (N + 1)*P + 2.*e_layer
# Let w_channel be the size of the channel between fluid box surrounding the assembly
    and the nearest water tunnel border.
# We have:
#     w_tunnel = (N + 1)*P + 2.*e_layer + 2.*w_channel

# compute w_channel
w_channel = 0.5*(w_tunnel - (N + 1)*P) - e_layer

# h_up, length of the water tunnel upstream of the fluid box surrounding the assembly

# we express this legnth as a number of diameters of the spheres on top of the top grid,
     since these objects curve the streamlines
h_up = 30*d

# h_down, length of the water tunnel downstream of the fluid box surrounding the
    assembly

# we express this legnth as a number of diameters of the spheres on top of the top grid,
     since these objects curve the streamlines
h_down = 30*d

#
# build a symmetry plane of the assembly
#

# define a point on the plane
Point_7 = geompy.MakeVertex(0., 0., -h - 0.5*L_cyl)
# define the plane
# N.B. the size of the plane doesn't seem to matter :-D
Plane_1 = geompy.MakePlane(Point_7, OZ, P)

# A fuel assembly is made of nxn cylinders held toghether by a grid. We want to build
    the fluid domain surrounding
# the fuel assembly. We first define a pattern to reproduce nxn times: the fluid domain
    surrounding a cylinder, taking
# into account the spacer bands. We then build a bundle of this pattern, to model the
    nxn cylinders. We finallay add
# the fluid domain surrounding the frame.

# Cylinder Pattern Builder
class CylinderPatternBuilder():
    """An instance of this class is able to build the following pattern: the fluid
    surrounding a cylinder,
       taking into account the spacer bands.

       The spheres on top of the top grid lie in the domain {(x, y, z): 0 <= z <= P/2}.
       The top grid lies in the domain {(x, y, z): -h <= z <= 0}.
       The cylinder lies in the domain {(x, y, z): -(h + L_cyl) <= z <= -h}.
       The bottom grid lies in the domain {(x, y, z): -(2*h + L_cyl) <= z <= -(h + L_cyl
    )}.
       The spheres under the bottom of the bottomm grid lie in the domain {(x, y, z): -(
    P/2 + 2*h + L_cyl) <= z <= -(2*h + L_cyl)}.

       The builder thus builds a pattern lying in the domain {(x, y, z): |x| <= P/2, |y|
     <= P/2, -(P/2 + 2*h + L_cyl) <= z <= P/2}.
       Note that the plane {(x, y, z): z = -(h + L_cyl/2)} is a plane of symmetry of the
     pattern.
       """

    def getResult(self):
        """This method builds the pattern and return the built object."""

        #
        # fluid box
```

```
#

#
# fluid box upstream of the plate
#

Box_up = geompy.MakeBoxDXDYDZ(P, P, 0.5*P)
geompy.TranslateDXDYDZ(Box_up, -0.5*P, -0.5*P, 0.)

#
# tool to cut the fluid box
#

tool = geompy.MakeBoxDXDYDZ(P, e, P)
geompy.TranslateDXDYDZ(tool, -0.5*P, -0.5*e, -0.5*P)

#
# build a second tool, using rotation
#

tool_2 = geompy.MakeRotation(tool, OZ, 0.5*math.pi)

#
# sphere a the end of the cylinder
#
Sphere_1 = geompy.MakeSphereR(r)

#
# cut the fluid box upstream the plate using the first tool
# + common of the box with the tool
# => three objects interesection of the fluid box
#

# * cut the fluid box upstream the plate using the first tool
Cut_1 = geompy.MakeCut(Box_up, tool)
# * cut the result of the previous cut using the sphere
Cut_1 = geompy.MakeCut(Cut_1, Sphere_1)

# * common of the box with the tool
Common_1 = geompy.MakeCommon(Box_up, tool)

# * second sphere rotated to cut correctly Common_1
Sphere_2 = geompy.MakeRotation(Sphere_1, OY, 0.5*math.pi)
geompy.Rotate(Sphere_2, OX, 0.5*math.pi)

# * cut the result of the previous cut using the rotated sphere
Common_1 = geompy.MakeCut(Common_1, Sphere_2)

Compound_1 = geompy.MakeCompound([Cut_1, Common_1])

#
# cut the fluid box upstream the plate using the second tool
# + common of the box with the tool
# => nine objects interesection of the fluid box
#

Cut_2 = geompy.MakeCut(Compound_1, tool_2)
Common_2 = geompy.MakeCommon(Compound_1, tool_2)
Compound_2 = geompy.MakeCompound([Cut_2, Common_2])

#
# Explode the "pool cue chalk" (i.e. "craie de billard")
#
Solids = geompy.ExtractShapes(Compound_2, geompy.ShapeType["SOLID"], True)

#
# look for the solid in the domain {x > 0, y > 0, z > 0}
#

Solid_1 = None
```

```
        # loop over the solids
        for Sol in Solids:
            # get the center of mass of the current solid
            G = geompy.MakeCDG(Sol)
            # get the coordinates of the center of mass
            xG, yG, zG = geompy.PointCoordinates(G)
            # skip the solid centered on an axis
            if abs(xG) < 0.5*e:
                continue
            if abs(yG) < 0.5*e:
                continue
            # at this point, xG != 0 and yG != 0
            # if both xG and yG are positive,
            if xG > 0. and yG > 0.:
                # we found the solid
                Solid_1 = Sol
                break

        #
        # tool to partition the domain
        #

        tool_3 = geompy.MakeBox(0., 0., 0., P, P, P)

        #
        # build a first tool by rotation
        #
        tool_31 = geompy.MakeRotation(tool_3, OZ, -0.25*math.pi)

        #
        # compute the common part of Solid_1 and tool_31
        #
        Common_3 = geompy.MakeCommonList([Solid_1, tool_31], True)

        #
        # build a second tool by rotation
        #
        tool_32 = geompy.MakeRotation(tool_3, OY, +0.25*math.pi)
        geompy.TranslateDXDYDZ(tool_32, 0., -e, 0.)

        #
        # compute the common part of Common_3 and tool_32
        #
        Common_4 = geompy.MakeCommonList([Common_3, tool_32], True)

        #
        # build a third tool by rotation
        #
        tool_33 = geompy.MakeRotation(tool_3, OX, +0.25*math.pi)

        #
        # cut Solid_1, first using tool_31, then using tool_33
        #
        Cut_2 = geompy.MakeCut(Solid_1, tool_31)
        Cut_3 = geompy.MakeCut(Cut_2, tool_33)

        #
        # cut Solid_1, first using Common_4, then using Cut_3, to finish the partition
        #
        Cut_4 = geompy.MakeCut(Solid_1, Common_4)
        Cut_5 = geompy.MakeCut(Cut_4, Cut_3)

        #
        # look for the solid in the domain {x > e/2, |y| <= e/2, z > 0}
        #

        Solid_2 = None
        # loop over the solids
        for Sol in Solids:
            # get the center of mass of the current solid
```

```
            G = geompy.MakeCDG(Sol)
            # get the coordinates of the center of mass
            xG, yG, zG = geompy.PointCoordinates(G)
            # if both xG > e/2 and |yG| <= e/2,
            if xG > 0.5*e and abs(yG) < 0.5*e:
                # we found the solid
                Solid_2 = Sol
                break

        #
        # cut Solid_2, first using tool_32
        #
        Cut_6 = geompy.MakeCut(Solid_2, tool_32)

        #
        # compute the common part of Solid_2 and tool_32
        #

        # N.B. use common with Solid_2 and tool_32 ends up with solids breaking the full
     hexaedra algorithm of the Mesh module :'(
        #
        # Actually, the problem appears where a part of the boundary is cut by the
    sphere...
        # We cut this solid in Common_1, to obtain a solid far from the boundaries of
    the inital box ;-) !

        #
        # build a fourth tool by rotation of tool_32
        #
        tool_34 = geompy.MakeRotation(tool_32, OY, -0.25*math.pi)

        #
        # cut Common_1 using tool_34
        #
        Cut_7 = geompy.MakeCut(Common_1, tool_34)

        #
        # rotate tool_34
        #
        geompy.Rotate(tool_34, OY, -0.75*math.pi)

        #
        # cut Cut_7 using tool_34
        #
        Cut_8 = geompy.MakeCut(Cut_7, tool_34)

        #
        # rotate Cut_8 to obtain the common part of Solid_2 and tool_32
        #
        Common_5 = geompy.MakeRotation(Cut_8, OY, 0.5*math.pi)

        #
        # build a compound from the solids partitionning a quarter of the "pull cue
    chalk"
        #
        Compound_3 = geompy.MakeCompound([Common_4, Cut_3, Cut_5, Common_5, Cut_6])

        #
        # build the three other quarters by rotation
        #
        Rotation_1 = geompy.MakeRotation(Compound_3, OZ, 0.5*math.pi)
        Rotation_2 = geompy.MakeRotation(Compound_3, OZ, math.pi)
        Rotation_3 = geompy.MakeRotation(Compound_3, OZ, 1.5*math.pi)

        #
        # get the last solid from Compound_2
        #

        #
        # look for the solid in the domain {|x| <= e/2, |y| <= e/2, z > 0}
```

```
    #

    Solid_3 = None
    # loop over the solids
    for Sol in Solids:
        # get the center of mass of the current solid
        G = geompy.MakeCDG(Sol)
        # get the coordinates of the center of mass
        xG, yG, zG = geompy.PointCoordinates(G)
        # if both |xG| < e/2 and |yG| <= e/2,
        if abs(xG) < 0.5*e and abs(yG) < 0.5*e:
            # we found the solid
            Solid_3 = Sol
            break

    #
    # build a compound from the solids partitionning the full "pull cue chalk"
    #
    Compound_4 = geompy.MakeCompound([Compound_3, Rotation_1, Rotation_2, Rotation_3
, Solid_3])

    #
    # remove duplicated edges and faces
    #
    Glue_1 = geompy.MakeGlueEdges(Compound_4, 1e-07)
    Glue_2 = geompy.MakeGlueFaces(Glue_1, 1e-07)

    #
    # look for faces in Glue_2 to build the fluid box arround the grid using
extrusion
    #

    # build useful points
    Point_1 = geompy.MakeVertex(r/math.sqrt(2.), r/math.sqrt(2.), 0.)
    Point_2 = geompy.MakeVertex(0.5*P, 0.5*P, 0.)
    Point_3 = geompy.MakeVertex(0.5*e, 0.5*P, 0.)
    Point_4 = geompy.MakeVertex(0.5*P, 0.5*e, 0.)
    Point_5 = geompy.MakeVertex(0.5*P, -0.5*e, 0.)

    # point on the sphere, such that: x > 0, y=e/2 and z=0
    y6 = 0.5*e
    z6 = 0.
    x6 = math.sqrt(r**2 - y6**2 - z6**2)
    Point_6 = geompy.MakeVertex(x6, y6, z6)

    # get corresponding vertices
    Vertex_1 = geompy.GetVertexNearPoint(Glue_2, Point_1)
    Vertex_2 = geompy.GetVertexNearPoint(Glue_2, Point_2)
    Vertex_3 = geompy.GetVertexNearPoint(Glue_2, Point_3)
    Vertex_4 = geompy.GetVertexNearPoint(Glue_2, Point_4)
    Vertex_5 = geompy.GetVertexNearPoint(Glue_2, Point_5)
    Vertex_6 = geompy.GetVertexNearPoint(Glue_2, Point_6)

    #
    # look for the first face
    #

    # get two edges of the face
    Edge_12 = geompy.GetEdge(Glue_2, Vertex_1, Vertex_2)
    Edge_24 = geompy.GetEdge(Glue_2, Vertex_2, Vertex_4)

    # get the face
    Face_1 = geompy.GetFaceByEdges(Glue_2, Edge_12, Edge_24)

    #
    # look for the second face
    #

    # get on edge of the face
    Edge_23 = geompy.GetEdge(Glue_2, Vertex_2, Vertex_3)
```

```
    # get the face
    Face_2 = geompy.GetFaceByEdges(Glue_2, Edge_12, Edge_23)

    #
    # look for the third face
    #

    # get two edges of the face
    Edge_45 = geompy.GetEdge(Glue_2, Vertex_4, Vertex_5)
    Edge_46 = geompy.GetEdge(Glue_2, Vertex_4, Vertex_6)

    # get the face
    Face_3 = geompy.GetFaceByEdges(Glue_2, Edge_45, Edge_46)

    #
    # build the fluid domain around around the spacerbands
    #

    # extrude Face_1 and Face_2, along the spacerband
    Extrusion_1 = geompy.MakePrismVecH(Face_1, OZ, -h)
    Extrusion_2 = geompy.MakePrismVecH(Face_2, OZ, -h)

    #
    # build a compound from the solids partitionning a quarter of the fluid domain
around the spacerbands
    #
    Compound_5 = geompy.MakeCompound([Extrusion_1, Extrusion_2])

    #
    # build the three other quarters by rotation
    #
    Rotation_4 = geompy.MakeRotation(Compound_5, OZ, 0.5*math.pi)
    Rotation_5 = geompy.MakeRotation(Compound_5, OZ, math.pi)
    Rotation_6 = geompy.MakeRotation(Compound_5, OZ, 1.5*math.pi)

    #
    # build a compound from the solids partitionning the full fluid domain around
the spacerbands
    #
    Compound_6 = geompy.MakeCompound([Compound_5, Rotation_4, Rotation_5, Rotation_6
])

    #
    # build the fluid domain around around the cylinder
    #

    # The cylinder is thinner than the grid.
    # Extrude the fluid domain from Face_1 and Face_2 is thus not sufficient.
    # We need a hollow cylinder of fluid around the cylinder.

    #
    # create a disk of radius r; the external radius of the hollow cylinder
    #
    Disk_1 = geompy.MakeDiskR(r, 1)

    #
    # create a disk of radius r_cyl; the internal radius of the hollow cylinder
    #
    Disk_2 = geompy.MakeDiskR(r_cyl, 1)

    #
    # make the hollow disk
    #
    Cut_9 = geompy.MakeCut(Disk_1, Disk_2)

    #
    # create two rectangles to cut the hollow disk
    #
    tool_4 = geompy.MakeFaceHW(P, P, 1)
```

```
    tool_5 = geompy.MakeFaceHW(0.5*P, e, 1)
    geompy.TranslateDXDYDZ(tool_5, 0.25*P, 0., 0.)

    #
    # first tool: a tool to extract the part of the hollow disk in the domain {x >=
e/2, y >= e/2: x, y}
    #
    tool_41 = geompy.MakeTranslation(tool_4, 0.5*(P + e), 0.5*(P + e), 0.)

    #
    # extract the common part of the hollow disk and the domain {x >= e/2, y >= e/2:
 x, y}
    #
    Common_6 = geompy.MakeCommon(Cut_9, tool_41)

    #
    # second tool: a tool to partition the "quarter" of hollow disk
    #
    tool_42 = geompy.MakeTranslation(tool_4, 0.5*P, 0.5*P, 0.)
    geompy.Rotate(tool_42, OZ, -0.25*math.pi)

    #
    # compute the partition of the "quarter" of hollow disk
    #
    Cut_10 = geompy.MakeCut(Common_6, tool_42)
    Common_7 = geompy.MakeCommon(Common_6, tool_42)

    #
    # extract the common part of the hollow disk and the domain {|x| <= e/2, |y| <=
e/2: x, y}
    #
    Common_8 = geompy.MakeCommon(Cut_9, tool_5)

    #
    # extrude the faces along the cylinder
    #
    Extrusion_3 = geompy.MakePrismVecH(Face_1, OZ, -L_cyl)
    Extrusion_4 = geompy.MakePrismVecH(Face_2, OZ, -L_cyl)
    Extrusion_5 = geompy.MakePrismVecH(Face_3, OZ, -L_cyl)
    Extrusion_6 = geompy.MakePrismVecH(Cut_10, OZ, -L_cyl)
    Extrusion_7 = geompy.MakePrismVecH(Common_7, OZ, -L_cyl)
    Extrusion_8 = geompy.MakePrismVecH(Common_8, OZ, -L_cyl)

    #
    # build a compound from the solids partitionning a quarter of the fluid domain
around the cylinder
    #
    Compound_7 = geompy.MakeCompound([Extrusion_3, Extrusion_4, Extrusion_5,
Extrusion_6, Extrusion_7, Extrusion_8])

    # position the fluid domain
    geompy.TranslateDXDYDZ(Compound_7, 0., 0., -h)

    #
    # build the three other quarters by rotation
    #
    Rotation_7 = geompy.MakeRotation(Compound_7, OZ, 0.5*math.pi)
    Rotation_8 = geompy.MakeRotation(Compound_7, OZ, math.pi)
    Rotation_9 = geompy.MakeRotation(Compound_7, OZ, 1.5*math.pi)

    #
    # build a compound from the solids partitionning the full fluid domain around
the cylinder
    #
    Compound_8 = geompy.MakeCompound([Compound_7, Rotation_7, Rotation_8, Rotation_9
])

    #
    # build the second grid, using mirror reflection
    #
```

```
    # relfect the first grid
    Mirror_1 = geompy.MakeMirrorByPlane(Glue_2, Plane_1)
    Mirror_2 = geompy.MakeMirrorByPlane(Compound_6, Plane_1)

    #
    # build a compound from the different parts of the fluid domain surrounding the
cylinder + the two grids
    #
    Compound_9 = geompy.MakeCompound([Glue_2, Compound_6, Compound_8, Mirror_1,
Mirror_2])

    #
    # remove duplicated edges and faces
    #
    Glue_3 = geompy.MakeGlueEdges(Compound_9, 1e-07)
    Pattern = geompy.MakeGlueFaces(Glue_3, 1e-07)

    # return the built cylinder pattern
    return Pattern

def setMeshSize(self, Shape, Mesh, dr, dth, dz, dz2, transformation=None):
    """This method set the mesh size in a part of a mesh corresponding to a cylinder
 pattern.

       Inputs:

       - Shape: the geometrical object to be meshed, in which some parts corresponds
 to a cylinder pattern,
       - Mesh: the mesh obtained from the given geometrical object,
       - dr: mesh size in the radial direction,
       - dth: mesh size in the orthoradial direction,
       - dz: mesh size in the Oz direction, near the grids.
       - dz2: mesh size in the Oz direction, far from the grids.

       Optional inputs:

       - transformation: the geometrical transformation to apply to the reference
cylinder pattern to obtain
           the corrsponding cylinder pattern in the given geometrical object.
    """

    # if no transformation is given
    if transformation is None:
        # the transformation is identity
        transformation = lambda theObject: theObject

    ####
    #
    # specify mesh size in the orthoradial direction (cylinder + sphere)
    #
    ####

    # start the timer to compute the time elapsed in GEOM module, during the
definition of the mesh
    mesh_def_GEOM_timer.start()

    # list of edges used to define the mesh size in the orthoradial direction
    Edges_th = []

    #
    # get edges on the circle {(x, y, z): x^2 + y^2 = r^2, z = 0}
    #

    # get edge near point (0, r, 0)
    Point_ref = geompy.MakeVertex(0., r, 0.)
    Point = transformation(Point_ref)
    Edge = geompy.GetEdgeNearPoint(Shape, Point)
    Edges_th.append(Edge)
```

```
# get edge near point (r, 0, 0)
Point_ref = geompy.MakeVertex(r, 0., 0.)
Point = transformation(Point_ref)
Edge = geompy.GetEdgeNearPoint(Shape, Point)
Edges_th.append(Edge)

# P_1, point on the sphere, such that: x > 0, y=+e/2 and z=0
y1 = 0.5*e
z1 = 0.
x1 = math.sqrt(r**2 - y1**2 - z1**2)
Point_1_ref = geompy.MakeVertex(x1, y1, z1)
Point_1 = transformation(Point_1_ref)

# P_2, point on the sphere, such that: x = y > 0 and z=0
x2 = 0.5*math.sqrt(2.)*r
y2 = x2
z2 = 0.
Point_2_ref = geompy.MakeVertex(x2, y2, z2)
Point_2 = transformation(Point_2_ref)

# P_3, point on the sphere, such that: x=+e/2, y > 0 and z=0
x3 = y1
y3 = x1
z3 = 0.
Point_3_ref = geompy.MakeVertex(x3, y3, z3)
Point_3 = transformation(Point_3_ref)

# get corresponding vertices
Vertex_1 = geompy.GetVertexNearPoint(Shape, Point_1)
Vertex_2 = geompy.GetVertexNearPoint(Shape, Point_2)
Vertex_3 = geompy.GetVertexNearPoint(Shape, Point_3)

# get the edges (P_1, P_2) and (P_2, P_3)
Edge_12 = geompy.GetEdge(Shape, Vertex_1, Vertex_2)
Edges_th.append(Edge_12)
Edge_23 = geompy.GetEdge(Shape, Vertex_2, Vertex_3)
Edges_th.append(Edge_23)

# P_4, point on the sphere, such that: x < 0, y=-e/2 and z=0
y4 = -y1
z4 = z1
x4 = -x1
Point_4_ref = geompy.MakeVertex(x4, y4, z4)
Point_4 = transformation(Point_4_ref)

# P_5, point on the sphere, such that: x = y < 0 and z=0
x5 = -x2
y5 = -y2
z5 = z2
Point_5_ref = geompy.MakeVertex(x5, y5, z5)
Point_5 = transformation(Point_5_ref)

# P_6, point on the sphere, such that: x=-e/2, y < 0 and z=0
x6 = -x3
y6 = -y3
z6 = z3
Point_6_ref = geompy.MakeVertex(x6, y6, z6)
Point_6 = transformation(Point_6_ref)

# get corresponding vertices
Vertex_4 = geompy.GetVertexNearPoint(Shape, Point_4)
Vertex_5 = geompy.GetVertexNearPoint(Shape, Point_5)
Vertex_6 = geompy.GetVertexNearPoint(Shape, Point_6)

# get the edges (P_4, P_5) and (P_5, P_6)
Edge = geompy.GetEdge(Shape, Vertex_4, Vertex_5)
Edges_th.append(Edge)
Edge = geompy.GetEdge(Shape, Vertex_5, Vertex_6)
Edges_th.append(Edge)
```

```
# P_7, point on the sphere, such that: x = y = z > 0
x7 = r/math.sqrt(3.)
y7 = x7
z7 = x7
Point_7_ref = geompy.MakeVertex(x7, y7, z7)
Point_7 = transformation(Point_7_ref)

# get corresponding vertex
Vertex_7 = geompy.GetVertexNearPoint(Shape, Point_7)

# get the edge (P_2, P_7)
Edge_27 = geompy.GetEdge(Shape, Vertex_2, Vertex_7)
Edges_th.append(Edge_27)

# compute  P'_2 as the reflection of P_2 and P'_7 as the relfection of P_7
Mirror_1_ref = geompy.MakeMirrorByPlane(Point_2_ref, Plane_1)
Mirror_1 = transformation(Mirror_1_ref)
Mirror_2_ref = geompy.MakeMirrorByPlane(Point_7_ref, Plane_1)
Mirror_2 = transformation(Mirror_2_ref)

# get corresponding vertices
Vertex_M1 = geompy.GetVertexNearPoint(Shape, Mirror_1)
Vertex_M2 = geompy.GetVertexNearPoint(Shape, Mirror_2)

# get the edge (P'_2, P'_7)
Edge = geompy.GetEdge(Shape, Vertex_M1, Vertex_M2)
Edges_th.append(Edge)

#
# get edges on the circle {(x, y, z): x^2 + y^2 = r^2, z = -h}
#

# get edge near point (0, r, -h)
Point_ref = geompy.MakeVertex(0., r, -h)
Point = transformation(Point_ref)
Edge = geompy.GetEdgeNearPoint(Shape, Point)
Edges_th.append(Edge)

# get edge near point (-r, 0, -h)
Point_ref = geompy.MakeVertex(-r, 0., -h)
Point = transformation(Point_ref)
Edge = geompy.GetEdgeNearPoint(Shape, Point)
Edges_th.append(Edge)

# get edge near point (0, -r, -h)
Point_ref = geompy.MakeVertex(0., -r, -h)
Point = transformation(Point_ref)
Edge = geompy.GetEdgeNearPoint(Shape, Point)
Edges_th.append(Edge)

# get edge near point (r, 0, -h)
Point_ref = geompy.MakeVertex(r, 0., -h)
Point = transformation(Point_ref)
Edge = geompy.GetEdgeNearPoint(Shape, Point)
Edges_th.append(Edge)

#
# get edges on the circle {(x, y, z): x^2 + y^2 = r^2, z = -2*h - L_cyl}
#

# get edge near point (0, r, -2*h - L_cyl)
Point_ref = geompy.MakeVertex(0., r, -2*h - L_cyl)
Point = transformation(Point_ref)
Edge = geompy.GetEdgeNearPoint(Shape, Point)
Edges_th.append(Edge)

# get edge near point (-r, 0, -2*h - L_cyl)
Point_ref = geompy.MakeVertex(-r, 0., -2*h - L_cyl)
Point = transformation(Point_ref)
Edge = geompy.GetEdgeNearPoint(Shape, Point)
```

```
    Edges_th.append(Edge)

    # get edge near point (0, -r, -2*h - L_cyl)
    Point_ref = geompy.MakeVertex(0., -r, -2*h - L_cyl)
    Point = transformation(Point_ref)
    Edge = geompy.GetEdgeNearPoint(Shape, Point)
    Edges_th.append(Edge)

    # get edge near point (r, 0, -2*h - L_cyl)
    Point_ref = geompy.MakeVertex(r, 0., -2*h - L_cyl)
    Point = transformation(Point_ref)
    Edge = geompy.GetEdgeNearPoint(Shape, Point)
    Edges_th.append(Edge)

    # stop the timer to compute the time elapsed in GEOM module, during the
definition of the mesh
    mesh_def_GEOM_timer.stop()

    #
    # define a sub-mesh on each of these edges
    #

    # start the timer to compute the time elapsed in SMESH module, during the
definition of the mesh
    mesh_def_SMESH_timer.start()

    for i_Edge, Edge in enumerate(Edges_th):
        # get the length of the current edge
        length, area, volume = geompy.BasicProperties(Edge)
        # compute the number of segments
        nseg = int(length/dth)

        # ensure there is at least one element in the segment
        nseg = max(nseg, 1)

        # define the sub-mesh
        #    * Algorithm: "Wire Discretisation"
        #    * Hypothesis: "Number of segments = nseg"
        Regular_1D = Mesh.Segment(geom=Edge)
        Regular_1D.NumberOfSegments(nseg)
        # propagate this definition
        Regular_1D.Propagation()

    # stop the timer to compute the time elapsed in SMESH module, during the
definition of the mesh
    mesh_def_SMESH_timer.stop()

    ####
    #
    # specify mesh size in the orthoradial direction (cylinder + sphere)
    #
    ####

    # start the timer to compute the time elapsed in GEOM module, during the
definition of the mesh
    mesh_def_GEOM_timer.start()

    # list of edges used to define the mesh size in the radial direction
    Edges_r = []

    # P_8, point on the cylinder, such that: x > 0, y=+e/2 and z=-h
    y8 = 0.5*e
    x8 = math.sqrt(r_cyl**2 - y8**2)
    z8 = -h
    Point_8_ref = geompy.MakeVertex(x8, y8, z8)
    Point_8 = transformation(Point_8_ref)

    # P_9, point such that: x > 0, y=+e/2, x**2 + y**2 = r and z=-h
    y9 = 0.5*e
    x9 = math.sqrt(r**2 - y9**2)
```

```
    z9 = -h
    Point_9_ref = geompy.MakeVertex(x9, y9, z9)
    Point_9 = transformation(Point_9_ref)

    # P_10, point such that: x=+P/2, y=+e/2 and z=-h
    Point_10_ref = geompy.MakeVertex(0.5*P, 0.5*e, -h)
    Point_10 = transformation(Point_10_ref)

    # get corresponding vertices
    Vertex_8 = geompy.GetVertexNearPoint(Shape, Point_8)
    Vertex_9 = geompy.GetVertexNearPoint(Shape, Point_9)
    Vertex_10 = geompy.GetVertexNearPoint(Shape, Point_10)

    # get the edges (P_8, P_9) and (P_9, P_10)
    Edge = geompy.GetEdge(Shape, Vertex_8, Vertex_9)
    Edges_r.append(Edge)
    Edge = geompy.GetEdge(Shape, Vertex_9, Vertex_10)
    Edges_r.append(Edge)

    # stop the timer to compute the time elapsed in GEOM module, during the
definition of the mesh
    mesh_def_GEOM_timer.stop()

    #
    # define a sub-mesh on each of these edges
    #

    # start the timer to compute the time elapsed in SMESH module, during the
definition of the mesh
    mesh_def_SMESH_timer.start()

    for Edge in Edges_r:
        # get the length of the current edge
        length, area, volume = geompy.BasicProperties(Edge)
        # compute the number of segments
        nseg = int(length/dr)

        # ensure there is at least one element in the segment
        nseg = max(nseg, 1)

        # define the sub-mesh
        #    * Algorithm: "Wire Discretisation"
        #    * Hypothesis: "Number of segments = nseg"
        #Regular_1D = Mesh.Segment(geom=Edge)
        #Regular_1D.NumberOfSegments(nseg)
        Regular_1D = Mesh.Segment(geom=Edge) #modif raison égo en r
        NumberOfSegments_1=Regular_1D.NumberOfSegments(nseg,1,[])
        NumberOfSegments_1.SetScaleFactor( 4 )
        NumberOfSegments_1.SetReversedEdges([])
        # propagate this definition
        Regular_1D.Propagation()

    # stop the timer to compute the time elapsed in SMESH module, during the
definition of the mesh
    mesh_def_SMESH_timer.stop()

    ####
    #
    # specify mesh size in the Oz direction
    #
    ####

    # start the timer to compute the time elapsed in GEOM module, during the
definition of the mesh
    mesh_def_GEOM_timer.start()

    # list of edges used to define the mesh size in the Oz direction
    Edges_z = []

    # the mesh size in the domain such that 0 <= z <= P/2 is related to the
```

```
discretization of the sphere

    #
    # specify the mesh size in the parts of the fluid domain located between two
spacer bands of the top grid
    #

    # get edges in the domain such that -h <= z <= 0
    # N.B. we need four edges, since this domain is not connex due to the spacer
bands

    # P_11, point such that: x > 0, y=+e/2, x^2 + y^2 = r^2 and z=-h/2
    y11 = 0.5*e
    x11 = math.sqrt(r**2 - y11**2)
    z11 = -0.5*h
    Point_11_ref = geompy.MakeVertex(x11, y11, z11)

    # P_12, point such that: x=-e/2, y > 0, x^2 + y^2 = r^2 and z=-h/2
    Point_12_ref = geompy.MakeRotation(Point_11_ref, OZ, 0.5*math.pi)

    # P_13, point such that: x < 0, y=-e/2, x^2 + y^2 = r^2 and z=-h/2
    Point_13_ref = geompy.MakeRotation(Point_11_ref, OZ, math.pi)

    # P_14, point such that: x=+e/2, y < 0, x^2 + y^2 = r^2 and z=-h/2
    Point_14_ref = geompy.MakeRotation(Point_11_ref, OZ, 1.5*math.pi)

    # get edge near point P_11
    Point_11 = transformation(Point_11_ref)
    Edge = geompy.GetEdgeNearPoint(Shape, Point_11)
    Edges_z.append(Edge)

    # get edge near point P_12
    Point_12 = transformation(Point_12_ref)
    Edge = geompy.GetEdgeNearPoint(Shape, Point_12)
    Edges_z.append(Edge)

    # get edge near point P_13
    Point_13 = transformation(Point_13_ref)
    Edge = geompy.GetEdgeNearPoint(Shape, Point_13)
    Edges_z.append(Edge)

    # get edge near point P_14
    Point_14 = transformation(Point_14_ref)
    Edge = geompy.GetEdgeNearPoint(Shape, Point_14)
    Edges_z.append(Edge)

    #
    # specify the mesh size in the parts of the fluid domain located between two
spacer bands of the top grid
    #

    # get an edge in the domain such that -2*h - L_cyl <= z <= -h - L_cyl
    # N.B. we need four edges, since this domain is not connex due to the spacer
bands

    # P_15, point such that: x > 0, y=+e/2, x^2 + y^2 = r^2 and z=-3*h/2 - L_cyl
    Point_15_ref = geompy.MakeMirrorByPlane(Point_11_ref, Plane_1)

    # P_16, point such that: x=-e/2, y > 0, x^2 + y^2 = r^2 and z=-3*h/2 - L_cyl
    Point_16_ref = geompy.MakeMirrorByPlane(Point_12_ref, Plane_1)

    # P_17, point such that: x < 0, y=-e/2, x^2 + y^2 = r^2 and z=-3*h/2 - L_cyl
    Point_17_ref = geompy.MakeMirrorByPlane(Point_13_ref, Plane_1)

    # P_18, point such that: x=+e/2, y < 0, x^2 + y^2 = r^2 and z=-3*h/2 - L_cyl
    Point_18_ref = geompy.MakeMirrorByPlane(Point_14_ref, Plane_1)

    # get edge near point P_15
    Point_15 = transformation(Point_15_ref)
    Edge = geompy.GetEdgeNearPoint(Shape, Point_15)
```

```
    Edges_z.append(Edge)

    # get edge near point P_16
    Point_16 = transformation(Point_16_ref)
    Edge = geompy.GetEdgeNearPoint(Shape, Point_16)
    Edges_z.append(Edge)

    # get edge near point P_17
    Point_17 = transformation(Point_17_ref)
    Edge = geompy.GetEdgeNearPoint(Shape, Point_17)
    Edges_z.append(Edge)

    # get edge near point P_18
    Point_18 = transformation(Point_18_ref)
    Edge = geompy.GetEdgeNearPoint(Shape, Point_18)
    Edges_z.append(Edge)

    # the mesh size in the domain such that -2*h - L_cyl - P/2 <= z <= -2*h - L_cyl
is related to the discretization of the sphere

    # stop the timer to compute the time elapsed in GEOM module, during the
definition of the mesh
    mesh_def_GEOM_timer.stop()

    #
    # define a sub-mesh on each of these edges
    #

    # start the timer to compute the time elapsed in SMESH module, during the
definition of the mesh
    mesh_def_SMESH_timer.start()

    for Edge in Edges_z:
        # get the length of the current edge
        length, area, volume = geompy.BasicProperties(Edge)
        # compute the number of segments
        nseg = int(length/dz)

        # ensure there is at least one element in the segment
        nseg = max(nseg, 1)

        # define the sub-mesh
        #    * Algorithm: "Wire Discretisation"
        #    * Hypothesis: "Number of segments = nseg"
        Regular_1D = Mesh.Segment(geom=Edge)
        Regular_1D.NumberOfSegments(nseg)
        # propagate this definition
        Regular_1D.Propagation()

    # stop the timer to compute the time elapsed in SMESH module, during the
definition of the mesh
    mesh_def_SMESH_timer.stop()

    #
    # specify the mesh size in the parts of the fluid domain surrounding the
cylinders
    #

    # start the timer to compute the time elapsed in GEOM module, during the
definition of the mesh
    mesh_def_GEOM_timer.start()

    # domain such that -h - L_cyl <= z <= -h

    # P_10, point such that: x > 0, y=+e/2, x^2 + y^2 = r^2 and z=-h - L_cyl/2
    y19 = 0.5*e
    x19 = math.sqrt(r**2 - y11**2)
    z19 = -h - 0.5*L_cyl
    Point_19_ref = geompy.MakeVertex(x19, y19, z19)
    Point_19 = transformation(Point_19_ref)
```

```
    # get the edge near point P_19
    Edge = geompy.GetEdgeNearPoint(Shape, Point_19)

    # stop the timer to compute the time elapsed in GEOM module, during the
definition of the mesh
    mesh_def_GEOM_timer.stop()

    # start the timer to compute the time elapsed in SMESH module, during the
definition of the mesh
    mesh_def_SMESH_timer.start()

    # the legnth of this edge is L_cyl

    # we split this edge in three parts:
    #
    # +---+--------------+---+
    #   1         2        3
    #

    # the length of part 1 is h,     so that part 1 corresponds to domain {(x, y, z)
: -2*h <= z <= -h
    # the length of part 2 is L_cyl, so that part 2 corresponds to domain {(x, y, z)
: -L_cyl <= z <= -2*h
    # the length of part 3 is h,     so that part 3 corresponds to domain {(x, y, z)
: -h - L_cyl <= z <= -L_cyl

    # partition of the [0, 1] semgent corresponding to this partion of the edge
    #
    #     1          2            3
    # +-------+--------------+--------+
    # 0    h/L_cyl      1 - h/L_cyl   1


    # N.B. due to symmetry, part 1 and 3 are interchangeable :-D

    # mesh size in parts 1 and 3: dz
    # mesh size in part 2: dz2

    # compute the number of segments in parts 1 and 3
    nseg = int(h/dz)
    # ensure there is at least one element in the segment
    nseg = max(nseg, 1)

    # compute the number of segments in parts 2
    nseg2 = int((L_cyl - 2*h)/dz2)
    # ensure there is at least one element in the segment
    nseg2 = max(nseg2, 1)

    # define the sub-mesh
    #    * Algorithm: "Wire Discretisation"
    #    * Hypothesis: "Fixed Point 1D"
    Regular_1D = Mesh.Segment(geom=Edge)
    Regular_1D.FixedPoints1D([ 0., h/L_cyl, 1. - h/L_cyl, 1. ],[ nseg, nseg2, nseg
],[])
    # propagate this definition
    Regular_1D.Propagation()

    # stop the timer to compute the time elapsed in SMESH module, during the
definition of the mesh
    mesh_def_SMESH_timer.stop()

    #
    # compute the mesh size along the directions Ox, Oy and Oz in the domains where
the
    # the discretisation of the sphere is prescribed
    #

    # N.B. the time elapsed in the following part of the code is note measured by a
sepcific timer.
```

```
        # get the length of the edge (P_2, P_3)
        length, area, volume = geompy.BasicProperties(Edge_23)
        # compute nseg, the number of segments to be set on this edge
        nseg = int(length/dth)

        # This number of segments will be propagated to the edge lying in the domain {(x
    , y, z): e/2 <= x <= P/2, y = P/2, z = 0}
        # Thus the mesh size along the dorection Ox will be:
        #     Dx = (P/2 - e/2)/nseg

        # compute Dx
        Dx = (0.5*P - 0.5*e)/nseg

        # get the length of the edge (P_1, P_2)
        length, area, volume = geompy.BasicProperties(Edge_12)
        # compute nseg, the number of segments to be set on this edge
        nseg = int(length/dth)

        # This number of segments will be propagated to the edge lying in the domain {(x
    , y, z): x = P/2, e/2 <= y <= P/2, z = 0}
        # Thus the mesh size along the dorection Oy will be:
        #     Dy = (P/2 - e/2)/nseg

        # compute Dy
        Dy = (0.5*P - 0.5*e)/nseg

        # get the length of the edge (P_2, P_7)
        length, area, volume = geompy.BasicProperties(Edge_27)
        # compute nseg, the number of segments to be set on this edge
        nseg = int(length/dth)

        # This number of segments will be propagated to the edge lying in the domain {(x
    , y, z): x = P/2, y = P/2, 0 <= z = 0}
        # Thus the mesh size along the dorection Oz will be:
        #     Dz = (P/2)/nseg

        # compute Dz
        Dz = 0.5*P/nseg

        # return the mesh size along the directions Ox, Oy and Oz in the domains where
    the
        # the discretisation of the sphere is prescribed
        return Dx, Dy, Dz

# Frame Pattern Builder
class FramePatternBuilder():
    """An instance of this class is able to build the following pattern: the fluid
    surrounding the frame of the two grids.

       The top grid lies in the domain {(x, y, z): -h <= z <= 0}.
       The cylinder lies in the domain {(x, y, z): -(h + L_cyl) <= z <= -h}.
       The bottom grid lies in the domain {(x, y, z): -(2*h + L_cyl) <= z <= -(h + L_cyl
    )}.

       The builder thus builds a pattern lying in the domain {(x, y, z): P/2 <= x <= P +
     e_layer, |y| <= P/2, -(P/2 + 2*h + L_cyl) <= z <= P/2}.
       Note that the plane {(x, y, z): z = -(h + L_cyl/2)} is a plane of symmetry of the
     pattern.
    """

    def getResult(self):
        """This method builds the pattern and return the built object."""

        #
        # build the fluid layer between the cylinder pattern and the frame of the top
    grid
        #

        # on the top of the top grid (z >= 0)
```

```
    Box_17 = geompy.MakeBoxDXDYDZ(L, 0.5*(P - e), 0.5*P)
    geompy.TranslateDXDYDZ(Box_17, +0.5*P, -0.5*P, 0.)
    Box_18 = geompy.MakeBoxDXDYDZ(L, e, 0.5*P)
    geompy.TranslateDXDYDZ(Box_18, +0.5*P, -0.5*e, 0.)
    Box_19 = geompy.MakeTranslation(Box_17, 0., e + 0.5*(P - e), 0.)

    # at the top grid level (-h <= z <= 0)
    Box_20 = geompy.MakeBoxDXDYDZ(L, 0.5*(P - e), h)
    geompy.TranslateDXDYDZ(Box_20, +0.5*P, -0.5*P, -h)
    Box_21 = geompy.MakeTranslation(Box_20, 0., e + 0.5*(P - e), 0.)

    # between the two grids (-h - L_cyl <= z <= -h)
    Box_22 = geompy.MakeBoxDXDYDZ(L, 0.5*(P - e), L_cyl)
    geompy.TranslateDXDYDZ(Box_22, +0.5*P, -0.5*P, -L_cyl - h)
    Box_23 = geompy.MakeBoxDXDYDZ(L, e, L_cyl)
    geompy.TranslateDXDYDZ(Box_23, +0.5*P, -0.5*e, -L_cyl - h)
    Box_24 = geompy.MakeTranslation(Box_22, 0., e + 0.5*(P - e), 0.)

    #
    # build the fluid box around the frame of the top grid
    #

    # fluid domain on the top of the top grid
    Box_2 = geompy.MakeBoxDXDYDZ(e_frame, 0.5*(P - e), 0.5*P)
    geompy.TranslateDXDYDZ(Box_2, +0.5*P + L, -0.5*P, 0.)
    Box_3 = geompy.MakeBoxDXDYDZ(e_frame, e, 0.5*P)
    geompy.TranslateDXDYDZ(Box_3, +0.5*P + L, -0.5*e, 0.)
    Box_4 = geompy.MakeTranslation(Box_2, 0., e + 0.5*(P - e), 0.)

    # fluid domain between the two grids
    Box_5 = geompy.MakeBoxDXDYDZ(e_frame, 0.5*(P - e), L_cyl)
    geompy.TranslateDXDYDZ(Box_5, +0.5*P + L, -0.5*P, -L_cyl - h)
    Box_6 = geompy.MakeBoxDXDYDZ(e_frame, e, L_cyl)
    geompy.TranslateDXDYDZ(Box_6, +0.5*P + L, -0.5*e, -L_cyl - h)
    Box_7 = geompy.MakeTranslation(Box_5, 0., e + 0.5*(P - e), 0.)

    #
    # fluid domain beyond the grid
    #

    # on the top of the top grid (z >= 0)
    Box_8 = geompy.MakeBoxDXDYDZ(e_layer, 0.5*(P - e), 0.5*P)
    geompy.TranslateDXDYDZ(Box_8, +0.5*P + L + e_frame, -0.5*P, 0.)
    Box_9 = geompy.MakeBoxDXDYDZ(e_layer, e, 0.5*P)
    geompy.TranslateDXDYDZ(Box_9, +0.5*P + L + e_frame, -0.5*e, 0.)
    Box_10 = geompy.MakeTranslation(Box_8, 0., e + 0.5*(P - e), 0.)

    # at the top grid level (-h <= z <= 0)
    Box_11 = geompy.MakeBoxDXDYDZ(e_layer, 0.5*(P - e), h)
    geompy.TranslateDXDYDZ(Box_11, +0.5*P + L + e_frame, -0.5*P, -h)
    Box_12 = geompy.MakeBoxDXDYDZ(e_layer, e, h)
    geompy.TranslateDXDYDZ(Box_12, +0.5*P + L + e_frame, -0.5*e, -h)
    Box_13 = geompy.MakeTranslation(Box_11, 0., e + 0.5*(P - e), 0.)

    # between the two grids (-h - L_cyl <= z <= -h)
    Box_14 = geompy.MakeBoxDXDYDZ(e_layer, 0.5*(P - e), L_cyl)
    geompy.TranslateDXDYDZ(Box_14, +0.5*P + L + e_frame, -0.5*P, -L_cyl - h)
    Box_15 = geompy.MakeBoxDXDYDZ(e_layer, e, L_cyl)
    geompy.TranslateDXDYDZ(Box_15, +0.5*P + L + e_frame, -0.5*e, -L_cyl - h)
    Box_16 = geompy.MakeTranslation(Box_14, 0., e + 0.5*(P - e), 0.)

    #
    # build a compound from the different parts of the fluid domain surrounding the
top grid
    #
    Compound_10 = geompy.MakeCompound([Box_17, Box_18, Box_19, Box_20, Box_21, Box_2
, Box_3, Box_4, Box_8, Box_9, Box_10, Box_11, Box_12, Box_13])

    # relfect the fluid domain surrounding the top grid
    Mirror_7 = geompy.MakeMirrorByPlane(Compound_10, Plane_1)
```

```
    #
    # build a compound from the different parts of the fluid domain surrounding the
cylinders + the two grids
    #
    Compound_11 = geompy.MakeCompound([Compound_10, Mirror_7, Box_5, Box_6, Box_7,
Box_14, Box_15, Box_16, Box_22, Box_23, Box_24])

    #
    # remove duplicated edges and faces
    #
    Glue_4 = geompy.MakeGlueEdges(Compound_11, 1e-07)
    Pattern = geompy.MakeGlueFaces(Glue_4, 1e-07)

    # return the built frame pattern
    return Pattern

def setMeshSize(self, Shape, Mesh, dx, dz, transformation=None):
    """This method set the mesh size in a part of a mesh corresponding to a frame
pattern.

       Inputs:

       - Shape: the geometrical object to be meshed, in which some parts corresponds
 to a frame pattern,
       - Mesh: the mesh obtained from the given geometrical object,
       - dx: mesh size in the Ox direction,
       - dz: mesh size in the Oz direction.

       Optional inputs:

       - transformation: the geometrical transformation to apply to the reference
frame pattern to obtain
           the corresponding frame pattern in the given geometrical object.

       Please note that mesh size in given in the Oy direction, since this mesh size
 depends on the
       discretization of the cylinder pattern.
    """

    # if no transformation is given
    if transformation is None:
        # the transformation is identity
        transformation = lambda theObject: theObject

    ####
    #
    # specify mesh size in the Ox direction
    #
    ####

    # start the timer to compute the time elapsed in GEOM module, during the
definition of the mesh
    mesh_def_GEOM_timer.start()

    # list of edges used to define the mesh size in the Ox direction
    Edges_x = []

    #
    # get an edge in the domain {(x, y, z): P/2 <= x <= P/2 + L
    #

    # get edge near point (P/2 + L/2, e/2, 0)
    Point_ref = geompy.MakeVertex(0.5*P + 0.5*L, 0.5*e, 0.)
    Point = transformation(Point_ref)
    Edge = geompy.GetEdgeNearPoint(Shape, Point)
    Edges_x.append(Edge)

    #
    # get an edge in the domain {(x, y, z): P - e_frame <= x <= P
```

```
    # N.B. 1- we recall L + e_frame = P/2
    #      2- we need three edges, since this domain is not connex due to the two
grids
    #

    # get edge near point (P - e_frame/2, e/2, 0)
    Point_ref = geompy.MakeVertex(P - 0.5*e_frame, 0.5*e, 0.)
    Point = transformation(Point_ref)
    Edge = geompy.GetEdgeNearPoint(Shape, Point)
    Edges_x.append(Edge)

    # get edge near point (P - e_frame/2, e/2, -h)
    Point_ref = geompy.MakeVertex(P - 0.5*e_frame, 0.5*e, -h)
    Point = transformation(Point_ref)
    Edge = geompy.GetEdgeNearPoint(Shape, Point)
    Edges_x.append(Edge)

    # get edge near point (P - e_frame/2, e/2, -2*h - L_cyl)
    Point_ref = geompy.MakeVertex(P - 0.5*e_frame, 0.5*e, -2.*h - L_cyl)
    Point = transformation(Point_ref)
    Edge = geompy.GetEdgeNearPoint(Shape, Point)
    Edges_x.append(Edge)

    #
    # get an edge in the domain {(x, y, z): P <= x <= P + e_layer
    #

    # get edge near point (P + e_layer/2, e/2, 0)
    Point_ref = geompy.MakeVertex(P + 0.5*e_layer, 0.5*e, 0.)
    Point = transformation(Point_ref)
    Edge = geompy.GetEdgeNearPoint(Shape, Point)
    Edges_x.append(Edge)

    # stop the timer to compute the time elapsed in GEOM module, during the
definition of the mesh
    mesh_def_GEOM_timer.stop()

    #
    # define a sub-mesh on each of these edges
    #

    # start the timer to compute the time elapsed in SMESH module, during the
definition of the mesh
    mesh_def_SMESH_timer.start()

    for Edge in Edges_x:
        # get the length of the current edge
        length, area, volume = geompy.BasicProperties(Edge)
        # compute the number of segments
        nseg = int(length/dx)

        # ensure there is at least two elements in the segment
        nseg = max(nseg, 8) #2 par 8 modif 28012021 mesh ortho spacer band

        # define the sub-mesh
        #    * Algorithm: "Wire Discretisation"
        #    * Hypothesis: "Number of segments = nseg"
        Regular_1D = Mesh.Segment(geom=Edge)
        Regular_1D.NumberOfSegments(nseg)
        # propagate this definition
        Regular_1D.Propagation()

    # stop the timer to compute the time elapsed in SMESH module, during the
definition of the mesh
    mesh_def_SMESH_timer.stop()

    ####
    #
    # specify mesh size in the Oz direction
    #
```

```
        ####

        # start the timer to compute the time elapsed in GEOM module, during the
    definition of the mesh
        mesh_def_GEOM_timer.start()

        # N.B. The mesh size in the Oz direction is set by the cylinder pattern builder.
        #      However, due to the frame of one grid, the fluid domain surrounding a
        #      grid is not connex.

        # list of edges used to define the mesh size in the Oz direction
        Edges_z = []

        #
        # get an edge in the domain {(x, y, z): x >= P, -h <= z <= 0}
        #

        # get edge near point (P, e/2, -h/2)
        Point_ref = geompy.MakeVertex(P, 0.5*e, -0.5*h)
        Point = transformation(Point_ref)
        Edge = geompy.GetEdgeNearPoint(Shape, Point)
        Edges_z.append(Edge)

        #
        # get an edge in the domain {(x, y, z): x >= P, -2*h - L_cyl <= z <= -h - L_cyl}
        #

        # get edge near point (P, e/2, -3*h/2 - L_cyl)
        Point_ref = geompy.MakeVertex(P, 0.5*e, -1.5*h - L_cyl)
        Point = transformation(Point_ref)
        Edge = geompy.GetEdgeNearPoint(Shape, Point)
        Edges_z.append(Edge)

        # stop the timer to compute the time elapsed in GEOM module, during the
    definition of the mesh
        mesh_def_GEOM_timer.stop()

        #
        # define a sub-mesh on each of these edges
        #

        # start the timer to compute the time elapsed in SMESH module, during the
    definition of the mesh
        mesh_def_SMESH_timer.start()

        for Edge in Edges_z:
            # get the length of the current edge
            length, area, volume = geompy.BasicProperties(Edge)
            # compute the number of segments
            nseg = int(length/dz)

            # ensure there is at least one element in the segment
            nseg = max(nseg, 1)

            # define the sub-mesh
            #    * Algorithm: "Wire Discretisation"
            #    * Hypothesis: "Number of segments = nseg"
            Regular_1D = Mesh.Segment(geom=Edge)
            Regular_1D.NumberOfSegments(nseg)
            # propagate this definition
            Regular_1D.Propagation()

        # stop the timer to compute the time elapsed in SMESH module, during the
    definition of the mesh
        mesh_def_SMESH_timer.stop()

# Corner Pattern Builder
class CornerPatternBuilder():
    """An instance of this class is able to build the following pattern: the fluid
    surrounding the corner of the frame of the two grids.
```

```
    The top grid lies in the domain {(x, y, z): -h <= z <= 0}.
    The cylinder lies in the domain {(x, y, z): -(h + L_cyl) <= z <= -h}.
    The bottom grid lies in the domain {(x, y, z): -(2*h + L_cyl) <= z <= -(h + L_cyl
)}.

    The builder thus builds a pattern lying in the domain {(x, y, z): P/2 <= x <= P +
 e_layer, -P/2 -e_layer <= y <= -P/2, -(P/2 + 2*h + L_cyl) <= z <= P/2}.
    Note that the plane {(x, y, z): z = -(h + L_cyl/2)} is a plane of symmetry of the
 pattern.
"""

def getResult(self):
    """This method builds the pattern and return the built object."""

    #
    # build the part of the fluid domain surrounding the the corners of the frame of
 the two grids
    #

    # domain such that z => 0
    Box_17 = geompy.MakeBoxDXDYDZ(e_frame, e_frame, 0.5*P)
    geompy.TranslateDXDYDZ(Box_17, +0.5*P + L, -0.5*P - L - e_frame, 0.)
    Box_18 = geompy.MakeBoxDXDYDZ(e_layer, e_frame, 0.5*P)
    geompy.TranslateDXDYDZ(Box_18, +0.5*P + L + e_frame, -0.5*P - L - e_frame, 0.)
    Box_19 = geompy.MakeBoxDXDYDZ(e_layer, e_layer, 0.5*P)
    geompy.TranslateDXDYDZ(Box_19, +0.5*P + L + e_frame, -0.5*P - L - e_frame -
e_layer, 0.)
    Box_20 = geompy.MakeBoxDXDYDZ(e_frame, e_layer, 0.5*P)
    geompy.TranslateDXDYDZ(Box_20, +0.5*P + L, -0.5*P - L - e_frame - e_layer, 0.)

    Box_28 = geompy.MakeBoxDXDYDZ(L, L, 0.5*P)
    geompy.TranslateDXDYDZ(Box_28, +0.5*P, -0.5*P - L, 0.)
    Box_29 = geompy.MakeBoxDXDYDZ(e_frame, L, 0.5*P)
    geompy.TranslateDXDYDZ(Box_29, +0.5*P + L, -0.5*P - L, 0.)
    Box_30 = geompy.MakeBoxDXDYDZ(e_layer, L, 0.5*P)
    geompy.TranslateDXDYDZ(Box_30, +0.5*P + L + e_frame, -0.5*P - L, 0.)
    Box_31 = geompy.MakeBoxDXDYDZ(L, e_frame, 0.5*P)
    geompy.TranslateDXDYDZ(Box_31, +0.5*P, -0.5*P - L - e_frame, 0.)
    Box_32 = geompy.MakeBoxDXDYDZ(L, e_layer, 0.5*P)
    geompy.TranslateDXDYDZ(Box_32, +0.5*P, -0.5*P - L - e_frame - e_layer, 0.)

    # domain such that -h <= z <= 0
    Box_21 = geompy.MakeBoxDXDYDZ(e_layer, e_frame, h)
    geompy.TranslateDXDYDZ(Box_21, +0.5*P + L + e_frame, -0.5*P - L - e_frame, -h)
    Box_22 = geompy.MakeBoxDXDYDZ(e_layer, e_layer, h)
    geompy.TranslateDXDYDZ(Box_22, +0.5*P + L + e_frame, -0.5*P - L - e_frame -
e_layer, -h)
    Box_23 = geompy.MakeBoxDXDYDZ(e_frame, e_layer, h)
    geompy.TranslateDXDYDZ(Box_23, +0.5*P + L, -0.5*P - L - e_frame - e_layer, -h)

    Box_33 = geompy.MakeBoxDXDYDZ(L, L, h)
    geompy.TranslateDXDYDZ(Box_33, +0.5*P, -0.5*P - L, -h)
    Box_34 = geompy.MakeBoxDXDYDZ(e_layer, L, h)
    geompy.TranslateDXDYDZ(Box_34, +0.5*P + L + e_frame, -0.5*P - L, -h)
    Box_35 = geompy.MakeBoxDXDYDZ(L, e_layer, h)
    geompy.TranslateDXDYDZ(Box_35, +0.5*P, -0.5*P - L - e_frame - e_layer, -h)

    # domain such that -h - L_cyl <= z <= -h
    Box_24 = geompy.MakeBoxDXDYDZ(e_frame, e_frame, L_cyl)
    geompy.TranslateDXDYDZ(Box_24, +0.5*P + L, -0.5*P - L - e_frame, -h - L_cyl)
    Box_25 = geompy.MakeBoxDXDYDZ(e_layer, e_frame, L_cyl)
    geompy.TranslateDXDYDZ(Box_25, +0.5*P + L + e_frame, -0.5*P - L - e_frame, -h -
L_cyl)
    Box_26 = geompy.MakeBoxDXDYDZ(e_layer, e_layer, L_cyl)
    geompy.TranslateDXDYDZ(Box_26, +0.5*P + L + e_frame, -0.5*P - L - e_frame -
e_layer, -h - L_cyl)
    Box_27 = geompy.MakeBoxDXDYDZ(e_frame, e_layer, L_cyl)
    geompy.TranslateDXDYDZ(Box_27, +0.5*P + L, -0.5*P - L - e_frame - e_layer, -h -
L_cyl)
```

```
        Box_36 = geompy.MakeBoxDXDYDZ(L, L, L_cyl)
        geompy.TranslateDXDYDZ(Box_36, +0.5*P, -0.5*P - L, -h - L_cyl)
        Box_37 = geompy.MakeBoxDXDYDZ(e_frame, L, L_cyl)
        geompy.TranslateDXDYDZ(Box_37, +0.5*P + L, -0.5*P - L, -h - L_cyl)
        Box_38 = geompy.MakeBoxDXDYDZ(e_layer, L, L_cyl)
        geompy.TranslateDXDYDZ(Box_38, +0.5*P + L + e_frame, -0.5*P - L, -h - L_cyl)
        Box_39 = geompy.MakeBoxDXDYDZ(L, e_frame, L_cyl)
        geompy.TranslateDXDYDZ(Box_39, +0.5*P, -0.5*P - L - e_frame, -h - L_cyl)
        Box_40 = geompy.MakeBoxDXDYDZ(L, e_layer, L_cyl)
        geompy.TranslateDXDYDZ(Box_40, +0.5*P, -0.5*P - L - e_frame - e_layer,-h - L_cyl
    )

        #
        # build a compound from the different parts of the fluid domain surrounding the
    top grid (z >= -h)
        #
        Compound_12 = geompy.MakeCompound([Box_17, Box_18, Box_19, Box_20, Box_28,
    Box_29, Box_30, Box_31, Box_32, Box_33, Box_34, Box_35, Box_21, Box_22, Box_23])

        # relfect the fluid domain surrounding the top grid
        Mirror_8 = geompy.MakeMirrorByPlane(Compound_12, Plane_1)

        #
        # build a compound from the different parts of the fluid domain surrounding the
    cylinders + the two grids
        #
        Compound_13 = geompy.MakeCompound([Box_24, Box_25, Box_26, Box_27, Box_36,
    Box_37, Box_38, Box_39, Box_40, Compound_12, Mirror_8])

        #
        # remove duplicated edges and faces
        #
        Glue_5 = geompy.MakeGlueEdges(Compound_13, 1e-07)
        Pattern = geompy.MakeGlueFaces(Glue_5, 1e-07)

        # return the built corner pattern
        return Pattern

# The following class is a Builder: an instance of this class builds a new
# shape by applying a transformation to a given shape
class TransformationBuilder:
    # constructor
    def __init__(self, i, j):
        self._i = i
        self._j = j

    # apply the transformation to a given shape to build a new one
    def __call__(self, shape):
        return geompy.MakeTranslation(shape, self._i*P, self._j*P, 0.)

# Assembly Builder
class AssemblyBuilder():
    """An instance of this class is able to build the following assembly: the fluid
    surrounding a NxN cylinders,
       fuel assembly, including the two grids and the spheres at the extremities of each
     cylinder.

       The spheres on top of the top grid lie in the domain {(x, y, z): |x| <= N*P/2, |y
    | <= N*P/2, 0 <= z <= P/2}.
       The top grid lies in the domain {(x, y, z): |x| <= (N + 1)*P/2 + e_layer, |y| <=
    (N + 1)*P/2 + e_layer, -h <= z <= 0}.
       The cylinders lie in the domain {(x, y, z): |x| <= N*P/2, |y| <= N*P/2, -(h +
    L_cyl) <= z <= -h}.
       The bottom grid lies in the domain {(x, y, z): |x| <= (N + 1)*P/2 + e_layer, |y|
    <= (N + 1)*P/2 + e_layer, -(2*h + L_cyl) <= z <= -(h + L_cyl)}.
       The spheres under the bottom of the bottomm grid lie in the domain {(x, y, z):  |
    x| <= N*P/2, |y| <= N*P/2, -(P/2 + 2*h + L_cyl) <= z <= -(2*h + L_cyl)}.

       The builder first builds a fluid box, surrounding the assembly, lying in the
```

```
    domain {(x, y, z): |x| <= (N + 1)*P/2 + e_layer, |y| <= (N + 1)*P/2 + e_layer, -(P/2
     + 2*h + L_cyl) <= z <= P/2}.
       The builder thes complete the fluid domain to take into account the water tunnel
    borders and some fluid upstream and downstream of the fluid box.
       Finally the builder builds a fluid domain lying in the domain {(x, y, z): |x| <=
    w_tunnel/2, |y| <= w_tunnel/2, -(P/2 + 2*h + L_cyl + h_down) <= z <= P/2 + h_up}.

       Note that the plane {(x, y, z): z = -(h + L_cyl/2)} is a plane of symmetry of the
     assembly, but not of the complete fluid domain if h_up /= h_down.
       """
    # constructor
    def __init__(self, N):
        self._N = N

        # is N even?
        N_is_even = (self._N % 2 == 0)

        #
        # transformation to apply to the cylinder pattern to build each part of the
    assembly
        #

        l_index = []
        k = self._N // 2
        for i in range(self._N):
            l_index.append(-k + i)

        l_shift = copy.deepcopy(l_index)

        # * if N is odd, a k such that N = 2*k + 1 exists and
        #   l_shift = [-k, ..., 0, ..., k]
        # * if N is even, a k such that N = 2*k exists and
        #   l_shift = [-k, ..., 0, ..., k -1]

        # The cylinder pattern is duplicated and moved in the union of the domains {(x,
    y, z): -i*P/2 <= x <= +i*P/2, -j*P/2 <= y <= +j*P/2}, for i, j in l_shift.
        # The bundle of cylinder patterns lies in the domain:
        #     * {(x, y, z): -k*P/2 <= x <= +k*P/2, -k*P/2 <= y <= +k*P/2}, if N is odd,
        #     * {(x, y, z): -k*P/2 <= x <= +(k - 1)*P/2, -k*P/2 <= y <= +(k - 1)*P/2},
    if N is even.
        # Consequently,
        #     * the bundle of cylinders is symmetric with respect to the planes {(x, y,
    z): x = 0} and {(x, y, z): y = 0}, if N is odd,
        #     * the bundle of cylinders is symmetric with respect to the planes {(x, y,
    z): x = -P/2} and {(x, y, z): y = -P/2}, if N is even.
        # We want the assembly to be symmetric with respect to the planes {(x, y, z): x
    = 0} and {(x, y, z): y = 0}, for all N.
        # We thus have to apply an additional translation: (x, y, z) -> (x + P/2, y + P
    /2, z) to the assembly, if N is even.
        # To do so, we apply this translation to each pattern used to build the assembly
    , if N is even.

        # take into account the additional translation: (x, y, z) -> (x + P/2, y + P/2,
    z), if N is even

        if N_is_even:
            for i in range(self._N):
                l_shift[i] += 0.5

        self._transformations_cylinder = []
        self._names_cylinder = []
        for i in range(self._N):
            for j in range(self._N):
                transformation = TransformationBuilder(l_shift[i], l_shift[j])
                self._transformations_cylinder.append(transformation)
                name = (l_index[i], l_index[j])
                self._names_cylinder.append(name)

        #
        # transformation to apply to the frame pattern to build the part of the fluid
```

```
    domain in the domain {(x, y, z): N*P/2 <= x <= N*P/2 + e_frame + e_layer, -N*P/2 <=
    y <= +N*P/2}
        #

        self._transformations_frame = []
        # initialisation: assume N is odd
        i = k
        # if N is even
        if N_is_even:
            # the domain is not symmetric
            i = k - 1
            # take into account the additional translation
            i += 0.5
        for j in l_shift:
            transformation = TransformationBuilder(i, j)
            self._transformations_frame.append(transformation)

        #
        # transformation to apply to the corner
        #

        # initialisation: assume N is odd
        i = k
        j = -k
        # if N is even
        if N_is_even:
            # the domain is not symmetric
            i = k - 1
            # take into account the additional translation
            i += 0.5
            j += 0.5
        self._transformation_corner = TransformationBuilder(i, j)

        #
        # transformations to apply to the spacer band, aligned with the Ox axis
        #

        self._transformations_spacer_band_x = []
        for j in l_shift:
            transformation = TransformationBuilder(0, j)
            self._transformations_spacer_band_x.append(transformation)

        #
        # transformations to apply to the spacer band, aligned with the Ox axis
        #

        self._transformations_spacer_band_y = []
        for i in l_shift:
            transformation = TransformationBuilder(i, 0)
            self._transformations_spacer_band_y.append(transformation)

    def getResult(self):
        """This method builds the assembly and return the built object."""

        #
        # build the cylinder pattern
        #

        # new cylinder pattern builder
        CPB = CylinderPatternBuilder()

        # build the cylinder pattern
        CylinderPattern = CPB.getResult()

        parts = []
        # loop over the parts
        for transformation in self._transformations_cylinder:
            # apply the transformation to the pattern, to build the new the part and out
     it at its final position
            part = transformation(CylinderPattern)
```

```
            # append the part to the list of parts
            parts.append(part)

        # build the assembly
        Assembly = geompy.MakeCompound(parts)

        #
        # remove duplicated edges and faces
        #
        Assembly = geompy.MakeGlueEdges(Assembly, 1e-07)
        Assembly = geompy.MakeGlueFaces(Assembly, 1e-07)

        #
        # build the frame pattern
        #

        # new frame pattern builder
        FPB = FramePatternBuilder()

        # buiild the frame pattern
        FramePattern = FPB.getResult()

        parts_2 = []
        # loop over the parts
        for transformation in self._transformations_frame:
            # apply the transformation to the pattern, to build the new the part and put
     it at its final position
            part = transformation(FramePattern)
            # append the part to the list of parts
            parts_2.append(part)

        # build a compound from the parts
        Compound_11 = geompy.MakeCompound(parts_2)

        #
        # build the three other quarters by rotation
        #
        Rotation_10 = geompy.MakeRotation(Compound_11, OZ, 0.5*math.pi)
        Rotation_11 = geompy.MakeRotation(Compound_11, OZ, math.pi)
        Rotation_12 = geompy.MakeRotation(Compound_11, OZ, 1.5*math.pi)

        #
        # build the corner pattern
        #

        # new corner pattern builder
        CPB = CornerPatternBuilder()

        # build the corner pattern
        CornerPattern = CPB.getResult()

        geompy.addToStudy( CylinderPattern, 'CylinderPattern' )
        geompy.addToStudy( FramePattern, 'FramePattern' )
        geompy.addToStudy( CornerPattern, 'CornerPattern' )

        # apply the transformation to the corner pattern to put the corner at its final
    position
        Corner = self._transformation_corner(CornerPattern)

        #
        # build the three other corners by rotation
        #
        Rotation_13 = geompy.MakeRotation(Corner, OZ, 0.5*math.pi)
        Rotation_14 = geompy.MakeRotation(Corner, OZ, math.pi)
        Rotation_15 = geompy.MakeRotation(Corner, OZ, 1.5*math.pi)

        # build a compound for the fluid box surrounding the assembly
        Compound_12 = geompy.MakeCompound([Assembly, Compound_11, Rotation_10,
    Rotation_11, Rotation_12, Corner, Rotation_13, Rotation_14, Rotation_15])
```

```
    # look for the faces of Fluid_domain lying in the plane {(x, y, z): x=(N + 1)*P
/2 + e_layer}
    Axis_1 = geompy.MakeTranslation(OX, 0.5*(self._N + 1)*P + e_layer, 0., 0.)
    Faces = geompy.GetShapesOnPlane(Compound_12, geompy.ShapeType["FACE"], Axis_1,
GEOM.ST_ON)
    # build a compound from these faces
    Compound_13 = geompy.MakeCompound(Faces)

    # extrude these faces from the fluid box surrounding the assembly to the water
tunnel border lying in the plane {(x, y, z): x=w_tunnel/2}
    Extrusion_1 = geompy.MakePrismVecH(Compound_13, Axis_1, w_channel)

    # N.B. Extrustion_1 lies in the domain {(x, y, z): (N + 1)*P/2 + e_layer <= x <=
 w_tunnel/2,  |y| <= (N + 1)*P/2 + e_layer, -(P/2 + 2*h + L_cyl) <= z <= P/2}

    # look for the faces of Exrtusion_1 lying in the plane {(x, y, z): y=-(N + 1)*P
/2 + e_layer}
    Axis_2 = geompy.MakeRotation(Axis_1, OZ, -0.5*math.pi)
    Faces = geompy.GetShapesOnPlane(Extrusion_1, geompy.ShapeType["FACE"], Axis_2,
GEOM.ST_ON)
    # build a compound from these faces
    Compound_14 = geompy.MakeCompound(Faces)

    # extrude these faces from the Extrusion_1 to the water tunnel border lying in
the plane {(x, y, z): y=-w_tunnel/2}
    Extrusion_2 = geompy.MakePrismVecH(Compound_14, Axis_2, w_channel)

    # N.B. Extrustion_2 lies in the domain {(x, y, z): (N + 1)*P/2 + e_layer <= x <=
 w_tunnel/2,  -w_tunnel/2 <= y <= -(N + 1)*P/2 + e_layer, -(P/2 + 2*h + L_cyl) <= z
<= P/2}

    # build the part of the fluid domain lying in the domain {(x, y, z): (N + 1)*P/2
 + e_layer <= x <= w_tunnel/2, -w_tunnel/2 <= y <= +(N + 1)*P/2 + e_layer, -(P/2 +
2*h + L_cyl) <= z <= P/2}
    Compound_14 = geompy.MakeCompound([Extrusion_1, Extrusion_2])

    #
    # complete the domain between the fluid box surrounding the assembly and the
water tunnel borders by rotation
    #
    Rotation_16 = geompy.MakeRotation(Compound_14, OZ, 0.5*math.pi)
    Rotation_17 = geompy.MakeRotation(Compound_14, OZ, math.pi)
    Rotation_18 = geompy.MakeRotation(Compound_14, OZ, 1.5*math.pi)

    # build the part of the fluid domain lying in the domain {(x, y, z): |x| <=
w_tunnel/2, |y| <= w_tunnel/2, -(P/2 + 2*h + L_cyl) <= z <= P/2}
    Compound_15 = geompy.MakeCompound([Compound_12, Compound_14, Rotation_16,
Rotation_17, Rotation_18])

    # look for the faces of Compound_15 lying in the plane {(x, y, z): z=+P/2}
    Axis_3 = geompy.MakeTranslation(OZ, 0., 0., 0.5*P)
    Faces = geompy.GetShapesOnPlane(Compound_15, geompy.ShapeType["FACE"], Axis_3,
GEOM.ST_ON)
    # build a compound from these faces
    Compound_16 = geompy.MakeCompound(Faces)

    # extrude these faces of Compound_15 in order to build the part of the water
tunnel upstream of the fluid box surrounding the assembly
    Extrusion_3 = geompy.MakePrismVecH(Compound_16, Axis_3, h_up)

    # N.B. Extrusion_3 lies in the domain {(x, y, z): |x| <= w_tunnel/2, |y| <=
w_tunnel/2, P/2 <= z <= P/2 + h_up}

    # look for the faces of Compound_15 lying in the plane {(x, y, z): z=-(P/2 + 2*h
 + L_cyl)}
    Axis_4 = geompy.MakeMirrorByPlane(Axis_3, Plane_1)
    Faces = geompy.GetShapesOnPlane(Compound_15, geompy.ShapeType["FACE"], Axis_4,
GEOM.ST_ON)
    # build a compound from these faces
    Compound_17 = geompy.MakeCompound(Faces)
```

```
    # extrude these faces of Compound_15 in order to build the part of the water
tunnel downstream of the fluid box surrounding the assembly
    Extrusion_4 = geompy.MakePrismVecH(Compound_17, Axis_4, h_down)

    # N.B. Extrusion_4 lies in the domain {(x, y, z): |x| <= w_tunnel/2, |y| <=
w_tunnel/2, -(P/2 + 2*h + L_cyl + h_down)<= z <= -(P/2 + 2*h + L_cyl)}

    # build a compound for the fluid domain
    Fluid_domain = geompy.MakeCompound([Compound_15, Extrusion_3, Extrusion_4])

    #
    # remove duplicated edges and faces
    #
    Fluid_domain = geompy.MakeGlueEdges(Fluid_domain, 1e-07)
    Fluid_domain = geompy.MakeGlueFaces(Fluid_domain, 1e-07)

    return Fluid_domain

def getGroups(self, Fluid_domain):
    """This method builds and return groups from a given fluid domain, built by an
instance of this class.

       The groups built are:
           - assembly_walls, the walls of the assembly,
           - tunnel_walls, the walls of the water tunnel,
           - walls, union of the two sets of walls,
           - inlet, the inlet of the water tunnel,
           - outlet, the outlet of the water tunnel.
    """

    ####
    #
    # Create group of faces on the boundary of each cylinder +  the two grids
    #
    ####

    # build a group to contain the faces on the walls of the assembly
    assembly_walls = geompy.CreateGroup(Fluid_domain, geompy.ShapeType["FACE"])

    # define the domain, wrt the reference cylinder pattern

    # look for the faces in the domain {(x, y, z): |x| <= r, |y| <= r, -h <= z <= 0}

    # make a box corresponding to this domain
    Box_1 = geompy.MakeBoxDXDYDZ(2.*r, 2.*r, h)
    geompy.TranslateDXDYDZ(Box_1, -r, -r, -h)

    groups_cylinder={}

    # loop over the parts
    for i_cylinder, transformation in enumerate(self._transformations_cylinder):
        # build the axis of the cylinder corresponding to the current part
        axis_cyl = transformation(OZ)

        #
        # create group of faces on the cylinder
        #

        name = self._names_cylinder[i_cylinder]
        groups_cylinder[name] = geompy.CreateGroup(Fluid_domain, geompy.ShapeType["
FACE"])


        # look for the faces on the cylinder
        l_IDs = geompy.GetShapesOnCylinderIDs(Fluid_domain, geompy.ShapeType["FACE"
], axis_cyl, r_cyl, GEOM.ST_ON)

        # add these faces to the group assembly_walls
        geompy.UnionIDs(assembly_walls, l_IDs)
```

```
        geompy.UnionIDs(groups_cylinder[name], l_IDs)

        #
        # create group of faces on the top grid
        #

        # build the box corresponding to the current part
        Box_part = transformation(Box_1)

        # look for the faces in and on the box
        l_IDs = geompy.GetShapesOnBoxIDs(Box_part, Fluid_domain, geompy.ShapeType["
FACE"], GEOM.ST_ONIN)

        # add these faces to the group assembly_walls
        geompy.UnionIDs(assembly_walls, l_IDs)

        #
        # create group of faces on sphere on the top of the top grid
        #

        # build the center of the sphere corresponding to the current part
        Center_part = transformation(O)

        # look for the faces on the sphere on top of the grid
        l_IDs = geompy.GetShapesOnSphereIDs(Fluid_domain, geompy.ShapeType["FACE"],
Center_part, r, GEOM.ST_ON)

        # add these faces to the group assembly_walls
        geompy.UnionIDs(assembly_walls, l_IDs)

        #
        # create group of faces on the bottom grid
        #

        # Idea: apply the mirror reflection to the shapes used to find faces of the
first grid

        # reflect Box_part
        Mirror_3 = geompy.MakeMirrorByPlane(Box_part, Plane_1)

        # look for the faces in and on the box
        l_IDs = geompy.GetShapesOnBoxIDs(Mirror_3, Fluid_domain, geompy.ShapeType["
FACE"], GEOM.ST_ONIN)

        # add these faces to the group assembly_walls
        geompy.UnionIDs(assembly_walls, l_IDs)

        # reflect the center of the sphere
        Mirror_6 = geompy.MakeMirrorByPlane(Center_part, Plane_1)

        # look for the faces on the sphere on top of the grid
        l_IDs = geompy.GetShapesOnSphereIDs(Fluid_domain, geompy.ShapeType["FACE"],
Mirror_6, r, GEOM.ST_ON)

        # add these faces to the group assembly_walls
        geompy.UnionIDs(assembly_walls, l_IDs)

    # build a tool to find the faces on the spacer bands, aligned with axis Ox, of
the top grid
    # N.B. We have:
    #          L = P/2 - e_frame
    #      so that:
    #          N*P + 2*L = (N + 1)*P - 2*e_frame
    spacer_band_x = geompy.MakeBoxDXDYDZ(self._N*P + 2.*L, e, h)
    geompy.TranslateDXDYDZ(spacer_band_x, -0.5*(self._N*P + 2.*L), -0.5*e, -h)

    # build a second tool to find the faces on the spacer bands, aligned with axis
Oy, of the top grid
    spacer_band_y = geompy.MakeRotation(spacer_band_x, OZ, 0.5*math.pi)
```

```
    # for each spacer band aligned with the Ox axis
    for transformation in self._transformations_spacer_band_x:
        #
        # create group of faces on the spacer bands of the top grid
        #

        # build the current spacer band
        spacer_band_top = transformation(spacer_band_x)

        # look for the faces on the spacer band
        l_IDs = geompy.GetShapesOnBoxIDs(spacer_band_top, Fluid_domain, geompy.
ShapeType["FACE"], GEOM.ST_ON)

        # add these faces to the group assembly_walls
        geompy.UnionIDs(assembly_walls, l_IDs)

        #
        # create group of faces on the spacer bands of the bottom grid
        #

        # reflect spacer band
        spacer_band_bottom = geompy.MakeMirrorByPlane(spacer_band_top, Plane_1)

        # look for the faces on the spacer band
        l_IDs = geompy.GetShapesOnBoxIDs(spacer_band_bottom, Fluid_domain, geompy.
ShapeType["FACE"], GEOM.ST_ON)

        # add these faces to the group assembly_walls
        geompy.UnionIDs(assembly_walls, l_IDs)

    # for each spacer band aligned with the Oy axis
    for transformation in self._transformations_spacer_band_y:
        #
        # create group of faces on the spacer bands of the top grid
        #

        # build the current spacer band
        spacer_band_top = transformation(spacer_band_y)

        # look for the faces on the spacer band
        l_IDs = geompy.GetShapesOnBoxIDs(spacer_band_top, Fluid_domain, geompy.
ShapeType["FACE"], GEOM.ST_ON)

        # add these faces to the group assembly_walls
        geompy.UnionIDs(assembly_walls, l_IDs)

        #
        # create group of faces on the spacer bands of the bottom grid
        #

        # reflect spacer band
        spacer_band_bottom = geompy.MakeMirrorByPlane(spacer_band_top, Plane_1)

        # look for the faces on the spacer band
        l_IDs = geompy.GetShapesOnBoxIDs(spacer_band_bottom, Fluid_domain, geompy.
ShapeType["FACE"], GEOM.ST_ON)

        # add these faces to the group assembly_walls
        geompy.UnionIDs(assembly_walls, l_IDs)

    # build a tool to find the faces on the frame of the top grid
    Box_28 = geompy.MakeBoxDXDYDZ(e_frame, (self._N + 1)*P - e_frame, h)
    geompy.TranslateDXDYDZ(Box_28, 0.5*(self._N + 1)*P - e_frame, -0.5*(self._N + 1)
*P, -h)

    # loop over the four part of the frame
    for i in range(4):
        # make the box corresponding to the current part of the frame of the top
grid
```

```
        Rotation_16 = geompy.MakeRotation(Box_28, OZ, i*0.5*math.pi)
        # look for the faces on the current part of frame of the top grid
        l_IDs = geompy.GetShapesOnBoxIDs(Rotation_16, Fluid_domain, geompy.ShapeType
["FACE"], GEOM.ST_ON)
        # add these faces to the group assembly_walls
        geompy.UnionIDs(assembly_walls, l_IDs)
        #
        # make the box corresponding to the current part of the frame of the bottom
grid
        Mirror_7 = geompy.MakeMirrorByPlane(Rotation_16, Plane_1)
        # look for the faces on the current part of frame of the bottom grid
        l_IDs = geompy.GetShapesOnBoxIDs(Mirror_7, Fluid_domain, geompy.ShapeType["
FACE"], GEOM.ST_ON)
        # add these faces to the group assembly_walls
        geompy.UnionIDs(assembly_walls, l_IDs)

    # build a group to contain the faces on the inlet of the water tunnel
    inlet = geompy.CreateGroup(Fluid_domain, geompy.ShapeType["FACE"])

    # look for the faces in the plane {(x, y, z): z=+P/2 + h_up}
    Axis_1 = geompy.MakeTranslation(OZ, 0., 0., 0.5*P + h_up)
    l_IDs = geompy.GetShapesOnPlaneIDs(Fluid_domain, geompy.ShapeType["FACE"],
Axis_1, GEOM.ST_ON)
    # add these faces to the group inlet
    geompy.UnionIDs(inlet, l_IDs)

    # build a group to contain the faces on the outlet of the water tunnel
    outlet = geompy.CreateGroup(Fluid_domain, geompy.ShapeType["FACE"])

    # look for the faces in the plane {(x, y, z): z=-(P/2 + 2*h + L_cyl + h_down)}
    Axis_2 = geompy.MakeTranslation(OZ, 0., 0., -(0.5*P + 2.*h + L_cyl + h_down))
    l_IDs = geompy.GetShapesOnPlaneIDs(Fluid_domain, geompy.ShapeType["FACE"],
Axis_2, GEOM.ST_ON)
    # add these faces to the group outlet
    geompy.UnionIDs(outlet, l_IDs)

    # build a group to contain the faces on the walls of the water tunnel
    tunnel_walls = geompy.CreateGroup(Fluid_domain, geompy.ShapeType["FACE"])

    # look for the faces in the plane {(x, y, z): x=+w_tunnel/2}
    Axis_3 = geompy.MakeTranslation(OX, 0.5*w_tunnel, 0., 0.)
    l_IDs = geompy.GetShapesOnPlaneIDs(Fluid_domain, geompy.ShapeType["FACE"],
Axis_3, GEOM.ST_ON)
    # add these faces to the group tunnel_walls
    geompy.UnionIDs(tunnel_walls, l_IDs)

    # look for the faces in the plane {(x, y, z): x=-w_tunnel/2}
    Axis_4 = geompy.MakeTranslation(OX, -0.5*w_tunnel, 0., 0.)
    l_IDs = geompy.GetShapesOnPlaneIDs(Fluid_domain, geompy.ShapeType["FACE"],
Axis_4, GEOM.ST_ON)
    # add these faces to the group tunnel_walls
    geompy.UnionIDs(tunnel_walls, l_IDs)

    # look for the faces in the plane {(x, y, z): y=+w_tunnel/2}
    Axis_5 = geompy.MakeTranslation(OY, 0., 0.5*w_tunnel, 0.)
    l_IDs = geompy.GetShapesOnPlaneIDs(Fluid_domain, geompy.ShapeType["FACE"],
Axis_5, GEOM.ST_ON)
    # add these faces to the group tunnel_walls
    geompy.UnionIDs(tunnel_walls, l_IDs)

    # look for the faces in the plane {(x, y, z): y=-w_tunnel/2}
    Axis_6 = geompy.MakeTranslation(OY, 0., -0.5*w_tunnel, 0.)
    l_IDs = geompy.GetShapesOnPlaneIDs(Fluid_domain, geompy.ShapeType["FACE"],
Axis_6, GEOM.ST_ON)
    # add these faces to the group tunnel_walls
    geompy.UnionIDs(tunnel_walls, l_IDs)

    # union of all the walls
    walls = geompy.UnionGroups(assembly_walls, tunnel_walls)
```

```
        return assembly_walls, tunnel_walls, walls, inlet, outlet, groups_cylinder

    def setMeshSize(self, Shape, Mesh, dr, dth, dz, dz2):
        """This method set the mesh size in a part of a mesh corresponding to a N x N
    assembly.

           Inputs:

           - Shape: the geometrical object to be meshed, corresponding to a N x N
    assembly,
           - Mesh: the mesh obtained from the given geometrical object,
           - dr: mesh size in the radial direction,
           - dth: mesh size in the orthoradial direction,
           - dz: mesh size in the Oz direction, near the grids.
           - dz2: mesh size in the Oz direction, far from the grids.
        """

        # new cylinder pattern builder
        CPB = CylinderPatternBuilder()

        # loop over the parts
        for transformation in self._transformations_cylinder:
            Dx, Dy, Dz = CPB.setMeshSize(Shape, Mesh, dr, dth, dz, dz2, transformation)

        # new frame pattern builder
        FPB = FramePatternBuilder()

        # loop over the parts
        for transformation in self._transformations_frame:
            # for k in {0, 1, 2, 3}
            for k in range(4):
                # we define the composition of the current transformation and a rotation
     of angle k*pi/2, around axis Oz
                transformation_2 = lambda theObject: geompy.MakeRotation(transformation(
    theObject), OZ, 0.5*k*math.pi)

                # we set the mesh size on the frame pattern, taking into account the
    current rotation
                FPB.setMeshSize(Shape, Mesh, Dx, dz, transformation_2)

        ####
        #
        # specify mesh size in the domain between the fluid box surrounding the assembly
     and the water tunnel borders
        #
        ####

        # start the timer to compute the time elapsed in GEOM module, during the
    definition of the mesh
        mesh_def_GEOM_timer.start()

        # list of edges used to define the mesh size in the Ox direction or the Oy
    direction
        Edges_xy = []

        # define point ((w_channel + w_tunnel/2)/2, -w_tunnel/2, 0)
        Point_ref = geompy.MakeVertex(0.5*(w_channel + 0.5*w_tunnel), 0.5*w_tunnel, 0.)
        for k in range(4):
            # apply the rotation of angle k*pi/2, around axis Oz, to this point
            Point = geompy.MakeRotation(Point_ref, OZ, 0.5*k*math.pi)
            # get the edge near this point
            Edge = geompy.GetEdgeNearPoint(Shape, Point)
            # add the edge to the list
            Edges_xy.append(Edge)

        # get the edge near pount ((w_channel + w_tunnel/2)/2, -w_tunnel/2, 0)
        Edge = Edges_xy[0]

        # stop the timer to compute the time elapsed in GEOM module, during the
    definition of the mesh
```

```
    mesh_def_GEOM_timer.stop()

    # the legnth of this edge is w_channel

    # we split this edge in two parts:
    #
    # +---------------+---+
    #         1         2
    #

    # the length of part 1 is w_channel - e_frame, so that part 1 corresponds to
domain {(x, y, z): w_tunnel/2 - w_channel <= x <= w_tunnel/2 - e_frame
    # the length of part 2 is e_frame            , so that part 2 corresponds to
domain {(x, y, z): w_tunnel/2 - e_frame <= x <= w_tunnel/2

    # partition of the [0, 1] semgent corresponding to this partion of the edge
    #
    #           1                   2
    # +--------------------+--------------+
    # 0          1 - e_frame/w_channel    1

    # During the simulation, the ALE module will change the size of part 1.
    # The maximal dispalcement is A. Consequently, the minimal size of this part
    # is w_channel - e_frame - A

    # We want the mesh size in in part 1 to be Dx, when part 1 reaches its minimal
size
    # (i.e. when the displacement is maximal)

    # compute the number of segments in part 1
    nseg = int((w_channel - e_frame - A)/Dx)
    # ensure there is at least one element in the segment
    nseg = max(nseg, 1)

    # We want the mesh size in part 2 to be Dx, during the whole simulation.

    # compute the number of segments in part 2
    nseg2 = int(e_frame/Dx)
    # ensure there is at least one element in the segment
    nseg2 = max(nseg2, 1)

    # start the timer to compute the time elapsed in SMESH module, during the
definition of the mesh
    mesh_def_SMESH_timer.start()

    for Edge in Edges_xy:
        # define the sub-mesh
        #    * Algorithm: "Wire Discretisation"
        #    * Hypothesis: "Fixed Point 1D"
        Regular_1D = Mesh.Segment(geom=Edge)
        Regular_1D.FixedPoints1D([ 0., 1. - e_frame/w_channel, 1. ],[ nseg, nseg2
],[])
        # propagate this definition
        Regular_1D.Propagation()

    # stop the timer to compute the time elapsed in SMESH module, during the
definition of the mesh
    mesh_def_SMESH_timer.stop()

    ####
    #
    # specify mesh size in the domains upstream and downstream of the fluid box
surrounding the assembly
    #
    ####

    # start the timer to compute the time elapsed in GEOM module, during the
definition of the mesh
    mesh_def_GEOM_timer.start()
```

```
        # list of edges used to define the mesh size in the Oz direction
        Edges_z = []

        #
        # get an edge in the domain {(x, y, z): x >= 0, y >= 0, 0 <= z <= h_up}
        #

        # get edge near point (+w_tunnel/2, +w_tunnel/2, +h_up/2)
        Point = geompy.MakeVertex(0.5*w_tunnel, 0.5*w_tunnel, +0.5*h_up)
        Edge = geompy.GetEdgeNearPoint(Shape, Point)
        Edges_z.append(Edge)

        #
        # get an edge in the domain {(x, y, z): x >= 0, y >= 0, -(P/2 + 2*h + L_cyl +
    h_down) <= z <= -(P/2 + 2*h + L_cyl)}
        #

        # get edge near point (+w_tunnel, +w_tunnel/2, -(P/2 + 2*h + L_cyl + h_down/2))
        Point = geompy.MakeVertex(0.5*w_tunnel, 0.5*w_tunnel, -(0.5*P + 2.*h + L_cyl +
    0.5*h_down))
        Edge = geompy.GetEdgeNearPoint(Shape, Point)
        Edges_z.append(Edge)

        # stop the timer to compute the time elapsed in GEOM module, during the
    definition of the mesh
        mesh_def_GEOM_timer.stop()

        #
        # define a sub-mesh on each of these edges
        #

        # start the timer to compute the time elapsed in SMESH module, during the
    definition of the mesh
        mesh_def_SMESH_timer.start()

        for Edge in Edges_z:
            # get the length of the current edge
            length, area, volume = geompy.BasicProperties(Edge)
            # compute the number of segments
            nseg = int(length/dz2)

            # ensure there is at least one element in the segment
            nseg = max(nseg, 1)

            # define the sub-mesh
            #    * Algorithm: "Wire Discretisation"
            #    * Hypothesis: "Number of segments = nseg"
            Regular_1D = Mesh.Segment(geom=Edge) #modif raison égo
            NumberOfSegments_1=Regular_1D.NumberOfSegments(nseg,1,[])
            NumberOfSegments_1.SetScaleFactor( 10 )
            NumberOfSegments_1.SetReversedEdges([])
            # propagate this definition
            Regular_1D.Propagation()

        # stop the timer to compute the time elapsed in SMESH module, during the
    definition of the mesh
        mesh_def_SMESH_timer.stop()

####
#
# build the 5x5 assembly
#
####

# start the timer to compute the time elapsed in the building of the fluid domain
geom_building_timer.start()

# new assembly builder
AB = AssemblyBuilder(N)
```

```
# build the fluid domain surrounding the assembly
Fluid_domain = AB.getResult()

# stop the timer to compute the time elapsed in the building of the fluid domain
geom_building_timer.stop()

# print the time elapsed in the building of the fluid domain
print("+--> Elapsed time in the building of the fluid domain: ", geom_building_timer.
    elapsed() )

# start the timer to compute the time elapsed in the definition of the groups, in the
    geoemtry
geom_def_groups_timer.start()

# get the groups
assembly_walls, tunnel_walls, walls, inlet, outlet, groups_cylinder = AB.getGroups(
    Fluid_domain)

# stop the timer to compute the time elapsed in the definition of the groups, in the
    geometry
geom_def_groups_timer.stop()

# print the time elapsed in the definition of the groups
print("+--> Elapsed time in the definition of the groups, in the geometry: ",
    geom_def_groups_timer.elapsed() )

# show the geometrical objects
geompy.addToStudy( O, 'O' )
geompy.addToStudy( OX, 'OX' )
geompy.addToStudy( OY, 'OY' )
geompy.addToStudy( OZ, 'OZ' )
geompy.addToStudy( Fluid_domain, 'Fluid_domain' )
geompy.addToStudy( assembly_walls, 'assembly_walls' )
geompy.addToStudy( tunnel_walls, 'tunnel_walls' )
geompy.addToStudy( walls, 'walls' )
geompy.addToStudy( inlet, 'inlet' )
geompy.addToStudy( outlet, 'outlet' )
for name in groups_cylinder:

    i=name[0]
    j=name[1]
    cyl_i=str(abs(i))
    cyl_j=str(abs(j))
    if (i>0):
        cyl_i="p"+cyl_i
    if (j>0):
        cyl_j="p"+cyl_j
    if (i<0):
        cyl_i="m"+cyl_i
    if (j<0):
        cyl_j="m"+cyl_j
    if (i==0):
        cyl_i="_"+cyl_i
    if (j==0):
        cyl_j="_"+cyl_j


    geompy.addToStudy(groups_cylinder[name], cyl_i + cyl_j)

####
#
# Define and compute the mesh of the fluid domain
#
####
raffinage_r=4
raffinage_z=2
# mesh size:
#     * near the walls of the cylinders
dr = (1/raffinage_r)*0.5*mm
#     * in the orthoradial direction (no wall condition)
```

```
dth = 0.5*0.83*mm#*0.5
#     * near the grids (wall condition)
dz = (1/raffinage_z)*1.66*mm
#     * far from the grids (no wall condition)
dz2 = 2.*dz#*0.5

# N.B. the mesh sizes above are defined with respect to specific parts of the geoemtry.
#      Nevertheless, they will be affected to other parts of the geometry in order to
    define a structured grid of the fluid domain.

# start the timer to compute the time elapsed in the definition of the mesh
mesh_definition_timer.start()

# new mesh fo the fluid_domain
Mesh_1 = smesh.Mesh(Fluid_domain)

#
# Global parameters
#

#    - 3D
#        * Algorithm: "Hexahedron (i, j, k)"
Hexa_3D = Mesh_1.Hexahedron(algo=smeshBuilder.Hexa)
#    - 2D
#        * Algortihm: "Quadrangle: Mapping"
Quadrangle_2D = Mesh_1.Quadrangle(algo=smeshBuilder.QUADRANGLE)
#    - 1D
#        * Algorithm: "Wire Discretisation"
Regular_1D = Mesh_1.Segment()
#        * Hypothesis: "Number of segments = 1"
Number_of_Segments_1 = Regular_1D.NumberOfSegments(1)

AB.setMeshSize(Shape=Fluid_domain, Mesh=Mesh_1, dr=dr, dth=dth, dz=dz, dz2=dz2)

# stop the timer to compute the time elapsed in the definition of the mesh
mesh_definition_timer.stop()

# print the time elapsed in the definition of the mesh
print("+--> Elapsed time in the definition of the mesh: ", mesh_definition_timer.elapsed
    () )
print("    +--> Elapsed time in GEOM module: ", mesh_def_GEOM_timer.elapsed() )
print("    +--> Elapsed time in SMESH module: ", mesh_def_SMESH_timer.elapsed() )

#
# Compute the mesh
#

# start the timer to compute the time elapsed in the computation of the mesh
mesh_computation_timer.start()

isDone = Mesh_1.Compute()

# stop the timer to compute the time elapsed in the computation of the mesh
mesh_computation_timer.stop()

# print the time elapsed in the computation of the mesh
print("+--> Elapsed time in the computation of the mesh: ", mesh_computation_timer.
    elapsed() )

#
# Add groups from the groups defined in the geometry
#

# start the timer to compute the time elapsed in the definition of the groups, in the
    mesh
mesh_def_groups_timer.start()

Mesh_1.GroupOnGeom(walls, 'walls', SMESH.FACE)
Mesh_1.GroupOnGeom(assembly_walls, 'assembly_walls', SMESH.FACE)
Mesh_1.GroupOnGeom(tunnel_walls, 'tunnel_walls', SMESH.FACE)
```

```
Mesh_1.GroupOnGeom(inlet, 'inlet', SMESH.FACE)
Mesh_1.GroupOnGeom(outlet, 'outlet', SMESH.FACE)
for name in groups_cylinder:

    i=name[0]
    j=name[1]
    cyl_i=str(abs(i))
    cyl_j=str(abs(j))
    if (i>0):
        cyl_i="p"+cyl_i
    if (j>0):
        cyl_j="p"+cyl_j
    if (i<0):
        cyl_i="m"+cyl_i
    if (j<0):
        cyl_j="m"+cyl_j
    if (i==0):
        cyl_i="_"+cyl_i
    if (j==0):
        cyl_j="_"+cyl_j


    Mesh_1.GroupOnGeom(groups_cylinder[name], cyl_i + cyl_j, SMESH.FACE)

# stop the timer to compute the time elapsed in the definition of the groups, in the
    mesh
mesh_def_groups_timer.stop()

# print the time elapsed in the definition of the groups, in the mesh
print("+--> Elapsed time in the definition of the groups, in the mesh: ",
    mesh_def_groups_timer.elapsed() )

#
# Write the mesh in a MED file
#

# start the timer to compute the time consummed to write the mesh in a MED file
mesh_write_timer.start()

Mesh_1.ExportMED( r'/scratch/g42679/ellez_ALE/Ellez_r'+str(raffinage_r)+'adapt_z'+str(
    raffinage_z)+'_dth2_f4.med', auto_groups=0,minor=33,overwrite=1,meshPart=None,
    autoDimension=1)

# stop the timer to compute the time consummed to write the mesh in a MED file
mesh_write_timer.stop()

# print the time consummed to write the mesh in a MED file
print("+--> Time consummed to write the mesh in a MED file: ", mesh_write_timer.elapsed
    ())

if salome.sg.hasDesktop():
  salome.sg.updateObjBrowser()

# stop the global timer
global_timer.stop()

print("-> Stop the global timer")

# print the total elapsed time
print("+--> Total elapsed time: ", global_timer.elapsed() )
```

C.2 Description du modèle numérique Code_Saturne

setup.xml

— Pré-conditions : Le fichier setup.xml doit être placé dans le dossier "DATA" du cas de calcul Code_Saturne.

— Exécution : Le fichier est automatiquement lu au lancement du calcul.
— Post-conditions : Sans objet.

```
<?xml version="1.0" encoding="utf-8"?><Code_Saturne_GUI case="5x5_RANS" solver_version="
    6.0" study="brennilis_ALE" version="2.0">
  <additional_scalars>
    <users/>
  </additional_scalars>
  <analysis_control>
    <output>
      <listing_printing_frequency>1</listing_printing_frequency>
      <mesh id="-1" label="Fluid domain" type="cells">
        <all_variables status="on"/>
        <location>all[]</location>
        <writer id="-1"/>
      </mesh>
      <mesh id="-2" label="Boundary" type="boundary_faces">
        <all_variables status="on"/>
        <location>all[]</location>
        <writer id="-1"/>
      </mesh>
      <mesh id="1" label="assembly_walls" type="boundary_faces">
        <all_variables status="on"/>
        <location>assembly_walls</location>
      </mesh>
      <probe name="1" status="on">
        <probe_x>0.075</probe_x>
        <probe_y>0</probe_y>
        <probe_z>-0.665</probe_z>
      </probe>
      <probe name="2" status="on">
        <probe_x>0.075</probe_x>
        <probe_y>0.0134</probe_y>
        <probe_z>-0.665</probe_z>
      </probe>
      <probe name="3" status="on">
        <probe_x>0.075</probe_x>
        <probe_y>0.0268</probe_y>
        <probe_z>-0.665</probe_z>
      </probe>
      <probe name="4" status="on">
        <probe_x>0.075</probe_x>
        <probe_y>-0.0134</probe_y>
        <probe_z>-0.665</probe_z>
      </probe>
      <probe name="5" status="on">
        <probe_x>0.075</probe_x>
        <probe_y>-0.0268</probe_y>
        <probe_z>-0.665</probe_z>
      </probe>
      <probe name="6" status="on">
        <probe_x>0.075</probe_x>
        <probe_y>0</probe_y>
        <probe_z>-0.498</probe_z>
      </probe>
      <probe name="7" status="on">
        <probe_x>0.075</probe_x>
        <probe_y>0</probe_y>
        <probe_z>-0.331</probe_z>
      </probe>
      <probe name="8" status="on">
        <probe_x>0.075</probe_x>
        <probe_y>0</probe_y>
        <probe_z>-0.832</probe_z>
      </probe>
      <probe name="9" status="on">
        <probe_x>0.075</probe_x>
        <probe_y>0</probe_y>
        <probe_z>-1</probe_z>
      </probe>
```

```
      <probe_format choice="DAT"/>
      <probe_recording_frequency>1</probe_recording_frequency>
      <writer id="-1" label="results">
        <directory name="postprocessing"/>
        <format name="ensight" options="separate_meshes"/>
        <frequency period="time_value">1.0</frequency>
        <output_at_end status="on"/>
        <output_at_start status="on"/>
        <time_dependency choice="transient_coordinates"/>
      </writer>
    </output>
    <profiles/>
    <scalar_balances/>
    <time_averages/>
    <time_parameters>
      <maximum_time>14</maximum_time>
      <property label="CourantNb" name="courant_number"/>
      <property label="FourierNb" name="fourier_number">
        <postprocessing_recording status="off"/>
      </property>
      <time_passing>0</time_passing>
      <time_step_ref>0.00025</time_step_ref>
    </time_parameters>
  </analysis_control>
  <boundary_conditions>
    <boundary label="BC_2" name="1" nature="wall">assembly_walls</boundary>
    <boundary label="BC_3" name="2" nature="inlet">inlet</boundary>
    <boundary label="BC_4" name="3" nature="outlet">outlet</boundary>
    <boundary label="BC_5" name="4" nature="wall">tunnel_walls</boundary>
    <inlet field_id="none" label="BC_3">
      <ale choice="fixed_boundary"/>
      <turbulence choice="hydraulic_diameter">
        <hydraulic_diameter>0.11</hydraulic_diameter>
      </turbulence>
      <velocity_pressure choice="norm" direction="normal">
        <norm>0.8</norm>
      </velocity_pressure>
    </inlet>
    <outlet field_id="none" label="BC_4">
      <ale choice="fixed_boundary"/>
    </outlet>
    <wall field_id="none" label="BC_2">
      <ale choice="fixed_displacement">
        <formula>A=0.0025;
f=4.0;
t_min=7.0;


mesh_x=0;
mesh_y=0;
mesh_z=0;
if (t&gt;=t_min) mesh_x=A*sin(2*pi*f*t)*(1-exp(-0.8*(t-t_min)^3));</formula>
      </ale>
      <velocity_pressure choice="off"/>
    </wall>
    <wall field_id="none" label="BC_5">
      <ale choice="fixed_boundary"/>
      <velocity_pressure choice="off"/>
    </wall>
  </boundary_conditions>
  <calculation_management>
    <block_io/>
    <partitioning/>
    <start_restart>
      <frozen_field status="off"/>
      <restart_rescue>0</restart_rescue>
      <restart_with_auxiliary status="on"/>
    </start_restart>
  </calculation_management>
  <lagrangian model="off"/>
```

```
  <numerical_parameters>
    <gradient_reconstruction choice="0"/>
    <gradient_transposed status="on"/>
    <hydrostatic_pressure status="off"/>
    <pressure_relaxation>1</pressure_relaxation>
    <velocity_pressure_algo choice="simplec">
      <piso_sweep_number>1</piso_sweep_number>
    </velocity_pressure_algo>
    <velocity_pressure_coupling status="off"/>
    <wall_pressure_extrapolation>0</wall_pressure_extrapolation>
  </numerical_parameters>
  <physical_properties>
    <fluid_properties>
      <material choice="Water"/>
      <method choice="Cathare">
        <reference>WaterLiquid</reference>
      </method>
      <property choice="thermal_law" label="Density" name="density">
        <initial_value>1.17862</initial_value>
        <probes_recording status="off"/>
      </property>
      <property choice="constant" label="DiffDyn" name="dynamic_diffusion">
        <initial_value>0.01</initial_value>
        <listing_printing status="off"/>
        <postprocessing_recording status="off"/>
      </property>
      <property choice="thermal_law" label="LamVisc" name="molecular_viscosity">
        <initial_value>1.83e-05</initial_value>
        <probes_recording status="off"/>
      </property>
      <reference_pressure>101325</reference_pressure>
      <reference_temperature>293.15</reference_temperature>
    </fluid_properties>
    <gravity>
      <gravity_x>0</gravity_x>
      <gravity_y>0</gravity_y>
      <gravity_z>0</gravity_z>
    </gravity>
    <notebook/>
    <omega>
      <omega_x>0</omega_x>
      <omega_y>0</omega_y>
      <omega_z>0</omega_z>
    </omega>
  </physical_properties>
  <solution_domain>
    <extrusion/>
    <faces_cutting status="off"/>
    <joining/>
    <mesh_smoothing status="off"/>
    <meshes_list>
      <mesh name="tranche_Ellez_r4adapt_z2_dth2_f4.med"/>
    </meshes_list>
    <periodicity/>
    <thin_walls/>
    <volumic_conditions>
      <zone groundwater_law="off" head_losses="off" id="1" initialization="on" label="
    all_cells" mass_source_term="off" momentum_source_term="off" porosity="off"
    scalar_source_term="off" thermal_source_term="off">all[]</zone>
    </volumic_conditions>
  </solution_domain>
  <thermophysical_models>
    <ale_method status="on">
      <displacement_prediction_alpha>0.5</displacement_prediction_alpha>
      <displacement_prediction_beta>0</displacement_prediction_beta>
      <fluid_initialization_sub_iterations>24000</fluid_initialization_sub_iterations>
      <formula>mesh_viscosity_1 = 1;
xlim=0.0414;
ylim=0.0414;
```

```
zlim1=0.1;
zlim2=-1.4;

xbord=0.0725;
ybord=0.0725;

if (x&gt;=-xlim &amp;&amp; x&lt;=xlim &amp;&amp; y&gt;=-ylim &amp;&amp; y&lt;=ylim &amp
    ;&amp; z&lt;=zlim1 &amp;&amp; z&gt;=zlim2)
mesh_viscosity_1 = 1e10;

if(x&gt;=xbord || x&lt;=-xbord || y&gt;=ybord || y&lt;=-ybord)
mesh_viscosity_1 = 1e10;</formula>
      <implicitation_precision>1e-05</implicitation_precision>
      <max_iterations_implicitation>1</max_iterations_implicitation>
      <mesh_viscosity type="isotrop"/>
      <monitor_point_synchronisation status="off"/>
      <property label="mesh_vi1" name="mesh_viscosity_1"/>
      <stress_prediction_alpha>2</stress_prediction_alpha>
      <variable dimension="3" label="Mesh Velocity" name="mesh_velocity">
        <blending_factor>0</blending_factor>
        <rhs_reconstruction>1</rhs_reconstruction>
      </variable>
    </ale_method>
    <atmospheric_flows model="off"/>
    <compressible_model model="off"/>
    <gas_combustion model="off"/>
    <groundwater_model model="off"/>
    <hgn_model model="off"/>
    <joule_effect model="off"/>
    <porosities/>
    <radiative_transfer model="off"/>
    <reference_values>
      <length/>
    </reference_values>
    <solid_fuels model="off"/>
    <source_terms/>
    <thermal_scalar model="off"/>
    <turbomachinery model="off">
      <joining/>
    </turbomachinery>
    <turbulence model="k-omega-SST">
      <gravity_terms status="on"/>
      <initialization choice="reference_value" zone_id="1"/>
      <property label="TurbVisc" name="turbulent_viscosity">
        <postprocessing_recording status="off"/>
      </property>
      <reference_velocity>0.8</reference_velocity>
      <variable label="k" name="k">
        <blending_factor>0</blending_factor>
        <postprocessing_recording status="off"/>
        <rhs_reconstruction>1</rhs_reconstruction>
        <solver_precision>1e-05</solver_precision>
      </variable>
      <variable label="omega" name="omega">
        <blending_factor>0</blending_factor>
        <postprocessing_recording status="off"/>
        <rhs_reconstruction>1</rhs_reconstruction>
        <solver_precision>1e-05</solver_precision>
      </variable>
      <wall_function>7</wall_function>
    </turbulence>
    <velocity_pressure>
      <initialization>
        <formula zone_id="1">velocity[0] = 0.;
velocity[1] = 0.;
velocity[2] = 0.;</formula>
      </initialization>
      <property label="Stress" name="stress" support="boundary">
        <postprocessing_recording status="off"/>
      </property>
```

```
      <property label="Stress, normal" name="stress_normal" support="boundary">
        <postprocessing_recording status="off"/>
      </property>
      <property label="Stress, tangential" name="stress_tangential" support="boundary">
        <postprocessing_recording status="off"/>
      </property>
      <property label="total_pressure" name="total_pressure">
        <postprocessing_recording status="off"/>
      </property>
      <property label="Yplus" name="yplus" support="boundary"/>
      <variable label="Pressure" name="pressure">
        <rhs_reconstruction>2</rhs_reconstruction>
        <solver_precision>1e-05</solver_precision>
      </variable>
      <variable dimension="3" label="Velocity" name="velocity">
        <blending_factor>1</blending_factor>
        <rhs_reconstruction>1</rhs_reconstruction>
        <solver_precision>1e-05</solver_precision>
      </variable>
    </velocity_pressure>
  </thermophysical_models>
</Code_Saturne_GUI>
```

C.3 Calcul des résultantes sur l'assemblage et les cylindres

forces_local_global_tranche.f90

- Pré-conditions : Le fichier forces_local_global_tranche.f90 doit être placé dans le dossier "SRC" du cas de calcul Code_Saturne.
- Exécution : La procédure Fortran 90 contenue dans le fichier est automatiquement exécutée à chaque instant du calcul.
- Post-conditions : Plusieurs fichiers texte sont écrits dans le dossier "postprocessing" du cas de calcul Code_Saturne.
  - "resultante_globale.dat" contient 8 colonnes : indice du pas de temps, instant final du pas de temps, F_x, F_y, F_z, X, Y et Z. F_x, F_y et F_z sont les composantes de la résultante des efforts exercés par le fluide sur l'assemblage entier. X, Y et Z sont les coordonnées du centre de gravité de l'assemblage entier.
  - Pour $i_x \in \{-2,-1,0,+1,+2\}$ et $i_y \in \{-2,-1,0,+1,+2\}$, le fichier "sgn($i_x$)|$i_x$|_sgn($i_y$)|$i_y$|.dat" contient 8 colonnes : indice du pas de temps, instant final du pas de temps, F_x, F_y, F_z, X, Y et Z. F_x, F_y et F_z sont les composantes de la résultante des efforts exercés par le fluide sur le cylindre identifié par le couple (i_x, i_y). X, Y et Z sont les coordonnées du centre de gravité de ce cylindre. La fonction sgn est définie comme suit :

$$\mathrm{sgn}(x) = \begin{cases} \text{"p"}, \text{si } x < 0 \\ \text{"m"}, \text{si } x > 0 \\ \text{"\_"}, \text{si } x = 0 \end{cases} .$$

```
!-------------------------------------------------------------------------------

!                      Code_Saturne version 6.0-beta
!                      -------------------------
! This file is part of Code_Saturne, a general-purpose CFD tool.
!
! Copyright (C) 1998-2019 EDF S.A.
!
! This program is free software; you can redistribute it and/or modify it under
! the terms of the GNU General Public License as published by the Free Software
! Foundation; either version 2 of the License, or (at your option) any later
! version.
!
! This program is distributed in the hope that it will be useful, but WITHOUT
! ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
```

```
! FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
! details.
!
! You should have received a copy of the GNU General Public License along with
! this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
! Street, Fifth Floor, Boston, MA 02110-1301, USA.

!-------------------------------------------------------------------------------

!===============================================================================
! Purpose:
! -------

!> \file cs_user_extra_operations-global_efforts.f90
!>
!> \brief This function is called at the end of each time step, and has a very
!>  general purpose
!>  (i.e. anything that does not have another dedicated user subroutine)
!>
!> See \subpage cs_user_extra_operations_examples and
!> \subpage cs_user_extra_operations-nusselt_calculation for examples.
!>
!> This is an example of cs_user_extra_operations.f90 which
!> performs global efforts

!-------------------------------------------------------------------------------

!-------------------------------------------------------------------------------
! Arguments
!-------------------------------------------------------------------------------.
!  mode           name          role                                           !
!-------------------------------------------------------------------------------!
!> \param[in]     nvar          total number of variables
!> \param[in]     nscal         total number of scalars
!> \param[in]     dt            time step (per cell)
!-------------------------------------------------------------------------------

subroutine cs_f_user_extra_operations &
 ( nvar   , nscal  ,                                              &
   dt     )

!===============================================================================

!===============================================================================
! Module files
!===============================================================================

use paramx
use dimens, only: ndimfb
use pointe
use numvar
use optcal
use cstphy
use cstnum
use entsor
use lagran
use parall
use period
use ppppar
use ppthch
use ppincl
use mesh
use field
use field_operator
use turbomachinery

!===============================================================================

implicit none
```

```
! Arguments

integer          nvar   , nscal

double precision dt(ncelet)

! Local variables

!< [loc_var_dec]
integer          ifac
integer          ii, indx, indy, indt, islice
integer          ilelt  , nlelt
integer          file_index,N,k
integer,PARAMETER ::  nslice=64
character*20   :: file_name,file_name_x, file_name_y, file_name_z
character*4  :: group_name
character*7  :: group_name_cyl
character*7  :: group_name_x,group_name_y,group_name_z
character*2  :: cyl_ii,cyl_jj,tranche_ind
character*2    :: cyl_i,cyl_j

double precision :: zmax=-1.31,z0=-0.015
DOUBLE PRECISION :: delta

double precision xfor(3)
double precision xg(3)
double precision xfort(3, 0:nslice-1)
double precision area(1)
double precision, dimension(:,:), pointer :: bfprp_for

double precision xnod(3)
double precision xnodt(3,0:64)
integer, allocatable, dimension(:) :: lstelt
!< [loc_var_dec]

!===============================================================================

!===============================================================================
! Initialization
!===============================================================================
!debut resultante
if (iforbr.ge.0) call field_get_val_v(iforbr, bfprp_for)

! Allocate a temporary array for cells or interior/boundary faces selection
allocate(lstelt(max(ncel,nfac,nfabor)))

if(ntcabs.eq.1) then
  if (irangp.eq.0) then
        open(file="resultante_globale.dat",unit=impusr(2))
! write headings to the dat file FX, FY, FZ are the x,y,z efforts
        write(impusr(2),"(8(a,1x))") " TIME STEP  ","  TIME  ","     FX     ","     FY
       ","     FZ     ","    X     ","    Y     ","    Z     "
        close(unit=impusr(2))
  endif
endif



if (iforbr.ge.0) then

  do ii = 1, ndim
    xfor(ii) = 0.d0
    xnod(ii) = 0.d0

 enddo
    area(1) = 0.d0
  !==========
  call getfbr('assembly_walls', nlelt, lstelt)
  !==========
```

```
  do ilelt = 1, nlelt

    ifac = lstelt(ilelt)
    xnod(1)=xnod(1)+cdgfbo(1, ifac)*surfbn(ifac)
    xnod(2)=xnod(2)+cdgfbo(2, ifac)*surfbn(ifac)
    xnod(3)=xnod(3)+cdgfbo(3, ifac)*surfbn(ifac)

! Boundary Area
   area(1) = area(1) + surfbn(ifac)
! Global Efforts
    do ii = 1, ndim
      xfor(ii) = xfor(ii) + bfprp_for(ii, ifac)


    enddo


  enddo
! Compute the global sums of an array of real numbers in case of parellism.
  if (irangp.ge.0) then
    call parrsm(ndim,xfor)
    call parrsm(ndim,xnod)
    call parrsm(1,area)
    xnod(1)=xnod(1)/area(1)
    xnod(2)=xnod(2)/area(1)
    xnod(3)=xnod(3)/area(1)

  if (irangp.eq.0) then
! open the file
    open(file="resultante_globale.dat",unit=impusr(2),position="append")
! print  efforts for current timestep
    write(impusr(2),"(i12,1x,7(e12.5,1x))") ntcabs,ttcabs,xfor(1),xfor(2),xfor(3),xnod
    (1),xnod(2),xnod(3)
    close(unit=impusr(2))
  endif

! close



 endif
endif

!end resultante

!================================================================================
! Example: compute global efforts on a subset of faces
!================================================================================

!  If efforts have been calculated correctly:

! Open file
! ntcabs: Current absolute time step number


do indx=-2,2!-k,N-k-1 ! bouclage sur les crayons N=5 code en dur
    do indy=-2,2!-k,N-k-1
        write(cyl_ii,'(I1)')abs(indx)
      write(cyl_jj,'(I1)')abs(indy)

      if (indx>0) cyl_i='p'//cyl_ii

      if (indy>0) cyl_j='p'//cyl_jj

      if (indx<0) cyl_i='m'//cyl_ii

      if (indy<0) cyl_j='m'//cyl_jj

      if (indx==0) cyl_i='_'//cyl_ii
```

```
      if (indy==0) cyl_j='_'//cyl_jj

      group_name = trim(cyl_j) // trim(cyl_i)
      file_name = trim(group_name // '.dat')

        if(ntcabs.eq.1) then
            if (irangp.eq.0) then
                open(file=file_name,unit=impusr(1))
! write headings to the dat file FX, FY, FZ are the x,y,z efforts
                write(impusr(1),"(8(a,1x))") " TIME STEP  ","  TIME  ","     FX     ","
        FY     ","     FZ     ","    X     ","    Y     ","    Z     "
                close(unit=impusr(1))
            endif
        endif

        if (iforbr.ge.0) then

            do ii = 1, ndim
                xfor(ii) = 0.d0
                xnod(ii) = 0.d0
            enddo
            area(1) = 0.d0
            !==========
            call getfbr(group_name, nlelt, lstelt)
            !==========
print*," group_name= ",group_name," nlelt_resu= ",nlelt
            do ilelt = 1, nlelt

                ifac = lstelt(ilelt)
    xnod(1)=xnod(1) + cdgfbo(1, ifac)*surfbn(ifac)
    xnod(2)=xnod(2) + cdgfbo(2, ifac)*surfbn(ifac)
    xnod(3)=xnod(3) + cdgfbo(3, ifac)*surfbn(ifac)

! Boundary Area
                area(1) = area(1) + surfbn(ifac)
! Global Efforts
                do ii = 1, ndim
                    xfor(ii) = xfor(ii) + bfprp_for(ii, ifac)
                enddo


            enddo
! Compute the global sums of an array of real numbers in case of parellism.
            if (irangp.ge.0) then
                call parrsm(ndim,xfor)
                call parrsm(ndim,xnod)
                call parrsm(1,area)

    xnod(1)=xnod(1)/area(1)
    xnod(2)=xnod(2)/area(1)
    xnod(3)=xnod(3)/area(1)

                if (irangp.eq.0) then
! open the file
                    open(file=file_name,unit=impusr(1),position="append")
! print  efforts for current timestep
                    write(impusr(1),"(i12,1x,7(e12.5,1x))") ntcabs,ttcabs,xfor(1),xfor
    (2),xfor(3),xnod(1),xnod(2),xnod(3)
        close(unit=impusr(1))
                endif

! close

            endif

        endif
!< [End Global Efforts]
    enddo
enddo
```

```
!Tranches
delta=zmax/nslice
do indx=-2,2!-k,N-k-1 ! bouclage sur les crayons N=5 code en dur
    do indy=-2,2!-k,N-k-1
        write(cyl_ii,'(I1)')abs(indx)
      write(cyl_jj,'(I1)')abs(indy)

      if (indx>0) cyl_i='p'//cyl_ii

      if (indy>0) cyl_j='p'//cyl_jj

      if (indx<0) cyl_i='m'//cyl_ii

      if (indy<0) cyl_j='m'//cyl_jj

      if (indx==0) cyl_i='_'//cyl_ii

      if (indy==0) cyl_j='_'//cyl_jj

      !group_name = trim(cyl_j) // trim(cyl_i)
      !file_name = trim(group_name // '.dat')
      group_name_x= trim(cyl_j) // trim(cyl_i) // trim('_') // 'rx'
    group_name_y= trim(cyl_j) // trim(cyl_i) // trim('_')  // 'ry'
    group_name_z= trim(cyl_j) // trim(cyl_i) // trim('_')  // 'rz'
        file_name_x = trim(group_name_x // '.dat')
    file_name_y = trim(group_name_y // '.dat')
    file_name_z = trim(group_name_z // '.dat')


        if (iforbr.ge.0) then

            do ii = 1, ndim
                xfort(ii,:) = 0.d0
            enddo
            !==========
            call getfbr(group_name, nlelt, lstelt)
            !==========

            do ilelt = 1, nlelt

                ifac = lstelt(ilelt)
              !  recuperation des coordonnees du centre de gravite
                 do ii = 1, ndim
                   xg(ii) = cdgfbo(ii, ifac)
                 enddo

            !     calcul de la tranche a laquelle appartient la face
                  islice = int(floor(abs(xg(3)-z0)/delta))

            !     verification de la validite de la tranche
                  if ((islice.lt.0).or.(islice.gt.nslice-1)) then
                    write(6, *) "Tranche invalide... Verifier la routine de post-
    traitement !?"
                    call csexit(1)
                  endif

! Global Efforts
                do ii = 1, ndim
                    xfort(ii,islice) = xfort(ii,islice) + bfprp_for(ii, ifac)
                enddo


            enddo
! Compute the global sums of an array of real numbers in case of parellism.
            if (irangp.ge.0) then
                call parrsm(nslice,xfort(1,:))
                call parrsm(nslice,xfort(2,:))
                call parrsm(nslice,xfort(3,:))
            endif
```

```
            if (irangp.eq.0) then
          ! open the file
                        open(file=file_name_x,unit=impusr(1),position="append")
          ! print  efforts for current timestep
                        write(impusr(1), '(i12,1x)', advance='no') ntcabs
       write(impusr(1), '(e12.5,1x)', advance='no') ttcabs
       do islice=0,nslice-1
             write(impusr(1), "(e12.5,1x)", advance='no') xfort(1,islice)
             enddo
       write(impusr(1), *)
               close(unit=impusr(1))

                        open(file=file_name_y,unit=impusr(1),position="append")
          ! print  efforts for current timestep
                        write(impusr(1), '(i12,1x)', advance='no') ntcabs
       write(impusr(1), '(e12.5,1x)', advance='no') ttcabs
       do islice=0,nslice-1
             write(impusr(1), "(e12.5,1x)", advance='no') xfort(2,islice)
             enddo
       write(impusr(1), *)
               close(unit=impusr(1))

                        open(file=file_name_z,unit=impusr(1),position="append")
          ! print  efforts for current timestep
                        write(impusr(1), '(i12,1x)', advance='no') ntcabs
       write(impusr(1), '(e12.5,1x)', advance='no') ttcabs
       do islice=0,nslice-1
             write(impusr(1), "(e12.5,1x)", advance='no') xfort(3,islice)
             enddo
       write(impusr(1), *)
               close(unit=impusr(1))


        endif

         endif
!< [End Global Efforts]
     enddo
enddo
!Fin tranche

! Deallocate the temporary array
!  If efforts have been calculated correctly:

! Open file
! ntcabs: Current absolute time step number

! nslice=63
! xfort(:,:)=0.d0
! xnodt(:,:)=0.d0
! do indx=-2,2!-k,N-k-1 ! bouclage sur les crayons N=5 code en dur
!     do indy=-2,2!-k,N-k-1
!       do indt=0,nslice
!             write(cyl_ii,'(I1)')abs(indx)
!           write(cyl_jj,'(I1)')abs(indy)
!     if(nslice<10) then
!     write(tranche_ind,'(I1)')abs(nslice)
!     else
!     write(tranche_ind,'(I2)')abs(nslice)
!     endif
!
!           if (indx>0) cyl_i='p'//cyl_ii
!
!           if (indy>0) cyl_j='p'//cyl_jj
!
!           if (indx<0) cyl_i='m'//cyl_ii
!
!           if (indy<0) cyl_j='m'//cyl_jj
```

```
!
!           if (indx==0) cyl_i='_'//cyl_ii
!
!           if (indy==0) cyl_j='_'//cyl_jj
!
!
!         group_name_cyl = trim(cyl_j) // trim(cyl_i) // trim('_') // trim(tranche_ind)
!     group_name_x= trim(cyl_j) // trim(cyl_i) // trim('_') // 'rx'
!     group_name_y= trim(cyl_j) // trim(cyl_i) // trim('_')  // 'ry'
!     group_name_z= trim(cyl_j) // trim(cyl_i) // trim('_')  // 'rz'
!         file_name_x = trim(group_name_x // '.dat')
!     file_name_y = trim(group_name_y // '.dat')
!     file_name_z = trim(group_name_z // '.dat')
!
!             if(ntcabs.eq.1) then
!                 if (irangp.eq.0) then
!                     !open(file=file_name_x,unit=impusr(1))
!                     !write(impusr(1),"(8(a,1x))") " TIME STEP  ","  TIME  ","     T1
       ","     T2     ","     T3     ","    T4     ","    T5     ","    T6     "
!                     !close(unit=impusr(1))
!                     !open(file=file_name_y,unit=impusr(1))
!                     !write(impusr(1),"(8(a,1x))") " TIME STEP  ","  TIME  ","     T1
       ","     T2     ","     T3     ","    T4     ","    T5     ","    T6     "
!                     !close(unit=impusr(1))
!                     !open(file=file_name_z,unit=impusr(1))
!                     !write(impusr(1),"(8(a,1x))") " TIME STEP  ","  TIME  ","     T1
       ","     T2     ","     T3     ","    T4     ","    T5     ","    T6     "
!                     !close(unit=impusr(1))
!                 endif
!             endif
!
!             if (iforbr.ge.0) then
!
!                 do ii = 1, ndim
!
!                     xfort(ii,indt) = 0.d0
!                     xfor(ii)=0.d0
!                     xnod(ii)=0.d0
!                     !xnodt(ii,indt) = 0.d0
!
!                 enddo
!                 area(1) = 0.d0
!                 !==========
!                 call getfbr(group_name_cyl, nlelt, lstelt)
!                 !==========
!
!                 do ilelt = 1, nlelt
!
!                 ifac = lstelt(ilelt)
!         !xnodt(1,indt)=xnodt(1,indt) + cdgfbo(1, ifac)*surfbn(ifac)
!         !xnodt(2,indt)=xnodt(2,indt) + cdgfbo(2, ifac)*surfbn(ifac)
!         !xnodt(3,indt)=xnodt(3,indt) + cdgfbo(3, ifac)*surfbn(ifac)
!
!     ! Boundary Area
!                     area(1) = area(1) + surfbn(ifac)
!     ! Global Efforts
!                     do ii = 1, ndim
!                         !xfort(ii,indt) = xfort(ii,indt) + bfprp_for(ii, ifac)
!                         xfor(ii) = xfor(ii) + bfprp_for(ii, ifac)
!                     enddo
!
!
!                 enddo
!     ! Compute the global sums of an array of real numbers in case of parellism.
!                 if (irangp.ge.0) then
!                     !call parrsm(ndim,xfor)
!         !call parrsm(3*nslice,xfort)
!         call parrsm(ndim,xfor(:))
! !         call parrsm(nslice,xfort(1,:))
! !         call parrsm(nslice,xfort(2,:))
```

```
! !         call parrsm(nslice,xfort(3,:))
!             !call parrsm(nslice,xnodt(1,:))
!         !call parrsm(nslice,xnodt(2,:))
!         !call parrsm(nslice,xnodt(3,:))
!             call parrsm(1,area)
!         !call parssm(nlelt)
!  print*," group_name= ",group_name_cyl,"nlelt= ",nlelt," indt=",indt
!         !xnod(1)=xnod(1)/area(1)
!         !xnod(2)=xnod(2)/area(1)
!         !xnod(3)=xnod(3)/area(1)
!
!
!                 xfort(1,indt)=xfor(1)
!                 xfort(2,indt)=xfor(2)
!                 xfort(3,indt)=xfor(3)
!
!
!                 endif
!
!             endif
!     !< [End Global Efforts]
!         enddo
!     if (iforbr.ge.0) then
!                     if (irangp.eq.0) then
!           ! open the file
!                         open(file=file_name_x,unit=impusr(1),position="append")
!           ! print  efforts for current timestep
!                         write(impusr(1), '(i12,1x)', advance='no') ntcabs
!       write(impusr(1), '(e12.5,1x)', advance='no') ttcabs
!       do islice=1,nslice
!             write(impusr(1), "(e12.5,1x)", advance='no') xfort(1,islice)
!             enddo
!       write(impusr(1), *)
!               close(unit=impusr(1))
!
!                         open(file=file_name_y,unit=impusr(1),position="append")
!           ! print  efforts for current timestep
!                         write(impusr(1), '(i12,1x)', advance='no') ntcabs
!       write(impusr(1), '(e12.5,1x)', advance='no') ttcabs
!       do islice=1,nslice
!             write(impusr(1), "(e12.5,1x)", advance='no') xfort(2,islice)
!             enddo
!       write(impusr(1), *)
!               close(unit=impusr(1))
!
!                         open(file=file_name_z,unit=impusr(1),position="append")
!           ! print  efforts for current timestep
!                         write(impusr(1), '(i12,1x)', advance='no') ntcabs
!       write(impusr(1), '(e12.5,1x)', advance='no') ttcabs
!       do islice=1,nslice
!             write(impusr(1), "(e12.5,1x)", advance='no') xfort(3,islice)
!             enddo
!       write(impusr(1), *)
!               close(unit=impusr(1))
!
!                     endif
!     endif
!     enddo
! enddo
! Deallocate the temporary array

deallocate(lstelt)

return
end subroutine cs_f_user_extra_operations
```

C.4 Calcul des résultantes sur les couronnes

— Pré-conditions : Les fichiers cs_user_boundary_conditions.f90, cs_user_extra_operations.f90, cs_user_modules.f90, user_conf_bundle.h, user_minmax.h, user_recycling_inlet.h doivent être placés dans le dossier "SRC" du cas de calcul Code_Saturne.

— Exécution : Les procédures Fortran 90 contenues dans les fichiers sont automatiquement exécutées à chaque instant du calcul.

— Post-conditions : Plusieurs fichiers texte sont écrits dans le dossier "postprocessing" du cas de calcul Code_Saturne.

  — "resultante_globale.dat" contient 8 colonnes : indice du pas de temps, instant final du pas de temps, F_x, F_y, F_z, X, Y et Z. F_x, F_y et F_z sont les composantes de la résultante des efforts exercés par le fluide sur l'assemblage entier. X, Y et Z sont les coordonnées du centre de gravité de l'assemblage entier.

  — "F_Fp_upper_grid.dat" contient 7 colonnes : instant final du pas de temps, F_x, F_y, F_z, F_x^p, F_y^p et F_z^p. F_x, F_y et F_z sont les composantes de la résultante des efforts exercés par le fluide sur la grille du haut. F_x^p, F_y^p et F_z^p sont les composantes de la résultante des forces de pression exercées sur la grille du haut.

  — "F_Fp_lower_grid.dat" contient 7 colonnes : instant final du pas de temps, F_x, F_y, F_z, F_x^p, F_y^p et F_z^p. F_x, F_y et F_z sont les composantes de la résultante des efforts exercés par le fluide sur la grille du bas. F_x^p, F_y^p et F_z^p sont les composantes de la résultante des forces de pression exercées sur la grille du bas.

  — "F_Fp_tubes.dat" contient 7 colonnes : instant final du pas de temps, F_x, F_y, F_z, F_x^p, F_y^p et F_z^p. F_x, F_y et F_z sont les composantes de la résultante des efforts exercés par le fluide sur le faisceau de cylindres. F_x^p, F_y^p et F_z^p sont les composantes de la résultante des forces de pression exercées sur le faisceau de cylindres.

  — Les cellules qui composent la discrétisation du domaine occupé par le fluide peuvent être triées selon la coordonnée suivant l'axe (Oz) de leur barycentre. Le fichier "z.dat" représente la fonction qui associe à un numéro de coordonnée i_z la coordonnée $z(i_z)$ qui repère un ensemble de faces à la paroi de l'assemblage. La ligne numéro i_z du fichier "z.dat" contient la coordonnée $z(i_z)$.

  — Pour $n \in \{1, 2, 3, 4, 5\}$, le fichier "Force_lin_tot_n.dat" contient les données correspondant à la ligne de cylindres (i_x, i_y), pour $i_x = n-3$ et $i_y \in \{-2, -1, 0, +1, +2\}$. Chaque ligne du fichier est associée à un couple (t_f, i_z), où t_f est l'instant final d'un pas de temps et i_z correspond à une coordonnée $z(i_z)$ suivant l'axe (Oz). Chaque ligne du fichier contient 16 colonnes. La première colonne est l'instant t_f. Pour $i_y \in \{-2, -1, 0, +1, +2\}$, les trois colonnes numérotées $3(i_y + 2) + 2, 3(i_y + 2) + 3, 3(i_y + 2) + 4$ correspondent aux composantes F_x, F_y et F_z de la résultante des efforts exercés par le fluide sur faces à la paroi du cylindre (i_x, i_y), repérées par la coordonnée $z(i_z)$.

  — Pour $n \in \{1, 2, 3, 4, 5\}$, le fichier "Force_lin_p_n.dat" contient les données correspondant à la ligne de cylindres (i_x, i_y), pour $i_x = n - 3$ et $i_y \in \{-2, -1, 0, +1, +2\}$. Chaque ligne du fichier est associée à un couple (t_f, i_z), où t_f est l'instant final d'un pas de temps et i_z correspond à une coordonnée $z(i_z)$ suivant l'axe (Oz). Chaque ligne du fichier contient 16 colonnes. La première colonne est l'instant t_f. Pour $i_y \in \{-2, -1, 0, +1, +2\}$, les trois colonnes numérotées $3(i_y + 2) + 2, 3(i_y + 2) + 3, 3(i_y + 2) + 4$ correspondent aux composantes F_x^p, F_y^p et F_z^p de la résultante des efforts exercés par le fluide sur faces à la paroi du cylindre (i_x, i_y), repérées par la coordonnée $z(i_z)$.

cs_user_boundary_conditions.f90

```
!-------------------------------------------------------------------------------

!                      Code_Saturne version 6.1.1
!                      --------------------------
! This file is part of Code_Saturne, a general-purpose CFD tool.
!
! Copyright (C) 1998-2020 EDF S.A.
!
```

```
! This program is free software; you can redistribute it and/or modify it under
! the terms of the GNU General Public License as published by the Free Software
! Foundation; either version 2 of the License, or (at your option) any later
! version.
!
! This program is distributed in the hope that it will be useful, but WITHOUT
! ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
! FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
! details.
!
! You should have received a copy of the GNU General Public License along with
! this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
! Street, Fifth Floor, Boston, MA 02110-1301, USA.

!-------------------------------------------------------------------------------

!===============================================================================
! Function:
! ---------

!> \file cs_user_boundary_conditions.f90
!>
!> \brief User subroutine which fills boundary conditions arrays
!> (\c icodcl, \c rcodcl) for unknown variables.
!>
!> See \subpage cs_user_boundary_conditions_examples for examples.
!>
!> \section cs_user_boundary_conditions_intro Introduction
!>
!> Here one defines boundary conditions on a per-face basis.
!>
!> Boundary faces may be selected using the \ref getfbr subroutine.
!>
!> \code getfbr(string, nelts, lstelt) \endcode
!>  - string is a user-supplied character string containing selection criteria;
!>  - nelts is set by the subroutine. It is an integer value corresponding to
!>    the number of boundary faces verifying the selection criteria;
!>  - lstelt is set by the subroutine. It is an integer array of size nelts
!>    containing the list of boundary faces verifying the selection criteria.
!>
!> string may contain:
!>  - references to colors (ex.: 1, 8, 26, ...)
!>  - references to groups (ex.: inlet, group1, ...)
!>  - geometric criteria (ex. x < 0.1, y >= 0.25, ...)
!>
!> These criteria may be combined using logical operators (\c and,\c or) and
!> parentheses.
!>
!> \par Example
!> \code 1 and (group2 or group3) and y < 1 \endcode
!> will select boundary faces
!> of color 1, belonging to groups 'group2' or 'group3' and with face center
!> coordinate y less than 1.
!>
!> Operators priority, from highest to lowest:
!>  '( )' > 'not' > 'and' > 'or' > 'xor'
!>
!> Similarly, interior faces and cells can be identified using the \ref getfac
!> and \ref getcel subroutines (respectively). Their syntax are identical to
!> \ref getfbr syntax.
!>
!> For a more thorough description of the criteria syntax, see the user guide.
!>
!>
!> \section bc_types Boundary condition types
!>
!> Boundary conditions may be assigned in two ways.
!>
!>
!> \subsection std_bcs For "standard" boundary conditions:
```

```
!>
!> One defines a code in the \c itypfb
!> array (of dimensions number of boundary faces).
!> This code will then be used by a non-user subroutine to assign the
!> following conditions.
!> The available codes are:
!>  - \c ientre: Inlet
!>  - \c isolib: Free outlet
!>  - \c isymet: Symmetry
!>  - \c iparoi: Wall (smooth)
!>  - \c iparug: Rough wall
!>
!> These integers are defined elsewhere (in paramx.f90 module).
!> Their value is greater than or equal to 1 and less than or  equal to
!> ntypmx (value fixed in paramx.h)
!>
!> In addition, some values must be defined:
!>  - Inlet (more precisely, inlet/outlet with prescribed flow, as the flow
!>           may be prescribed as an outflow):
!>    - Dirichlet conditions on variables other than pressure are mandatory
!>      if the flow is incoming, optional if the flow is outgoing (the code
!>      assigns zero flux if no Dirichlet is specified); thus,
!>      at face \c ifac, for the variable \c ivar: \c rcodcl(ifac, ivar, 1)
!>
!>
!>  - Smooth wall: (= impermeable solid, with smooth friction)
!>    - Velocity value for sliding wall if applicable:
!>                  - \c rcodcl(ifac, iu, 1) = fluid velocity in the x direction
!>                  - \c rcodcl(ifac, iv, 1) = fluid velocity in the y direction
!>                  - \c rcodcl(ifac, iw, 1) = fluid velocity in the z direction
!>    - Specific code and prescribed temperature value at wall if applicable:
!>                  - \c icodcl(ifac, ivar)    = 5
!>                  - \c rcodcl(ifac, ivar, 1) = prescribed temperature
!>    - Specific code and prescribed flux value at wall if applicable:
!>                  - \c icodcl(ifac, ivar)    = 3
!>                  - \c rcodcl(ifac, ivar, 3) = prescribed flux
!>    .
!>    Note that the default condition for scalars (other than k and epsilon)
!>    is homogeneous Neumann.
!>
!>
!>  - Rough wall: (= impermeable solid, with rough friction)
!>    - Velocity value for sliding wall if applicable:
!>                  - \c rcodcl(ifac, iu, 1) = fluid velocity in the x direction
!>                  - \c rcodcl(ifac, iv, 1) = fluid velocity in the y direction
!>                  - \c rcodcl(ifac, iw, 1) = fluid velocity in the z direction
!>    - Value of the dynamic roughness height to specify in
!>                  - \c rcodcl(ifac, iu, 3)
!>    - Value of the scalar roughness height (if required) to specify in
!>                  - \c rcodcl(ifac, iv, 3) (values for iw are not used)
!>    - Specific code and prescribed temperature value at wall if applicable:
!>                  - \c icodcl(ifac, ivar)    = 6
!>                  - \c rcodcl(ifac, ivar, 1) = prescribed temperature
!>    - Specific code and prescribed flux value at rough wall, if applicable:
!>                  - \c icodcl(ifac, ivar)    = 3
!>                  - \c rcodcl(ifac, ivar, 3) = prescribed flux
!>    .
!>    Note that the default condition for scalars (other than k and epsilon)
!>    is homogeneous Neumann.
!>
!>  - Symmetry (= slip wall):
!>    - Nothing to specify
!>
!>  - Free outlet (more precisely free inlet/outlet with prescribed pressure)
!>    - Nothing to prescribe for pressure and velocity. For scalars and
!>      turbulent values, a Dirichlet value may optionally be specified.
!>      The behavior is as follows:
!>          - pressure is always handled as a Dirichlet condition
!>          - if the mass flow is inflowing:
!>              one retains the velocity at infinity
```

```
!>              Dirichlet condition for scalars and turbulent values
!>               (or zero flux if the user has not specified a
!>                Dirichlet value)
!>          - if the mass flow is outflowing:
!>              one prescribes zero flux on the velocity, the scalars,
!>              and turbulent values
!>    .
!>    Note that the pressure will be reset to p0 on the first free outlet
!>    face found.
!>
!>
!> \subsection nonstd_bcs For "non-standard" conditions:
!>
!> Other than (inlet, free outlet, wall, symmetry), one defines
!>  - on one hand, for each face:
!>    - an admissible \c itypfb value (i.e. greater than or equal to 1 and
!>      less than or equal to \c ntypmx; see its value in paramx.h).
!>      The values predefined in paramx.h:
!>      \c ientre, \c isolib, \c isymet, \c iparoi, \c iparug are in this range,
!>      and it is preferable not to assign one of these integers to \c itypfb
!>      randomly or in an inconsiderate manner. To avoid this, one may use
!>      \c iindef if one wish to avoid checking values in paramx.h. \c iindef
!>      is an admissible value to which no predefined boundary condition
!>      is attached.
!>      Note that the \c itypfb array is reinitialized at each time step to
!>      the non-admissible value of 0. If one forgets to modify \c itypfb for
!>      a given face, the code will stop.
!>
!>  - and on the other hand, for each face and each variable:
!>    - a code
!>                      - \c icodcl(ifac, ivar)
!>    - three real values
!>                      - \c rcodcl(ifac, ivar, 1)
!>                      - \c rcodcl(ifac, ivar, 2)
!>                      - \c rcodcl(ifac, ivar, 3)
!>
!> \anchor icodcl \anchor rcodcl
!> The value of \c icodcl is taken from the following:
!>  - 1: Dirichlet      (usable for any variable)
!>  - 3: Neumann        (usable for any variable)
!>  - 4: Symmetry       (usable only for the velocity and components of
!>                       the Rij tensor)
!>  - 5: Smooth wall    (usable for any variable except for pressure)
!>  - 6: Rough wall     (usable for any variable except for pressure)
!>  - 9: Free outlet    (usable only for velocity)
!>  - 13: Dirichlet for the advection operator and
!>        Neumann for the diffusion operator
!>
!> The values of the 3 \c rcodcl components are:
!>  - \c rcodcl(ifac, ivar, 1):
!>     - Dirichlet for the variable          if \c icodcl(ifac, ivar) = 1 or 13
!>     - Wall value (sliding velocity, temp) if \c icodcl(ifac, ivar) = 5
!>     .
!>     The dimension of \c rcodcl(ifac, ivar, 1) is that of the
!>     resolved variable, for instance:
!>        - U (velocity in m/s),
!>        - T (temperature in degrees)
!>        - H (enthalpy in J/kg)
!>        - F (passive scalar in -)
!>  - \c rcodcl(ifac, ivar, 2):
!>       "exterior" exchange coefficient (between the prescribed value
!>                        and the value at the domain boundary)
!>                        rinfin = infinite by default
!>     - For velocities U,                in kg/(m2 s):
!>        \c rcodcl(ifac, ivar, 2) =          (viscl+visct) / d
!>     - For the pressure P,              in  s/m:
!>        \c rcodcl(ifac, ivar, 2) =                      dt / d
!>     - For temperatures T,              in Watt/(m2 degres):
!>        \c rcodcl(ifac, ivar, 2) = Cp*(viscls+visct/turb_schmidt) / d
!>     - For enthalpies H,                in kg /(m2 s):
```

```
!>        \c rcodcl(ifac, ivar, 2) =    (viscls+visct/turb_schmidt) / d
!>     - For other scalars F              in:
!>        \c rcodcl(ifac, ivar, 2) =    (viscls+visct/turb_schmidt) / d
!>            (d has the dimension of a distance in m)
!>
!>  - \c rcodcl(ifac, ivar, 3) if \c icodcl(ifac, ivar) = 3 or 13:
!>      Flux density (< 0 if gain, n outwards-facing normal)
!>     - For velocities U,                in kg/(m s2) = J:
!>        \c rcodcl(ifac, ivar, 3) =         -(viscl+visct) * (grad U).n
!>     - For pressure P,                  in kg/(m2 s):
!>        \c rcodcl(ifac, ivar, 3) =                    -dt * (grad P).n
!>     - For temperatures T,              in Watt/m2:
!>        \c rcodcl(ifac, ivar, 3) = -Cp*(viscls+visct/turb_schmidt) * (grad T).n
!>     - For enthalpies H,                in Watt/m2:
!>        \c rcodcl(ifac, ivar, 3) = -(viscls+visct/turb_schmidt) * (grad H).n
!>     - For other scalars F              in:
!>        \c rcodcl(ifac, ivar, 3) = -(viscls+visct/turb_schmidt) * (grad F).n
!>
!>  - \c rcodcl(ifac, ivar, 3) if \c icodcl(ifac, ivar) = 6:
!>      Roughness for the rough wall law
!>     - For velocities U, dynamic roughness
!>         \c rcodcl(ifac, iu, 3) = roughd
!>     - For other scalars, thermal roughness
!>         \c rcodcl(ifac, iv, 3) = rough
!>
!>
!> Note that if the user assigns a value to \c itypfb equal to \c ientre, \c isolib,
!> \c isymet, \c iparoi, or \c iparug and does not modify \c icodcl (zero value by
!>  default), \c itypfb will define the boundary condition type.
!>
!> To the contrary, if the user prescribes \c icodcl(ifac, ivar) (nonzero),
!> the values assigned to \c rcodcl will be used for the considered face
!> and variable (if \c rcodcl values are not set, the default values will
!> be used for the face and variable, so:
!>                          - \c rcodcl(ifac, ivar, 1) = 0.d0
!>                          - \c rcodcl(ifac, ivar, 2) = rinfin
!>                          - \c rcodcl(ifac, ivar, 3) = 0.d0)
!>
!> Especially, one may have for example:
!>  - set \c itypfb(ifac) = \c iparoi which prescribes default wall
!>    conditions for all variables at face ifac,
!>  - and define IN ADDITION for variable ivar on this face specific
!>    conditions by specifying \c icodcl(ifac, ivar) and the 3 \c rcodcl values.
!>
!> The user may also assign to \c itypfb a value not equal to \c ientre, \c isolib,
!> \c isymet, \c iparoi, \c iparug, \c iindef but greater than or equal to 1 and less
!> than or equal to ntypmx (see values in param.h) to distinguish groups
!> or colors in other subroutines which are specific to the case and in
!> which itypfb is accessible. In this case though it will be necessary
!> to prescribe boundary conditions by assigning values to icodcl and to
!> the 3 \c rcodcl fields (as the value of \c itypfb will not be predefined in
!> the code).
!>
!>
!> \subsection comp_bcs Boundary condition types for compressible flows
!>
!> For compressible flows, only predefined boundary conditions may
!> be assigned among: \c iparoi, \c isymet, \c iesicf, \c isspcf, \c isopcf, \c iephcf,
    \c ieqhcf
!>
!>  - \c iparoi : standard wall
!>  - \c isymet : standard symmetry
!>
!>  - \c iesicf, \c isspcf, \c isopcf, \c iephcf, \c ieqhcf : inlet/outlet
!>
!> For inlets/outlets, we can prescribe
!> a value for turbulence and passive scalars in \c rcodcl(.,.,1)
!> for the case in which the mass flux is incoming. If this is not
!> done, a zero flux condition is applied.
!>
```

```
!> - \c iesicf: prescribed inlet/outlet (for example supersonic inlet)
!>           the user prescribes the velocity and all thermodynamic variables
!> - \c isspcf: supersonic outlet
!>           the user does not prescribe anything
!> - \c isopcf: subsonic outlet with prescribed pressure
!>           the user presribes the pressure
!> - \c iephcf: mixed inlet with prescribed total pressure and enthalpy
!>           the user prescribes the total pressure and total enthalpy
!> - \c ieqhcf: subsonic inlet with prescribed mass and enthalpy flow
!>           to be implemented
!>
!>
!> \subsection cons_rul Consistency rules
!>
!> A few consistency rules between \c icodcl codes for variables with
!> non-standard boundary conditions:
!>
!>  - Codes for velocity components must be identical
!>  - Codes for Rij components must be identical
!>  - If code (velocity or Rij) = 4
!>    one must have code (velocity and Rij) = 4
!>  - If code (velocity or turbulence) = 5
!>    one must have code (velocity and turbulence) = 5
!>  - If code (velocity or turbulence) = 6
!>    one must have code (velocity and turbulence) = 6
!>  - If scalar code (except pressure or fluctuations) = 5
!>    one must have velocity code = 5
!>  - If scalar code (except pressure or fluctuations) = 6
!>    one must have velocity code = 6
!>
!>
!> \remarks
!>   - Caution: to prescribe a flux (nonzero) to Rij, the viscosity to take
!>              into account is viscl even if visct exists
!>              (visct=rho cmu k2/epsilon)
!>   - One have the ordering array for boundary faces from the previous time
!>       step (except for the fist one, where \c itrifb has not been set yet).
!>   - The array of boundary face types \c itypfb has been reset before
!>       entering the subroutine.
!>
!>
!> \subsubsection cs_user_bc_cell_id Cell values of some variables
!>
!> Cell value field ids
!>
!> - Density:                          \c icrom
!> - Dynamic molecular viscosity:      \c iviscl
!> - Turbulent viscosity:              \c ivisct
!> - Specific heat:                    \c icp
!> - Diffusivity(lambda):              \c field_get_key_int(ivarfl(isca(iscal)), &
!>                                        kivisl, ...)
!>
!>
!> \subsubsection fac_id Faces identification
!>
!> - Density:                                \c field id \c ibrom
!> - Boundary mass flux (for convecting \c ivar):
!>     field id \c iflmab
!>     using \c field_get_key_int(ivarfl(ivar), kbmasf, iflmab)
!> - For other values: take as an approximation the value in the adjacent cell
!>                     i.e. as above with \c iel = ifabor(ifac).
!>
!> Please refer to the
!> <a href="../../theory.pdf#boundary"><b>boundary conditions</b></a>
!> section of the theory guide for more informations.
!-------------------------------------------------------------------------------

!-------------------------------------------------------------------------------
! Arguments
!-------------------------------------------------------------------------------.
```

```
!  mode           name          role                                                !
!-----------------------------------------------------------------------------!
!> \param[in]     nvar          total number of variables
!> \param[in]     nscal         total number of scalars
!> \param[out]    icodcl        boundary condition code:
!>                               - 1 Dirichlet
!>                               - 2 Radiative outlet
!>                               - 3 Neumann
!>                               - 4 sliding and
!>                                 \f$ \vect{u} \cdot \vect{n} = 0 \f$
!>                               - 5 smooth wall and
!>                                 \f$ \vect{u} \cdot \vect{n} = 0 \f$
!>                               - 6 rough wall and
!>                                 \f$ \vect{u} \cdot \vect{n} = 0 \f$
!>                               - 9 free inlet/outlet
!>                                 (input mass flux blocked to 0)
!>                               - 13 Dirichlet for the advection operator and
!>                                    Neumann for the diffusion operator
!> \param[in]     itrifb        indirection for boundary faces ordering
!> \param[in,out] itypfb        boundary face types
!> \param[out]    izfppp        boundary face zone number
!> \param[in]     dt            time step (per cell)
!> \param[in,out] rcodcl        boundary condition values:
!>                               - rcodcl(1) value of the dirichlet
!>                               - rcodcl(2) value of the exterior exchange
!>                                 coefficient (infinite if no exchange)
!>                               - rcodcl(3) value flux density
!>                                 (negative if gain) in w/m2 or roughness
!>                                 in m if icodcl=6
!>                                 -# for the velocity \f$ (\mu+\mu_T)
!>                                    \gradt \, \vect{u} \cdot \vect{n}  \f$
!>                                 -# for the pressure \f$ \Delta t
!>                                    \grad P \cdot \vect{n}  \f$
!>                                 -# for a scalar \f$ cp \left( K +
!>                                     \dfrac{K_T}{\sigma_T} \right)
!>                                     \grad T \cdot \vect{n} \f$
!-------------------------------------------------------------------------------

subroutine cs_f_user_boundary_conditions &
 ( nvar   , nscal  ,                                              &
   icodcl , itrifb , itypfb , izfppp ,                            &
   dt     ,                                                       &
   rcodcl )

!===============================================================================

!===============================================================================
! Module files
!===============================================================================

use paramx
use numvar
use optcal
use cstphy
use cstnum
use entsor
use parall
use period
use ppppar
use ppthch
use coincl
use cpincl
use ppincl
use ppcpfu
use atchem
use atincl
use atsoil
use ctincl
use cs_fuel_incl
use mesh
```

```
use field
use turbomachinery
use iso_c_binding
use cs_c_bindings

use user_module

!===============================================================================

implicit none

include "user_minmax.h"
include "user_recycling_inlet.h"
include "user_conf_bundle.h"

! Arguments

integer          nvar   , nscal

integer          icodcl(nfabor,nvar)
integer          itrifb(nfabor), itypfb(nfabor)
integer          izfppp(nfabor)

double precision dt(ncelet)
double precision rcodcl(nfabor,nvar,3)
double precision, allocatable, dimension(:,:) :: vel_inlet

! Local variables
integer iel, idim, ifac
integer ii, jj


! INSERT_VARIABLE_DEFINITIONS_HERE

!integer, allocatable, dimension(:) :: lstelt
integer nentre, nsolib, nparoi, nheated
double precision uref2, xdh, xitur


!===============================================================================

!===============================================================================
! Initialization
!===============================================================================
!allocate(lstelt(nfabor))  ! temporary array for boundary faces selection

! INSERT_ADDITIONAL_INITIALIZATION_CODE_HERE

!===============================================================================
! Assign boundary conditions to boundary faces here

! For each subset:
! - use selection criteria to filter boundary faces of a given subset
! - loop on faces from a subset
!   - set the boundary condition for each face
!===============================================================================

! INSERT_MAIN_CODE_HERE
! minmax
allocate(vel_inlet(nfabor,3))
vel_inlet = 0.d0
do ifac = 1, nfabor
  vel_inlet(ifac,1) = 0.d0
  vel_inlet(ifac,2) = 0.d0
  vel_inlet(ifac,3) = -0.8d0
enddo
ipass_minmax = ipass_minmax + 1
if(ipass_minmax.eq.1) then
  xmin = 1.d20
  xmax = -1.d20
```

```
  ymin = 1.d20
  ymax = -1.d20
  zmin = 1.d20
  zmax = -1.d20
  do ifac = 1, nfabor
    xmin = min(xmin,cdgfbo(1,ifac))
    xmax = max(xmax,cdgfbo(1,ifac))
    ymin = min(ymin,cdgfbo(2,ifac))
    ymax = max(ymax,cdgfbo(2,ifac))
    zmin = min(zmin,cdgfbo(3,ifac))
    zmax = max(zmax,cdgfbo(3,ifac))
  enddo
  if(irangp.ge.0) then
    call parmin(xmin)
    call parmax(xmax)
    call parmin(ymin)
    call parmax(ymax)
    call parmin(zmin)
    call parmax(zmax)
  endif
  write(nfecra,*) "user_boundary xmin xmax ",xmin,xmax
  write(nfecra,*) "user_boundary ymin ymax ",ymin,ymax
  write(nfecra,*) "user_boundary zmin zmax ",zmin,zmax
endif
if(indic_recycling_inlet.eq.1) then
  allocate(vel_recycling_inlet(nfabor,3,n_recycling_inlet))
  vel_recycling_inlet = 0.d0
  iperturb(1) = 1
  ipass_recycling_inlet = ipass_recycling_inlet + 1

  wbulk(1) = -0.8d0
  turbintensity(1) = 0.1d0
  turblengthscale(1) = 0.0375d0
  dz_upward(1) = -(0.3367-0.270033)
  z_recycling_inlet(1) = zmax
  if(ipass_recycling_inlet.eq.1) then
    do icl = 1, n_recycling_inlet
      write(nfecra,*) "************* recycling at the inlet *",icl,"*********** "
      write(nfecra,*) "Recycling Bulk velocity = ",wbulk(icl)
      write(nfecra,*) "Recycling length = ", dz_upward(icl)
      if(iperturb(icl).eq.1) then
        write(nfecra,*) "turbulence intensity for initial SEM = ", turbintensity(icl)
        write(nfecra,*) "Turbulent Length scale for initial SEM = ",turblengthscale(icl)
      endif
      write(nfecra,*) "******************************************************** "
    enddo
  endif
! if dz_upward < 0 ipos = 2 otherwise ipos = 1 (could be implemented dynamically)
  ipos_recycling(1) = 2
  call field_get_key_int(ivarfl(iu), kimasf, iflmas)
  call field_get_val_s(iflmas, imasfl)
  call field_get_val_prev_v(ivarfl(iu), vela)
  call field_get_val_v(ivarfl(iu), vel)
  if(ipass_recycling_inlet.eq.1) then
    allocate(lstelt_recycling_inlet(nfabor,n_recycling_inlet))  ! temporary array for
    boundary faces selection
    allocate(ifaccycloc(nfac,2))
    allocate(iconnectcyc(nfabor,2))
    do icl = 1, n_recycling_inlet
      surfinlet(icl) = 0.d0
      zmin_recycling_inlet(icl) = 1.d20
      zmax_recycling_inlet(icl) = -1.d20
      nlelt_recycling_inlet(icl) = 0
      do ifac = 1, nfabor
! we perform this kind of test but could be replaced by a getfbr
        if(abs(cdgfbo(3,ifac)-z_recycling_inlet(icl)).lt.zeps_recycling_inlet) then
          surfinlet(icl) = surfinlet(icl) + surfbn(ifac)
          zmin_recycling_inlet(icl) = min(zmin_recycling_inlet(icl),cdgfbo(3,ifac))
          zmax_recycling_inlet(icl) = max(zmax_recycling_inlet(icl),cdgfbo(3,ifac))
          nlelt_recycling_inlet(icl) = nlelt_recycling_inlet(icl) + 1
```

```
          lstelt_recycling_inlet(nlelt_recycling_inlet,icl) = ifac
        endif
      enddo
      if(irangp.ge.0) then
        call parsom(surfinlet(icl))
        call parmin(zmin_recycling_inlet(icl))
        call parmax(zmax_recycling_inlet(icl))
      endif
      nleltcycam(icl) = nlelt_recycling_inlet(icl)
      if(irangp.ge.0) call parcpt(nleltcycam(icl))
      write(nfecra,*) "icl nb faces ",icl,nlelt_recycling_inlet(icl)
      write(nfecra,*) "surface zmin zmax ",surfinlet(icl),zmin_recycling_inlet(icl),
    zmax_recycling_inlet(icl)
      if(abs(zmin_recycling_inlet(icl)-zmax_recycling_inlet(icl)).gt.
    zeps_recycling_inlet) then
        write(nfecra,*) "arret non planar inlet surface ",icl,zmin_recycling_inlet(icl),
    zmax_recycling_inlet(icl)
        call csexit(1)
      endif
!
!
      ilelt_recycling_inlet = 0
      if(icl.eq.1) then
        do ifac = 1, nfac
          jj = ifacel(2,ifac)
          if(abs(cdgfac(3,ifac)-(zmin_recycling_inlet(icl)+dz_upward(icl))).lt.
    zeps_recycling_inlet.and.&
             jj.le.ncel) then
            ilelt_recycling_inlet = ilelt_recycling_inlet + 1
          endif
        enddo
      endif
      nleltcycavloc(icl) = ilelt_recycling_inlet
      nleltcycav(icl) = ilelt_recycling_inlet
      if(irangp.ge.0) call parcpt(nleltcycav(icl))
      write(nfecra,*) "nbr faces distantes ",nleltcycav(icl)
      if(nleltcycam(icl).ne.nleltcycav(icl)) then
        write(nfecra,*) "arret dans boundaries icl = ", icl
        write(nfecra,*) "nleltcycam et nleltcycav differents ",nleltcycam(icl),
    nleltcycav(icl)
        call csexit(1)
      endif
      allocate(xcyclocav(nleltcycavloc(icl)))
      allocate(ycyclocav(nleltcycavloc(icl)))
      allocate(xcycglobav(nleltcycav(icl)))
      allocate(ycycglobav(nleltcycav(icl)))
!
      ilelt_recycling_inlet= 0
      if(icl.eq.1) then
        do ifac = 1, nfac
          jj = ifacel(2,ifac)
          if(abs(cdgfac(3,ifac)-(zmin_recycling_inlet(icl)+dz_upward(icl))).lt.
    zeps_recycling_inlet.and.&
             jj.le.ncel) then
            ilelt_recycling_inlet = ilelt_recycling_inlet + 1
            xcyclocav(ilelt_recycling_inlet) = cdgfac(1,ifac)
            ycyclocav(ilelt_recycling_inlet) = cdgfac(2,ifac)
            ifaccycloc(ilelt_recycling_inlet,icl) = ifac
          endif
        enddo
      endif
      if(irangp.ge.0) then
        call paragv(nleltcycavloc(icl),nleltcycav(icl),xcyclocav,xcycglobav)
        call paragv(nleltcycavloc(icl),nleltcycav(icl),ycyclocav,ycycglobav)
      endif
      allocate(nconnectcyc(nfabor))
      istop = 0
      istop2 = 0
      distmin = 1.d20
      do ilelt_recycling_inlet = 1, nlelt_recycling_inlet(icl)
```

```
        ifac = lstelt_recycling_inlet(ilelt_recycling_inlet,icl)
        iconnectcyc(ilelt_recycling_inlet,icl) = 0
        nconnectcyc(ilelt_recycling_inlet) = 0
        do ileltcycav = 1, nleltcycav(icl)
          if(abs(cdgfbo(1,ifac)-xcycglobav(ileltcycav)).lt.xyeps_recycling_inlet.and.&
             abs(cdgfbo(2,ifac)-ycycglobav(ileltcycav)).lt.xyeps_recycling_inlet) then
            iconnectcyc(ilelt_recycling_inlet,icl) = ileltcycav
            nconnectcyc(ilelt_recycling_inlet) = nconnectcyc(ilelt_recycling_inlet) + 1
          endif
        enddo
        if(iconnectcyc(ilelt_recycling_inlet,icl).eq.0) then
          istop = istop + 1
          do ileltcycav = 1, nleltcycav(icl)
            distp = sqrt((cdgfbo(1,ifac)-xcycglobav(ileltcycav))**2+(cdgfbo(2,ifac)-
    ycycglobav(ileltcycav))**2)
            distmin = min(distp,distmin)
          enddo
        endif
        if(nconnectcyc(ilelt_recycling_inlet).gt.1) then
          istop2 = istop2 + 1
        endif
      enddo
      if(distmin.gt.1.d0) distmin = 0.d0
      if(irangp.ge.0) call parmax(distmin)
      if(irangp.ge.0) then
        call parcpt(istop)
        call parcpt(istop2)
      endif
      if(istop.ne.0) then
        write(nfecra,*) "arret du programme icl = ", icl
        write(nfecra,*) "pb de connectivite istop = ",istop
        write(nfecra,*) "augmenter xyeps_recycling_inlet a ",distmin
        call csexit(1)
      endif
      if(istop2.ne.0) then
        write(nfecra,*) "arret du programme icl = ", icl
        write(nfecra,*) "pb de connectivite istop2 = ",istop2
        call csexit(1)
      endif

      deallocate(xcyclocav)
      deallocate(ycyclocav)
      deallocate(xcycglobav)
      deallocate(ycycglobav)
      deallocate(nconnectcyc)
!
    enddo
  endif

  do icl = 1, n_recycling_inlet
    allocate(ucyclocav(nleltcycavloc(icl)))
    allocate(vcyclocav(nleltcycavloc(icl)))
    allocate(wcyclocav(nleltcycavloc(icl)))
    allocate(ucycglobav(nleltcycav(icl)))
    allocate(vcycglobav(nleltcycav(icl)))
    allocate(wcycglobav(nleltcycav(icl)))
    if(iperturb(icl).eq.1.and.ipass_recycling_inlet.eq.3) then
      write(nfecra,*) "perturbation du champ de vitesse ",icl
      zinlet = zmin_recycling_inlet(icl) + dz_upward(icl)
      write(nfecra,*) "zinlet = ", zinlet
      call inisem (ipos_recycling(icl),wbulk(icl),turbintensity(icl),turblengthscale(icl
    ),zinlet)
    endif
    fluxw(icl) = 0.d0
    fluxwfac(icl) = 0.d0
    do ilelt_recycling_inlet = 1, nleltcycavloc(icl)
      ifac = ifaccycloc(ilelt_recycling_inlet,icl)
      ii = ifacel(1,ifac)
      jj = ifacel(2,ifac)
      ucyclocav(ilelt_recycling_inlet) = pond(ifac)*vela(1,ii)+(1.d0-pond(ifac))*vela(1,
```

```
    jj)
      vcyclocav(ilelt_recycling_inlet) = pond(ifac)*vela(2,ii)+(1.d0-pond(ifac))*vela(2,
    jj)
      wcyclocav(ilelt_recycling_inlet) = pond(ifac)*vela(3,ii)+(1.d0-pond(ifac))*vela(3,
    jj)
      surf = sqrt(surfac(1,ifac)**2+surfac(2,ifac)**2+surfac(3,ifac)**2)
      fluxw(icl) = fluxw(icl) + ro0*wcyclocav(ilelt_recycling_inlet)*surf
      fluxwfac(icl) = fluxwfac(icl) + imasfl(ifac) * surfac(3,ifac)/surf
    enddo
    if(irangp.ge.0) then
      call parsom(fluxw(icl))
      call parsom(fluxwfac(icl))
    endif

    if(irangp.ge.0) then
      call paragv(nleltcycavloc(icl),nleltcycav(icl),ucyclocav,ucycglobav)
      call paragv(nleltcycavloc(icl),nleltcycav(icl),vcyclocav,vcycglobav)
      call paragv(nleltcycavloc(icl),nleltcycav(icl),wcyclocav,wcycglobav)
    endif
    if(ipass_recycling_inlet.le.2.and.isuite.eq.0) then
      do ilelt_recycling_inlet = 1, nleltcycav(icl)
        ucycglobav(ilelt_recycling_inlet) = 0.d0
        vcycglobav(ilelt_recycling_inlet) = 0.d0
        wcycglobav(ilelt_recycling_inlet) = wbulk(icl)
      enddo
      fluxw(icl) = wbulk(icl)*ro0*surfinlet(icl)
    endif
    fluxwtarget(icl) = wbulk(icl)*ro0*surfinlet(icl)

    write(nfecra,*) "cs_user_boundary icl fluxtarget fluxw fluxwfac ", icl, fluxwtarget(
    icl), fluxw(icl), fluxwfac(icl)
    do idim = 1, 3
      uvwmin(icl,idim) = 1.d20
      uvwmax(icl,idim) = -1.d20
    enddo

    do ilelt_recycling_inlet = 1, nlelt_recycling_inlet(icl)
      ifac = lstelt_recycling_inlet(ilelt_recycling_inlet,icl)
      iel = ifabor(ifac)
!      itypfb(ifac) = ientre
!      rcodcl(ifac,iu,1) = ucycglobav(iconnectcyc(ilelt_recycling_inlet,icl))
!      rcodcl(ifac,iv,1) = vcycglobav(iconnectcyc(ilelt_recycling_inlet,icl))
!      rcodcl(ifac,iw,1) = wcycglobav(iconnectcyc(ilelt_recycling_inlet,icl))*wbulk(icl)
    *ro0*surfinlet(icl)/fluxw(icl)
      vel_recycling_inlet(ifac,1,icl) = ucycglobav(iconnectcyc(ilelt_recycling_inlet,icl
    ))
      vel_recycling_inlet(ifac,2,icl) = vcycglobav(iconnectcyc(ilelt_recycling_inlet,icl
    ))
      vel_recycling_inlet(ifac,3,icl) = wcycglobav(iconnectcyc(ilelt_recycling_inlet,icl
    ))*wbulk(icl)*ro0*surfinlet(icl)/fluxw(icl)
      do idim = 1, 3
        uvwmin(icl,idim) = min(uvwmin(icl,idim),vel_recycling_inlet(ifac,idim,icl))
        uvwmax(icl,idim) = max(uvwmax(icl,idim),vel_recycling_inlet(ifac,idim,icl))
      enddo
    enddo
    do idim = 1, 3
      call parmin(uvwmin(icl,idim))
      call parmax(uvwmax(icl,idim))
      write(nfecra,*) "user boundaries uvw min max ",icl,idim,uvwmin(icl,idim),uvwmax(
    icl,idim)
    enddo
    deallocate(ucyclocav)
    deallocate(vcyclocav)
    deallocate(wcyclocav)
    deallocate(ucycglobav)
    deallocate(vcycglobav)
    deallocate(wcycglobav)
!
  enddo
endif
```

```
!
ipass_rods = ipass_rods + 1
if(ipass_rods.eq.1) then
  allocate(ifac_ijtube(nfabor,2))
  xG = (Lx - dble(nx-1)*pitch)/2.d0
  write(nfecra,*) "user_boundary xG = ",xG
  do ix = 1, nx
    do iy = 1, ny
      center_tube(1,ix,iy) = xG + dble(ix-1)*pitch - Lx/2.d0
      center_tube(2,ix,iy) = xG + dble(iy-1)*pitch - Ly/2.d0
      nfac_ijtube(ix,iy) = 0
    enddo
  enddo
  do ix = 1, nx
    write(nfecra,*) "x center_tubes ",ix,(center_tube(1,ix,iy),iy = 1, ny)
    write(nfecra,*) "y center_tubes ",ix,(center_tube(2,ix,iy),iy = 1, ny)
  enddo
  do ix = 1, nx
    do iy = 1, nx
      minxr(ix,iy) = 1.d20
      maxxr(ix,iy) = -1.d20
    enddo
  enddo
  do ifac = 1, nfabor
    xx = cdgfbo(1,ifac)
    yy = cdgfbo(2,ifac)
    zz = cdgfbo(3,ifac)
    ifac_ijtube(ifac,1) = 0
    ifac_ijtube(ifac,2) = 0
    do ix = 1, nx
      do iy = 1, ny
        xr = sqrt((xx-center_tube(1,ix,iy))**2+(yy-center_tube(2,ix,iy))**2)
        if(xr.lt.dtube/2.d0+reps.and.zz.lt.zmax_bundle.and.zz.gt.zmin_bundle) then
          minxr(ix,iy) = min(xr,minxr(ix,iy))
          maxxr(ix,iy) = max(xr,maxxr(ix,iy))
          ifac_ijtube(ifac,1) = ix
          ifac_ijtube(ifac,2) = iy
          nfac_ijtube(ix,iy) = nfac_ijtube(ix,iy) + 1
          goto 10
        endif
      enddo
    enddo
 10 continue
  enddo
  if(irangp.ge.0) then
    do ix = 1, nx
      do iy = 1, ny
        call parmin(minxr(ix,iy))
        call parmax(maxxr(ix,iy))
        call parcpt(nfac_ijtube(ix,iy))
      enddo
    enddo
  endif
  do ix = 1, nx
    write(nfecra,*) "minxr ",ix, (minxr(ix,iy),iy = 1, ny)
    write(nfecra,*) "maxxr ",ix, (maxxr(ix,iy),iy = 1, ny)
    write(nfecra,*) "nfac_ijtube ",ix, (nfac_ijtube(ix,iy),iy = 1, ny)
  enddo
endif
nentre = 0
nsolib = 0
nparoi = 0
nheated = 0
if(indic_recycling_inlet.eq.1) then
  do ifac = 1, nfabor
    do idim = 1, 3
      vel_inlet(ifac,idim) = vel_recycling_inlet(ifac,idim,1)
    enddo
  enddo
endif
```

```
do ifac = 1, nfabor
  xx = cdgfbo(1,ifac)
  yy = cdgfbo(2,ifac)
  zz = cdgfbo(3,ifac)
  itube = ifac_ijtube(ifac,1)
  jtube = ifac_ijtube(ifac,2)

  if(abs(zz-zmax).lt.zeps) then
    itypfb(ifac) = ientre
    nentre = nentre + 1
    rcodcl(ifac,iu,1) = vel_inlet(ifac,1)
    rcodcl(ifac,iv,1) = vel_inlet(ifac,2)
    rcodcl(ifac,iw,1) = vel_inlet(ifac,3)
   if (nscal.gt.0) then
     ii = 1
     icodcl(ifac, isca(ii))    = 1
     rcodcl(ifac, isca(ii), 1) = 0.d0
    endif
  elseif(abs(zz-zmin).lt.zeps) then
    nsolib = nsolib + 1
    itypfb(ifac) = isolib
  else
    nparoi = nparoi + 1
    itypfb(ifac) = iparoi
    if (nscal.gt.0) then
      ii = 1
      if(itube.ne.0) then
        nheated = nheated + 1
        icodcl(ifac, isca(ii))    = 3
        rcodcl(ifac, isca(ii), 3) = -1.d0
      else
        icodcl(ifac, isca(ii))    = 3
        rcodcl(ifac, isca(ii), 3) = 0.d0
      endif
    endif
  endif
enddo
deallocate(vel_inlet)
if(indic_recycling_inlet.eq.1) then
  deallocate(vel_recycling_inlet)
endif
if(ipass_rods.le.2) then
  if(irangp.ge.0) then
    call parcpt(nentre)
    call parcpt(nsolib)
    call parcpt(nparoi)
    call parcpt(nheated)
  endif
  write(nfecra,*) "nbr inlet faces ",nentre
  write(nfecra,*) "nbr outlet faces ",nsolib
  write(nfecra,*) "nbr wall faces ",nparoi
  write(nfecra,*) "nbr heated faces ",nheated
endif

! We would need it in extraoperation
!if(ntcabs.eq.ntmabs) deallocate(ifac_ijtube)
!deallocate(lstelt_recycling_inlet)


!--------
! Formats
!--------

!----
! End
!----

!deallocate(lstelt)  ! temporary array for boundary faces selection

return
```

```
end subroutine cs_f_user_boundary_conditions

!> \section examples Examples
!>   Several examples are provided
!>   \ref cs_user_boundary_conditions_examples "here".
```

cs_user_extra_operations.f90

```
!-------------------------------------------------------------------------------

!                      Code_Saturne version 6.0.2-patch
!                      --------------------------
! This file is part of Code_Saturne, a general-purpose CFD tool.
!
! Copyright (C) 1998-2019 EDF S.A.
!
! This program is free software; you can redistribute it and/or modify it under
! the terms of the GNU General Public License as published by the Free Software
! Foundation; either version 2 of the License, or (at your option) any later
! version.
!
! This program is distributed in the hope that it will be useful, but WITHOUT
! ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
! FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
! details.
!
! You should have received a copy of the GNU General Public License along with
! this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
! Street, Fifth Floor, Boston, MA 02110-1301, USA.

!-------------------------------------------------------------------------------

!===============================================================================
! Purpose:
! -------

!> \file cs_user_extra_operations.f90
!>
!> \brief This function is called at the end of each time step, and has a very
!>  general purpose
!>  (i.e. anything that does not have another dedicated user subroutine)
!>
!> See \subpage cs_user_extra_operations_examples and
!> \subpage cs_user_extra_operations-nusselt_calculation for examples.
!>
!-------------------------------------------------------------------------------

!-------------------------------------------------------------------------------
! Arguments
!-------------------------------------------------------------------------------.
!  mode           name          role                                           !
!-------------------------------------------------------------------------------!
!> \param[in]     nvar          total number of variables
!> \param[in]     nscal         total number of scalars
!> \param[in]     dt            time step (per cell)
!-------------------------------------------------------------------------------

subroutine cs_f_user_extra_operations &
 ( nvar   , nscal  ,                                              &
   dt     )

!===============================================================================

!===============================================================================
! Module files
!===============================================================================

use paramx
use dimens, only: ndimfb
use pointe
```

```
use numvar
use optcal
use cstphy
use cstnum
use entsor
use lagran
use parall
use period
use ppppar
use ppthch
use ppincl
use mesh
use field
use field_operator
use turbomachinery
use cs_c_bindings
use user_module
!use force_total_dep



!===============================================================================

implicit none
!
include "user_conf_bundle.h"
!
integer icalcforce
parameter(icalcforce = 1)
double precision zeps_force, xyeps_force
parameter (zeps_force = 1.d-5)
parameter (xyeps_force = 1.d-5)
integer nzglob
common /iforce/ nzglob
! Arguments

integer          nvar   , nscal
double precision dt(ncelet)
!
double precision, dimension(:,:), pointer :: bfprp_for
double precision, allocatable, dimension(:,:,:,:) :: forcesat, forcep
double precision, allocatable, dimension(:) :: zloc, zglob_real
double precision, allocatable, dimension(:,:,:) :: xfac_ring
double precision, dimension(:), pointer :: cvar_pr
double precision, allocatable, dimension(:,:) :: grad

!
integer idim, ifac, iel, iz, jz
integer nzloc, nfac_ring
integer nzglob_remove, nzglob_real
integer iunit, junit
integer ivar, inc, iccocg
integer n_lower_grid, n_upper_grid, n_tubes
double precision var_pr_fac, xfac, yfac, zfac
double precision Force_tot_lower_grid(3), Force_p_lower_grid(3)
double precision Force_tot_upper_grid(3), Force_p_upper_grid(3)
double precision Force_tot_tubes(3), Force_p_tubes(3)
!!!! resultante globale variable !!!!
!integer          ifac
integer          ii
integer          ilelt  , nlelt
integer          file_index,N,k,nslice
double precision xfor(3)
double precision area(1)
double precision xnod(3)
integer, allocatable, dimension(:) :: lstelt
!!!!
!
integer ipass_force
data    ipass_force /0/
```

```
save    ipass_force
character*300 filewrite
!
!debut resultante
if (iforbr.ge.0) call field_get_val_v(iforbr, bfprp_for)

! Allocate a temporary array for cells or interior/boundary faces selection
allocate(lstelt(max(ncel,nfac,nfabor)))

if(ntcabs.eq.1) then
  if (irangp.eq.0) then
        open(file="resultante_globale.dat",unit=impusr(15))
! write headings to the dat file FX, FY, FZ are the x,y,z efforts
        write(impusr(15),"(8(a,1x))") " TIME STEP  ","  TIME  ","     FX     ","     FY
        ","     FZ     ","    X     ","    Y     ","    Z     "
        close(unit=impusr(15))
  endif
endif



if (iforbr.ge.0) then

  do ii = 1, ndim
    xfor(ii) = 0.d0
    xnod(ii) = 0.d0

 enddo
    area(1) = 0.d0
  !==========
  call getfbr('assembly_walls', nlelt, lstelt)
  !==========

  do ilelt = 1, nlelt

    ifac = lstelt(ilelt)
    xnod(1)=xnod(1)+cdgfbo(1, ifac)*surfbn(ifac)
    xnod(2)=xnod(2)+cdgfbo(2, ifac)*surfbn(ifac)
    xnod(3)=xnod(3)+cdgfbo(3, ifac)*surfbn(ifac)

! Boundary Area
   area(1) = area(1) + surfbn(ifac)
! Global Efforts
    do ii = 1, ndim
      xfor(ii) = xfor(ii) + bfprp_for(ii, ifac)


    enddo


  enddo
! Compute the global sums of an array of real numbers in case of parellism.
  if (irangp.ge.0) then
    call parrsm(ndim,xfor)
    call parrsm(ndim,xnod)
    call parrsm(1,area)
    xnod(1)=xnod(1)/area(1)
    xnod(2)=xnod(2)/area(1)
    xnod(3)=xnod(3)/area(1)

  if (irangp.eq.0) then
 !open the file
    open(file="resultante_globale.dat",unit=impusr(15),position="append")
! print  efforts for current timestep
    write(impusr(15),"(i12,1x,7(e12.5,1x))") ntcabs,ttcabs,xfor(1),xfor(2),xfor(3),xnod
    (1),xnod(2),xnod(3)
    close(unit=impusr(15))
  endif

 !close
```

```
 endif
endif

!end resultante
ipass_force = ipass_force + 1
if(ipass_force.eq.1) write(nfecra,*) "extraoperations iforbr = ",iforbr
if(ipass_force.eq.1.and.icalcforce.eq.1) then
  allocate(iconnect_upper_grid(nfabor))
  !write(nfecra,*) "allocating iconnect_upper_grid"
  allocate(iconnect_lower_grid(nfabor))
  !write(nfecra,*) "allocating iconnect_lower_grid"
  allocate(iconnect_tubes(nfabor))
  !write(nfecra,*) "allocating iconnect_tubes"
  iconnect_upper_grid = 0
  iconnect_lower_grid = 0
  iconnect_tubes = 0
  !write(nfecra,*) "setting to zero"
  if(icalcforce.eq.1.and.iforbr.le.0) then
    write(nfecra,*) "The program stops in extraoperations "
    write(nfecra,*) "iforbr is equal to ", iforbr, "and it should be positive"
    stop
  endif
  nzloc = 0
  !write(nfecra,*) "entering iteration on tube"
  do ifac = 1, nfabor
  write(nfecra,*) "begin iteration on tube"
    itube = ifac_ijtube(ifac,1)
    jtube = ifac_ijtube(ifac,2)
    !write(nfecra,*) "itube= ",itube,"jtube= ",jtube
    if(itube.eq.1.and.jtube.eq.1) then
      nzloc = nzloc + 1
    endif
  enddo
  nzglob = nzloc
  if(irangp.ge.0) call parcpt(nzglob)
  write(nfecra,*) "extraoperations nzglob = ",nzglob
  allocate(zloc(nzloc),zglob(nzglob), zglob_real(nzglob))
  write(nfecra,*) "allocating zglob et al"
  nzloc = 0
  n_upper_grid = 0
  n_lower_grid = 0
  n_tubes = 0
  do ifac = 1, nfabor
    itube = ifac_ijtube(ifac,1)
    jtube = ifac_ijtube(ifac,2)
    xfac = cdgfbo(1,ifac)
    yfac = cdgfbo(2,ifac)
    zfac = cdgfbo(3,ifac)
    !write(nfecra,*) "cdg face"
    if(itube.eq.1.and.jtube.eq.1) then
      nzloc = nzloc + 1
      zloc(nzloc) = cdgfbo(3,ifac)
    endif
    if((zfac.gt.zmin_bundle+zeps_force.and.zfac.lt.zmax_bundle-zeps_force).and.&
       (xfac.gt.-xgrid-xyeps_force.and.xfac.lt.xgrid+xyeps_force).and.&
       (yfac.gt.-ygrid-xyeps_force.and.yfac.lt.ygrid+xyeps_force)) then
      iconnect_tubes(ifac) = 1
      n_tubes = n_tubes + 1
    endif
    if(zfac.lt.zmin_bundle+zeps_force.and.zfac.gt.zmin_assembly-zeps_force.and.&
       (xfac.gt.-xgrid-xyeps_force.and.xfac.lt.xgrid+xyeps_force).and.&
       (yfac.gt.-ygrid-xyeps_force.and.yfac.lt.ygrid+xyeps_force)) then
       iconnect_lower_grid(ifac) = 1
       n_lower_grid = n_lower_grid + 1
    endif
    if(zfac.gt.zmax_bundle-zeps_force.and.zfac.lt.zmax_assembly+zeps_force.and.&
       (xfac.gt.-xgrid-xyeps_force.and.xfac.lt.xgrid+xyeps_force).and.&
```

```
       (yfac.gt.-ygrid-xyeps_force.and.yfac.lt.ygrid+xyeps_force)) then
       iconnect_upper_grid(ifac) = 1
       n_upper_grid = n_upper_grid + 1
     endif
   enddo
   if(irangp.ge.0) then
     call parcpt(n_upper_grid)
     call parcpt(n_lower_grid)
     call parcpt(n_tubes)
     call paragv(nzloc,nzglob,zloc,zglob)
   endif
   write(nfecra,*) "extraoperations n_upper_grid = ",n_upper_grid
   write(nfecra,*) "extraoperations n_lower_grid = ",n_lower_grid
   write(nfecra,*) "extraoperations n_tubes = ",n_tubes
   nzglob_remove = 0
   do iz = 1, nzglob
     do jz = iz + 1, nzglob
       if(abs(zglob(iz)-zglob(jz)).lt.zeps_force.and.abs(zglob(iz)).lt.1.D12) then
         zglob(jz) = 1.d20
         nzglob_remove = nzglob_remove + 1
       endif
     enddo
   enddo
   write(nfecra,*) "extraoperations nzglob_remove = ",nzglob_remove
   nzglob_real = 0
   do iz = 1, nzglob
     if(zglob(iz).lt.1.D12) then
       nzglob_real = nzglob_real + 1
       zglob_real(nzglob_real) = zglob(iz)
     endif
   enddo
   write(nfecra,*) "extraoperations nzglob_real = ",nzglob_real
   deallocate(zglob)
   nzglob = nzglob_real
   allocate(zglob(nzglob))
   do iz = 1, nzglob
     zglob(iz) = zglob_real(iz)
   enddo
   if(irangp.le.0) then
     iunit = impusr(1)
     open(file="z.dat",unit=iunit)
     rewind(iunit)
     do iz = 1, nzglob
       write(iunit,*) zglob(iz)
     enddo
     close(iunit)
   endif
   call sort(nzglob,zglob)
   allocate(iconnectz(nfabor))
   do ifac = 1, nfabor
     do iz = 1, nzglob
       if(abs(cdgfbo(3,ifac)-zglob(iz)).lt.zeps_force) then
         iconnectz(ifac) = iz
         goto 10
       endif
     enddo
  10 continue
   enddo
!
! verifications
!
   allocate(xfac_ring(nx,ny,nzglob))
   do itube = 1, nx
     do jtube = 1, ny
       do iz = 1, nzglob
         xfac_ring(itube,jtube,iz) = 0.d0
       enddo
     enddo
   enddo
   do ifac = 1, nfabor
```

```
    itube = ifac_ijtube(ifac,1)
    jtube = ifac_ijtube(ifac,2)
    if(itube.ne.0) then
     iz = iconnectz(ifac)
     xfac_ring(itube,jtube,iz) = xfac_ring(itube,jtube,iz) + 1.d0
    endif
  enddo
  if(irangp.ge.0) call parrsm(nx*ny*nzglob,xfac_ring(1,1,1))
  do itube = 1, nx
    do jtube = 1, ny
      do iz = 1, nzglob
        nfac_ring = int(xfac_ring(itube,jtube,iz))
        if(nfac_ring.ne.int(xfac_ring(1,1,1))) then
          write(nfecra,*) "The program stops "
          write(nfecra,*) "itube jtube iz nfac_ring = ", itube, jtube, iz, nfac_ring
          stop
        endif
      enddo
    enddo
  enddo
  if(irangp.le.0) then
    do itube = 1, nx
      write(filewrite,'(A14,I1.1,A4)') 'Force_lin_tot_',itube,'.dat'
      iunit = impusr(itube)
      open(file=filewrite,unit=iunit)
      rewind(iunit)
      write(filewrite,'(A12,I1.1,A4)') 'Force_lin_p_',itube,'.dat'
      iunit = impusr(nx+itube)
      open(file=filewrite,unit=iunit)
      rewind(iunit)
    enddo
    iunit = impusr(2*nx+1)
    open(file="F_Fp_upper_grid.dat",unit=iunit)
    rewind(iunit)
    iunit = impusr(2*nx+2)
    open(file="F_Fp_lower_grid.dat",unit=iunit)
    rewind(iunit)
    iunit = impusr(2*nx+3)
    open(file="F_Fp_tubes.dat",unit=iunit)
    rewind(iunit)
  endif
  deallocate(xfac_ring)
!
  deallocate(zloc,zglob_real)
endif
if(iforbr.ge.0.and.icalcforce.eq.1) then
  allocate(forcesat(3,nzglob,nx,ny))
  allocate(forcep(3,nzglob,nx,ny))
  forcesat = 0.d0
  forcep = 0.d0
  call field_get_val_v(iforbr, bfprp_for)
  ivar = ipr
  inc = 1
  iccocg = 1
  allocate(grad(3,ncelet))
  call field_get_val_s(ivarfl(ivar), cvar_pr)
  call field_gradient_scalar(ivarfl(ivar), 0, 0, inc, iccocg, grad)
  do idim = 1, 3
    Force_tot_lower_grid(idim) = 0.d0
    Force_p_lower_grid(idim) = 0.d0
    Force_tot_upper_grid(idim) = 0.d0
    Force_p_upper_grid(idim) = 0.d0
    Force_tot_tubes(idim) = 0.d0
    Force_p_tubes(idim) = 0.d0
  enddo
  do ifac = 1, nfabor
    itube = ifac_ijtube(ifac,1)
    jtube = ifac_ijtube(ifac,2)
    iel = ifabor(ifac)
    var_pr_fac =   cvar_pr(iel)       &
```

```
                + diipb(1,ifac)*grad(1,iel)  &
                + diipb(2,ifac)*grad(2,iel)  &
                + diipb(3,ifac)*grad(3,iel)
    if(itube.ne.0) then
      iz = iconnectz(ifac)
      do idim = 1, 3
        forcesat(idim,iz,itube,jtube) = forcesat(idim,iz,itube,jtube) + bfprp_for(idim,
    ifac)
        forcep(idim,iz,itube,jtube) = forcesat(idim,iz,itube,jtube) + var_pr_fac*surfbo(
    idim,ifac)
      enddo
    endif
    if(iconnect_lower_grid(ifac).eq.1) then
      do idim = 1, 3
        Force_tot_lower_grid(idim) = Force_tot_lower_grid(idim) + bfprp_for(idim,ifac)
        Force_p_lower_grid(idim) = Force_tot_lower_grid(idim) + var_pr_fac*surfbo(idim,
    ifac)
      enddo
    endif
    if(iconnect_upper_grid(ifac).eq.1) then
      do idim = 1, 3
        Force_tot_upper_grid(idim) = Force_tot_upper_grid(idim) + bfprp_for(idim,ifac)
        Force_p_upper_grid(idim) = Force_tot_upper_grid(idim) + var_pr_fac*surfbo(idim,
    ifac)
      enddo
    endif
    if(iconnect_tubes(ifac).eq.1) then
      do idim = 1, 3
        Force_tot_tubes(idim) = Force_tot_tubes(idim) + bfprp_for(idim,ifac)
        Force_p_tubes(idim) = Force_tot_tubes(idim) + var_pr_fac*surfbo(idim,ifac)
      enddo
    endif
  enddo
!
  if(irangp.ge.0) then
    call parrsm(3*nzglob*nx*ny,forcesat(1,1,1,1))
    call parrsm(3*nzglob*nx*ny,forcep(1,1,1,1))
    call parrsm(3,Force_tot_upper_grid(1))
    call parrsm(3,Force_p_upper_grid(1))
    call parrsm(3,Force_tot_lower_grid(1))
    call parrsm(3,Force_p_lower_grid(1))
    call parrsm(3,Force_tot_tubes(1))
    call parrsm(3,Force_p_tubes(1))
  endif
!
  if(irangp.le.0) then
    do itube = 1, nx
      iunit = impusr(itube)
      junit = impusr(nx+itube)
      do iz = 1, nzglob
        write(iunit,3000) ((forcesat(idim,iz,itube,jtube),idim=1,3),jtube=1,ny)
        write(junit,3000) ((forcep(idim,iz,itube,jtube),idim=1,3),jtube=1,ny)
      enddo
    enddo
    iunit = impusr(2*nx+1)
    write(iunit,2000) dtref*dble(ntcabs),(Force_tot_upper_grid(idim),idim=1,3),(
    Force_p_upper_grid(idim),idim=1,3)
    iunit = impusr(2*nx+2)
    write(iunit,2000) dtref*dble(ntcabs),(Force_tot_lower_grid(idim),idim=1,3),(
    Force_p_lower_grid(idim),idim=1,3)
    iunit = impusr(2*nx+3)
    write(iunit,2000) dtref*dble(ntcabs),(Force_tot_tubes(idim),idim=1,3),(Force_p_tubes
    (idim),idim=1,3)
  endif
  deallocate(forcesat)
  deallocate(forcep)
  deallocate(grad)
endif

if(ntcabs.eq.ntmabs.and.icalcforce.eq.1) then
```

```
  deallocate(zglob)
  deallocate(iconnectz)
  if(irangp.le.0) then
    do itube = 1, nx
      iunit = impusr(itube)
      close(iunit)
      iunit = impusr(nx+itube)
      close(iunit)
    enddo
    iunit = impusr(2*nx+1)
    close(iunit)
    iunit = impusr(2*nx+2)
    close(iunit)
    iunit = impusr(2*nx+3)
    close(iunit)
  endif
endif

!--------
! Formats
!--------
3000 format(15E15.6)
2000 format(7E15.6)

!----
! End
!----

return
end subroutine cs_f_user_extra_operations
!===============================================================================
subroutine sort(n,tab)
!===============================================================================

implicit none

integer n
double precision tab(n)

integer ns, ii, jj, kk
double precision tabmin

ns = 1
do ii = 1, n-1
  tabmin = 1.d20
  kk = 0
  do jj = ns,n
    if(tabmin.gt.tab(jj)) then
      tabmin = tab(jj)
      kk = jj
    endif
  enddo
  tab(kk) = tab(ns)
  tab(ns) = tabmin
  ns = ns + 1
enddo

return
end subroutine sort
```

cs_user_modules.f90

```
!-------------------------------------------------------------------------------

!                      Code_Saturne version 6.1.1
!                      --------------------------
! This file is part of Code_Saturne, a general-purpose CFD tool.
!
! Copyright (C) 1998-2020 EDF S.A.
!
```

```
! This program is free software; you can redistribute it and/or modify it under
! the terms of the GNU General Public License as published by the Free Software
! Foundation; either version 2 of the License, or (at your option) any later
! version.
!
! This program is distributed in the hope that it will be useful, but WITHOUT
! ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
! FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
! details.
!
! You should have received a copy of the GNU General Public License along with
! this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
! Street, Fifth Floor, Boston, MA 02110-1301, USA.

!-------------------------------------------------------------------------------

!===============================================================================
! Purpose:
! -------

!> \file cs_user_modules.f90
!>
!> \brief User-defined module: it allows to create any user array.
!>
!> See \subpage cs_user_modules for examples.
!>
!> This file is compiled before all other user Fortran files.
!> To ensure this, it must not be renamed.
!>
!> The user may define an arbitrary number of modules here, even though
!> only one is defined in the example.
!
!> \cond DOXYGEN_SHOULD_SKIP_THIS
module user_module

!-------------------------------------------------------------------------------
implicit none
! for recycling inlets in LES
integer, allocatable, dimension(:,:) ::  ifaccycloc, iconnectcyc
integer, allocatable, dimension(:,:) ::  lstelt_recycling_inlet
!needed for bundles
integer, allocatable, dimension(:,:) :: ifac_ijtube
!needed for the forces
integer, allocatable, dimension(:) :: iconnectz, iconnect_upper_grid,
    iconnect_lower_grid, iconnect_tubes
double precision, allocatable, dimension(:) :: zglob
!-------------------------------------------------------------------------------
end module user_module

!> (DOXYGEN_SHOULD_SKIP_THIS) \endcond
```

user_conf_bundle.h

```
integer nx, ny
parameter (nx = 5)
parameter (ny = 5)
integer itube, jtube, ix, iy
integer nfac_ijtube(nx,ny)
double precision dtube, pitch, Lx, Ly, xg, xgrid, ygrid
double precision zmin_assembly, zmax_assembly
double precision zmin_bundle, zmax_bundle, reps
double precision minxr(nx,ny), maxxr(nx,ny)
double precision center_tube(2,nx,ny)
double precision xx,yy,zz,xr

parameter (dtube = 0.01d0)
parameter (pitch = 0.0134d0)
parameter (Lx = 0.15d0)
parameter (Ly = 0.15d0)
parameter (xgrid = 0.0402d0)
```

```
parameter (ygrid = 0.0402d0)
parameter (zmin_bundle = -1.3099d0)
parameter (zmax_bundle = -0.0151d0)
parameter (zmin_assembly = -1.35d0)
parameter (zmax_assembly = 0.006d0)
parameter (reps = 1.d-3)

integer ipass_rods
save ipass_rods
data ipass_rods /0/
```

user_minmax.h

```
double precision xeps, yeps, zeps
parameter (xeps = 1.d-6)
parameter (yeps = 1.d-6)
parameter (zeps = 1.d-6)
double precision xmin, xmax, ymin, ymax, zmin, zmax
common /xminxmax/ xmin, xmax, ymin, ymax, zmin, zmax
integer ipass_minmax
data ipass_minmax /0/
save ipass_minmax
```

user_recycling_inlet.h

```
integer indic_recycling_inlet, n_recycling_inlet
parameter (indic_recycling_inlet = 0)
parameter (n_recycling_inlet = 1)
double precision xyeps_recycling_inlet, zeps_recycling_inlet
parameter (xyeps_recycling_inlet = 3.d-5)
parameter (zeps_recycling_inlet = 1.d-5)
!
integer nleltcycam(n_recycling_inlet), nleltcycav(n_recycling_inlet),nleltcycavloc(
    n_recycling_inlet)
integer nlelt_recycling_inlet(n_recycling_inlet)
common /ibound/ nleltcycam, nleltcycav, nleltcycavloc, nlelt_recycling_inlet
double precision surfinlet(n_recycling_inlet), zmin_recycling_inlet(n_recycling_inlet),
    zmax_recycling_inlet(n_recycling_inlet)
double precision dz_upward(n_recycling_inlet), z_recycling_inlet(n_recycling_inlet)
common /rbound/ surfinlet,zmin_recycling_inlet,zmax_recycling_inlet

integer, allocatable, dimension(:) :: nconnectcyc
double precision, allocatable, dimension(:) :: xcyclocav, ycyclocav
double precision, allocatable, dimension(:) :: xcycglobav, ycycglobav

double precision, allocatable, dimension(:) :: ucyclocav, vcyclocav, wcyclocav
double precision, allocatable, dimension(:) :: ucycglobav, vcycglobav, wcycglobav
double precision, allocatable, dimension(:,:,:) :: vel_recycling_inlet
double precision, dimension(:), pointer :: imasfl
double precision, dimension(:,:), pointer :: vel, vela

integer icl, ilelt_recycling_inlet, iflmas
integer          ileltcycav
integer istop, istop2, iperturb(n_recycling_inlet), ipos_recycling(n_recycling_inlet)
double precision fluxw(n_recycling_inlet), fluxwfac(n_recycling_inlet),wbulk(
    n_recycling_inlet), fluxwtarget(n_recycling_inlet)
double precision turbintensity(n_recycling_inlet), turblengthscale(n_recycling_inlet)
double precision surf, zinlet
double precision distp, distmin, uvwmin(n_recycling_inlet,3), uvwmax(n_recycling_inlet
    ,3)

integer ipass_recycling_inlet
save ipass_recycling_inlet
data ipass_recycling_inlet /0/
```

Annexe D

Scripts de post-traitement des forces issues des simulations numériques

D.1 Calcul des évolutions du coefficient d'amortissement et du coefficient de masse ajoutée en fonction de la fréquence de sollicitation

postprocess_archive_force_local_global.py

- Pré-conditions : Le fichier postprocess_archive_force_local_global.py doit être placé dans le dossier "RESU" d'un cas de calcul Code_Saturne pour lequel le calcul des résultantes sur l'assemblage et les cylindres (forces_local_global_tranches.f90) a été réalisé.
- Exécution : Dans un terminal, se placer dans le répertoire "RESU" du cas de calcul Code_Saturne et exécuter la commande "python3 postprocess_archive_force_local_global.py".
- Post-conditions : Un dossier extract_*result_file* est créé pour chaque sous-dossier *result_file* présent dans le dossier RESU. Pour chaque dossier *result_file*, pour chaque fichier texte "*file*.dat" présent dans le dossier, un fichier binaire "*file*.npy" est écrit dans le dossier extract_*result_file*. Chaque fichier binaire "*file*.npy" contient les mêmes informations que le fichier "*file*.dat". En revanche, le fichier binaire occupe moins d'espace disque et est plus rapide à traiter que le fichier texte.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Aug 6 10:22:10 2021

@author: g42679
"""
# A copier a la racine du dossier RESU ou avant
# Faire tourner sur le CLUSTER sur un noeud BM
# ATTENTION les fichiers crayons etant gros il faudra peut être faire tourner le script
    plusieurs fois.

import numpy as np
import matplotlib.pyplot as plt
import os as nonosse
import os.path
from os import path


path_to_RESU_file=""

########### listage automatique des dossiers a traiter ###########
#Si presence de dossier n'utilisant pas la routine V2 dans RESU, liste a rentrer a la
    main.
```

```
list_file=nonosse.listdir(path_to_RESU_file+'.') #["d5_f4_u08_r4_part1","
    d5_f5_u08_r4_part1"]

###########

for result_file in list_file:
    if result_file.endswith("_u08_r4a_v1") == True and result_file[0:7] != 'extract': #
    Selection des dossiers parcourus, condition a modifier suivant arborescence
        print("current file = ",result_file)
        time_series_path=path_to_RESU_file+result_file
        time_series_extract_path="extract_"+result_file
        list_fichier=np.sort(nonosse.listdir(time_series_path))


        nonosse.system("mkdir " +time_series_extract_path)
        ########### RESULTANTE GLOBALE AVEC DEPLACEMENT ###########

        force_assemblage=np.loadtxt(time_series_path+"/resultante_globale.dat",skiprows
    =1) # extrait a part a cause du header
        np.save(time_series_extract_path+"/"+result_file+"_resultante_globale",
    force_assemblage)

        ############

        ########### CAPTEUR PRESSION ###########

        force_assemblage=np.loadtxt(time_series_path+"/monitoring/probes_Pressure.dat",
    skiprows=10) # extrait a part a cause du header
        np.save(time_series_extract_path+"/"+result_file+"_capteurs_pression",
    force_assemblage)

        ############

        ########### Chargement fichiers texte et sauvegarde en binaire ###########
        flag=1
        if flag==1:
            print("Archive creation start")
            for file in list_fichier:
                if file.endswith(".dat") == True:
                    print("Processing file = ",file)
                    if file.endswith("rx.dat") == True or file.endswith("ry.dat") ==
    True or file.endswith("rz.dat") == True:
                        if path.isfile(time_series_extract_path+"/"+result_file+"_"+file
    [0:-4]+".npy")==False: # evite de recharger ce qui a deja ete fait
                            current_file_data=np.loadtxt(time_series_path+"/"+file)
                            np.save(time_series_extract_path+"/"+result_file+"_"+file
    [0:-4],current_file_data)
                        else:
                             print("this file is already archived")
                    else:
                        if path.isfile(time_series_extract_path+"/"+result_file+"_"+file
    [0:-4]+".npy")==False: # evite de recharger ce qui a deja ete fait
                            current_file_data=np.loadtxt(time_series_path+"/"+file,
    skiprows=1)
                            np.save(time_series_extract_path+"/"+result_file+"_"+file
    [0:-4],current_file_data)
                        else:
                            print("this file is already archived")
        else:
            current_file_data=0
```

postprocess_analysis_force_local_global.py

— Pré-conditions : Le fichier postprocess_analysis_force_local_global.py doit être placé doit être placé dans le dossier "RESU" du cas de calcul Code_Saturne. Le script postprocess_archive_force_local_global.py doit avoir été exécuté dans ce répertoire.

— Exécution : Dans un terminal, se placer dans le répertoire "RESU" du cas de calcul Code_Saturne et exécuter la commande "python3 postprocess_analysis_force_local_global.py".

— Post-conditions : Deux fichiers sont écrits dans le sous-dossier "extract_figure" du dossier "RESU" du cas de calcul Code_Saturne (le répertoire "extract_figure" est créé s'il n'existe pas).
  — Le fichier "coeff_amort_TLP.png" correspond à la FIGURE 5.9. Chaque dossier contenant des résultats de calcul correspond à une fréquence de sollicitation f_0 donnée et donc à un point sur les courbes représentatives de deux fonctions : 1- l'évolution du coefficient C_{amort} en fonction de la fréquence f_0, obtenue à partir des efforts exercés sur l'assemblage entier et 2- l'évolution du coefficient C_{amort} en fonction de la fréquence f_0, obtenue à partir des efforts exercés sur le faisceau de cylindres. Les données nécessaires pour tracer les autres courbes représentatives sont écrites en dur dans le script.
  — Le fichier "coeff_masse_TLP.png" correspond à la FIGURE 5.11. Chaque dossier contenant des résultats de calcul correspond à une fréquence de sollicitation f_0 donnée et donc à un point sur les courbes représentatives de deux fonctions : 1- l'évolution du coefficient C_{masse} en fonction de la fréquence f_0, obtenue à partir des efforts exercés sur l'assemblage entier et 2- l'évolution du coefficient C_{masse} en fonction de la fréquence f_0, obtenue à partir des efforts exercés sur le faisceau de cylindres. Les données nécessaires pour tracer les autres courbes représentatives sont écrites en dur dans le script.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib.cm as centimetre
from mpl_toolkits.mplot3d import axes3d
import scipy
# from scipy.signal import butter, sosfilt, sosfreqz
import os as nonosse
import os.path
from os import path
from Ellez_minotaure import phase_synchronisee
from collections import OrderedDict
from scipy.signal import butter, sosfilt, sosfreqz

def butter_bandpass(lowcut, highcut, fs, order=5):
        nyq = 0.5 * fs
        low = lowcut / nyq
        high = highcut / nyq
        sos = butter(order, [low, high], analog=False, btype='band', output='sos')
        return sos

def butter_bandpass_filter(data, lowcut, highcut, fs, order=5):
        sos = butter_bandpass(lowcut, highcut, fs, order=order)
        y = sosfilt(sos, data)
        return y


list_file=["d5_f1_u08_r4a_v1","d5_f2_u08_r4a_v1","d5_f3_u08_r4a_v1","d5_f4_u08_r4a_v1","
    d5_f5_u08_r4a_v1"]
# list_file=["d5_f1_u08_r4a_v1"]

freq=np.zeros(len(list_file))
CN=np.zeros(len(list_file))
ki=np.zeros(len(list_file))

CN_faisceau=np.zeros(len(list_file))
ki_faisceau=np.zeros(len(list_file))

CN_nobord=np.zeros((len(list_file),25))
ki_nobord=np.zeros((len(list_file),25))

phi_pression=np.zeros((len(list_file),9))

i=0
figure_file="extract_figure/"
#nonosse.system("mkdir " + "extract_figure")
list_cylinder=['m2m2','m2m1','m2_0','m2p1','m2p2','m1m2','m1m1','m1_0','m1p1','m1p2','
```

```
    _0m2','_0m1','_0_0','_0p1','_0p2','p1m2','p1m1','p1_0','p1p1','p1p2','p2m2','p2m1','
    p2_0','p2p1','p2p2']
for result_file in list_file:
    plt.close("all")
    print("################")
    print(result_file)
    time_series_extract_path="extract_"+result_file
    epaisseur_couronne=np.zeros((64,25))

    ########### Chargements des resultantes et pressions ###########
    resultante_globale=np.load(time_series_extract_path+"/"+result_file+"
    _resultante_globale.npy")
    pressions=np.load(time_series_extract_path+"/"+result_file+"_capteurs_pression.npy")
    ###########

    ########### Assignation des vecteurs utiles ###########
    t=resultante_globale[:,1]
    x=resultante_globale[:,5]
    F_x=resultante_globale[:,2]
    F_y=resultante_globale[:,3]
    F_z=resultante_globale[:,4]
    x_cyl=np.zeros((len(t),25))
    F_cyl_x=np.zeros((len(t),25))
    F_cyl_y=np.zeros((len(t),25))
    F_cyl_z=np.zeros((len(t),25))

    Contribution_cyl_x=np.zeros((25))

    for Ncyl in range(0,len(list_cylinder)):

        F_cyl_tempo=np.load(time_series_extract_path+"/"+result_file+'_'+list_cylinder[
    Ncyl]+".npy")
        x_cyl[:,Ncyl]=F_cyl_tempo[:,5]
        F_cyl_x[:,Ncyl]=F_cyl_tempo[:,2]


        ###### TLP STYLE #######
        f=float(result_file[4])
        A=0.0025
        rho=1000.0
        U=0.9
        D=0.01
        L=1.3
        N=25
        ohm=2*np.pi*f
        dt=t[1]-t[0]
        ind_t_deb=np.amax(np.where(t<10))
        ind_t_peak=+ind_t_deb+np.argmax(F_x[ind_t_deb:])

            ###### ECHELLE ASSEMBLAGE #######
        junk,tcross_x=phase_synchronisee(temps = t[ind_t_deb:], signal = x[ind_t_deb:]-
    np.mean(x[ind_t_deb:]), seuil = 0.05)
        junk,tcross_Fx=phase_synchronisee(temps = t[ind_t_deb:], signal = F_x[ind_t_deb
    :]-np.mean(F_x[ind_t_deb:]), seuil = 0.05)
        dephasage=np.mean((tcross_Fx[-5:-1]-tcross_x[-5:-1]))

        # print("dephasage = ",dephasage)

        C=(np.amax(F_x[ind_t_deb:])-np.mean(F_x[ind_t_deb:]))
        phi=dephasage*ohm-np.pi/2


        CN[i]=(C*np.cos(phi))/(0.5*rho*U*D*A*ohm*N*L)
        ki[i]=-C*np.sin(phi)/(rho*(N*L*np.pi*0.25*D**2)*A*ohm*ohm)

        # print("Cylindre = ",list_cylinder[Ncyl], np.around(ki_nobord[i,Ncyl],2), " ki
    "," CN= ",np.around(CN_nobord[i,Ncyl]),2)

        ###### ECHELLE CYLINDRE #######
        junk,tcross_x=phase_synchronisee(temps = t[ind_t_deb:], signal = x_cyl[ind_t_deb
```

```
:,Ncyl]-np.mean(x_cyl[ind_t_deb:,Ncyl]), seuil = 0.05)
    junk,tcross_Fx=phase_synchronisee(temps = t[ind_t_deb:], signal = F_cyl_x[
ind_t_deb:,Ncyl]-np.mean(F_cyl_x[ind_t_deb:,Ncyl]), seuil = 0.05)
    dephasage=np.mean((tcross_Fx[-5:-1]-tcross_x[-5:-1]))
    # print("dephasage = ",dephasage)

    C=(np.amax(F_cyl_x[ind_t_deb:,Ncyl])-np.mean(F_cyl_x[ind_t_deb:,Ncyl]))
    phi=dephasage*ohm-np.pi/2

    CN_nobord[i,Ncyl]=(C*np.cos(phi))/(0.5*rho*U*D*A*ohm*L)
    ki_nobord[i,Ncyl]=-C*np.sin(phi)/(rho*(L*np.pi*0.25*D**2)*A*ohm*ohm)
    # print("Cylindre = ",list_cylinder[Ncyl], np.around(ki_nobord[i,Ncyl],2), " ki
"," CN= ",np.around(CN_nobord[i,Ncyl]),2)

    ###### PRESSIONS #######
    # for jj in range(1,10):
    #     f_inf=0.9#np.amax([f-0.5,0.5])
    #     f_sup=5.1#np.amin([f+0.5,5.5])
    #     filtered_pression=butter_bandpass_filter(pressions[ind_t_deb:,jj]-np.mean(
pressions[ind_t_deb:,jj]), f_inf, f_sup, 1/dt, order=5)
    #     junk,tcross_x=phase_synchronisee(temps = t[ind_t_deb:], signal = x[
ind_t_deb:], seuil = 0.2)
    #     junk,tcross_pression=phase_synchronisee(temps = t[ind_t_deb:], signal =
pressions[ind_t_deb:,jj]-np.mean(pressions[ind_t_deb:,jj]), seuil = 0.05)
    #     dephasage=np.mean((tcross_pression[-5:-1]-tcross_x[-5:-1]))
    #     phi_pression[i,jj-1]=dephasage*ohm
    #     if jj==6:
    #         plt.figure('pression P5',figsize=(19,10))
    #         plt.plot(t[ind_t_deb:]-t[ind_t_deb],pressions[ind_t_deb:,jj]-np.mean(
pressions[ind_t_deb:,jj]),'k')
    #         plt.xlabel('t-7.0 (s)',size=20)
    #         plt.ylabel(r'$P6-\overline{P6}$ (Pa)',size=20)
    #         plt.xlim([0,4])
    #         plt.xticks(fontsize=20)
    #         plt.yticks(fontsize=20)
    #         plt.grid('on')
    #         plt.savefig(figure_file + "P6_" + result_file[4] + "Hz")

###### ECHELLE FAISCEAU #######
junk,tcross_x=phase_synchronisee(temps = t[ind_t_deb:], signal = x[ind_t_deb:]-np.
mean(x[ind_t_deb:]), seuil = 0.1)
junk,tcross_Fx=phase_synchronisee(temps = t[ind_t_deb:], signal = np.sum(F_cyl_x[
ind_t_deb:,:],axis=1)-np.mean(np.sum(F_cyl_x[ind_t_deb:,:],axis=1)), seuil = 0.1)
dephasage=np.mean((tcross_Fx[-5:-1]-tcross_x[-5:-1]))
    # print("dephasage = ",dephasage)

C=(np.amax(np.sum(F_cyl_x[ind_t_deb:,:],axis=1))-np.mean(np.sum(F_cyl_x[ind_t_deb
:,:],axis=1)))
phi=dephasage*ohm-np.pi/2

CN_faisceau[i]=(C*np.cos(phi))/(0.5*rho*U*D*A*ohm*L*N)
ki_faisceau[i]=-C*np.sin(phi)/(rho*(N*L*np.pi*0.25*D**2)*A*ohm*ohm)
print("ki = ",np.around(ki[i],3)," CN= ",np.around(CN[i],3))
print("ki_faisceau = ",np.around(ki_faisceau[i],3)," CN_faisceau = ",np.around(
CN_faisceau[i],3))

# print("ki_mean = ",np.around(np.mean(ki_nobord[i,:]),2)," CN_mean = ",np.around(np
.mean(CN_nobord[i,:]),2))
# print("ki_std = ",np.around(np.std(ki_nobord[i,:]),2)," CN_std = ",np.around(np.
std(CN_nobord[i,:]),2))
# amplitude_pression=(np.amax(pressions[ind_t_deb:,:],axis=0)-np.mean(pressions[
ind_t_deb:,:],axis=0))
# print("moyenne phi = ",np.around(np.mean(phi_pression[i,:]),3)," moyenne pression
= ",np.around(np.mean(amplitude_pression),3))
# print("std phi = ",np.around(np.std(phi_pression[i,:]),3)," std P= ",np.around(np.
std(amplitude_pression-np.mean(amplitude_pression)),3))

# print("contrib moy = ",np.around(np.mean(Contribution_cyl_x),3),"contrib std = ",
np.around(np.std(Contribution_cyl_x),3))
print("################")
```

```
plt.figure(1,figsize=(19,10))
plt.subplot(1,2,1)
# plt.plot(np.linspace(1,25,25),CN_nobord[i,:],'k',marker='*',ls='--')
plt.plot([1,2,3,4,5],CN_nobord[i,0:5],'k',marker='*',ls='--',label='Ligne 1')
plt.plot([1,2,3,4,5],CN_nobord[i,5:10],'r',marker='*',ls='--',label='Ligne 2')
plt.plot([1,2,3,4,5],CN_nobord[i,10:15],'g',marker='*',ls='--',label='Ligne 3')
plt.plot([1,2,3,4,5],CN_nobord[i,15:20],'b',marker='*',ls='--',label='Ligne 4')
plt.plot([1,2,3,4,5],CN_nobord[i,20:25],'orange',marker='*',ls='--',label='Ligne 5')
plt.grid('on')
plt.xlim([1,5])
plt.ylim([0.0,0.3])
plt.xticks([1,2,3,4,5],['-2','-1','0','1','2'],fontsize=20)
plt.yticks(fontsize=20)
plt.xlabel('$i_y$ cylindre (-)',size=20)
plt.ylabel('$C_{amort}$ (/)',size=20)
plt.legend(['$i_x$=-2','$i_x$=-1','$i_x$=0','$i_x$=1','$i_x$=2'],fontsize=20)
plt.subplot(1,2,2)
# plt.plot(np.linspace(1,25,25),ki_nobord[i,:],'k',marker='*',ls='--')
plt.plot([1,2,3,4,5],ki_nobord[i,0:5],'k',marker='*',ls='--',label='Ligne 1')
plt.plot([1,2,3,4,5],ki_nobord[i,5:10],'r',marker='*',ls='--',label='Ligne 2')
plt.plot([1,2,3,4,5],ki_nobord[i,10:15],'g',marker='*',ls='--',label='Ligne 3')
plt.plot([1,2,3,4,5],ki_nobord[i,15:20],'b',marker='*',ls='--',label='Ligne 4')
plt.plot([1,2,3,4,5],ki_nobord[i,20:25],'orange',marker='*',ls='--',label='Ligne 5')
plt.grid('on')
plt.xlim([1,5])
plt.ylim([0,2.2])
plt.xticks([1,2,3,4,5],['-2','-1','0','1','2'],fontsize=20)
plt.yticks(fontsize=20)
plt.xlabel('$i_y$ cylindre (-)',size=20)
plt.ylabel('$C_{masse}$ (/)',size=20)
plt.legend(['$i_x$=-2','$i_x$=-1','$i_x$=0','$i_x$=1','$i_x$=2'],fontsize=20,loc='
lower center')
# plt.legend(["maximum","moyenne","minimum"],fontsize=20)
plt.savefig(figure_file + "Cxy_" + result_file[4] + "Hz")

###### CONTRIBUTION CRAYON #######
Contribution_cyl_x[:]=100*F_cyl_x[ind_t_peak,:]/np.sum(F_cyl_x[ind_t_peak,:])
# print("contrib moy = ",np.around(np.mean(Contribution_cyl_x),3),"contrib std = ",
np.around(np.std(Contribution_cyl_x),3))
plt.figure(2,figsize=(19,10))
# plt.subplot(1,2,1)
# plt.plot(np.linspace(1,25,25),Contribution_cyl_x[i,:],'k',marker='*',ls='--')
plt.plot([1,2,3,4,5],Contribution_cyl_x[0:5],'k',marker='*',ls='--',label='Ligne 1')
plt.plot([1,2,3,4,5],Contribution_cyl_x[5:10],'r',marker='*',ls='--',label='Ligne 2'
)
plt.plot([1,2,3,4,5],Contribution_cyl_x[10:15],'g',marker='*',ls='--',label='Ligne 3
')
plt.plot([1,2,3,4,5],Contribution_cyl_x[15:20],'b',marker='*',ls='--',label='Ligne 4
')
plt.plot([1,2,3,4,5],Contribution_cyl_x[20:25],'orange',marker='*',ls='--',label='
Ligne 5')
plt.grid('on')
plt.xlim([1,5])
plt.ylim([0.0,5])
plt.xticks([1,2,3,4,5],['-2','-1','0','1','2'],fontsize=20)
plt.yticks(fontsize=20)
plt.xlabel('$i_y$ cylindre (-)',size=20)
plt.ylabel(r'$ \dfrac{F_{x\_cyl}}{F_{x\_tot}} $ (%)',size=20)
plt.legend(['$i_x$=-2','$i_x$=-1','$i_x$=0','$i_x$=1','$i_x$=2'],fontsize=20)
plt.savefig(figure_file + "Contrib_cyl_" + result_file[4] + "Hz")

plt.figure(7,figsize=(19,10))
max_plot=np.amax(np.sum(F_cyl_x[:,:],axis=1))*1.1
# plt.title("Force X vs deplacement impose \n amplitude = "+str(1000*A)+"mm
frequence = "+str(f)+"Hz")
plt.plot(t,np.sum(F_cyl_x[:,:],axis=1),'k')
# plt.plot(t,F_x/np.amax(F_x))
plt.xlabel('t (s)',size=20)
plt.ylabel('Fx (N)',size=20)
```

```
    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)
    plt.plot([7.0,7.0],[-max_plot,max_plot],'--',color='r')
    plt.plot([(7.0+3.0),(7.0+3.0)],[-max_plot,max_plot],'--',color='b')
    plt.legend(['Fx (sans grille)','Debut çforage','Debut regime stabilise'],fontsize=20)
    plt.savefig(figure_file + "Fx_faisceau_" + result_file[4] + "Hz")

    freq[i]=f
    i=i+1

plt.figure(4,figsize=(19,10))
plt.plot(freq,ki,color='k',marker='*',ls='',ms=10)
plt.plot(freq,ki_faisceau,color='k',marker='o',ls='',ms=10)
# for x,y in zip(freq,ki):
#     label=str(np.around(y,2))
#     plt.annotate(label, (x,y),textcoords="offset points",xytext=(0,10),ha='center',
    fontsize=20)
plt.plot(freq,1.45*np.ones(len(freq)),color='b',ls=':',ms=10)
plt.plot(freq,0.6*np.ones(len(freq)),color='r',ls='--',ms=10)
plt.plot(freq,1.25*np.ones(len(freq)),color='green',ls='--',ms=10)

plt.xlabel('f (Hz)',size=20)
plt.ylabel('$C_{masse}$ (-)',size=20)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.grid('on')
plt.ylim([0,2.2])
plt.legend(['CFD U=0.9m/s','CFD sans grille U=0.9m/s','Essais Divaret (2014)','Essais U
    =0.9m/s','Potentiel 2D'],fontsize=20)
plt.savefig(figure_file +"coeff_masse_TLP.png")
# plt.title('Coeff masse ajoutee TLP applique a l\'assemblage complet')

plt.figure(5,figsize=(19,10))
plt.plot(freq,CN,color='k',marker='*',ls='',ms=10)
plt.plot(freq,CN_faisceau,color='k',marker='o',ls='',ms=10)
# CN_xp=np.array([1.01,0.5,0.34,0.25,0.2])
plt.plot(freq,0.18*np.ones(len(freq)),color='b',ls=':',ms=10)
plt.plot(freq,0.32*np.ones(len(freq)),color='r',ls='--',ms=10)
# for x,y in zip(freq,CN):
#     label=str(np.around(y,2))
#     plt.annotate(label, (x,y),textcoords="offset points",xytext=(-0,-30),ha='center',
    fontsize=20)
    # label=str(np.around(z,2))
    # plt.annotate(label, (x,z),textcoords="offset points",xytext=(25,25),ha='center',
    fontsize=20)

plt.xlabel('f (Hz)',size=20)
plt.ylabel('$C_{amort}$ (-)',size=20)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.grid('on')
plt.ylim([0,0.4])
plt.legend(['CFD U=0.9m/s','CFD sans grille U=0.9m/s','Essais Divaret (2014)','Essais U
    =0.9m/s'],fontsize=20)
plt.savefig(figure_file +"coeff_amort_TLP.png")

# plt.figure(6,figsize=(19,10))
# # plt.title("Dephasage pression/deplacement")
# nombre_capteur=[1,2,3,4,5,6,7,8,9]
# couleurs=['k','k','red','red','g','red','red','k','k']
# for N_capteur,couleur in zip(nombre_capteur,couleurs):
#     plt.plot([1,2,3,4,5],phi_pression[0:5,N_capteur-1],'o')
# plt.xlabel('f (Hz)',size=20)
# plt.ylabel('$\phi (rad)$',size=20)
# plt.xticks(fontsize=20)
# plt.yticks(fontsize=20)
# # plt.ylim([0,np.pi])
# plt.legend(['p5','p6','p7','p4','p3','p9','p8','p2','p1'],fontsize=20)
# plt.savefig(figure_file + "pressions_" + result_file[4] + "Hz")
```

```
# plt.figure(7,figsize=(19,10))
# plt.plot(t[ind_t_deb:],F_x[ind_t_deb:]/np.amax(F_x[ind_t_deb:]))
# plt.plot(t[ind_t_deb:],np.sum(F_cyl_x[ind_t_deb:,:],axis=1)/np.amax(np.sum(F_cyl_x[
    ind_t_deb:,:],axis=1)))
```

D.2 Calcul des densités linéiques de force

postprocess_archive_force_couronnes.py

- Pré-conditions : Le fichier postprocess_archive_force_couronnes.py doit être placé dans le dossier "RESU" d'un cas de calcul Code_Saturne pour lequel le calcul des résultantes sur les couronnes a été réalisé.
- Exécution : Dans un terminal, se placer dans le répertoire "RESU" du cas de calcul Code_Saturne et exécuter la commande "python3 postprocess_archive_force_couronnes.py".
- Post-conditions : Un dossier extract_*result_file* est créé pour chaque sous-dossier *result_file* présent dans le dossier RESU. Pour chaque dossier *result_file*, pour chaque fichier texte "*file*.dat" présent dans le dossier, 15 fichiers binaires sont écrit dans le dossier extract_*result_file*. Chaque fichier binaire correspond à un triplet (i_x, i_y, *comp*), où (i_x, i_y) correspond à un cylindre parmi les 5 cylindres correspondant au fichier "*file*.dat" et *comp* correspond à une composante de la résultante de la force stockée dans le fichier "*file*.dat". Dans chaque fichier, la première colonne est l'instant à la fin du pas de temps t_f et les colonnes suivantes sont des coordonnées selon l'axe (Oz).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Aug 6 10:22:10 2021

@author: g42679
"""
# A copier a la racine du dossier RESU ou avant
# Faire tourner sur le CLUSTER sur un noeud BM
# ATTENTION les fichiers crayons etant gros il faudra peut etre faire tourner le script
    plusieurs fois.

import numpy as np
import matplotlib.pyplot as plt
import os as nonosse
import os.path
from os import path


path_to_RESU_file=""

########### listage automatique des dossiers a traiter ###########
#Si presence de dossier n'utilisant pas la routine V2 dans RESU, liste a rentrer a la
    main.

list_file=nonosse.listdir(path_to_RESU_file+'.') #["d5_f4_u08_r4_part1","
    d5_f5_u08_r4_part1"]

###########

for result_file in list_file:
    if result_file.endswith("_u08_r4_part1") == True and result_file[0:7] != 'extract':
    # Selection des dossiers parcourus, condition a modifier suivant arborescence
        print("current file = ",result_file)
        time_series_path=path_to_RESU_file+result_file
        time_series_extract_path="extract_"+result_file
        list_fichier=np.sort(nonosse.listdir(time_series_path))


        nonosse.system("mkdir " +time_series_extract_path)
        ########### RESULTANTE GLOBALE AVEC DEPLACEMENT ###########
```

```
    force_assemblage=np.loadtxt(time_series_path+"/resultante_globale.dat",skiprows
=1) # extrait a part a cause du header
    np.save(time_series_extract_path+"/"+result_file+"_resultante_globale",
force_assemblage)

    ############

    ############ CAPTEUR PRESSION ############

    force_assemblage=np.loadtxt(time_series_path+"/monitoring/probes_Pressure.dat",
skiprows=10) # extrait a part a cause du header
    np.save(time_series_extract_path+"/"+result_file+"_capteurs_pression",
force_assemblage)

    ############

    ############ Chargement fichiers texte et sauvegarde en binaire ############
    flag=1
    if flag==1:
        print("Archive creation start")
        for file in list_fichier:
            if file.endswith(".dat") == True:
                print("Processing file = ",file)
                if file.endswith("resultante_globale.dat") == False:
                    if path.isfile(time_series_extract_path+"/"+result_file+"_"+file
[0:-4]+".npy")==False: # evite de recharger ce qui a deja ete fait
                        current_file_data=np.loadtxt(time_series_path+"/"+file)
                        np.save(time_series_extract_path+"/"+result_file+"_"+file
[0:-4],current_file_data)
                    else:
                        print("this file is already archived")
    else:
        current_file_data=0

    ############

    ############ Chargements necessaires au decoupage ############
    resu=np.load(time_series_extract_path+"/"+result_file+"_resultante_globale.npy")
    t=resu[:,1]
    x=resu[:,5]
    z=np.load(time_series_extract_path+"/"+result_file+"_z.npy")
    ############

    ############ Decoupage des fichiers couronnes force tot ############

    print("reshaping result from V2 to V1")
    for number_corona_file in [1,2,3,4,5]:

        current_corona_file="/"+result_file+"_Force_lin_tot_"+str(number_corona_file
)+".npy"
        force_lin_tot=np.load(time_series_extract_path+"/"+current_corona_file)

        N_z=len(z)
        N_dt=int(len(force_lin_tot[:,1])/N_z)

        c_ligne_r=np.zeros((N_dt,N_z,5,3))

        for ii in range (0,N_dt):
            for dim in [0,1,2]:
                for jj in [0,1,2,3,4]:
                    c_ligne_r[ii,:,jj,dim]=force_lin_tot[ii*N_z:(ii+1)*N_z,3*jj+dim]
 # on passe de la structure par bloc de la routine v2 a des vecteurs individuels par
 crayons comme en v1
    ############

    ############ Decoupage des fichiers couronnes force P ############

    print("reshaping result from V2 to V1")
    for number_corona_file in [1,2,3,4,5]:
```

```
            current_corona_file="/"+result_file+"_Force_lin_p_"+str(number_corona_file)+
    ".npy"
            force_lin_p=np.load(time_series_extract_path+"/"+current_corona_file)

            N_z=len(z)
            N_dt=int(len(force_lin_p[:,1])/N_z)

            c_ligne_p_r=np.zeros((N_dt,N_z,5,3))

            for ii in range (0,N_dt):
                for dim in [0,1,2]:
                    for jj in [0,1,2,3,4]:
                        c_ligne_p_r[ii,:,jj,dim]=force_lin_tot[ii*N_z:(ii+1)*N_z,3*jj+
    dim] # on passe de la structure par bloc de la routine v2 a des vecteurs individuels
     par crayons comme en v1
        ############

        ############ sauvegarde des fichiers individuelles en binaire ############

            print("saving binaries for file : ",current_corona_file)
            for Ncyl in [1,2,3,4,5]:
                np.save(time_series_extract_path+"/"+result_file+"_"+"c"+str(5*(
    number_corona_file-1)+Ncyl)+"_rx", c_ligne_r[:,:,Ncyl-1,0])
                np.save(time_series_extract_path+"/"+result_file+"_"+"c"+str(5*(
    number_corona_file-1)+Ncyl)+"_ry", c_ligne_r[:,:,Ncyl-1,1])
                np.save(time_series_extract_path+"/"+result_file+"_"+"c"+str(5*(
    number_corona_file-1)+Ncyl)+"_rz", c_ligne_r[:,:,Ncyl-1,2])

                np.save(time_series_extract_path+"/"+result_file+"_"+"p_c"+str(5*(
    number_corona_file-1)+Ncyl)+"_rx", c_ligne_p_r[:,:,Ncyl-1,0])
                np.save(time_series_extract_path+"/"+result_file+"_"+"p_c"+str(5*(
    number_corona_file-1)+Ncyl)+"_ry", c_ligne_p_r[:,:,Ncyl-1,1])
                np.save(time_series_extract_path+"/"+result_file+"_"+"p_c"+str(5*(
    number_corona_file-1)+Ncyl)+"_rz", c_ligne_p_r[:,:,Ncyl-1,2])

        ############


############
```

postprocess_analysis_force_couronnes.py

- Pré-conditions : Le fichier postprocess_analysis_force_couronnes.py doit être placé doit être placé dans le dossier "RESU" du cas de calcul Code_Saturne. Le script postprocess_archive_force_couronnes.py doit avoir été exécuté dans ce répertoire.
- Exécution : Dans un terminal, se placer dans le répertoire "RESU" du cas de calcul Code_Saturne et exécuter la commande "python3 postprocess_analysis_force_couronnes.py".
- Post-conditions : Pour chaque fréquence de sollicitation f_0, le fichier "densite_lineique_*f0*Hz.png" contient le graphique représentant la densité linéique de force le long de cinq cylindres (*cf.* FIGURE 5.17).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Aug  9 11:48:00 2021

@author: g42679
"""

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib.cm as centimetre
from mpl_toolkits.mplot3d import axes3d
import scipy
# from scipy.signal import butter, sosfilt, sosfreqz
import os as nonosse
```

```
import os.path
from os import path
from Ellez_minotaure import phase_synchronisee


# list_file=["d5_f1_u08_r4_part1","d5_f2_u08_r4_part1","d5_f3_u08_r4_part1","
    d5_f4_u08_r4_part1","d5_f5_u08_r4_part1"]
# list_file=["d5_f1_u08_r4_part1"]
list_file=["d5_f5_u08_r4_part1"]

contrib_grille_haute=np.zeros((len(list_file),3))
contrib_grille_basse=np.zeros((len(list_file),3))
contrib_crayons=np.zeros((len(list_file),3))
F_tot=np.zeros((len(list_file),3))
max_crayon=np.zeros((len(list_file),25,3))
mean_crayon=np.zeros((len(list_file),25,3))
min_crayon=np.zeros((len(list_file),25,3))
distribution_crayon=np.zeros((len(list_file),25,3))
# densite_crayon=np.zeros((len(list_file),25,3))
deviation_locale_crayon=np.zeros((len(list_file),25,3))
deviation_redux_locale_crayon=np.zeros((len(list_file),25,3))
moyenne_locale_crayon=np.zeros((len(list_file),25,3))

f_fonda_dep=np.zeros(len(list_file))
# module_fonda=np.zeros((len(list_file),12))
# phase_fonda=np.zeros((len(list_file),12))

CN=np.zeros(len(list_file))
ki=np.zeros(len(list_file))

CN_nobord=np.zeros((len(list_file),25))
ki_nobord=np.zeros((len(list_file),25))

i=0
figure_file="extract_figure/"
nonosse.system("mkdir " + "extract_figure")

for result_file in list_file:
    plt.close("all")
    print(result_file)
    time_series_extract_path="extract_"+result_file

    ############ Chargements des resultantes et pressions ############
    resultante_globale=np.load(time_series_extract_path+"/"+result_file+"
    _resultante_globale.npy")
    force_tot_crayons=np.load(time_series_extract_path+"/"+result_file+"_F_Fp_tubes.npy"
    )
    force_upper_grid=np.load(time_series_extract_path+"/"+result_file+"_F_Fp_upper_grid.
    npy")
    force_lower_grid=np.load(time_series_extract_path+"/"+result_file+"_F_Fp_lower_grid.
    npy")
    z=np.load(time_series_extract_path+"/"+result_file+"_z.npy")
    # pressions=np.load(time_series_extract_path+"/"+result_file+"_capteurs_pression.npy
    ")
    ############

    ############ Assignation des vecteurs utiles ############
    t=resultante_globale[:,1]
    x=resultante_globale[:,5]
    F_x=resultante_globale[:,2]
    F_y=resultante_globale[:,3]
    F_z=resultante_globale[:,4]
    F_cyl_x=np.zeros((len(t),len(z),25))
    F_cyl_y=np.zeros((len(t),len(z),25))
    # F_cyl_z=np.zeros((len(t),len(z),25))
    z=abs(np.sort(z))
    z=abs(np.sort(z))
    epaisseur=np.zeros(len(z))
    densite_crayon=np.zeros((len(list_file),25,len(z),3))
    # resultante_crayon=np.zeros((len(list_file),25,3))
```

```
    CN_lin=np.zeros((len(list_file),25,len(z),1))
    ki_lin=np.zeros((len(list_file),25,len(z),1))
    # F_p_cyl_x=np.zeros((len(t),len(z),25))

    dt=t[1]-t[0]
    for Ncyl in range(0,25):
        F_cyl_x[:,:,Ncyl]=np.load(time_series_extract_path+"/"+result_file+"_c"+str(Ncyl
    +1)+"_rx.npy")
        F_cyl_y[:,:,Ncyl]=np.load(time_series_extract_path+"/"+result_file+"_c"+str(Ncyl
    +1)+"_ry.npy")
        # F_cyl_z[:,:,Ncyl]=np.load(time_series_extract_path+"/"+result_file+"_c"+str(
    Ncyl+1)+"_rz.npy")
        # F_p_cyl_x[:,:,Ncyl]=np.load(time_series_extract_path+"/"+result_file+"_p_c"+
    str(Ncyl+1)+"_rx.npy")
        # max_crayon[Ncyl]=np.amax(np.sum(F_cyl_x[:,:,Ncyl],axis=1))
        # mean_crayon[Ncyl]=np.mean(np.sum(F_cyl_x[:,:,Ncyl],axis=1))
        # min_crayon[Ncyl]=np.amin(np.sum(F_cyl_x[:,:,Ncyl],axis=1))

    ############


    ############ DEPHASAGE PRESSION/DEPLACEMENT ############
    nfft = 2**15
    overlap = nfft/2
    fsample=1.0/dt
    ind_t_deb=np.amax(np.where(t<9))
    ind_t_fin=np.amax(np.where(t<13))
    PSD_x2, freqs_x2 = mlab.psd(x[ind_t_deb:ind_t_fin]-np.mean(x[ind_t_deb:ind_t_fin]),
    nfft, fsample, detrend=None, window=mlab.window_hanning, noverlap=overlap,
    scale_by_freq=True)
    f_fonda_dep[i]=freqs_x2[np.argmax(PSD_x2)]

    # for jj in range(1,10):
    #     CSD, freqs_csd = mlab.csd(x[ind_t_deb:ind_t_fin]-np.mean(x[ind_t_deb:ind_t_fin
    ]),pressions[ind_t_deb:ind_t_fin,jj]-np.mean(pressions[ind_t_deb:ind_t_fin,jj]),
    nfft, fsample, detrend=None, window=mlab.window_hanning, noverlap=overlap,
    scale_by_freq=True)
    #     PSD_P, freqs_PSD_P = mlab.csd(x[ind_t_deb:ind_t_fin]-np.mean(x[ind_t_deb:
    ind_t_fin]),pressions[ind_t_deb:ind_t_fin,jj]-np.mean(pressions[ind_t_deb:ind_t_fin,
    jj]), nfft, fsample, detrend=None, window=mlab.window_hanning, noverlap=overlap,
    scale_by_freq=True)
    #     module=np.abs(CSD)
    #     phase=np.arctan2(np.real(CSD),np.imag(CSD))
    #     module_fonda[i,jj-1]=module[np.argmax(CSD)]
    #     phase_fonda[i,jj-1]=phase[np.argmax(CSD)]
        # plt.figure(jj*10)
        # plt.plot(freqs_PSD_P,PSD_P)
        # print(jj)


    ############ Contribution des crayons a la resultante ############
    ind_t_deb=np.amax(np.where(t<11))
    ind_t0_deb=np.amax(np.where(t<8))
    ind_min_z=np.amax(np.where(z<=0.1))
    ind_max_z=np.amax(np.where(z<=1.2))
    for dim in [1,2,3]:
        ind_peak_eval=ind_t_deb+np.argmax(F_x[ind_t_deb:ind_t_fin])
        resul=abs(resultante_globale[ind_peak_eval,dim+1])
        # contrib_grille_haute[i,dim-1]=(np.amin((force_upper_grid[ind_t_deb:,dim]))/np.
    amin((resultante_globale[:,dim+1])))
        # contrib_grille_basse[i,dim-1]=(np.amin((force_lower_grid[ind_t_deb:,dim]))/np.
    amin((resultante_globale[:,dim+1])))
        # contrib_crayons[i,dim-1]=(np.amin((force_tot_crayons[ind_t_deb:,dim]))/np.amin
    ((resultante_globale[:,dim+1])))
        # contrib_grille_haute[i,dim-1]=np.around(100*((resul-abs(force_tot_crayons[
    ind_peak_eval,dim])-abs(force_lower_grid[ind_peak_eval,dim]))/resul),1)
        # contrib_grille_basse[i,dim-1]=np.around(100*((resul-abs(force_tot_crayons[
    ind_peak_eval,dim])-abs(force_upper_grid[ind_peak_eval,dim]))/resul),1)
        # contrib_crayons[i,dim-1]=np.around(100*((resul-abs(force_upper_grid[
    ind_peak_eval,dim])-abs(force_lower_grid[ind_peak_eval,dim]))/resul),1)
```

```
    contrib_grille_haute[i,dim-1]=np.around(100*(abs(force_upper_grid[ind_peak_eval,
dim])/abs(resul)),1)
    contrib_grille_basse[i,dim-1]=np.around(100*(abs(force_lower_grid[ind_peak_eval,
dim])/abs(resul)),1)
    contrib_crayons[i,dim-1]=np.around(100*(abs(force_tot_crayons[ind_peak_eval,dim
])/abs(resul)),1)
    F_tot[i,dim-1]=resul
for Ncyl in range(0,25):
    distribution_crayon[i,Ncyl,0]=(np.sum(F_cyl_x[ind_peak_eval,:,Ncyl],axis=0))/
resultante_globale[ind_peak_eval,2]
    distribution_crayon[i,Ncyl,1]=(np.sum(F_cyl_y[ind_peak_eval,:,Ncyl],axis=0))/
resultante_globale[ind_peak_eval,3]
    # distribution_crayon[i,Ncyl,2]=(np.sum(F_cyl_z[ind_peak_eval,:,Ncyl],axis=0))/
resultante_globale[ind_peak_eval,4]

    ###### TLP STYLE #######
    f=float(result_file[4])
    A=0.0025
    rho=1000.0
    U=0.9
    D=0.01
    L=1.3
    ohm=2*np.pi*f
    resultante_crayon=np.sum(F_cyl_x[ind_t_deb:,:,Ncyl],axis=1)
    # resultante_crayon=np.sqrt(np.sum(F_cyl_x[ind_t_deb:,:,Ncyl],axis=1)**2+np.sum(
F_cyl_y[ind_t_deb:,:,Ncyl],axis=1)**2)

    junk,tcross_x=phase_synchronisee(temps = t[ind_t_deb:], signal = x[ind_t_deb:],
seuil = 0.1)
    junk,tcross_Fx=phase_synchronisee(temps = t[ind_t_deb:], signal =
resultante_crayon-np.mean(resultante_crayon), seuil = 0.1)
    dephasage=np.mean((tcross_Fx[-2:-1]-tcross_x[-2:-1]))
    print("dephasage = ",dephasage)

    C=(np.amax(resultante_crayon)-np.mean(resultante_crayon))
    phi=dephasage*ohm-np.pi/2

    CN_nobord[i,Ncyl]=(C*np.cos(phi))/(0.5*rho*U*D*A*ohm*L)
    ki_nobord[i,Ncyl]=-C*np.sin(phi)/(rho*(L*np.pi*0.25*D**2)*A*ohm*ohm)

    for Ncouronne in range(0,len(z)-1):
        epaisseur[Ncouronne]=abs(z[Ncouronne+1]-z[Ncouronne])
        densite_crayon[i,Ncyl,Ncouronne,0]=F_cyl_x[ind_peak_eval,Ncouronne,Ncyl]/
epaisseur[Ncouronne]
        densite_crayon[i,Ncyl,Ncouronne,1]=F_cyl_y[ind_peak_eval,Ncouronne,Ncyl]/
epaisseur[Ncouronne]
        # densite_crayon[i,Ncyl,Ncouronne,2]=F_cyl_z[ind_peak_eval,Ncouronne,Ncyl]/
epaisseur[Ncouronne]

    for Ncouronne_redux in range(ind_min_z,ind_max_z-1):

        junk,tcross_x_lin=phase_synchronisee(temps = t[ind_t0_deb:], signal = x[
ind_t0_deb:], seuil = 0.05)
        junk,tcross_Fx_lin=phase_synchronisee(temps = t[ind_t0_deb:], signal =
F_cyl_x[ind_t0_deb:,Ncouronne_redux,Ncyl]-np.mean(F_cyl_x[ind_t0_deb:,
Ncouronne_redux,Ncyl]), seuil = 0.05)
        dephasage_lin=np.mean(abs(tcross_Fx_lin[-5:-1]-tcross_x_lin[-5:-1]))
        # print("dephasage_lin = ",dephasage)
        C_lin=abs(np.amax(F_cyl_x[ind_t0_deb:,Ncouronne_redux,Ncyl])-np.amin(F_cyl_x
[ind_t0_deb:,Ncouronne_redux,Ncyl]))/2
        CN_lin[i,Ncyl,Ncouronne_redux]=(C_lin*np.cos(phi))/(0.5*rho*U*D*A*ohm*
epaisseur[Ncouronne_redux])
        ki_lin[i,Ncyl,Ncouronne_redux]=-C_lin*np.sin(phi)/(rho*(np.pi*0.25*D**2)*A*
ohm*ohm*epaisseur[Ncouronne_redux])



    moyenne_locale_crayon[i,Ncyl,0]=np.mean(densite_crayon[i,Ncyl,:,0])
    deviation_locale_crayon[i,Ncyl,0]=np.std(densite_crayon[i,Ncyl,:,0])
    deviation_redux_locale_crayon[i,Ncyl,0]=np.std(densite_crayon[i,Ncyl,ind_min_z:
```

```
ind_max_z,0])
    test=(np.sum(np.sum(F_cyl_x[ind_t_deb:,1:-2,:],axis=1),axis=1))
###########



###########

plt.figure(1,figsize=(19,10))
plt.plot(t,F_x)
plt.plot(t,force_upper_grid[:,1])
plt.plot(t,force_lower_grid[:,1])
# plt.plot(t,force_tot_crayons[:,1])
# plt.title("Resultante dans la direction du çforage " + result_file[4] + "Hz")
plt.xlabel('t (s)',size=20)
plt.ylabel('Fx (N)',size=20)
plt.legend(['F_x_assemblage','F_x_grille_haute','F_x_grille_basse'], loc="upper
right",fontsize=20)
plt.title("resultante crayon " + result_file[4] + "Hz")
plt.savefig(figure_file + "resultante_globale_" + result_file[4] + "Hz")

# addition_force=force_lower_grid[:,1]+force_upper_grid[:,1]+np.sum(np.sum(F_cyl_x,
axis=1),axis=1)
# addition_force=force_lower_grid[:,1]+force_upper_grid[:,1]+force_tot_crayons[:,1]
# plt.plot(t,addition_force)
plt.figure(2,figsize=(19,10))

# plt.title("Resultante le long de trois crayons pour un temps donne en regime
etabli a " + result_file[4] + "Hz")
plt.plot(z, densite_crayon[i,0,:,0] ,'--')
plt.plot(z, densite_crayon[i,6,:,0] ,'--')
plt.plot(z, densite_crayon[i,13,:,0] ,'--')
plt.plot(z, densite_crayon[i,18,:,0] ,'--')
plt.plot(z, densite_crayon[i,24,:,0] ,'--')
# plt.plot(z, densite_crayon[ind_peak_eval,:,6] ,'--')
# plt.plot(z, densite_crayon[ind_peak_eval,:,13] ,'--')
# plt.plot(z, densite_crayon[ind_peak_eval,:,18] ,'--')
# plt.plot(z, densite_crayon[ind_peak_eval,:,24] ,'--')
max_plot=2#np.amax(abs((densite_crayon[-1,24,:,0]*1.1)))
plt.plot([0.1,0.1],[-max_plot,max_plot],'--',color='k')
plt.plot([1.2,1.2],[-max_plot,max_plot],'--',color='k')
plt.xlabel('z (m)',size=20)
plt.ylabel('dFx (N/m)',size=20)
plt.ylim([-max_plot,max_plot])
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.grid('on')
plt.legend(["crayon 1","crayon 7","crayon 13","crayon 19","crayon 25","z=0.1m","z
=1.2m"],fontsize=20)
plt.savefig(figure_file + "densite_lineique_" + result_file[4] + "Hz")


plt.figure(3,figsize=(19,10))
plt.plot(t,np.sum(F_cyl_x[:,:,0],axis=1),'--')
plt.plot(t,np.sum(F_cyl_x[:,:,13],axis=1),'--')
plt.plot(t,np.sum(F_cyl_x[:,:,24],axis=1),'--')
plt.savefig(figure_file + "resultante_locale_" + result_file[4] + "Hz")

plt.figure(10,figsize=(19,10))
plt.subplot(2,1,1)
# plt.title("Resultante le long de trois crayons pour un temps donne en regime
etabli a " + result_file[4] + "Hz")
plt.plot(z, CN_lin[i,0,:] ,'--')
plt.plot(z, CN_lin[i,6,:] ,'--')
plt.plot(z, CN_lin[i,13,:] ,'--')
plt.plot(z, CN_lin[i,18,:] ,'--')
plt.plot(z, CN_lin[i,24,:] ,'--')
# plt.plot(z, densite_crayon[ind_peak_eval,:,6] ,'--')
# plt.plot(z, densite_crayon[ind_peak_eval,:,13] ,'--')
# plt.plot(z, densite_crayon[ind_peak_eval,:,18] ,'--')
```

```
    # plt.plot(z, densite_crayon[ind_peak_eval,:,24] ,'--')
    max_plot=2#np.amax(abs((densite_crayon[-1,24,:,0]*1.1)))
    # plt.plot([0.1,0.1],[-max_plot,max_plot],'--',color='k')
    # plt.plot([1.2,1.2],[-max_plot,max_plot],'--',color='k')
    plt.xlabel('z (m)',size=20)
    plt.ylabel('$C_{amort}$ (/)',size=20)
    # plt.ylim([0,0.5])
    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)
    plt.grid('on')
    plt.legend(["crayon 1","crayon 7","crayon 13","crayon 19","crayon 25"],fontsize=20)
    plt.subplot(2,1,2)
    # plt.title("Resultante le long de trois crayons pour un temps donne en regime
    etabli a " + result_file[4] + "Hz")
    plt.plot(z, ki_lin[i,0,:] ,'--')
    plt.plot(z, ki_lin[i,6,:] ,'--')
    plt.plot(z, ki_lin[i,13,:] ,'--')
    plt.plot(z, ki_lin[i,18,:] ,'--')
    plt.plot(z, ki_lin[i,24,:] ,'--')
    # plt.plot(z, densite_crayon[ind_peak_eval,:,6] ,'--')
    # plt.plot(z, densite_crayon[ind_peak_eval,:,13] ,'--')
    # plt.plot(z, densite_crayon[ind_peak_eval,:,18] ,'--')
    # plt.plot(z, densite_crayon[ind_peak_eval,:,24] ,'--')
    max_plot=2#np.amax(abs((densite_crayon[-1,24,:,0]*1.1)))
    # plt.plot([0.1,0.1],[-max_plot,max_plot],'--',color='k')
    # plt.plot([1.2,1.2],[-max_plot,max_plot],'--',color='k')
    plt.xlabel('z (m)',size=20)
    plt.ylabel('$C_{masse}$ (/)',size=20)
    # plt.ylim([0,2])
    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)
    plt.grid('on')
    plt.legend(["crayon 1","crayon 7","crayon 13","crayon 19","crayon 25"],fontsize=20)

    plt.savefig(figure_file + "Cz_" + result_file[4] + "Hz")

    plt.figure(20,figsize=(19,10))
    plt.subplot(2,1,1)
    # plt.title("Resultante le long de trois crayons pour un temps donne en regime
    etabli a " + result_file[4] + "Hz")
    for hihi in range (10,15):
        plt.plot(z, CN_lin[i,hihi,:] ,'--')
    # plt.plot(z, densite_crayon[ind_peak_eval,:,6] ,'--')
    # plt.plot(z, densite_crayon[ind_peak_eval,:,13] ,'--')
    # plt.plot(z, densite_crayon[ind_peak_eval,:,18] ,'--')
    # plt.plot(z, densite_crayon[ind_peak_eval,:,24] ,'--')
    max_plot=2#np.amax(abs((densite_crayon[-1,24,:,0]*1.1)))
    # plt.plot([0.1,0.1],[-max_plot,max_plot],'--',color='k')
    # plt.plot([1.2,1.2],[-max_plot,max_plot],'--',color='k')
    plt.xlabel('z (m)',size=20)
    plt.ylabel('$C_{amort}$ (/)',size=20)
    # plt.ylim([0,0.5])
    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)
    plt.grid('on')
    plt.legend(["crayon 1","crayon 7","crayon 13","crayon 19","crayon 25"],fontsize=20)
    plt.subplot(2,1,2)
    # plt.title("Resultante le long de trois crayons pour un temps donne en regime
    etabli a " + result_file[4] + "Hz")
    for hihi in range (10,15):
        plt.plot(z, ki_lin[i,hihi,:] ,'--')
    # plt.plot(z, densite_crayon[ind_peak_eval,:,6] ,'--')
    # plt.plot(z, densite_crayon[ind_peak_eval,:,13] ,'--')
    # plt.plot(z, densite_crayon[ind_peak_eval,:,18] ,'--')
    # plt.plot(z, densite_crayon[ind_peak_eval,:,24] ,'--')
    max_plot=2#np.amax(abs((densite_crayon[-1,24,:,0]*1.1)))
    # plt.plot([0.1,0.1],[-max_plot,max_plot],'--',color='k')
    # plt.plot([1.2,1.2],[-max_plot,max_plot],'--',color='k')
    plt.xlabel('z (m)',size=20)
    plt.ylabel('$C_{masse}$ (/)',size=20)
```

```
    # plt.ylim([0,2])
    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)
    plt.grid('on')
    plt.legend(["crayon 1","crayon 7","crayon 13","crayon 19","crayon 25"],fontsize=20)

    plt.savefig(figure_file + "Cz_" + result_file[4] + "Hz")

    # plt.figure(3,figsize=(19,10))
    # plt.plot(t,np.sum(F_cyl_x[:,:,0],axis=1),'--')
    # plt.plot(t,np.sum(F_cyl_x[:,:,13],axis=1),'--')
    # plt.plot(t,np.sum(F_cyl_x[:,:,24],axis=1),'--')
    # plt.savefig(figure_file + "resultante_locale_" + result_file[4] + "Hz")


    plt.figure(4,figsize=(19,10))
    plt.subplot(1,2,1)
    # plt.plot(np.linspace(1,25,25),CN_nobord[i,:],'k',marker='*',ls='--')
    plt.plot([1,2,3,4,5],CN_nobord[i,0:5],'k',marker='*',ls='--',label='Ligne 1')
    plt.plot([1,2,3,4,5],CN_nobord[i,5:10],'r',marker='*',ls='--',label='Ligne 2')
    plt.plot([1,2,3,4,5],CN_nobord[i,10:15],'g',marker='*',ls='--',label='Ligne 3')
    plt.plot([1,2,3,4,5],CN_nobord[i,15:20],'b',marker='*',ls='--',label='Ligne 4')
    plt.plot([1,2,3,4,5],CN_nobord[i,20:25],'orange',marker='*',ls='--',label='Ligne 5')
    plt.grid('on')
    plt.xlim([1,5])
    plt.ylim([0.15,0.35])
    plt.xticks([1,2,3,4,5],['1','2','3','4','5'],fontsize=20)
    plt.yticks(fontsize=20)
    plt.xlabel('Ny crayon (-)',size=20)
    plt.ylabel('$C_{amort}$ (/)',size=20)
    plt.legend(['Ligne 1','Ligne 2','Ligne 3','Ligne 4','Ligne 5'],fontsize=20,loc='
    lower center')
    plt.subplot(1,2,2)
    # plt.plot(np.linspace(1,25,25),ki_nobord[i,:],'k',marker='*',ls='--')
    plt.plot([1,2,3,4,5],ki_nobord[i,0:5],'k',marker='*',ls='--',label='Ligne 1')
    plt.plot([1,2,3,4,5],ki_nobord[i,5:10],'r',marker='*',ls='--',label='Ligne 2')
    plt.plot([1,2,3,4,5],ki_nobord[i,10:15],'v',marker='*',ls='--',label='Ligne 3')
    plt.plot([1,2,3,4,5],ki_nobord[i,15:20],'b',marker='*',ls='--',label='Ligne 4')
    plt.plot([1,2,3,4,5],ki_nobord[i,20:25],'orange',marker='*',ls='--',label='Ligne 5')
    plt.grid('on')
    plt.xlim([1,5])
    plt.ylim([1.25,1.6])
    plt.xticks([1,2,3,4,5],['1','2','3','4','5'],fontsize=20)
    plt.yticks(fontsize=20)
    plt.xlabel('Ny crayon (-)',size=20)
    plt.ylabel('$C_{masse}$ (/)',size=20)
    plt.legend(['Ligne 1','Ligne 2','Ligne 3','Ligne 4','Ligne 5'],fontsize=20,loc='
    lower center')
    # plt.legend(["maximum","moyenne","minimum"],fontsize=20)
    plt.savefig(figure_file + "Cxy_" + result_file[4] + "Hz")

    # plt.figure(6,figsize=(19,10))
    # plt.plot(t,F_x)
    # plt.plot(t,np.sum(np.sum(F_p_cyl_x,axis=1),axis=1))
    # # plt.plot(t,force_lower_grid[:,1])
    # plt.title("Resultante dans la direction du çforage " + result_file[4] + "Hz")
    # plt.xlabel('t (s)')
    # plt.ylabel('Fx (N)')
    # plt.legend(['F_x_assemblage','F_x_pression'], loc="upper right")
    # plt.title("resultante crayon " + result_file[4] + "Hz")
    ############
    print("mean distrib = ",np.around(np.mean(100*distribution_crayon[i,:,0]),2))
    print("std distrib = ",np.around(np.std(100*distribution_crayon[i,:,0]),2))

    print("mean locale = ",np.around(moyenne_locale_crayon[i,13,0],2))
    print("std redux locale = ",np.around(deviation_redux_locale_crayon[i,13,0],2))

    # print("f = ",result_file[4])
    print("CN mean = ",np.around(np.mean(CN_nobord[i,:]),3),"CN std = ",np.around(np.std
    (CN_nobord[i,:]),3))
```

```
    print("ki mean = ",np.around(np.mean(ki_nobord[i,:]),3),"ki std = ",np.around(np.std
    (ki_nobord[i,:]),3))

    print("CN lin mean = ",np.around(np.mean(CN_lin[i,13,ind_min_z:ind_max_z]),3),"CN
    lin std = ",np.around(np.std(CN_lin[i,13,ind_min_z:ind_max_z]),3))
    print("ki lin mean = ",np.around(np.mean(ki_lin[i,13,ind_min_z:ind_max_z]),3),"ki
    lin std = ",np.around(np.std(ki_lin[i,13,ind_min_z:ind_max_z]),3))
    i=i+1


    ############

CN=np.mean(CN_nobord,axis=1)
ki=np.mean(ki_nobord,axis=1)
# print(CN)
# print(ki)
# Monitoring point coordinates:
#       1  7.5000000e-02  0.0000000e+00 -6.6500000e-01 -> 5
#       2  7.5000000e-02  1.3400000e-02 -6.6500000e-01 -> 6
#       3  7.5000000e-02  2.6800000e-02 -6.6500000e-01 -> 7
#       4  7.5000000e-02 -1.3400000e-02 -6.6500000e-01 -> 4
#       5  7.5000000e-02 -2.6800000e-02 -6.6500000e-01 -> 3
#       6  7.5000000e-02  0.0000000e+00 -4.9800000e-01 -> 9
#       7  7.5000000e-02  0.0000000e+00 -3.3100000e-01 -> 8
#       8  7.5000000e-02  0.0000000e+00 -8.3200000e-01 -> 2
#       9  7.5000000e-02  0.0000000e+00 -1.0000000e+00 -> 1
# plt.figure(5,figsize=(19,10))
# # plt.title("Dephasage pression/deplacement")
# nombre_capteur=[1,2,3,4,5,6,7,8,9]
# couleurs=['k','k','red','red','g','red','red','k','k']
# for N_capteur,couleur in zip(nombre_capteur,couleurs):
#     plt.plot(f_fonda_dep[0:5],phase_fonda[0:5,N_capteur-1],ls='--',marker='+')
# plt.xlabel('F (Hz)',size=20)
# plt.ylabel('$\phi$ (rad)',size=20)
# plt.xticks(fontsize=20)
# plt.yticks(fontsize=20)
# plt.legend(['p5','p6','p7','p4','p3','p9','p8','p2','p1'],fontsize=20)
# plt.savefig(figure_file + "pressions_" + result_file[4] + "Hz")
if len(list_file)>4:
    plt.figure(7,figsize=(19,10))
    freqs=[1,2,3,4,5]
    plt.plot(freqs,CN,color='k',marker='*',ls='--',ms=10)
    for x,y in zip(freqs,CN):
        label=str(np.around(y,2))
        plt.annotate(label, (x,y),textcoords="offset points",xytext=(10,10),ha='center',
    fontsize=20)

    plt.xlabel('F (Hz)',size=20)
    plt.ylabel('$C_{amort}$ (-)',size=20)
    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)
    plt.grid('on')
    plt.ylim([0,0.8])
    plt.savefig(figure_file +"coeff_amort_nogrid_TLP.png")

    plt.figure(8,figsize=(19,10))

    plt.plot(freqs,ki,color='k',marker='*',ls='--',ms=10)
    for x,y in zip(freqs,ki):
        label=str(np.around(y,2))
        plt.annotate(label, (x,y),textcoords="offset points",xytext=(0,10),ha='center',
    fontsize=20)
    plt.xlabel('F (Hz)',size=20)
    plt.ylabel('$C_{masse}$ (-)',size=20)
    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)
    plt.grid('on')
    plt.ylim([1,2])
    # plt.legend(['CFD','TLP','Potentiel 2D'],fontsize=20)
    plt.savefig(figure_file +"coeff_masse_nogrid_TLP.png")
```

```
plt.figure(9,figsize=(19,10))
markers="+", "o", "d", ">", "*"
couleurs = "k", "r", "b", "g", "orange"
# freqs=[1,2,3,4,5]
freqs=[5]
for freq in freqs:
    marker=markers[freq-1]
    couleur=couleurs[freq-1]
    plt.plot(np.linspace(1,25,25),100*distribution_crayon[freq-1,:,0],ls='--',color=
'k',marker=marker,ms=6)
plt.xlabel('N crayon (-)',size=20)
plt.ylabel('$ F_{x\_crayon}/F_{xtot} (\%)$ ',size=20)
# plt.xlim([0,24])
plt.ylim([-10,10])
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.legend(['1Hz','2Hz','3Hz','4Hz','5Hz'],fontsize=20)
plt.grid('on')
plt.savefig(figure_file + "distribution_globale_fx_" + result_file[4] + "Hz")

plt.figure(10,figsize=(19,10))
markers="+", "o", "d", ">", "*"
couleurs = "k", "r", "b", "g", "orange"
for freq,marker,couleur in zip([1,2,3,4,5],markers,couleurs):
    plt.plot(np.reshape(np.linspace(1,25,25),(5,5)),np.reshape(moyenne_locale_crayon
[freq-1,:,0],(5,5)),ls='--',marker=marker,ms=8,color=couleur)
    # plt.errorbar(x=np.linspace(1,25,25),y=moyenne_locale_crayon[freq-1,:,0],yerr=
deviation_locale_crayon[freq-1,:,0],ls='--',marker=marker,ms=8,ecolor = 'k',color=
couleur,barsabove=True)
plt.xlabel('N crayon (-)',size=20)
plt.ylabel('$ dF_{x\_crayon\_mean} (N/m)$ ',size=20)
plt.xlim([1,25])
# plt.ylim([-10,10])
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.legend(['1Hz','2Hz','3Hz','4Hz','5Hz'],fontsize=20)
plt.savefig(figure_file + "distribution_locale_fx_moyz_" + result_file[4] + "Hz")

plt.figure(11,figsize=(19,10))
markers="+", "o", "d", ">", "*"
couleurs = "k", "r", "b", "g", "orange"
for freq,marker,couleur in zip([1,2,3,4,5],markers,couleurs):
    # plt.plot(np.linspace(1,25,25),deviation_locale_crayon[freq-1,:,0],ls='--',
marker=marker,ms=8,color=couleur)
    plt.errorbar(x=np.linspace(1,25,25),y=moyenne_locale_crayon[freq-1,:,0],xerr=0,
yerr=deviation_locale_crayon[freq-1,:,0],ls='--',marker=marker,ms=8,ecolor = couleur
,color=couleur,barsabove=True,capsize=20)
    # print("std = ",np.around(deviation_locale_crayon[freq-1,:,0],2))
    # print("")
    # print("mean = ",np.around(moyenne_locale_crayon[freq-1,:,0],2))

plt.xlabel('N crayon (-)',size=20)
plt.ylabel('$ dF_{x\_crayon\_mean\_z} (N/m)$ ',size=20)
plt.xlim([1,25])
# plt.ylim([-10,10])
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.legend(['1Hz','2Hz','3Hz','4Hz','5Hz'],fontsize=20)
plt.savefig(figure_file + "distribution_locale_fx_stdz_" + result_file[4] + "Hz")
```

Bibliographie

Adjiman, J. (2016). *Instabilités statiques d'un réseau de structures élancées sous écoulement axial.* Thèse de doctorat.

Brenneman, B., Shah, S. J., Williams, G. T. et Strumpell, J. H. (2003). Water confinement effects on fuel assembly motion and damping. *In Transactions of the 17th International Conference on Structural Mechanics in Reactor Technology (SMiRT 17)*, Prague, République Tchèque.

Chen, S.-S. (1987). *Flow-induced vibration of circular cylindrical structures.* Hemisphere Publishing Corporation, Washington.

Collard, B. (2005). Flow-induced damping of a pwr fuel assembly. *In IAEA-Tecdoc-1454, Structural behaviour of fuel assemblies for water cooled reactors*, pages 289–296, Cadarache.

Collard, B., Pisapia, S., Bellizzi, S. et Witters, F. (2003). Pwr fuel assembly modal testing and analysis. *In Proceedings of the ASME 2003 Pressure Vessels and Piping Conference*, PVP2003-2084, pages 147–152, Cleveland, OH, USA.

De Ridder, J., Doaré, O., Degroote, J., Van Tichelen, K., Schuurmans, P. et Vierendeels, J. (2015). Simulating the fluid forces and fluid-elastic instabilities of a clamped–clamped cylinder in turbulent axial flow. *Journal of Fluids and Structures*, 55:139–154.

Divaret, L. (2014). *Caractérisation des forces fluides s'exerçant sur un faisceau de cylindres oscillant latéralement en écoulement axial.* Thèse de doctorat.

Divaret, L., Cadot, O., Moussou, P. et Doaré, O. (2014). Normal forces exerted upon a long cylinder oscillating in an axial flow. *Journal of Fluid Mechanics*, 752:649–669.

Flamand, J. C., Maguin, J. C., Mattei, A., Rigaudeau, J. et Leroux, J. C. (1991). Influence of axial coolant flow on fuel assembly damping for the response to horizontal seismic loads. *In Transactions of the 11th International Conference on Structural Mechanics in Reactor Technology (SMiRT 11)*, C06/4, Tokyo, Japon.

Fujita, K. (1990). Flow-induced vibration and fluid-structure interaction in nuclear power plant components. *Journal of Wind Engineering and Industrial Aerodynamics*, 33(1):405–418.

Goldstein, H., Poole, C. et Safko, J. (2001). *Classical Mechancis.* Addison Wesley.

Guyon, P., Hulin, J. et Petit, L. (2001). *Hydrodynamique physique.* CNRS Editions, EDP Sciences.

Joly, A. (2018). *Forces fluides stationnaires exercées sur un cylindre déformé en écoulement axial et confiné à application au dimensionnement sismique des assemblages combustibles.* Thèse de doctorat.

Joly, A., de Buretel de Chassey, N., Martin, A., Cadot, O., Pastur, L. et Moussou, P. (2021). Direct measurement of steady fluid forces upon a deformed cylinder in confined axial flow. *Journal of Fluids and Structures*, 104:103326.

Lamb, H. (1932). *Hydrodynamics.* Dover.

Lee, K. e. a. (2013). Fuel assembly damping summary. *In Transactions of the Korean Nuclear Society Autumn Meeting.*

Lu, R. Y. et Seel, D. D. (2006). Pwr fuel assembly damping characteristics. *In Proceedings of the ICONE14 International Conference on Nuclear Engineering*, ICONE14-89535, Miami, FLA, USA.

Martin, J. (2004). Mécanismes de transformation de mouvement à contact local. *Techniques de l'ingénieur Transmission de puissance mécanique : accouplement, embrayage, freinage*, base documentaire TIB184DUO.(ref article b5910). fre.

Moretti, P. M. et Lowery, R. L. (1976). Hydrodynamic Inertia Coefficients for a Tube Surrounded by Rigid Tubes. *Journal of Pressure Vessel Technology*, 98(3):190–193.

Païdoussis, M. P. (1973). Dynamics of cylindrical structures subjected to axial flow. *Journal of Sound and Vibration*, 29(3):365–385.

Païdoussis, M. P. (2014). *Fluid-Structure Interactions : Slender Structures and Axial Flow*, volume 1. Elsevier Academic Press, Oxford, 2e édition.

Païdoussis, M. P. (2021). Dynamics of cylindrical structures in axial flow : A review. *Journal of Fluids and Structures*, 107:103374.

Pisapia, S., Collard, B., Bellizzi, S. et Mori, V. (2003). Modal testing and identification of a pwr fuel assembly. *In Transactions of the 17th International Conference on Structural Mechanics in Reactor Technology (SMiRT 17)*, Prague. Article C01-4.

Schettino, C., Gouvea, J. et Medeiros, N. (2013). Numerical analysis of the spacer grids compression strength. *In 2013 International Nuclear Atlantic Conference - INAC 2013*, Recife, Brésil.

Stokes, F. E. et King, R. A. (1979). Pwr fuel assembly dynamic characteristics. *In BNES Vibration in nuclear plant*, Keswick, UK.

Tanaka, M., Fujita, K., Hotta, A. et Kono, N. (1988). Parallel flow induced damping of pwr fuel assembly. *In Proceedings of the ASME 1988 Pressure Vessels and Piping Conference*, Pittsburgh, PA, USA.

Taylor, G. I. (1952). Analysis of the swimming of long and narrow animals. *Proceedings of the Royal Society of London A : Mathematical, Physical and Engineering Sciences*, 214:158–183.

Viallet, E., Bolsee, G., Ladouceur, B., Goubin, T. et Rigaudeau, J. (2003). Validation of pwr core seismic models with shaking table tests on interacting scale 1 fuel assemblies. *In Transactions of the 17th International Conference on Structural Mechanics in Reactor Technology (SMiRT 17)*, Prague, République Tchèque.

Viallet, E. et Kestens, T. (2003). Prediction of flow induced damping of a pwr fuel assembly in case of seismic and LOCA load case. *In Transactions of the 17th International Conference on Structural Mechanics in Reactor Technology (SMiRT 17)*, Prague, République Tchèque.

ECOLE DOCTORALE

Titre : Caractérisation de l'interaction fluide-structure d'un assemblage de coeur de réacteur sous forçage sismique

Mots clés : interaction fluide-structure, assemblage combustible, essais en canal hydraulique, écoulement axial, écoulement transverse, faisceau de cylindres

Résumé : Dans le cadre des études de sûreté, il est nécessaire de disposer de modèles validés des forces fluides s'exerçant sur des assemblages combustibles en cas de séisme. Dans la technologie REP (réacteur à eau pressurisée) utilisée en France, les assemblages sont constitués de faisceaux de crayons cylindriques disposés en réseau carré et soumis à un écoulement axial. En cas de séisme, ces assemblages sont soumis à des oscillations latérales. Ces vibrations peuvent mener à la déformation des assemblages. Des campagnes d'essais sur assemblage combustible réel ont été réalisées et ont mis en évidence l'existence de forces fluides dissipatives dont l'évolution et la répartition spatiale reste encore mal documentées. L'objectif des présents travaux est de contribuer à la description de ces forces en dynamique. Le modèle de référence pour la description de la dynamique des structures élancées sous écoulement axial est le modèle de TLP (Taylor-Lighthill-Païdoussis). Dans la littérature ce modèle a été validé pour une structure très proche du faisceau de cylindres. Des études statiques réalisées sur un cylindre dans un faisceau ont montré une influence non négligeable des extrémités sur les efforts exercés par le fluide sur la structure. La littérature propose peu de cas de validation du modèle TLP pour des structures plus complexes et proches d'un assemblage combustible. Afin d'évaluer la capacité du modèle à décrire les efforts fluides s'exerçant sur une structure plus proche d'un assemblage combustible, un nouveau banc d'essai a été conçu et mis en service. Ce banc d'essais permet de faire osciller latéralement un faisceau de cylindre 5×5 muni de grilles à chaque extrémité dans un écoulement axial. Des mesures de force au niveau de chaque grille donne accès à la résultante d'effort de la structure. Un modèle numérique complète cette approche expérimentale et donne accès à la répartition des efforts dans la structure. Les forces obtenues expérimentalement semble indiquer un effet important des grilles. Les forces issues des simulations confortent les observations faites sur un faisceau de cylindres et apportent de nouveaux éléments sur la répartition des efforts locaux ainsi que l'influence des extrémités.

Title : Characterisation of the fluid-structure interaction of a reactor core assembly under seismic forcing

Keywords : fluid-structure interaction, fuel assembly, water tunnel experiments, axial flow, cross flow, cylinder bundle

Abstract : Within the framework of safety studies, it is necessary to have validated models of the fluid forces exerted on fuel assemblies in the event of an earthquake. In the PWR (Pressurized Water Reactor) technology used in France, fuel assemblies are made of cylindrical rods bundles arranged in a square array and subjected to an axial flow. In case of an earthquake, these assemblies are subjected to lateral oscillations. These vibrations can lead to the deformation of the assemblies. Test campaigns on real fuel assemblies have been carried out and have revealed the existence of dissipative fluid forces whose evolution and spatial distribution are still poorly documented. The objective of the present work is to contribute to the description of these forces in dynamics. The reference model for the description of the dynamics of slender structures under axial flow is the TLP model (Taylor-Lighthill-Païdoussis). In the literature this model has been validated for a structure very close to the cylinder bundle. Static studies performed on a cylinder in a bundle have shown a non-negligible influence of the extremities on the forces exerted by the fluid on the structure. The literature proposes few validation cases of the TLP model for more complex structures close to a fuel assembly. In order to evaluate the capacity of the model to describe the fluid forces exerted on a structure closer to a fuel assembly, a new test bench was designed and put into service. This test rig allows to laterally oscillate a cylinder bundle 5×5 with grids at each end in an axial flow. Force measurements at each grid give access to the resultant force of the structure. A numerical model completes this experimental approach and gives access to the distribution of forces in the structure. The forces obtained experimentally seem to indicate an important effect of the grids. The forces obtained from the simulations confirm the observations made on a bundle of cylinders and bring new elements on the distribution of the local forces as well as the influence of the extremities.

Institut Polytechnique de Paris
91120 Palaiseau, France