

- Automatisation du traitement de données SAXS -

Conversion et Analyse incluses

Adrien TOULOUSE

Un compte-rendu présenté dans le cadre de
mon alternance au CEA
Encadré par **Guillaume FREYCHET**



Département de physique
Université de Montpellier
France
le 24 juillet 2025

Table des matières

1 Le Commissariat à l'énergie Atomique	6
1.1 Organisation du CEA	6
1.1.1 Organisation nationale	6
1.1.2 Le Site de Grenoble	6
1.1.3 Mon département	7
1.1.4 Mon laboratoire	8
1.2 Missions du LPMS	8
1.2.1 Aspect recherche	8
1.2.2 Aspect support	8
1.3 Ma mission dans le laboratoire	9
2 Planification de mes tâches et recherche bibliographique	10
2.1 Bibliographie	10
2.1.1 Les méthodes de mesure par rayons X	10
2.1.2 Dispositif expérimental	12
2.1.3 Les différents formats de données	14
2.2 Organisation	17
3 Mise en place de la conversion de données du format edf à hdf5	19
3.1 Problématiques	19
3.2 Création du fichier de structure de NXcanSAS	19
3.3 Créer le fichier de configuration	21
3.3.1 Structuration du fichier	21
3.3.2 Mise en place d'un GUI pour faciliter la construction d'un fichier configuration	22
3.3.3 Un exemple de création fichier de configuration	23
3.4 Conversion automatique des fichiers en hdf5	24
3.4.1 Gestion des fichiers	24
3.4.2 Conversion des données	25
3.4.3 Le GUI pour lancer la conversion automatique	27
3.4.4 Modification des fichiers NXcanSAS	28
3.4.5 Obstacles rencontrés	29
4 Mise en place du traitement des données automatiques	30
4.1 Les différentes étapes du traitement de données	30
4.1.1 Passage dans l'espace de Fourier	30
4.1.2 Le caking	32
4.1.3 Intégration verticale et horizontale	32
4.1.4 Intégration radiale et azimutale	34
4.1.5 Conversion en intensité absolue	35
4.2 Préparation de la réduction des données	36
4.2.1 Quelles opérations de réductions des données ?	36
4.2.2 Adaptation de l'acquisition des données	37

 TABLE DES MATIÈRES

4.2.3	Challenges et approches du déploiement des processus d'analyse des données	37
4.2.4	Introduction à la structure de NexusFile	38
4.3	Les méthodes de la classe NexusFile	38
4.3.1	Mise en place des méthodes de base	38
4.3.2	Mise en place des méthodes de traitement	40
4.4	Perspectives d'amélioration de la classe NexusFile	45
4.5	Mise en place du GUI	46
4.5.1	Construction du panneau d'entrée	46
4.5.2	Le panneau des paramètres	48
4.5.3	Lancement du traitement des données	49
4.5.4	Le panneau de console	50
5	Finitions et automatisation complète du traitement	51
5.1	Jenkins	51
5.1.1	Introduction	51
5.1.2	Mise à jour du programme	52
5.1.3	Lancement du programme	52
5.2	Le fichier .toml	53
6	Conclusion	54
7	Annexes	55
7.1	Bibliothèques Python utilisées	55

Remerciements

Je tiens à remercier Dr. Guillaume FREYCHET qui a été un excellent encadrant tout au long de mon alternance. Il a su répondre présent lorsque j'avais besoin d'aide et n'a pas hésité à me guider tout au long du projet.

Je remercie aussi Patrick QUEMERE qui a été d'une grande aide dans le déploiement du projet au niveau numérique. Il m'a énormément aidé dans l'acquisition de mes nouvelles compétences numériques.

Je voudrais aussi remercier le professeur Brahim GUIZAL qui m'a été d'une grande aide au niveau logistique et académique au cours de cette alternance.

Bien sûr je remercie grandement tous les collègues que j'ai pu rencontrer au cours de l'année et qui ont instauré une ambiance très accueillante et chaleureuse au sein du laboratoire.

Résumé

Le traitement de données d'expérience SAXS étant relativement complexe, il est important de mettre en place et de développer des outils qui permettent leur traitement de manière aisée afin de perdre le moins de temps possible. Dans ce rapport d'alternance, nous présentons une procédure qui a été mise en place afin d'automatiser à 100% ce traitement. Une interface graphique a également été développée pour modifier certaines métadonnées en cas d'échec de la conversion des données ou d'erreur d'enregistrement lors de l'expérience. Nous traitons aussi la manière dont nous avons développé une bibliothèque Python permettant à des utilisateurs plus expérimentés d'utiliser le programme directement dans des scripts Python.

Abstract

SAXS data treatment is relatively complex, so it is important to develop and to set up tools that allow easy data treatment and save time. In this report, we talk about a pipeline that was set up to completely automatize SAXS data treatment. We will also present a graphical user interface that also allows a user to change certain metadata that could've been saved improperly. We also discuss how we developed a Python library that lets more experienced users to use the program directly into their Python script.

1 Le Commissariat à l'énergie Atomique

1.1 Organisation du CEA

1.1.1 Organisation nationale

Le Commissariat à l'énergie Atomique (CEA) est un organisme qui a vu le jour presque en même temps que la bombe nucléaire. C'est le 18 octobre 1945 que l'ordonnance de création du CEA est signée par le chef du gouvernement de l'époque : Charles de Gaulle. Le CEA a été créé avec le but de répondre aux questions scientifiques et techniques autour de l'énergie nucléaire. Bien entendu, depuis 1945, le CEA a énormément évolué et ne se focalise plus uniquement sur l'énergie atomique mais s'est aussi développé dans la recherche de nouvelles énergies et technologies, par exemple en micro-électronique.

Il existe, à l'heure actuelle, 9 sites (sans compter le siège social et autres antennes locales) disséminés un peu partout en France, chacun possédant son propre directeur.

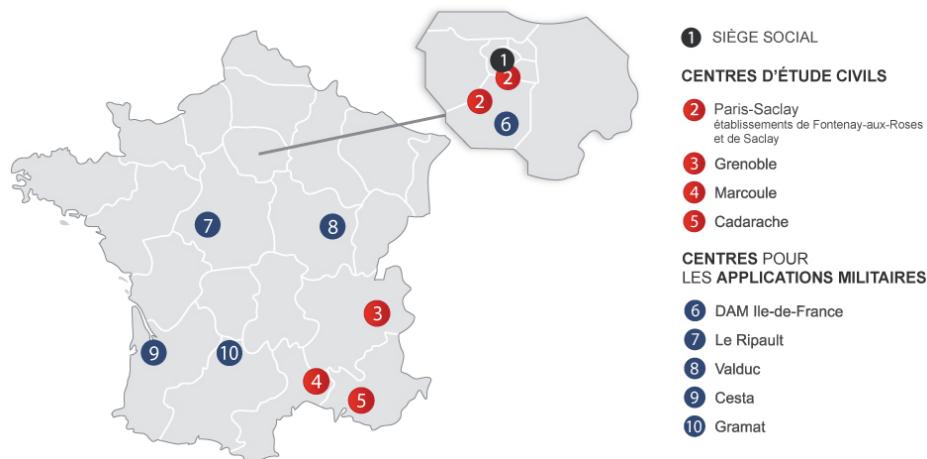


FIGURE 2 – Cartes des différents centres CEA en France [2]

1.1.2 Le Site de Grenoble

Du côté du CEA de Grenoble, le centre est séparé en trois instituts de recherche :

1. Le Laboratoire d'Électronique et de Technologie d'Information (LETI) : Il favorise le développement de nouvelles technologies en collaboration avec des industriels.
2. Le Laboratoire d'Innovation pour les Technologies des Énergies nouvelles et les Nanomatériaux (LITEN) : Il favorise le développement de nouvelles énergies et est dédié à la transition énergétique. Le LITEN se concentre surtout sur le développement de l'énergie solaire et les moyens de stocker l'énergie.
3. L'Institut de Recherche Interdisciplinaire de Grenoble (IRIG) : Il favorise le travail de recherche collaboratif à l'internationale. C'est principalement de la recherche fondamentale multidisciplinaire.

A l'échelle nationale, le CEA accueille environ 18 000 employés. A Grenoble c'est environ 5000 employés qui travaillent sur le site. Je fais personnellement parti du LETI qui compte environ

1 LE COMMISSARIAT À L'ÉNERGIE ATOMIQUE

2000 personnes. Le LETI est une structure complexe, par conséquent je vais décrire uniquement la structure menant à mon laboratoire. Voici un organigramme qui montre en partie l'organisation du LETI :

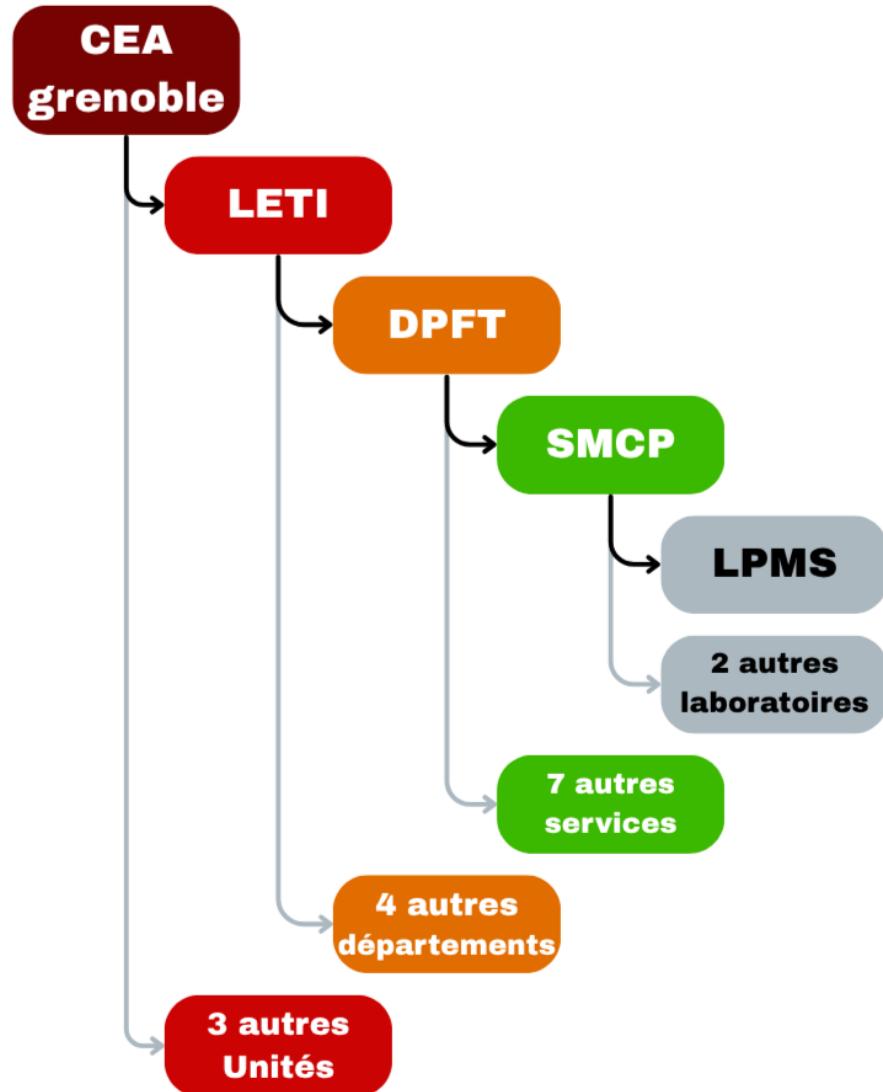


FIGURE 3 – Organigramme hiérarchique du LETI. Les flèches noires indiquent mon appartenance à chaque niveau

1.1.3 Mon département

Le LETI est dirigé par Sébastien DOUVE. Il s'occupe de 7 départements dont le Département des PlateForme Technologique (DPFT) dont je fais parti. Le DPFT, qui est administré par un chef de département, est composé de 7 services.

Je fais parti du Service de Métrologie et de Caractérisation Physique (SMCP) qui se charge d'étudier la structure et les propriétés d'objets nanométriques.

1.1.4 Mon laboratoire

Ce service (SMCP) est ensuite divisé en trois laboratoires, notamment le Laboratoire des Propriétés des Matériaux et Structure (LPMS) qui analyse plus en détail la structure et autres propriétés (optiques, mécaniques, électriques, structurales...) de structure nanométrique à l'aide d'expériences utilisant des rayonnements X, optiques, ... Les mesures sont effectuées à travers l'utilisation de nombreux équipements. Chaque laboratoire est géré par un responsable de laboratoire. Il gère une équipe dont mon tuteur fait partie : Guillaume FREYCHET.

Au sein de ce laboratoire se trouve plusieurs plateformes, j'ai eu l'occasion de travailler au sein de la PlateForme NanoCaractérisation (PFNC) qui intègre le Centre de Compétence Rayon X.

Enfin, on arrive à moi, Adrien TOULOUSE, alternant au CEA ! Il y a donc une chaîne de 7 personnes en partant du plus haut niveau pour arriver jusqu'à moi. Je n'ai rencontré mes supérieurs que jusqu'au niveau n+3.

1.2 Missions du LPMS

1.2.1 Aspect recherche

Au LPMS, et plus généralement dans le SMCP, la mission principale consiste à développer des nouvelles techniques pour caractériser les propriétés structurales et physico-chimiques d'objets nanométriques. Pour citer un exemple, le sujet de thèse d'un doctorant de mon laboratoire est la caractérisation de la rugosité de réseaux de lignes à des échelles inférieures au nanomètre.

L'étude de matériaux dans leurs conditions d'opération, appelée mesures in-situ ou operando, est importante. Par conséquent, de plus en plus d'expériences se font sous vide mais aussi sous conditions plus ou moins variées. La température, l'humidité ou la pression doivent ainsi être contrôlées dans la chambre qui contient l'échantillon analysé afin de pouvoir prédire comment va se comporter la structure nanométrique dans des conditions réelles.

1.2.2 Aspect support

Le SMCP/LPMS a aussi une fonction de "support" vis-à-vis des industriels. Ils sont en effet là pour aider les entreprises à déployer des nouveaux outils de métrologie dans leurs locaux ou réaliser la caractérisation de matériaux produits.

Pour se faire, une série d'analyses est réalisée sur les matériaux étudiés afin de savoir s'ils ont bien les bonnes propriétés pour être utilisables et/ou commercialisables et s'il faut ajuster les procédés de fabrication. Le SMCP/LPMS travaillant sur des objets de taille nanométrique, les entreprises qui sollicitent le laboratoire sont souvent des entreprises liées à la microélectronique et à la fabrication de micro-processeur ou autres composants informatiques qui demandent une haute précision afin de fonctionner.

Voici donc les principales missions de mon laboratoire, et plus généralement de mon service. Nous allons donc passer à l'explication concrète de ma mission.

1 LE COMMISSARIAT À L'ÉNERGIE ATOMIQUE

1.3 Ma mission dans le laboratoire

Voici l'intitulé de ma mission :

Automatisation de l'analyse de données en temps réel et inter-plateforme pour le SAXS

Dans cet intitulé, plusieurs choses sont à comprendre :

1. SAXS : acronyme de "Small Angle X-ray Scattering", ou en français "diffusion des rayons X aux petits angles". WAXS veut dire la même chose mais le S de Small devient un W de Wide donc l'étude des grands angles.
2. Analyse de données : il s'agit donc de traiter des données qui sortent d'un équipement SAXS.
3. Temps réel : le traitement doit se faire à mesure que les données expérimentales sont collectées.
4. Automatisation : Il ne doit pas y avoir d'intervention humaine, ou alors très peu.
5. inter-plateforme : l'automatisation doit pouvoir marcher pour plusieurs appareils et non pas qu'un seul.

Nous comprenons donc bien de quels enjeux il s'agit, mais il reste d'autres questions : Quelles données ? Comment les traiter ? Quel format utiliser ? Comment rendre les résultats exploitables par la communauté scientifique ?

La réponse à ces questions se trouvent dans l'énoncé de la mission qui m'a été confiée :

L'objectif de cette alternance est de convertir le format de données actuels de l'équipement SAXS en un format .hdf5 défini par la communauté pour permettre une analyse automatique et en temps réel des données collectées.

Le but va donc être de convertir des fichiers de données, initialement au format edf, en format hdf5. Afin d'organiser les métadonnées ainsi que les données brutes et les données traitées, le fichier hdf5 va suivre le standard NXcanSAS de NeXus. Cette standardisation permettra une meilleure compréhension entre équipes scientifiques du même milieu. Cette conversion, ainsi que quelques traitements de données, va devoir se faire de manière automatique.

La mission se fait donc en plusieurs étapes : récupérer les données de l'équipement, les convertir dans le format hdf5 avec le standard Nexus et enfin analyser les données.

2 Planification de mes tâches et recherche bibliographique

La première partie de mon alternance a consisté en une recherche bibliographique pour comprendre les différentes tâches à réaliser décrites ci-dessus. Voici les plusieurs points à explorer :

1. Les méthodes de caractérisations par rayons X
2. Le format edf
3. Le format hdf5 et le standard NeXus
4. La gestion de ces formats dans Python
5. Le traitement de données nécessaire pour l'analyse de données SAXS

2.1 Bibliographie

2.1.1 Les méthodes de mesure par rayons X

Les méthodes de mesure SAXS et WAXS reposent sur l'interaction entre lumière et matière. En effet, lorsqu'un photon interagit avec de la matière, il peut être absorbé ou diffusé. L'aspect adsorption de la matière peut être étudié par des techniques de spectroscopie telle que la fluorescence X. Dans notre cas, et comme son nom l'indique, la diffusion centrale des rayons X (SAXS en anglais) est basée sur les processus de diffusion élastique des rayons X. Il suffit donc d'exposer un matériau avec de la lumière et de placer un écran de l'autre côté de ce matériau pour collecter de l'information sur la structure électronique de l'échantillon.

Dans le cas du SAXS, des rayons X sont utilisés pour plusieurs raisons. Tout d'abord la pénétration des photons. L'absorption des photons par la matière est représenté par l'atténuation et peut être calculée avec la loi de Beer-Lambert :

$$I(r) = I_0 e^{-\mu r} \quad (1)$$

L'intensité transmise (i.e. le nombre de photons) à travers un échantillon d'épaisseur r est représentée par $I(r)$. Cette intensité perçue est proportionnelle à l'intensité initiale I_0 mais dépend aussi de ce qu'on appelle le coefficient d'atténuation linéique μ . Ce coefficient d'atténuation linéique dépend lui-même de plusieurs paramètres, mais principalement de l'énergie de la lumière qui est utilisée pour exposer le matériau et du matériau lui-même.

Le NIST a créé une base de données pour fournir ces coefficients [14]. Ces coefficients sont normalisés par la masse volumique ρ de l'élément, il faut donc multiplier la grandeur donnée par la base de données par ρ . Par exemple pour du Silicium pur $\rho = 2.329085 \text{ g.cm}^{-3}$, ce coefficient est de $1570 \text{ cm}^2 \cdot \text{g}^{-1}$ pour un rayonnement de 1keV et est de $0.1835 \text{ cm}^2 \cdot \text{g}^{-1}$ pour un rayonnement de 100keV. En multipliant par la masse volumique on obtient respectivement $\mu_1 = 3656 \text{ cm}^{-1}$ et $\mu_2 = 0.4273 \text{ cm}^{-1}$. En traçant la relation 1 pour ces deux coefficients, on obtient la figure 4.

La première raison pour laquelle on utilise des rayons X est que le phénomène de diffusion des rayons X est en compétition avec le phénomène d'absorption, utiliser une plus grande énergie permet de réduire le phénomène d'absorption et donc de permettre de récupérer suffisamment de signal sur le détecteur SAXS sans avoir à fabriquer des échantillons extrêmement fins.

2 PLANIFICATION DE MES TÂCHES ET RECHERCHE BIBLIOGRAPHIQUE

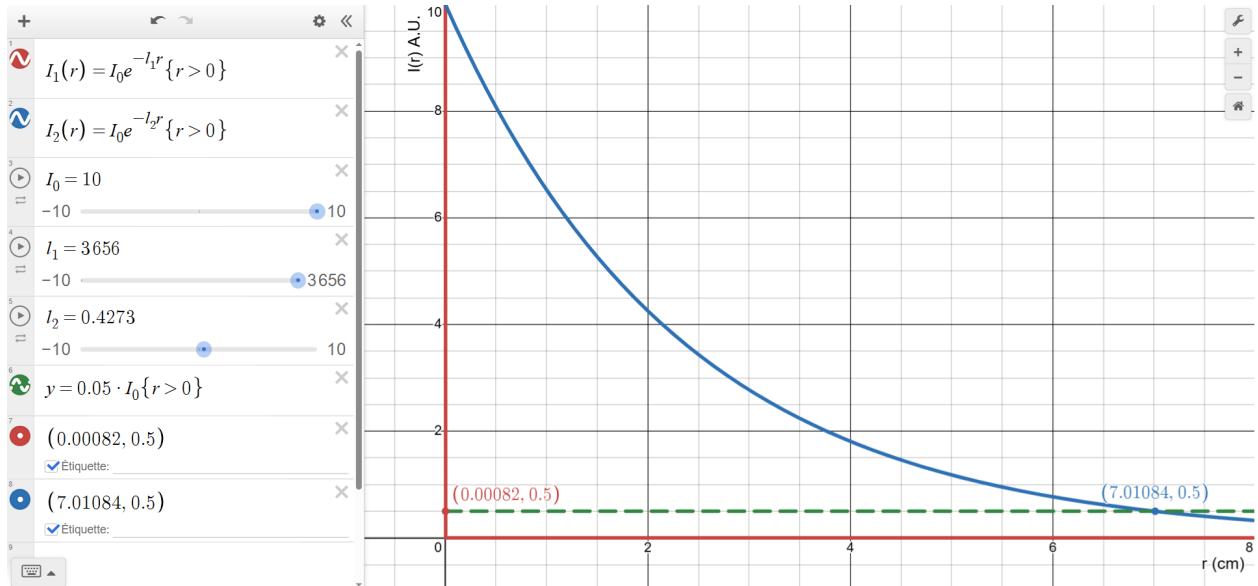


FIGURE 4 – On voit que le rayonnement s'atténue bien plus vite pour 1keV que pour 100keV. On atteint 5% de I_0 10^5 fois plus vite environ. On pénètre 0.0008 cm d'épaisseur pour 1kev et 7 cm d'épaisseur pour 100kev.

La deuxième raison pour laquelle on utilise les rayons X, c'est pour leurs petites longueurs d'ondes (de l'ordre de l'Ångström). Dans le contexte de la microélectronique, on cherche à caractériser des structures de tailles nanométriques. Ainsi, pour pouvoir les étudier, il faut utiliser des rayonnements qui ont des longueurs d'ondes plus faibles ou comparables à la taille des structures étudiées.

L'utilisation de rayons X implique plusieurs phénomènes physiques qu'il faudra prendre en compte. Par exemple : la diffusion des rayons X peut être élastique ou inélastique. Dans le cas inélastique (diffusion de Compton), le photon perd de l'énergie lors du phénomène de diffusion, le photon diffusé a ainsi une énergie différente du photon incident. Les phénomènes de diffusion inélastiques sont majoritaires lorsque l'énergie des photons est supérieure à 45 keV, ce qui correspond à une longueur d'onde d'environ 0.275 Å en utilisant la relation de Planck : $E = hf = \frac{hc}{\lambda}$. Dans les expériences SAXS et WAXS on utilise des photons d'énergie inférieure à 20 keV, et les phénomènes inélastiques peuvent donc être négligés.

On se concentre donc principalement sur les effets de diffusion élastique (diffusion de Thomson). Lors d'une diffusion élastique, le photon incident ne perd pas d'énergie.

Une expérience consiste ainsi à la mesure de l'intensité de la lumière incidente diffusée par un échantillon sur un écran bidimensionnel. La figure 5 schématise une configuration standard. La lumière va donc interagir avec l'échantillon et permettre d'en savoir plus sur la structure électronique interne de celui-ci sans le détruire. C'est un des avantages de ces méthodes de caractérisation,

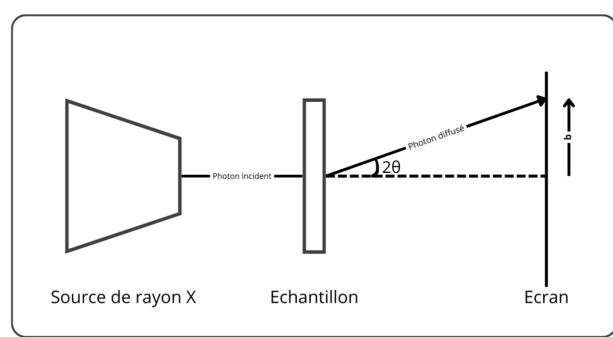


FIGURE 5 – Schéma représentant une configuration expérimentale de SAXS ou WAXS

elles ne sont pas destructives pour l'échantillon.

Les mesures présentées précédemment sont réalisées en transmission. Un seconde approche pour étudier la surface d'un échantillon est nommée GISAXS (Grazing Incidence SAXS). Le phénomène physique est identique mais la configuration expérimentale est différente, avec le faisceau de rayons X arrivant sur l'échantillon avec un angle d'incidence faible.

2.1.2 Dispositif expérimental

Les expériences de diffusion aux petits et grands angles (SAXS et WAXS respectivement), sont réalisées sur l'équipement XEUSS de la plateforme de nanocaractérisation du CEA Grenoble. Le XEUSS est un appareil fabriqué par Xenocs [15] très polyvalent. Il permet de faire des mesures SAXS et WAXS, avec possibilité de faire les deux en simultanée. Il est équipé de deux sources (une source Cu pour des rayons X émis de 8.047 keV et une source Mo pour des rayons X émis de 17.48 keV). Il est également possible de régler la collimation du faisceau incident. Une photo du XEUSS et sa schématisation sont présentées dans la figure 6.

Sur la figure 6a on a une photographie de l'équipement complet et sur la figure 6b une schématisation de celui-ci :

1. Le premier bloc concerne tout ce qui a un rapport avec la source de rayons X et le faisceau incident.
2. Le deuxième bloc représente la chambre où l'échantillon est installé, sur des portes échantillons spécifiques au type d'échantillon (éprouvette, poudre, suspension...)
3. Les troisièmes blocs représentent toute la partie détection du faisceau diffusé. Le sous bloc a est dédié à la détection SAXS, le bloc b est dédié à la détection WAXS. La figure 7 est une prise de vue plus détaillée sur la disposition des détecteurs SAXS et WAXS à l'intérieur de la chambre qui accueille l'échantillon.

On voit clairement que le détecteur SAXS a une plus grande marge de manœuvre quand il s'agit de la distance échantillon - détecteur par rapport au détecteur WAXS. Cela vient du fait que pour détecter la lumière diffusée aux petits angles, on veut mettre le plus de distance possible entre l'échantillon et le détecteur. A l'inverse, pour les grands angles, on veut que le détecteur soit le plus proche possible de l'échantillon.

On voit ici que le détecteur WAXS est positionné proche de l'échantillon avec un angle afin de couvrir une grande gamme angulaire. Une correction des effets géométriques dû à cette rotation sera effectuée lors de la réduction de données. Au contraire, le détecteur SAXS ne subit jamais de rotation il translate simplement pour s'approcher ou s'éloigner du détecteur.

Afin d'assurer correctement la collecte des données, l'équipement XEUSS est connecté à un "ordinateur équipement" qui contrôle la collecte et l'enregistrement des données. Cet ordinateur n'est pas directement connecté au réseau CEA pour des raisons de cybersécurité, il est cependant connecté à un serveur (nommé kiev). Il est possible d'écrire/transférer des fichiers depuis le PC équipement sur le serveur kiev mais pas l'inverse afin de garantir la sécurité de l'équipement. Il

2 PLANIFICATION DE MES TÂCHES ET RECHERCHE BIBLIOGRAPHIQUE

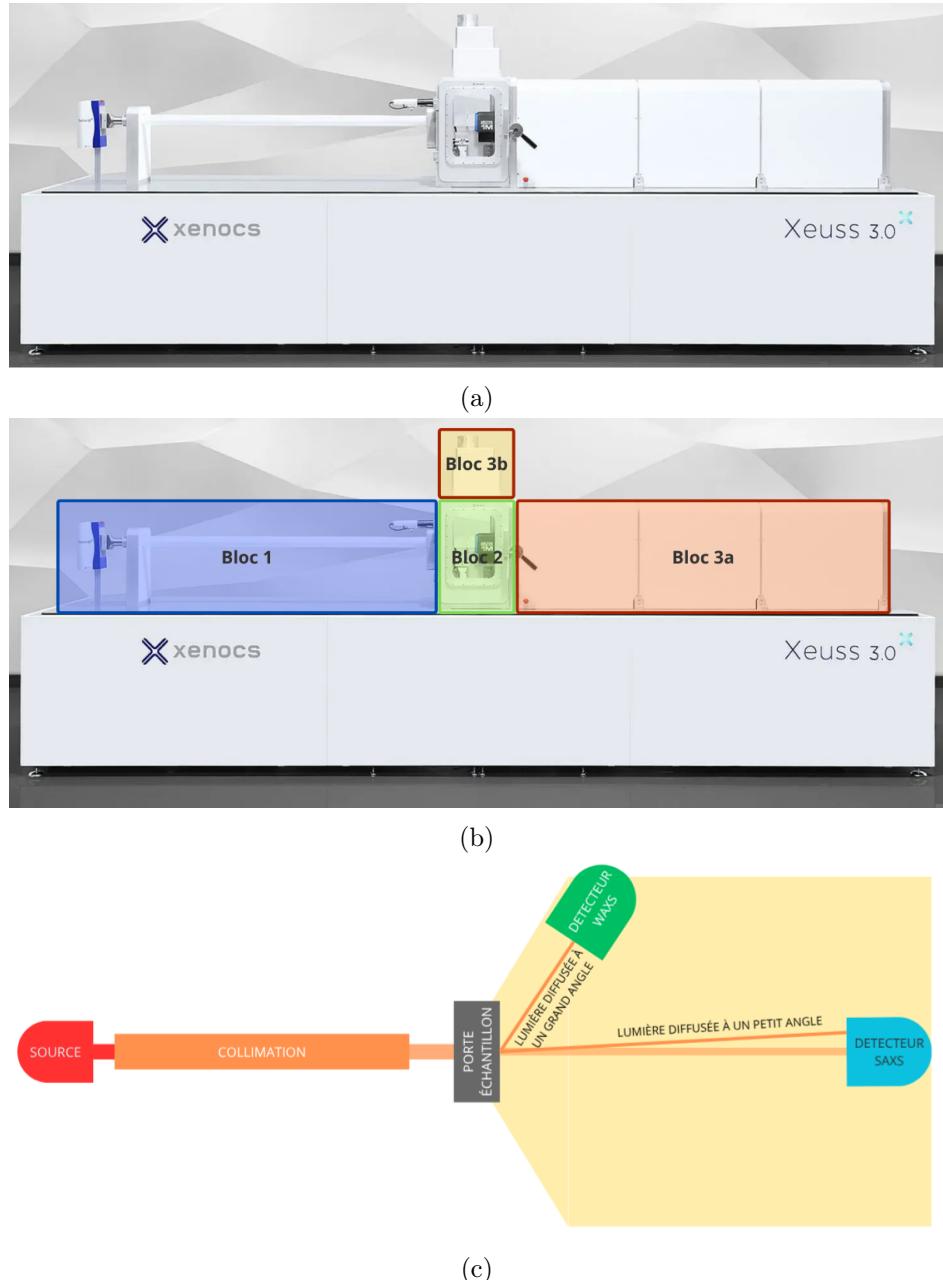


FIGURE 6 – Photo et schématisation du XEUSS, l'équipement utilisé pour les mesures SAXS/-WAXS. (a) : Photo du XEUSS [15] (b) : Schématisation par blocs (c) : Schématisation des éléments

faut voir cette connexion comme une rue à sens unique qui permet d'entrer dans le réseau depuis l'ordinateur équipement mais pas l'inverse.

Un second ordinateur est dédié à l'analyse des données. Sur cette machine n'importe quel utilisateur peut se connecter avec ses identifiants CEA. L'objectif sera donc d'installer mon programme sur cet ordinateur pour qu'il puisse être accessible pour tous les utilisateurs. Il faudra donc penser le programme pour qu'il se lance comme une "application" plutôt qu'un exécutable Python.

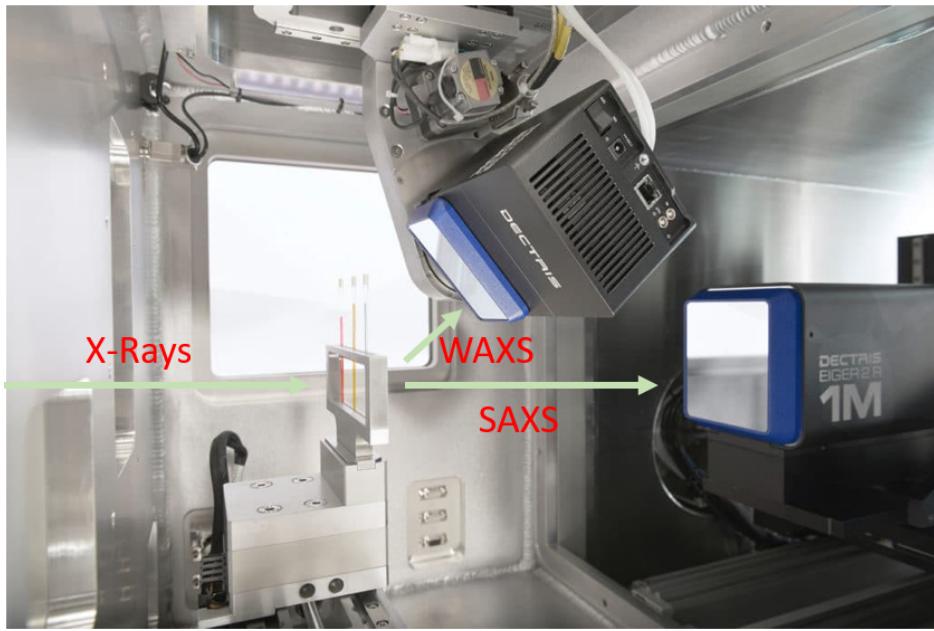


FIGURE 7 – Photographie de l'intérieur de la chambre échantillon avec le détecteur SAXS et WAXS.

2.1.3 Les différents formats de données

Le format de fichier edf [12]

Le format edf est un format de données initialement développé par l'European Synchrotron Radiation Facility (ESRF). Ce format de données est prévu pour stocker des données de SAXS provenant d'un synchrotron. Il est extrêmement simple mais peut quand même contenir plusieurs blocs de données. Il contient un en-tête général et il est possible d'avoir un en-tête par bloc de données qui contient des informations spécifiques à l'image. Dans chaque en-tête, il est possible d'entrer une paire "mot-clé = valeur" pour renseigner une propriété du bloc. Les en-têtes sont en ASCII afin de pouvoir les lire mais il n'est pas recommandé de modifier ce texte sous peine de corrompre le fichier en dépassant le nombre de caractères alloués au en-tête.

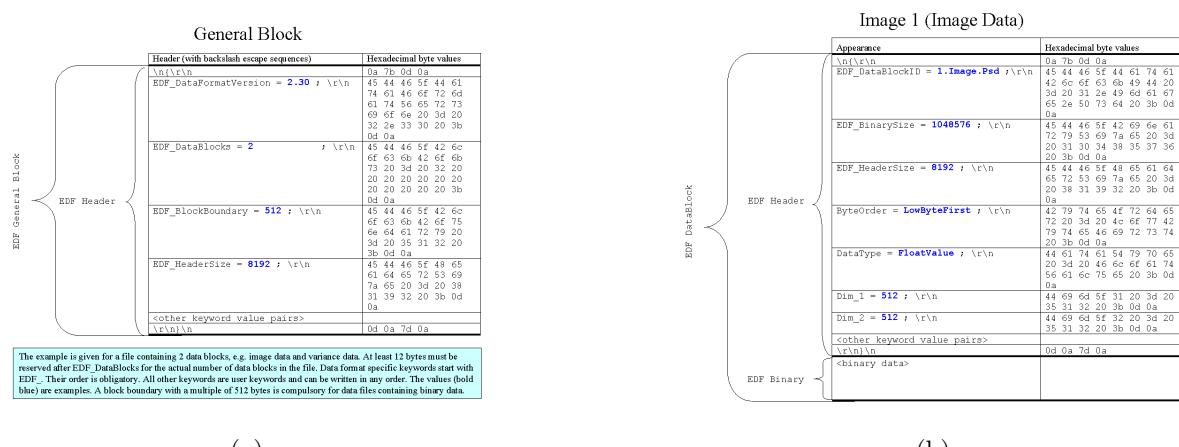


FIGURE 8 – Exemple de structure d'un fichier edf tel qu'exposé dans le papier de présentation du format EDF [4]. (a) : l'en-tête général d'un fichier edf avec plusieurs paires "mot-clé = valeur". (b) : l'en-tête d'un bloc de données les dimensions des données.

2 PLANIFICATION DE MES TÂCHES ET RECHERCHE BIBLIOGRAPHIQUE

Ce format soulève quelques problèmes :

1. Il est prévu pour le synchrotron ESRF, ce qui n'est pas forcément adapté aux manipulations en laboratoire.
2. Il ne contient que deux éléments principaux : les données enregistrées et l'en-tête. Cela rend la structuration des données assez compliquée d'autant plus que les données de l'en-tête sont sous la forme d'une paire "clé-valeur", sans contrainte sur le format de la clé ni sur la valeur.
3. Les applications développées pour visualiser les données sont ainsi très dépendantes de l'instrument pour lesquelles elles ont été créées et ne sont donc pas forcément exploitables pour d'autres instruments

Après ces quelques recherches, on comprend mieux pourquoi il est nécessaire de changer le format de stockage de ces données. Il faut un format de données qui soit plus universel.

Avant de modifier le format, il me faut tout d'abord trouver comment accéder aux données dans le fichier EDF. Fort heureusement, les chercheurs du CEA utilisaient déjà des notebook jupyter pour traiter leurs données. J'ai donc pu m'en servir pour savoir quelles bibliothèques étaient utilisées et quelles étaient les données importantes dans le fichier.

La bibliothèque utilisée est la bibliothèque Fabio. Cette bibliothèque ouvre un fichier edf sous forme d'objet Python qu'on peut facilement séparer en 2 morceaux :

1. l'en-tête : Il contient toutes les métadonnées liées à l'expérience. Il est fourni sous forme de dictionnaire. Cela est très utile car si on souhaite accéder à une métadonnée, par exemple la longueur d'onde, il suffit de trouver la clé associée et de l'utiliser pour l'extraire. En revanche, le nom de cette clé est choisi par celui qui a défini l'en-tête du fichier edf. Cette clé peut donc varier entre machines ou même entre deux versions du logiciel de la même machine ce qui peut poser des problèmes de maintenance.
2. Les données : Cet attribut contient un tableau numpy (numpy array) bidimensionnel de nombres, les dimensions de ce tableau représentent le nombre de pixel dans la direction verticale et horizontale du détecteur. Chaque valeur indique le nombre de photons détecté par un pixel du détecteur.

Le format hdf5 [\[6\]](#)

Le format hdf5 fait partie de la famille des hdf, Hierarchical Data Format. Il en existe deux versions : hdf4 et hdf5. Ici nous utiliserons la version 5 car plus récente. De plus le standard NeXus (détalé dans la partie suivante) est basé sur le format hdf5.

La structure de ce format est très simple et plus flexible que le format edf. On a donc 3 éléments principaux :

1. Les groupes : Ils peuvent contenir d'autres groupes ou des datasets, on peut les voir comme des "dossiers"
2. Les datasets : ce sont eux qui contiennent les données. Ces données peuvent être des images, tableaux, graphiques ou scalaires. On peut les voir comme des "fichiers"

3. Les attributs : Ce sont des informations supplémentaires qu'on peut donner à un groupe ou un dataset, ce qui permet de les décrire. On peut les voir comme des "propriétés"

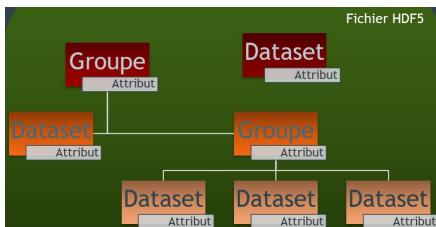


FIGURE 9 – Schéma représentant une structure possible de fichier hdf5

La présence des groupes, datasets et attributs pour la description rend, le format hdf5 idéal pour créer des fichiers de données bien structurés et compréhensibles.

Afin de pouvoir lire et écrire un fichier hdf5, nous utilisons la bibliothèque h5py. La structure arborescente d'un fichier hdf5 est illustrée sur la figure 9. Cette structure permet d'accéder à un groupe ou un dataset avec une syntaxe similaire au POSIX [9]. Pour un élément suivant le chemin "expérience/mesure/graphique", il suffit d'ouvrir le fichier avec h5py et on utilise la syntaxe objet ["expérience/mesure/graphique"] pour le récupérer. Cela rend l'accès aux données beaucoup plus clair et intuitif car ce format de chemin est déjà grandement utilisé par la plupart des systèmes d'exploitation.

Pour écrire, h5py nous offre de nombreuses méthodes qui permettent de créer des groupes, des éléments et des attributs dans un fichier hdf5.

Le standard NeXus [7]

Le format de données NeXus est développé par des scientifiques et des programmeurs qui ont pour but de créer un format de données standardisé et ainsi faciliter le partage et la comparaison des données au sein d'une communauté scientifique. Ce standard est en particulier destiné aux expériences faisant intervenir des neutrons, rayons X et muons. Il est basé sur le format hdf5 présenté plus haut. Pour atteindre cet objectif, le standard NeXus utilise les éléments du format hdf5 et définit des "*définitions d'application*".

Ces définitions d'application permettent de poser une convention et de se mettre d'accord sur la manière de mettre en forme le fichier hdf5 et les données qu'il contient. Pour se faire, sur le site officiel du standard NeXus se trouve toute une partie documentation pour chaque définition d'application. Il est même possible d'en créer en partant d'une "base" et en la spécifiant.

Il en existe beaucoup mais celle qui va nous intéresser principalement est la définition NXcanSAS [8], inspirée du format déjà existant canSAS, qui permet de stocker les données réduites d'une expérience SAXS.

Les noms des éléments dans le standard NeXus n'ont pas d'importance car la nature d'un élément est défini par son type NeXus. Pour le nommage des éléments j'ai donc pris la liberté de choisir une nomenclature bien spécifique afin de ne pas me perdre. C'est donc en utilisant ce standard en conjonction avec le format hdf5 qu'on pourra résoudre nos problèmes de format principaux.

La définition NXcanSAS est une définition qui a été développée par la communauté internationale de la diffusion aux petits angles (rayons X et neutrons). Il est conçu avec les spécificités des expériences de diffusion aux petits angles en tête et est donc tout à fait adapté aux données

2 PLANIFICATION DE MES TÂCHES ET RECHERCHE BIBLIOGRAPHIQUE

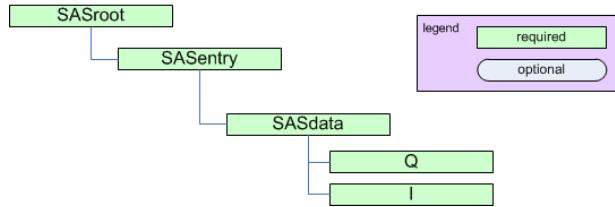


FIGURE 10 – Minimum requis pour former un fichier NXcanSAS

SAXS/WAXS, GISAXS/GIWAXS qui nous intéressent sur le XEUSS.

Le format NXcanSAS a une structure Nexus plutôt standard. A partir du point d'entrée nous avons :

1. Les groupes DATA. Il peut y en avoir autant que nécessaire, il faut juste que le nom des groupes soient uniques. Ils peuvent contenir l'information sur l'intensité, les vecteurs d'onde et les incertitudes sur ces différentes grandeurs.
2. Les groupes PROCESS. Ils permettent de décrire le genre de traitement que les groupes DATA ont subi. Il y en a un par groupe DATA et porte un nom similaire afin de pouvoir les associer facilement.
3. Le groupe INSTRUMENT. Il contient les informations concernant l'instrument subdivisé en plusieurs sous-groupes : SOURCE, APERTURE, COLIMATOR, DETECTOR
4. Le groupe SAMPLE. Il contient toutes les informations concernant l'échantillon qui a été mesuré. Parmi ces informations se trouvent, la position de l'échantillon dans la chambre, son épaisseur, température, sa transmission et bien d'autres.
5. Le groupe COLLECTION. Il contient des informations non critiques pour le traitement de données. Ce groupe peut contenir toutes les valeurs souhaitées et peut être structuré de la manière voulue. Il illustre toute la flexibilité du standard NeXus.

Cette définition d'application est donc tout à fait adaptée pour le stockage de données SAXS. De plus, la possibilité de mettre plusieurs groupes de données permet d'enregistrer les données dans leur état "brut", c'est-à-dire telles qu'elles sortent de la machine, mais aussi les données traitées accompagnées d'une description du traitement qu'elles ont subi. Cela permet de tracer les opérations d'analyse de données et de les rendre reproductibles par d'autres scientifiques de la communauté.

2.2 Organisation

Avec ces idées pour accomplir l'objectif de ma mission au CEA, j'ai mis au point un petit schéma (figure 11) me permettant de situer ce que je devais faire précisément.

Idéalement, toutes ces conversions et manipulations devront se faire sans que l'expérimentateur ait à toucher aux scripts Python. Par conséquent, il faudra développer une ou plusieurs interfaces graphiques. C'est de là que vient le launcher.py de la figure, il permettra de lancer les différentes interfaces graphiques de la figure 11.

Je me suis aussi fixé des dates butoirs pour ne pas prendre trop de retard :

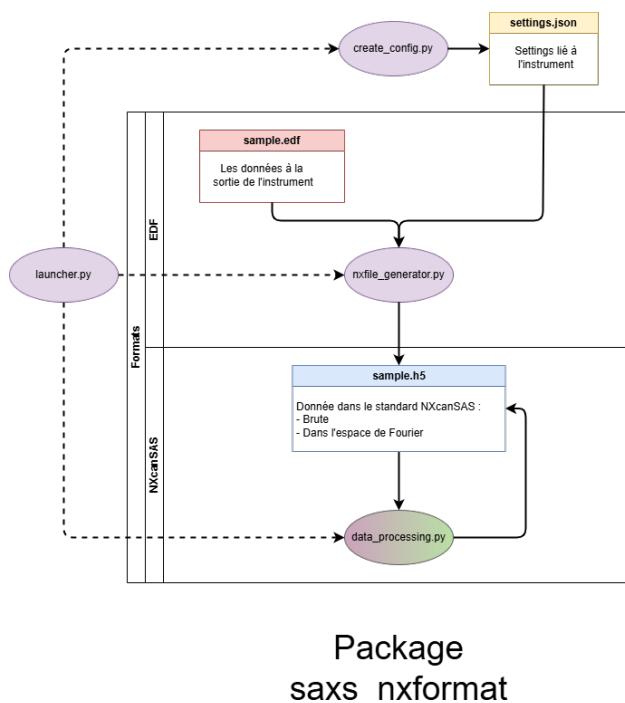


FIGURE 11 – Structure du package `sachs_nxformat`

1. Première période d’alternance : Faire mes recherches et avoir un code fonctionnel qui permet la conversion d’un fichier edf à un fichier hdf5 sous le standard NXcanSAS
2. Deuxième période d’alternance : Avoir un code fonctionnel qui permet de faire le traitement de données le plus basique
3. Dernière période d’alternance : Continuer à développer les processus et rendre un code qui correspond aux attentes

3 MISE EN PLACE DE LA CONVERSION DE DONNÉES DU FORMAT EDF À HDF5

3 Mise en place de la conversion de données du format edf à hdf5

3.1 Problématiques

La première grande étape a été de faire le passage des données du format edf à hdf5 en appliquant le standard NeXus. De base, nous avons un fichier edf avec un en-tête qui contient les métadonnées de l'expérience dans un format non standardisé et nous voulons ranger toutes ces métadonnées et données dans un fichier hdf5 qui suit la définition NXcanSAS.

Cette étape présente plusieurs problèmes :

1. La structure NXcanSAS est très complexe. Il faut trouver un moyen d'écrire facilement toutes les informations dans le fichier hdf5 tout en suivant cette structure.
2. La clé pour récupérer une certaine valeur dans le fichier edf est susceptible de changer au cours du temps et entre les machines.
3. Faire le lien entre l'en-tête du edf et les éléments de la structure NXcanSAS

Pour répondre à ces différentes problématiques, j'ai choisi de créer un fichier de configuration. Ce fichier aura la structure de la définition d'application NXcanSAS et contiendra les noms de clés nécessaires pour pouvoir récupérer les informations dans le fichier edf.

3.2 Crédation du fichier de structure de NXcanSAS

Le fichier de configuration doit avoir la même structure que NXcanSAS, il est donc nécessaire d'encoder la structure NXcanSAS avant de l'utiliser pour créer le fichier de configuration. Ce fichier sera appelé "fichier de structure" et sera la version vierge du fichier de configuration.

Pour se faire, il fallait que la structure NXcanSAS se trouve quelque part dans mon code. En effet la documentation [8] de NXcanSAS donne toute sa structure mais ne fournit pas de moyen de l'implémenter directement, et en entier, dans un script Python. Il a donc fallu trouver un moyen de pouvoir communiquer la structure NXcanSAS au programme. Pour se faire, et après plusieurs essais, j'ai fini par trouver une bonne manière de faire.

Afin de parvenir à recréer la structure d'un fichier NXcanSAS, j'ai choisi d'utiliser des dictionnaires, et ce pour trois raisons principales. La première est qu'un dictionnaire a l'avantage de pouvoir être "imbriqué" ce qui permet de refléter la structure arborescente d'un fichier hdf5. La deuxième est qu'un dictionnaire est un objet Python très pratique à explorer récursivement. Ainsi, comme un dictionnaire est facilement explorable, on pourra également le remplir aisément pour créer la structure NXcanSAS du fichier d'arrivée. Et finalement, Fabio utilise également des dictionnaires pour stocker l'en-tête des fichiers edf donc on a une continuité des types de variables utilisées.

La figure 12 est un extrait du fichier de structure NXcanSAS qui est présent dans mon code. Ce fichier n'a pas pour vocation d'être modifié et sert de référence.

Dedans, on peut voir le paramètre "SDD", tout en haut, qui est une clé. La valeur associé à SDD est un dictionnaire qui comprend 7 paires "clé-valeur" :

1. element type / dataset : SDD est donc un dataset dans le fichier hdf5
2. value / null : SDD n'a pas de valeur associée. C'est normal on est sur le fichier de structure

FIGURE 12 – Extrait du fichier de structure NXcanSAS.

3. possible value / [] : pas de valeur proposée
 4. EX_required / true : Ce champs est requis
 5. type / "NX_number" : le type de variable selon NeXus, ici un nombre.
 6. docstring / sample-detector distance : le SDD représente la distance détecteur échantillon
 7. content / ... : SDD possède du contenu, un attribut unité. On a donc un dictionnaire qui contient des paires "nom-attribut/dictionnaire". Le dictionnaire associé à l'attribut ayant exactement la même structure. C'est un exemple de l'imbrication des dictionnaire.

Avec cette approche, on peut adapter n'importe quelle définition d'application NeXus en dictionnaire Python. Par contre, plutôt que d'avoir ce dictionnaire très encombrant codé en tant que tel dans un script Python, il m'a paru plus pratique de l'enregistrer dans un fichier. Initialement, je l'avais enregistré dans un fichier texte, mais je me suis très vite rendu compte des limitations et je me suis donc renseigné un peu plus sur une bonne manière d'enregistrer des dictionnaires Python. J'ai fini par trouver le format .json qui est infiniment plus pratique pour charger et sauvegarder des dictionnaires. De plus le format .json est bien organisé et est facile à éditer (un simple bloc note suffit) si le besoin se présente.

3.3 Créer le fichier de configuration

3.3.1 Structuration du fichier

J'ai essayé plusieurs approches pour faire le lien entre les paramètres/données des fichiers edf et NXcanSAS. Voici une liste non-exhaustive des approches les plus réussies :

1. Un simple dictionnaire qui a pour clé et valeur le nom de la variable en NeXus et la clé associée dans l'en-tête edf. Le problème principal étant que les noms de certains paramètres ne sont pas uniques (par exemple les rotation ; "roll", "pitch" et "yaw" sont utilisées pour décrire les rotations du détecteur et de l'échantillon).
2. Une approche similaire où le nom de la variable est devenu le chemin de la variable. Cela résout le problème de l'unicité des noms de paramètres mais cela ne permet pas de renseigner les attributs comme l'unité de certaines grandeurs.

La spécificité de ce fichier de configuration a ainsi été de mettre dans le champ "value" de tous les éléments, la clé du en-tête edf qui lui correspond. Ainsi on peut obtenir directement la valeur. Prenons l'exemple de la longueur d'onde. La clé dans le fichier edf est "Wavelength". Ainsi la valeur de l'élément "incident_wavelength" dans mon fichier de configuration est "Wavelength". De cette manière, quand je génère le fichier NXcanSAS avec le fichier de configuration, je peux directement récupérer la longueur d'onde du fichier edf en exécutant la commande : `lambda = dict_edf[dict_config["value"]]` en dépliant un peu on a `lambda = dict_edf["Wavelength"]` puis `lambda = 1 nm` par exemple. Pour finir je n'aurais plus qu'à créer l'élément correspondant avec h5py.

En pratique, le fichier de structure est un .json qui sert "d'original vide". Ce fichier n'est pas modifié. Par contre il est copié et sert de base à la création d'un nouveau fichier configuration pour chaque nouvelle machine (i.e. chaque nouvelle structure de en-tête de fichier edf).

Le fichier de configuration présente quelques différences par rapport au fichier de structure :

1. La valeur est en fait le nom de la clé correspondante dans le fichier edf.
2. S'il y a un élément "possible value" alors l'utilisateur pourra choisir la valeur qu'il souhaite.
3. Pour les attributs du type unité, l'utilisateur n'a qu'à renseigner l'unité de la grandeur du fichier edf et si cette unité est supportée par le programme, elle sera convertie dans une unité du système SI plus appropriée.

Ainsi, le fichier configuration ne doit être créé qu'une seule fois pour un équipement. A moins de changement conséquent sur l'équipement ou sur les fichiers .edf sauvés, ce fichier reste inchangé. Il y a eu une fois où j'ai dû faire une mise à jour du fichier de configuration, lorsqu'on a ajouté les informations concernant le traitement en intensité absolue.

Comme mentionné précédemment, le fichier de configuration est dépendant de l'équipement. Il faut donc créer un fichier configuration par équipement. Par conséquent, j'ai mis en place un outil facilement manipulable pour les chercheurs qui ne savent pas forcément comment éditer un fichier json et ou placer les informations pertinentes.

3.3.2 Mise en place d'un GUI pour faciliter la construction d'un fichier configuration

L'objectif de ces travaux étant de proposer cet outil de conversion de données à la communauté, nous avons essayé de faciliter autant que possible le processus de création de ce fichier configuration. Pour ce faire, j'ai créé une interface graphique (Graphical User Interface (GUI) en anglais) pour éviter à l'utilisateur de faire un copier-coller et de remplir le fichier de configuration manuellement. Le GUI est assez simple et la création du fichier se fait en 3 étapes :

1. L'utilisateur charge un fichier .edf qui va servir de fichier de référence.
 2. toutes les paires "clé-valeur" de l'en-tête du fichier référence sont affichées. L'utilisateur choisit les clés qui contiennent des informations pertinentes selon lui.
 3. Toute la structure du fichier hdf5 est affichée avec une courte description pour chaque champs. L'utilisateur n'a qu'à renseigner les clés qu'il a gardé à l'étape précédente dans le champs approprié du fichier hdf5. Il peut aussi entrer une valeur manuellement ce qui mettra cette valeur par défaut dans tous les fichiers. Si l'utilisateur laisse un champs vide une valeur par défaut appropriée sera attribuée à ce champ.

L'utilisateur peut ensuite sauver le nouveau fichier de configuration sous le nom de son choix. Néanmoins, tout n'est pas modifiable de façon à toujours rester dans le cadre de la structure des fichiers hdf5. Par exemple, les données seront automatiquement intégrées au fichier hdf5 après que celui ci ait été généré.

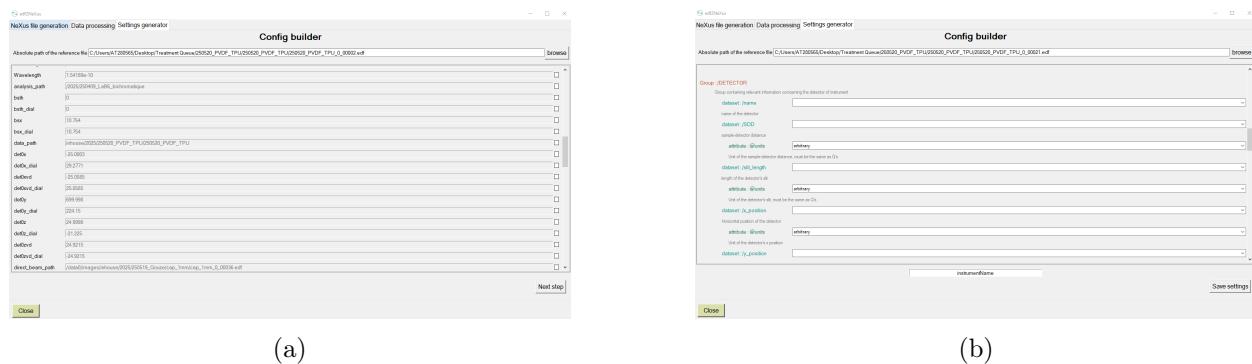


FIGURE 13 – Capture d'écran du GUI permettant la création d'un fichier de configuration. (a) : Première étape avec l'en-tête du fichier edf affiché. (b) : Deuxième étape avec la structure NXcanSAS affichée.

La fonction qui permet d'afficher le cadre défilant illustré sur la figure 13b est générale et fonctionne pour n'importe quelle définition NeXus, du moment que la structure utilisée soit similaire à celle présente dans le code. J'ai pu atteindre ce degrés de généralité en utilisant une fonction récursive qui parcours tous les éléments du dictionnaire représentatif de la structure NXcanSAS.

Une fois ce cadre rempli, une fonction va parcourir récursivement la structure et afficher tous les éléments qui constituent la définition NXcanSAS. Il suffit simplement de charger le fichier json dans un dictionnaire et de le passer en entrée de la fonction. Arriver à cette fonction a été assez fastidieux car il a fallu prendre en compte tous les cas limites comme l'attribut unité qui contient une unité de départ et une unité d'arrivée.

En appuyant sur le bouton sauvegarder, on récupère les valeurs renseignées dans les champs et on

3 MISE EN PLACE DE LA CONVERSION DE DONNÉES DU FORMAT EDF À HDF5

les injecte dans le fichier de configuration (qui est une copie conforme du fichier de la structure NXcanSAS) en utilisant une autre fonction recursive.

3.3.3 Un exemple de création fichier de configuration

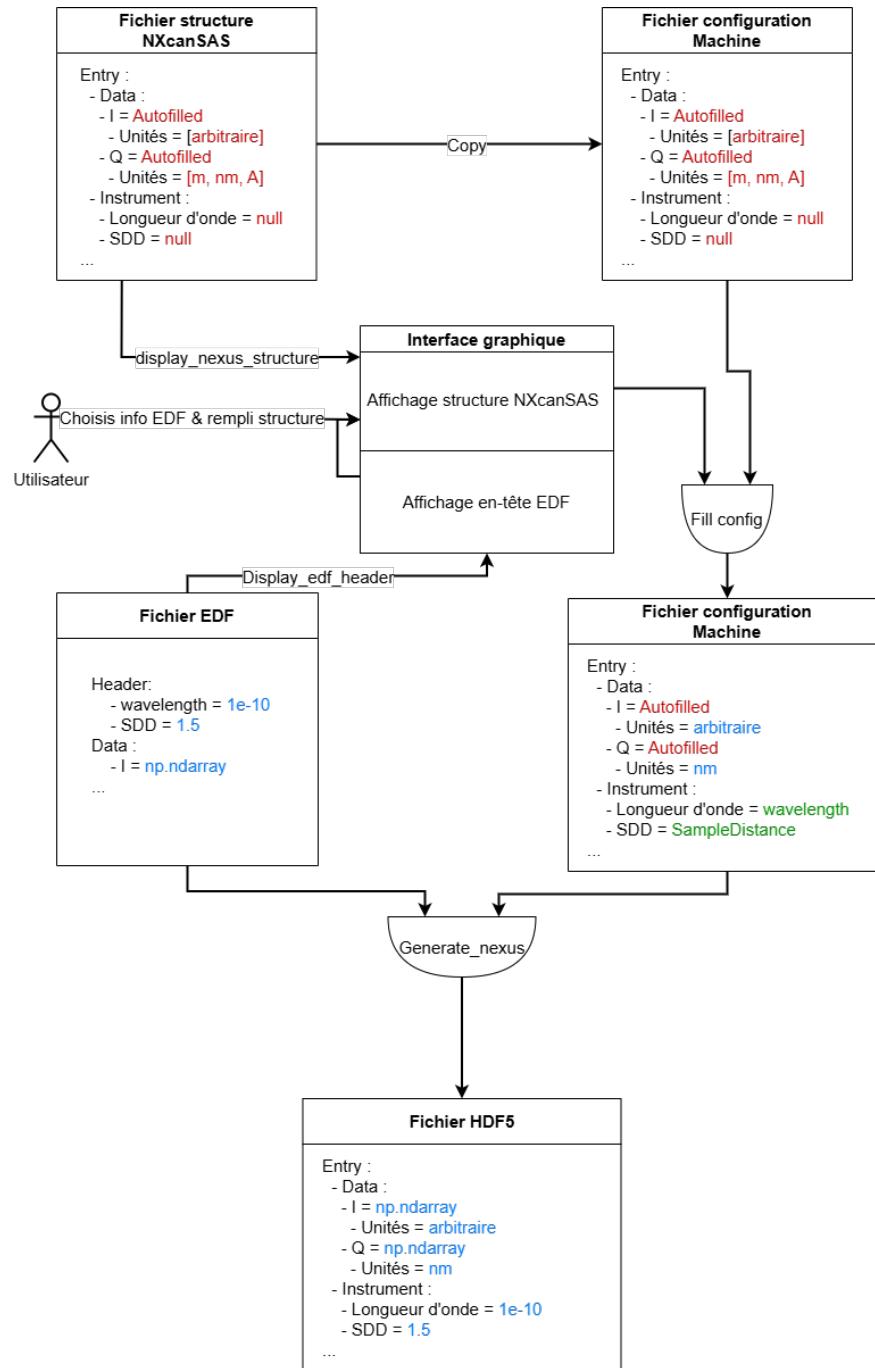


FIGURE 14 – Schéma récapitulant la conversion d'un fichier en hdf5

J'ai réalisé la figure 14 pour illustrer le fonctionnement de la conversion des fichiers du format edf au format hdf5. Les couleurs des éléments dans cette figure respectent la convention suivante :

1. En bleu : les valeurs devant être présentes dans le fichier hdf5 final
2. En vert : les clés des éléments dans l'en-tête du fichier edf

3. En rouge : les valeurs présentes par défaut dans le fichier de structure

Ce schéma met bien en évidence le fait que le fichier de configuration n'enregistre pas les valeurs du fichier edf mais bien les clés associées à ces valeurs. Cela permet de pouvoir récupérer la valeur depuis le fichier edf et de pouvoir la replacer au bon endroit dans le fichier hdf5.

On voit aussi que les données, l'intensité et les positions (coordonnées de \vec{Q} , \vec{q}_r , χ , 2θ ...) ne sont pas remplies car :

1. L'intensité est forcément positionnée dans l'attribut data du fichier edf quand il est ouvert via Fabio
2. Les positions sont calculées en fonction des dimensions de l'image, de la position du centre du faisceau et de la taille des pixels.

Il est important de noter que la partie haute du schéma correspond à la partie création du fichier de configuration qui demande une intervention de l'utilisateur (via l'interface graphique). Néanmoins, cette étape n'est pas nécessaire systématiquement. En effet, lorsque le fichier de configuration est créé, il permet ensuite d'automatiquement appliquer la fonction Generate nexus illustrée sur le schéma.

Ainsi, si le fichier configuration existe, un fichier hdf5 sera créé pour chaque nouveau fichier edf avec les valeurs désirées dans les emplacements prévus par NeXus. Nous allons détailler ce processus de conversion.

3.4 Conversion automatique des fichiers en hdf5

3.4.1 Gestion des fichiers

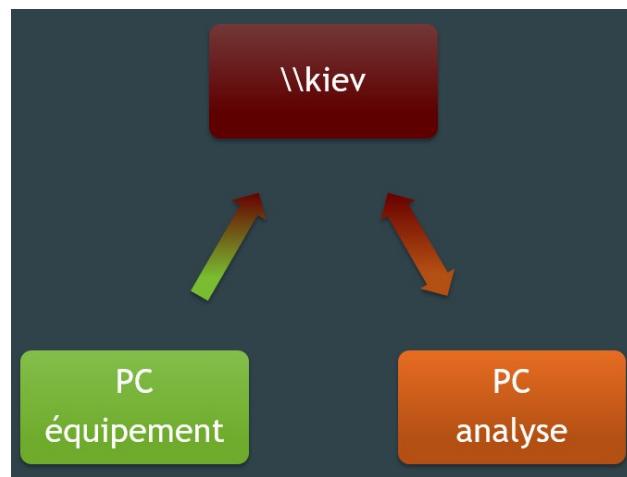


FIGURE 15 – Schéma récapitulant le déplacement des fichiers de données

La première problématique qui s'est posée lors de la conversion des fichiers a été la gestion des fichiers edf et hdf5. Il est important de savoir où sont générés les fichiers lorsqu'une mesure est prise. De plus, La structure et la sécurité des données au sein du CEA est d'une importance critique et par conséquent les procédures d'écriture et de copie des fichiers sont très réglementées. Ainsi il faut

3 MISE EN PLACE DE LA CONVERSION DE DONNÉES DU FORMAT EDF À HDF5

suivre des règles très précises.

Le XEUSS est piloté par un PC qui lui est dédié et qui n'est pas connecté sur le réseau CEA. Les fichiers edf sont ainsi écrits sur le PC équipement (en vert sur la figure 15). Il existe des procédures pour copier les fichiers sur le réseau mais pas de façon automatique et cela demande une présence physique.

C'est pour cela que mon tuteur à demander la mise en place d'un serveur sur lequel il serait possible d'envoyer les données automatiquement, c'est le serveur kiev (en rouge sur la figure 15). La réciproque n'est pas possible, il n'est pas possible de communiquer depuis le serveur kiev vers le PC équipement. L'avantage de cette approche est que le serveur kiev est lui accessible depuis le réseau CEA. Il est possible d'accéder aux données stockées sur le serveur kiev depuis un PC d'analyse (en orange sur la figure 15) situé sur le site.

A partir de cette architecture, on peut transférer les fichiers de données depuis le PC équipement vers le serveur kiev de manière automatique et régulière.

Pour ce faire j'ai utilisé le logiciel Cron [3] sur le PC équipement (opérant sous le système d'exploitation UNIX). Cron permet d'exécuter une commande périodiquement, la périodicité étant définie par l'utilisateur. Voici la commande cron utilisée :

```
0 */1 * * * cp -R -u "/chemin/donnée/PC équipement/" "/chemin/donnée/kiev/Treatment Queue/"
```

Cette commande peut se décomposer de la façon suivante :

1. 0 */1 * * * : C'est une expression spécifique à cron qui lui permet de savoir quand exécuter la commande qui suit. 0 pour la minute 0, */1 pour une fois par heure, * pour tous les jours, * pour tous les mois, * pour tous les jours de la semaine
2. cp -R -u "/chemin/donnée/PC équipement/" "/chemin/donnée/kiev/Treatment Queue/" : c'est la commande qui est exécutée. En l'occurrence, on copie tous les fichiers du dossier /chemin/donnée/PC équipement/ dans le dossier /chemin/donnée/kiev/Treatment Queue/ de manière récursive (option -R) et en mettant à jour (option -u). Par "mettre à jour" on veut dire que les fichiers qui sont déjà présents dans le dossier de destination ne seront pas copiés à nouveau. De plus si un fichier subit une modification dans le dossier source, il sera aussi modifié dans le dossier destination.

Maintenant, nous avons les fichiers edf qui se trouvent sur le serveur kiev qui est lui-même connecté au réseau interne du CEA. Cela nous permet de convertir les fichiers qui sont présents dessus en hdf5. De plus, ça nous permet aussi d'avoir rapidement accès aux données, depuis n'importe où sur le site.

3.4.2 Conversion des données

Avant de pouvoir commencer à convertir les fichiers, il faut trouver un moyen de vérifier s'ils ont déjà été traités. On ne peut pas faire quelque chose d'aussi simple que de modifier les fichiers afin de leurs mettre un marqueur indiquant qu'ils ont déjà été traités. En effet, si on venait à faire cela, le fichier serait modifié et le processus cron ferait une nouvelle copie du fichier sur le serveur kiev, entraînant une boucle infinie de conversion des données.

Pour contourner ce problème, la solution utilisée a été de détecter la présence du fichier dans l’arborescence finale. C'est-à-dire qu'au moment de la création du chemin d’enregistrement du fichier, je regarde si le chemin d’enregistrement existe déjà ou non. S'il existe déjà, on ne fait pas la conversion et on passe au fichier suivant. Cette procédure est illustrée sur la figure 16.

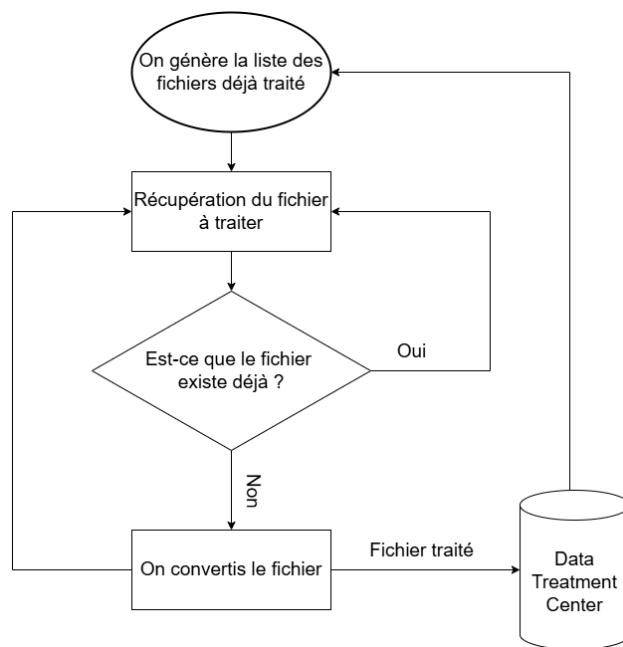


FIGURE 16 – Schéma expliquant la recherche et la gestion des fichiers qui permet de savoir si un fichier a déjà été traité ou non

Pour y arriver, j'ai créé une fonction qui se base sur l’arborescence existante du fichier à convertir, son nom et le nom du fichier de configuration utilisé. Ainsi, si on génère le chemin d’enregistrement d'un fichier à deux moments différents, on aura tout de même exactement le même chemin, ce qui permet de reconnaître si le fichier a déjà été converti ou non. Les chemins vont avoir le format suivant :

Treated Data / Instrument - {nom de l'instrument} / chemin original.

J'ai utilisé ce format en gardant en tête la problématique de portabilité sur plusieurs machines. De plus, en gardant le chemin original du fichier, on évite que deux fichiers de même noms finissent au même endroit, ce qui provoquerait un écrasement d'un des deux fichiers et les deux arborescences étant identiques, il est plus simple de retrouver les fichiers.

Après avoir identifié les fichiers à convertir, je vais détailler le processus de conversion du format edf au format hdf5 avec le standard NXcanSAS. Il s'agit de la dernière étape de la figure 14 qui est intégralement gérée par la fonction generate nexus. Comme indiqué sur la figure 14, la fonction a besoin du fichier de configuration, du fichier à traiter et du dossier d'enregistrement du fichier hdf5 pour fonctionner.

Je commence par définir le nom du fichier hdf5 en utilisant le nom de l'échantillon, le type de détecteur et le numéro de l'image (toutes ces informations étant stockées dans l'en-tête ou le nom

3 MISE EN PLACE DE LA CONVERSION DE DONNÉES DU FORMAT EDF À HDF5

du fichier edf). Ensuite, je crée un fichier hdf5 qui porte le nom que l'on vient de définir et qui sera stocké dans le dossier d'enregistrement passé en argument de la fonction. Pour remplir ce fichier avec toutes les données et métadonnées du fichier original, j'ai créé une fonction récursive nommée `fill_hdf5`. Cette fonction va prendre en entrée la structure du fichier de configuration sous forme de dictionnaire Python, le fichier hdf5 et l'élément que l'on est en train de remplir (élément parent). Le premier élément parent va donc être le fichier hdf5 lui-même (base case). La fonction va ensuite explorer chaque élément de la structure NXcanSAS et remplir petit à petit chaque élément de la structure à l'aide des clés edf trouvées dans le fichier de configuration.

La fonction retourne finalement le chemin complet du fichier hdf5 qui vient d'être créé, de cette manière il est toujours possible d'ouvrir le fichier après l'avoir généré si nécessaire.

3.4.3 Le GUI pour lancer la conversion automatique

Maintenant que nous avons tous les morceaux nécessaires à la conversion de fichiers edf en fichiers hdf5, il nous suffit de les assembler pour faire une "pipeline" qui va prendre les fichiers edf présents sur le serveur kiev et les convertir en fichiers hdf5 tout en appliquant les processus de traitement élémentaires. Pour accomplir cet objectif j'ai créé un autre GUI qui va permettre de contrôler le processus de conversion des fichiers, un panneau de contrôle en quelque sorte.

Pour entrer un peu plus dans le détail, la conversion automatique des fichiers se fait via une boucle while dans un thread daemon. Un thread daemon à l'avantage de s'arrêter dès que tous les autres thread sont terminés, donc si le panneau de contrôle est fermé pendant que le thread daemon tourne, le thread daemon sera terminé automatiquement.



FIGURE 17 – Interface graphique permettant le lancement de la conversion des données. Très simpliste avec seulement des boutons pour démarrer et arrêter le processus et une console affichant plusieurs informations

La boucle while est contrôlée par une variable booléenne. La valeur de cette variable est contrôlée par les boutons start (change en True) et stop (change en False) du panneau de contrôle. Le fonctionnement est très simple mais il y a beaucoup de petits éléments qui s'ajoutent par dessus.

La console permet d'afficher d'autres messages pour indiquer le statut de la conversion du fichier ou pour afficher la quantité de mémoire maximum consommée par le programme. Pour stopper le programme il suffit d'appuyer sur le bouton Stop dès que le programme a fini de traiter le fichier en court ou qu'il a fini de dormir, alors le programme s'arrête.

3.4.4 Modification des fichiers NXcanSAS

Lors de la conversion ou du traitement des données, il est possible que quelque chose se passe mal. Un moteur mal calibré qui renvoie des mauvaises valeurs de rotation ou de distance, un type d'expérience mal renseigné ou encore des informations manquantes peuvent causer des erreurs sur le traitement du fichier. Pour pouvoir vérifier si tout s'est bien passé et modifier certaines informations dans le fichier hdf5, il est toujours possible d'utiliser des applications de visualisation de fichier hdf5. Le format étant en effet beaucoup plus répandu, il existe des applications qui permettent de fouiller un fichier hdf5 et d'en modifier les données. L'exemple principal est la librairie HDFView créée par la même équipe que les formats hdf. Cette application est très simple d'utilisation et permet de modifier certaines données contenues dans un fichier hdf5.

Un exemple de l'interface de l'application est donné en figure 18.

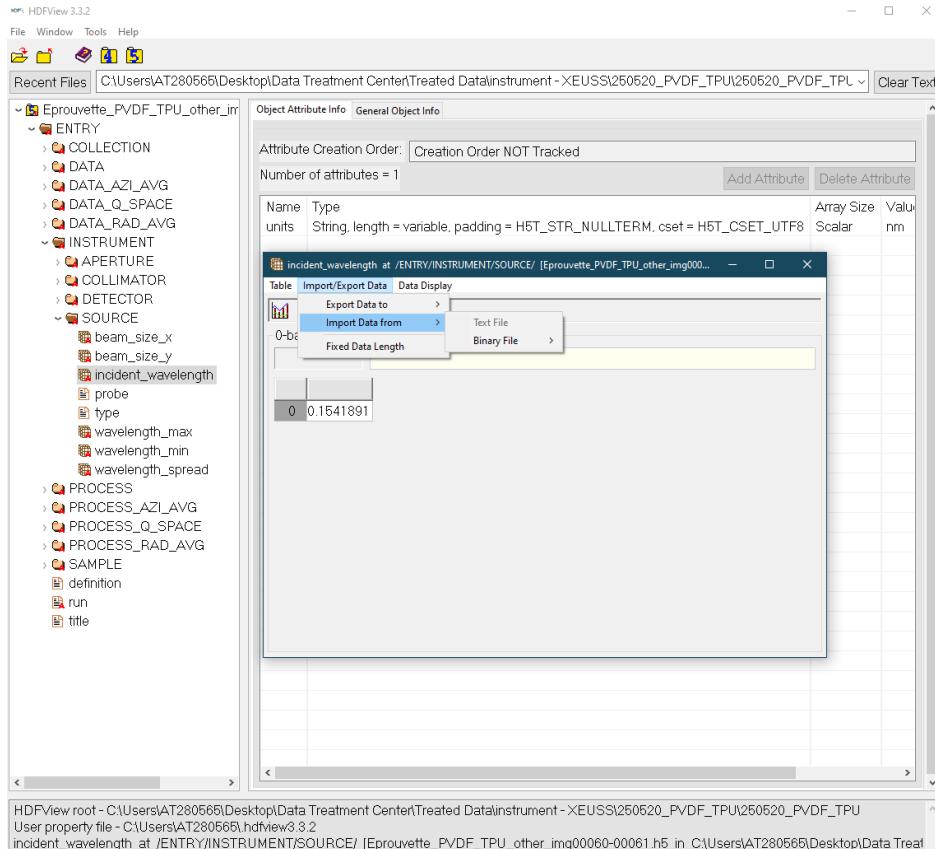


FIGURE 18 – Interface de l'application HDFView illustrant la comment accéder et modifier les données à l'intérieur du fichier hdf5.

3.4.5 Obstacles rencontrés

Le problème principal que j'ai pu rencontrer est la mémoire qui atteignait très vite des valeurs consommées trop importante. C'était un problème car les ordinateurs du CEA ne sont pas tous capables de soutenir un programme qui consomme constamment plus de 500 Mb de mémoire. De plus, la mémoire consommée semblait croître sans cesse, ce qui aurait éventuellement causer des problèmes au moment de l'automatisation.

La solution a été la commande `gc.collect()` qui permet d'invoquer manuellement le système de collecte de déchet de Python, ce qui a pour effet de supprimer tous les éléments inutilisés. On l'invoque manuellement car lorsque l'on est dans une boucle le système de collecte de déchets peut ne pas se déclencher tout seul. Cela permet de réinitialiser la mémoire consommée par le programme.

J'ai également rencontré des problèmes autour de la taille des fichiers. Il faut savoir que quand un dataset ou groupe d'un fichier hdf5 est supprimé, la taille que l'élément occupait n'est pas réutilisée. Ainsi lors de la conversion, certains éléments sont supprimés ce qui rend le fichier plus volumineux que ce qu'il devrait être.

Il existe une ligne de commande UNIX pour recompresser le fichier pour supprimer l'espace alloué mais non utilisé. Malheureusement elle ne pourrait pas être utilisé dans programme Python de manière pratique.

J'ai donc décidé de coder une version Python de cette commande de recompression, une fonction de "repack". Il s'avère que cette fonction est très simple à faire, il suffit de copier/coller toutes les données du fichier source dans un fichier de destination "temporaire" marqué par un .tmp en extension. Une fois la copie effectuée on supprime l'ancien fichier et on supprime le .tmp du nouveau fichier.

L'astuce repose sur le fait que l'on copie uniquement que les données, ce qui supprime l'espace alloué mais non utilisé et ramène le fichier à sa taille réelle. J'ai donc finalement trouvé une manière de faire la recompression des fichiers de manière automatique à la fermeture d'un fichier en implémentant cette fonction de "repack" au processus de fermeture du fichier.

Comme présenté dans cette partie, nous avons maintenant un système fiable et efficace qui permet de déplacer et de convertir les données. La prochaine partie présente la suite de mon travail qui, cette fois, porte sur l'automatisation du traitement de données.

4 Mise en place du traitement des données automatiques

4.1 Les différentes étapes du traitement de données

Une fois les données collectées et converties, une suite d'opération est appliquée sur les données expérimentales. En effet, l'information contenue dans les données expérimentales brutes n'est pas forcément exploitable. Dans cette première partie, je vais vous exposer les différentes opérations utilisées pour le traitement et la réduction des données.

4.1.1 Passage dans l'espace de Fourier

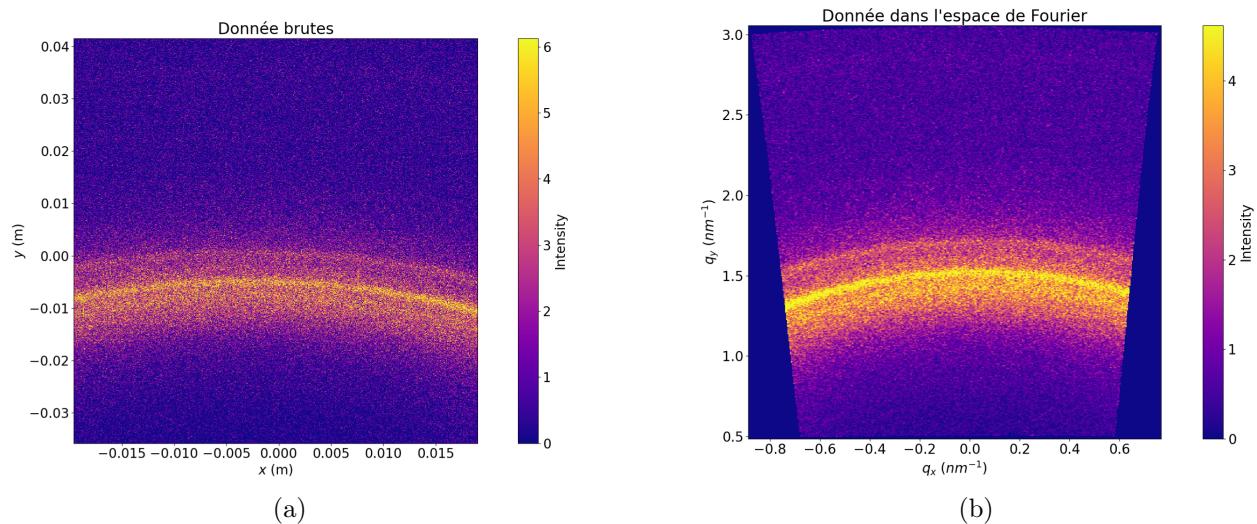


FIGURE 19 – (a) Exemple de données WAXS collectées sur le XEUS. L'image rectangulaire, représentative de la forme du détecteur et les coordonnées sont en unité de distance. (b) Image après conversion dans l'espace réciproque. On voit une distorsion de l'image initiale due à la prise en compte des rotations du détecteur. On a des coordonnées (q_x, q_y) exprimées en unité égale à l'inverse d'une distance

La première étape de l'analyse de données consiste à convertir les données brutes dans l'espace de Fourier (aussi appelé espace réciproque). Pour ce faire, l'angle de diffusion 2θ des rayons X est converti en un vecteur de diffusion \vec{Q} par l'équation suivante :

$$\|\vec{Q}\| = \frac{4\pi}{\lambda} \sin(2\theta) \quad (2)$$

λ étant la longueur d'onde des rayons X incidents.

Afin de précisément faire cette conversion, plusieurs informations sont nécessaires. Tout d'abord, les dimensions et la taille du pixel unitaire des détecteurs SAXS et WAXS (ici 75 μm pour les deux détecteurs).

Ensuite, il nous faut également connaître la position exacte de chaque détecteur par rapport à l'échantillon, comme illustré sur la figure 20. Même si l'équipement donne une valeur assez précise de chacune des trois distances et trois rotations illustrées sur la figure 20, une mesure sur un calibrant

4 MISE EN PLACE DU TRAITEMENT DES DONNÉES AUTOMATIQUES

est réalisée pour plus de précision.

Ainsi une poudre de Béhénate D'Argent (AgBeh) est mesurée. Il s'agit d'une poudre dont la distance inter-atomique est parfaitement calibrée donnant des anneaux à des positions connues dans le réseau réciproque, comme illustré sur la figure 21a. La librairie Python pyFAI permet ensuite, à partir de l'image de AgBeh, d'extraire la distance échantillon ainsi que les trois rotations rot_1 , rot_2 et rot_3 du détecteur. Elle calcule notamment l'intersection d'une sphère par rapport à un plan pour extraire ces rotations. Comme il n'existe pas une solution unique, on doit appliquer des restrictions sur les rotations, notamment pour le détecteur SAXS pour lequel les trois rotations sont fixées à 0 deg (contrainte du dispositif expérimental).

Les différents paramètres obtenus lors de la calibration sont stockés dans l'en-tête des futurs fichiers pour pouvoir automatiquement faire la conversion des futures données dans le réseau réciproque. Ainsi, on peut automatiquement déterminer le couple (q_x, q_y) associé à chaque pixel du détecteur. Les données ainsi converties ne sont donc plus impactées par les conditions géométriques de mesures et peuvent donc être comparées entre elles.

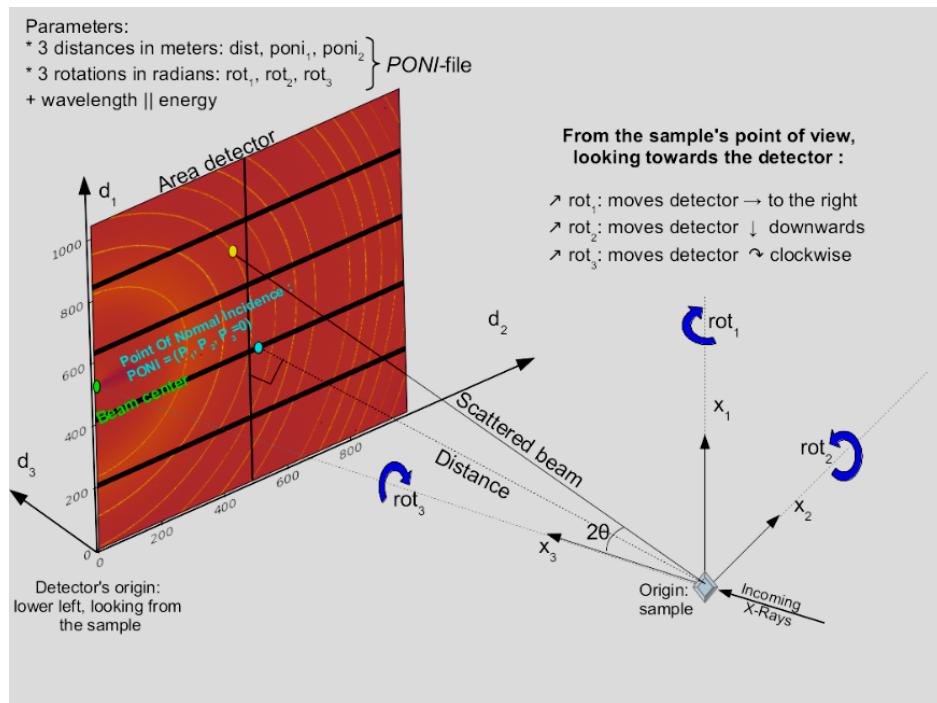


FIGURE 20 – Schéma de la géométrie utilisée par PyFAI [10]

4.1.2 Le caking

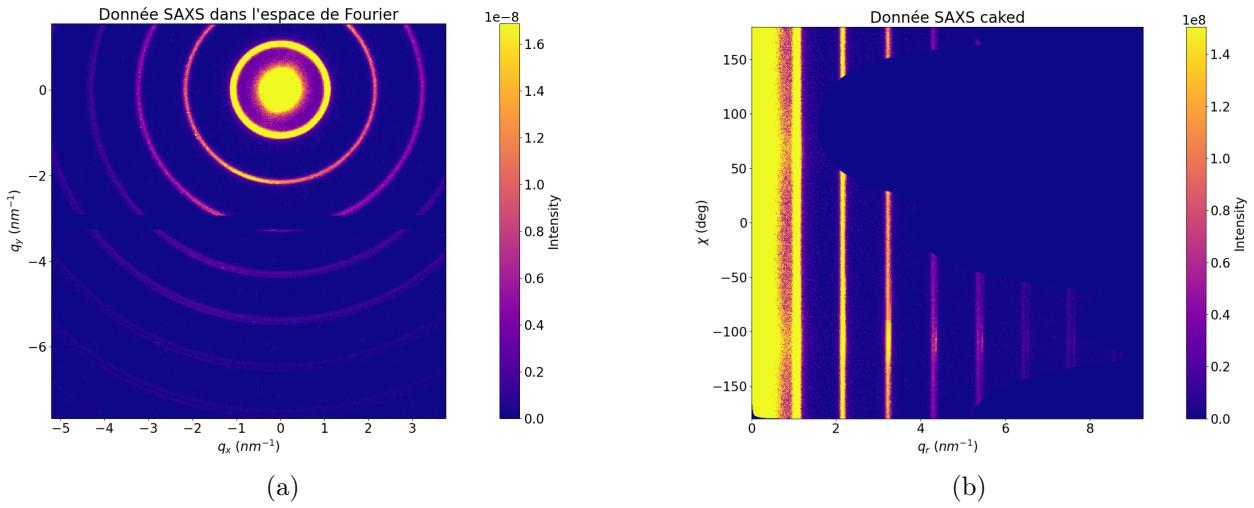


FIGURE 21 – (a) Données SAXS de l’AgBeh après conversion dans le réseau réciproque (q_x , q_y) et (b) après conversion dans le système de coordonnées (q_r , χ). Cela a pour effet de "déplier les anneaux" en lignes droites. On dit que ces données ont subi le processus de caking.

Le caking consiste à passer du repère (q_x , q_y) au repère (2θ , χ), ces angles sont représentés sur la figure 24. Pour se faire on peut tout simplement utiliser les formules suivantes qui exploitent le vecteur $\vec{Q} = \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix}$ l’origine du repère étant centré sur l’échantillon :

$$\cos(2\theta) = \frac{q_z}{\sqrt{q_x^2 + q_y^2 + q_z^2}} \quad (3)$$

$$\cos(\chi) = \frac{q_x}{\sqrt{q_x^2 + q_y^2}} \quad (4)$$

(5)

Ce qui nous permet, à l’aide des coordonnées cartésiennes 3D de \vec{Q} , de retrouver la paire d’angle (2θ , χ) correspondante. Alternativement on peut aussi passer au repère (q_r , χ) en utilisant l’équation 2.

Ce qui se passe concrètement au niveau de l’image, c’est que l’on "déplie" la figure de diffusion, transformant ainsi les cercles en lignes droites. Cela permet de voir à l’œil nu si les anneaux sont circulaires ou elliptiques et s’il y a une répartition non uniforme de l’intensité le long d’un anneau. Cette représentation est utile pour détecter de l’anisotropie dans un échantillon.

4.1.3 Intégration verticale et horizontale

A partir des données 2D dans l’espace de Fourier, il existe plusieurs façons d’extraire de l’information, cela dépend de l’organisation des matériaux que l’on étudie. Si l’organisation à l’intérieur du matériaux se fait sur une longue distance avec des directions préférentielles, on s’attend à une figure de diffusion anisotropique car la lumière va être diffusée dans des directions spécifiques. Dans ce cas c’est l’intégration verticale ou horizontale qui est favorisée.

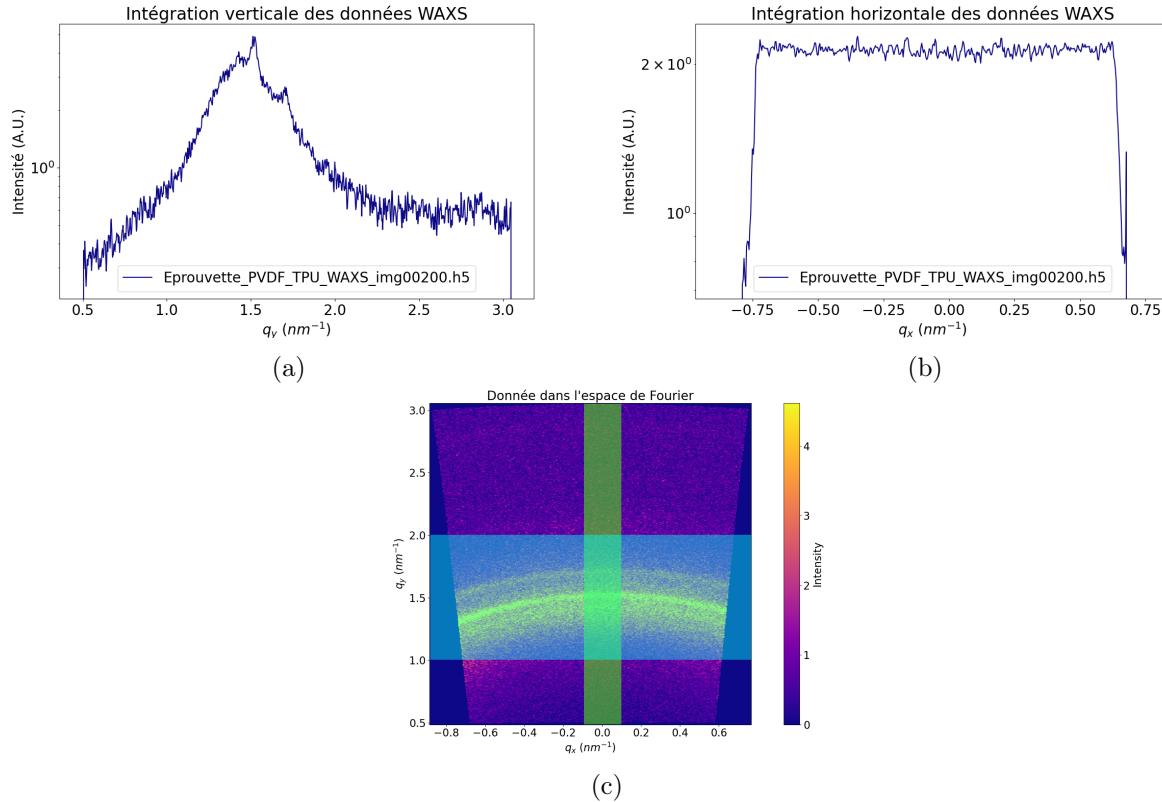


FIGURE 22 – Exemple d'intégration verticale et horizontale. (a) Profil vertical et (b) profil horizontal extrait respectivement de la figure de diffraction WAXS dans l'espace de Fourier (c).

En effet, on va vouloir faire une "moyenne" du signal sur une bande d'intérêt afin de pouvoir en tirer certaines conclusions. On peut résumer cette intégration par une moyenne des lignes (cas horizontal) ou des colonnes (cas vertical) du tableau d'intensité. Un exemple d'intégration verticale et horizontale est donnée dans la figure 22

4.1.4 Intégration radiale et azimutale

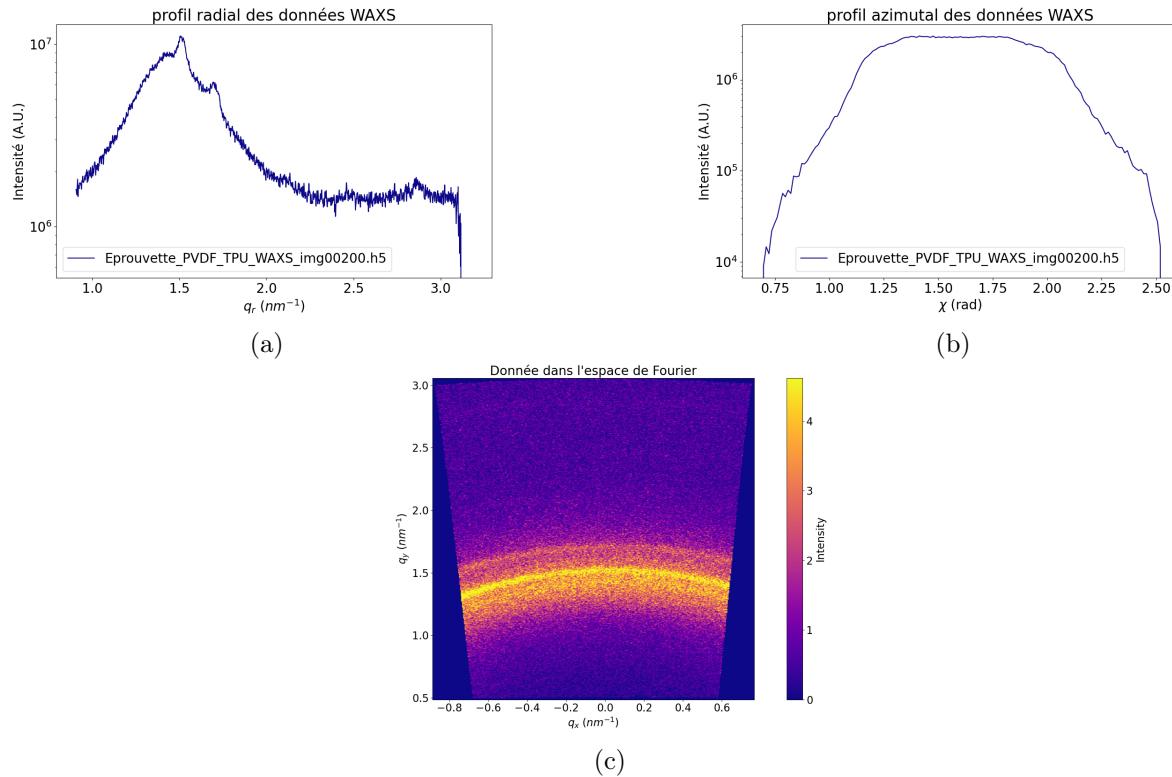


FIGURE 23 – Exemple d'intégration radiale et azimutale. (a) : Profil radial de la figure (c). On voit bien les 2 pics qui sont visibles sur l'image 2D. (b) : Profil azimuthal de la figure (c). On voit que l'intensité diffusée est à peu près constante (au bruit et bords près.) sur l'ensemble de l'arc de cercle. (c) : Figure de diffraction WAXS dans l'espace de Fourier

Dans des cas où le matériau est moins organisé ou de façon plus aléatoire, on va vouloir intégrer le signal suivant des angles plutôt que des positions cartésiennes car on va avoir des figures de diffusion isotropique. La matière désorganisée va avoir tendance à diffuser la lumière sans direction préférentielle ce qui donne des anneaux de diffusion. Cela apparaît dans le cas où la lumière est diffusée par une distance présente dans l'échantillon mais sans orientation préférentielle.

Pour représenter la position d'un point sur l'écran on peut utiliser un système de coordonnées avec 2 angles (2θ et χ) au lieu de coordonnées polaires.

Avec ce système de repérage avec 2 angles, on peut faire :

1. Une intégration radiale : On intègre par rapport à l'angle azimutal, ce qui donne un signal d'intensité en fonction de la distance par rapport au faisceau direct ($q_x = 0$, $q_y = 0$). Le résultat de cette opération est la moyenne de l'intensité en fonction de l'angle radial 2θ
2. Une intégration azimutale : On intègre par rapport à l'angle radial, ce qui donne un signal d'intensité en fonction de la direction de diffusion. Le résultat de cette opération est la moyenne de l'intensité en fonction de l'angle azimuthal χ

Un exemple de chaque opération est donné dans la figure 23.

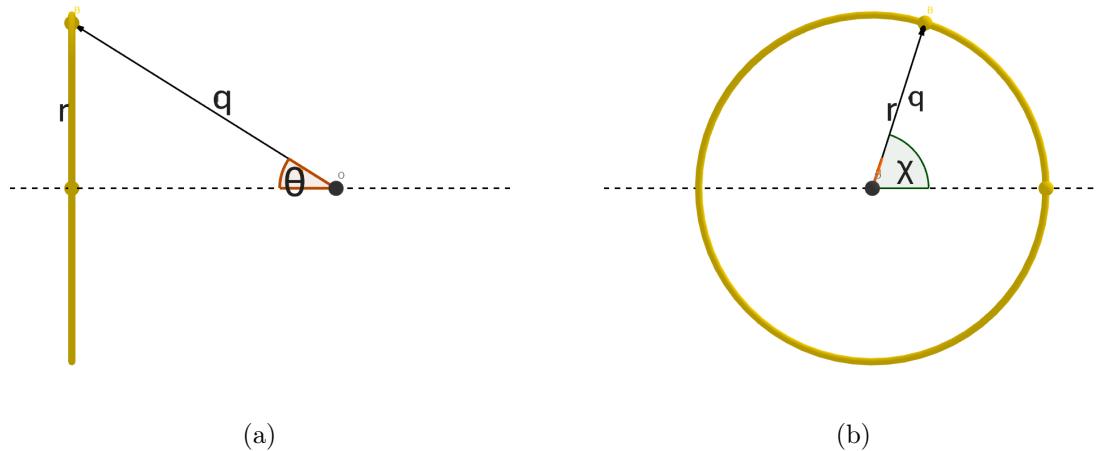


FIGURE 24 – Représentation de la géométrie de l'expérience. L'anneau jaune est sur le plan du détecteur et représente un anneau de diffraction. Coupe du dispositif expérimental SAXS / WAXS mettant en avant (a) l'angle 2θ , et (b) l'angle χ

4.1.5 Conversion en intensité absolue

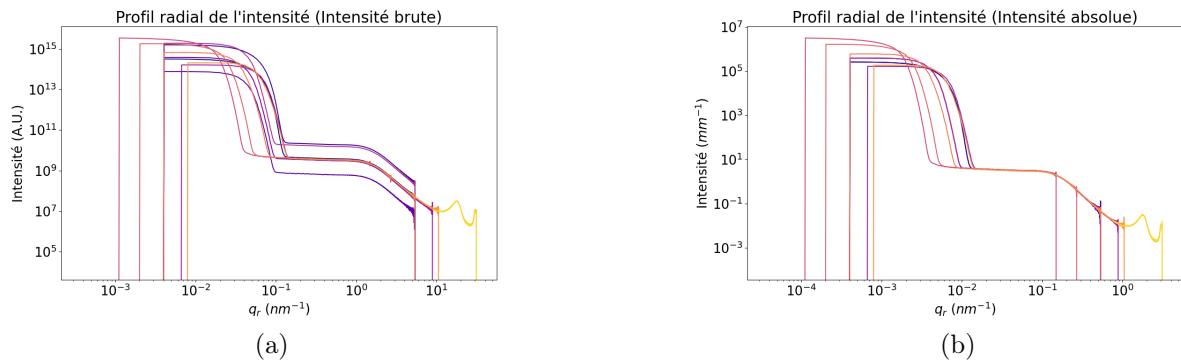


FIGURE 25 – Exemple d'application de l'intensité absolue avec plusieurs mesures sur du carbone vitreux dans des conditions expérimentales variées. (a) : Profils radiaux de l'intensité obtenus dans des conditions expérimentales variées. On voit plusieurs courbes qui ont la même forme mais ne sont pas à la même amplitude à cause des différents paramètres de l'expérience. (b) : Profil radial de l'intensité des différentes mesures après avoir appliqué plusieurs corrections pour obtenir l'intensité absolue pour chaque mesure. On voit que toutes les courbes se superposent, bien que les conditions de l'expérience et les détecteur ne soient pas les mêmes.

Une des problématiques importantes autour de la diffusion des rayons X aux petits angles est de convertir le signal mesuré sur un équipement donné en une intensité absolue, c'est-à-dire identique/universelle quelque soit l'équipement utilisé. Ainsi nous cherchons à convertir l'intensité mesurée sur un équipement, qui s'exprime en nombre de photons comptés par le détecteur, en une intensité absolue. L'intensité absolue est exprimée en $\text{cm}^{-1} \cdot \text{sr}^{-1}$. En pratique on ne mentionne pas le sté-radian, on se concentre uniquement sur l'inverse de la longueur, qui nous ramène à l'unité d'une section efficace macroscopique.

La procédure pour convertir l'intensité mesurée en intensité absolue est expliquée par une équipe

du NIST (National Institute of Standard and Technology) dans un papier nommé *NIST Standard Reference Material 3600 : Absolute Intensity Calibration Standard for Small-Angle X-ray Scattering* [1]. Cette publication détaille l'utilisation d'un matériau de référence afin de calibrer un appareil pour obtenir l'intensité absolue : le carbone vitreux. Grâce à ce carbone vitreux, ils sont capables d'en tirer un facteur d'échelle (SF) par lequel multiplier l'intensité. Dans ce papier la formule suivante est utilisée pour calculer le SF :

$$SF = \frac{1}{\tau e I_0(0) \Delta \Omega} \quad (6)$$

avec :

τ : La transmission de l'échantillon, i.e. le rapport entre l'intensité à $q_r = 0$ (faisceau direct) en présence de l'échantillon et l'intensité à $q_r = 0$ en l'absence de l'échantillon

e : L'épaisseur de l'échantillon. L'unité de l'intensité absolue est l'inverse de l'unité de cette épaisseur

$I_0(0)$: Intensité sur le détecteur en 0 en l'absence de l'échantillon

$\Delta \Omega$: L'angle solide pour chaque pixel du détecteur.

Nous avons légèrement modifié cette formule pour répondre à nos besoins. Nous utilisons donc la formule suivante pour calculer le facteur d'échelle :

$$SF = \frac{1}{\tau e I_0(0) t_e} \quad (7)$$

Il y a donc quelques différences :

1. Nous avons supprimé la normalisation par l'angle solide, cette correction est déjà effectuée lors de l'intégration radiale par pyFAI.
2. $I_0(0)$ et $I_s(0)$ ne sont plus les intensités au centre du faisceau incident mais sont la somme de l'intensité dans toute l'image. De cette manière on prend en compte tous les photons pour le calcul de la transmission.
3. On normalise par le temps d'exposition t_e . En effet dans le papier du NIST la durée de la mesure était fixée à une seconde, ils n'avaient donc pas pris en compte ce paramètre, mais nos expériences pouvant durer plus ou moins longtemps il était important d'ajouter ce paramètre.

Enfin, c'est en multipliant l'intensité par ce facteur d'échelle que l'on obtient l'intensité absolue. Un exemple d'application de l'intensité absolue est donnée en figure 25.

Maintenant que nous avons pu décrire les différentes étapes possibles de réduction des données. Il nous reste maintenant à définir comment les mettre en place et comment les automatiser.

4.2 Préparation de la réduction des données

4.2.1 Quelles opérations de réductions des données ?

Nous avons décrit les différentes étapes possibles de réduction des données. Il nous reste maintenant à définir quelles étapes de réduction sont nécessaires et obligatoires.

4 MISE EN PLACE DU TRAITEMENT DES DONNÉES AUTOMATIQUES

Certaines opérations, telles que la conversion dans l'espace de Fourier et l'intégration radiale, sont toujours réalisées car ce sont des opérations très standards et qui sont tout le temps nécessaires. Ensuite, étant donné que le processus de réduction des données dépend beaucoup des matériaux étudiés, nous avons eu l'idée d'ajouter un champ dans l'en-tête du fichier edf pour que l'utilisateur puisse indiquer des besoins spécifiques. Par exemple, pour la conversion en intensité absolue, il est demandé à l'utilisateur d'indiquer s'il souhaite appliquer l'intensité absolue avant l'acquisition des données. Pour cela, nous avons créé un champ dans l'en-tête des données pour spécifier la procédure d'analyse demandée.

Ainsi, du côté du code d'analyse, il nous suffit d'aller chercher cette indication dans l'en-tête pour permettre de lancer automatiquement la réduction de données.

4.2.2 Adaptation de l'acquisition des données

La procédure d'acquisition des données dépend fortement de l'analyse souhaitée à posteriori. Par exemple, comme nous l'avons décrit dans la section précédente, la conversion en intensité absolue nécessite la prise d'une image du faisceau sans échantillon pour pouvoir faire la normalisation. Ainsi mon tuteur a déployé, depuis l'environnement d'acquisition de données, des macros qui permettent d'établir des enchaînements d'acquisition de données pour collecter toutes les mesures nécessaires pour le futur traitement des données. En pratique, lorsque l'utilisateur arrive sur l'équipement, il a uniquement besoin de définir l'analyse de données souhaitée pour ces échantillons avant de démarrer les mesures.

4.2.3 Challenges et approches du déploiement des processus d'analyse des données

Une fois les données collectées, nous avons un fichier (ou un groupe de fichiers hdf5) qui doit pouvoir être ouvert et modifié afin de le faire passer par certains processus définis par l'utilisateur. Comme mentionné précédemment, cette procédure doit être automatique.

Comme pour la modification des en-têtes des fichiers hdf5 à posteriori, il est important de proposer un outil aux utilisateurs pour leurs permettre de refaire les différentes opérations de réduction de données s'ils les jugent mauvaises, fausses, ou veulent modifier l'analyse à réalisée à posteriori. La possibilité de sauvegarder les données à nouveau traitées est importante. Finalement, une interface graphique pour permettre aux utilisateurs d'ouvrir les fichiers hdf5 et de parcourir les fichiers rapidement était nécessaire.

Pour répondre à tous ces besoins, j'ai choisi d'utiliser la Programmation Orientée Objet (POO) qui répond à tous ces critères. On peut définir un constructeur qui permet d'ouvrir une liste de fichiers, définir une méthode pour fermer le groupe de fichiers correctement et définir autant de méthode que de processus applicables aux données.

De plus, il est possible d'inspecter la structure d'une classe et de ses méthodes, ce qui va permettre de développer un GUI dynamique. Dynamique dans le sens où si l'on ajoute une méthode à la classe le GUI se mettra à jour en conséquence. Utiliser une classe présentera beaucoup d'autres avantages.

4.2.4 Introduction à la structure de NexusFile

Le cœur du traitement des données repose donc sur une classe que j'ai nommé NexusFile. J'ai choisi ce nom car le but de cette classe est d'ouvrir un fichier hdf5 dans un format de données qui provient de Nexus. La structure de la classe est assez simple et est illustrée par un UML sur la figure 26.

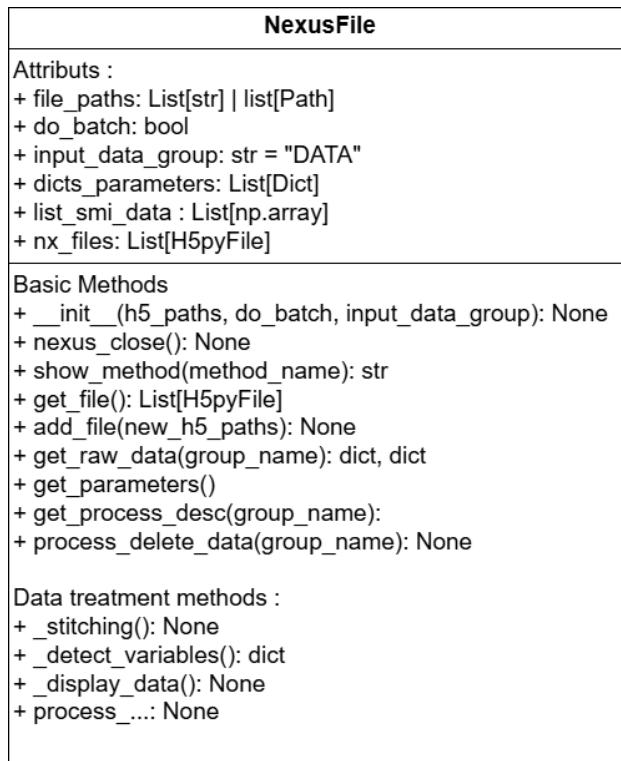


FIGURE 26 – Modélisation UML de la classe NexusFile

4.3 Les méthodes de la classe NexusFile

4.3.1 Mise en place des méthodes de base

La classe NexusFile est composée de plusieurs méthodes. Les méthodes nommées "méthodes de base" sont les méthodes qui ne concernent pas le traitement de données. Ces méthodes ne manipulent que la structure du fichier ou de l'objet NexusFile.

Le constructeur

Le constructeur est assez simple, il prend en entrée la liste des données qui vont être traitées ainsi que le nom du groupe de données qui va être utilisé pour faire le traitement. Outre l'assignement de quelques attributs de base à l'objet, le constructeur va aussi ouvrir l'intégralité des fichiers hdf5 qui ont été fournis ainsi qu'extraire les paramètres importants pour l'analyse de données, à savoir :

1. Le tableau numpy qui stocke l'intensité et les vecteurs positions associés. On ne peut traiter que les images 2D pour le moment.
2. La position du beamstop. Dans notre cas, le XEUS n'a pas de beamstop donc il est toujours égal à [0, 0]

4 MISE EN PLACE DU TRAITEMENT DES DONNÉES AUTOMATIQUES

3. La longueur d'onde du faisceau incident
4. L'angle incident du faisceau
5. Le nom du détecteur. Cette information est utilisée par la librairie smi_analysis pour connaître les dimensions de l'image
6. Les rotations du détecteur.
7. La position du centre du faisceau incident
8. La distance détecteur-échantillon

Les fichiers hdf5 restent ouverts jusqu'à leur fermeture via la méthode nexus_close ce qui permet d'améliorer les performances en évitant de fermer/ouvrir les fichiers après chaque fin/début de processus.

Nexus close

C'est cette méthode, très simple, qui va permettre de fermer les fichiers. La seule particularité de cette méthode est la présence de la fonction repack dont j'ai parlé auparavant afin que le fichier prenne le moins de place possible après qu'il ait été manipulé.

Show method

Cette méthode est une aide lors de l'utilisation la classe NexusFile. Cette aide est plutôt accessible via un script Python plutôt que par le GUI. Elle permet d'afficher la liste complète des méthodes ou la documentation complète de n'importe quelle méthode de la classe NexusFile. J'ai mis en place cette méthode car il me semblait qu'il était plus pratique pour un utilisateur d'avoir toutes les informations dont il a besoin concernant une méthode affichée dans son terminal plutôt que de devoir aller chercher dans une documentation internet qui peut s'avérer très lourde à lire.

get/add file

Ce sont tout simplement les méthodes qui permettent de récupérer ou d'ajouter des fichiers hdf5.

get raw data

Cette méthode est aussi conçue pour les utilisateurs qui se servent de cette classe dans un script Python plutôt que par GUI.

Elle permet de récupérer les données brutes d'un groupe de données, c'est-à-dire sous forme de tableau numpy. Les données sont récupérées sous forme de deux dictionnaires, un pour les paramètres (q_r par exemple) et l'autre pour les données (L'intensité). La clé du dictionnaire est le nom du fichier et la valeur associée est le tableau numpy correspondant.

get parameters

Tout comme la méthode get file, cette méthode permet de récupérer les paramètres qui ont été utilisés pour faire le traitement des données. On procède comme avec get raw data et on retourne un dictionnaire avec pour clé le nom du fichier et en valeur le dictionnaire de paramètres qui a servi pour l'analyse de ce fichier.

get process desc

Lorsqu'on applique un processus à un jeu de données d'un fichier grâce à la classe NexusFile, on crée aussi un groupe qui permet de décrire le processus utilisé sur les données. Cette méthode permet de récupérer cette description sous forme de dictionnaire qui a pour clé le nom du fichier et pour valeur la description du processus demandé. Cela permet de savoir ce qui a été fait sur un groupe de données facilement.

process delete data

Cette méthode permet tout simplement de supprimer un groupe de données du fichier hdf5.

C'est donc la fin de la description du fonctionnement des méthodes de base de la classe NexusFile. Grâce à ces méthodes on pourra plus facilement gérer le stockage et l'enregistrement des données dans le fichier hdf5.

4.3.2 Mise en place des méthodes de traitement

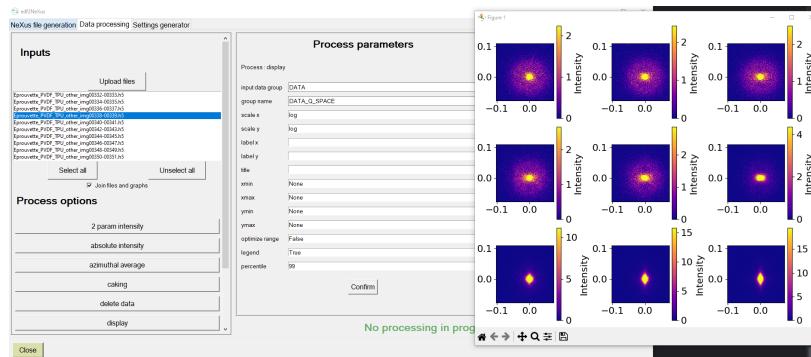


FIGURE 27 – Capture du gui en train d'afficher des données. la figure apparaissant dans une fenêtre matplotlib, il est tout à fait possible de la laisser ouverte et de continuer à utiliser le gui

Passons maintenant à la description précise des méthodes dont le nom commencent par "process_ ", les méthodes qui servent pour le traitement des données. De manière générale, les méthodes de traitement exécutent une boucle sur l'ensemble des fichiers pour traiter tous les fichiers qui sont ouverts dans l'objet NexusFile. De plus, les méthodes de traitement ont de manière générale des paramètres qui sont pour le traitement de données et des paramètres qui sont là pour contrôler l'affichage des données.

Outre ces paramètres, on a aussi les paramètres spécifiques aux méthodes. Il faut parfois leur donner des valeurs par défaut et on ne peut pas accepter de None. Afin de gérer les paramètres par défaut j'ai créé 2 dictionnaires, un dictionnaire initial _none _flags qui a pour paire "clé-valeur", nom paramètre/true ou false. Le deuxième dictionnaire contient la paire nom paramètre/valeur par défaut. Ainsi, pour chaque paramètre, on assigne ou non une valeur par défaut.

Finalement, une fois le processus terminé, on peut l'afficher, ce qui utilise une méthode nommée _display_data, ou sauvegarder les données. La méthode qui permet d'afficher les données est très longue, complexe et dépend de la dimension des données. N'ayant aucun intérêt physique, nous n'en

4 MISE EN PLACE DU TRAITEMENT DES DONNÉES AUTOMATIQUES

parlerons pas dans ce rapport.

Concernant l'enregistrement des données, on utilise une fonction `save_data` qui vient tout simplement sauvegarder correctement les données dans le fichier hdf5 mais crée aussi un autre groupe dans lequel sera enregistré la description du traitement qui vient d'être effectué.

Un exemple est donné en figure 28.

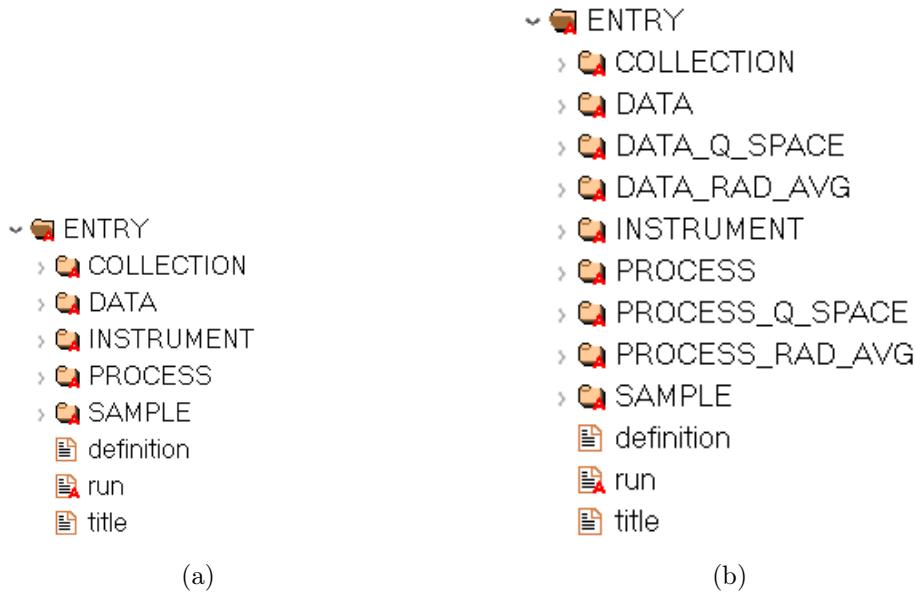


FIGURE 28 – Structure d'un fichier hdf5 avant (a) et après (b) traitement des données

process q space

Exception faite des paramètres généraux, cette méthode n'a pas de paramètre particulier. Elle combine plusieurs données ensemble si nécessaire à l'aide de la librairie Python `smi_analysis` et fait la conversion dans l'espace de Fourier.

Suite à ça on récupère les données d'intensité dans l'espace de Fourier et on construit les tableaux qui accueillent les vecteurs $\vec{q} = [q_x, q_y]$. Grâce à ces deux tableaux on peut remplir les datasets I et Q du groupe de données d'intérêts. Dans ce cas là le tableau d'intensité est en 2D et le tableau de vecteur \vec{q} est constitué de deux tableaux 2D, un contient les position q_x , l'autre les positions q_y . Une fois ce traitement effectuer on affiche et on sauvegarde dans le fichier hdf5 en fonction des paramètres que l'utilisateur a choisi. C'est donc ainsi qu'on obtient les données d'intensité dans l'espace de Fourier : $I(q_x, q_y)$.

process caking

Pour cette méthode, on a des paramètres qui sont des angles ou un nombre de points, ce qui ne pose aucun problème pour leurs valeurs par défaut. En revanche pour `rad_min` et `rad_max`, c'est un peu différent. En effet, par défaut, on veut couvrir toute la surface de l'image cela implique que le q_r minimum et maximum vont variés en fonction des données. De plus, on ne possède que les deux listes suivantes qui donnent l'intervalle des q_x et q_y : $[q_{x,min}, q_{x,max}]$ et $[q_{y,min}, q_{y,max}]$.

Pour le q_r maximum, l'image de base étant carré on prend la valeur maximale de q_x et de q_y en

valeur absolue et on applique la formule :

$$q_{r,max} = \sqrt{q_{x,max}^2 + q_{y,max}^2} \quad (8)$$

En revanche, pour le q_r minimum la situation est un peu plus délicate.

La position $q_r = 0$ correspond à la position du faisceau en coordonnées radiales. Donc lorsque le faisceau est présent sur l'image, on prend le `rad_min` à zéro. En revanche, si le faisceau est en dehors du détecteur, prendre le q_r minimum à 0 n'a pas de sens car il n'y a pas de signal au niveau du faisceau et des alentours. Ainsi, dans ce cas on prend la valeur minimale de q_x et de q_y en valeur absolue et on applique la formule :

$$q_{r,min} = \sqrt{q_{x,min}^2 + q_{y,min}^2} \quad (9)$$

On aurait pu prendre la norme minimale/maximale de l'ensemble des vecteur mais je n'ai pas pu employer cette technique car les valeurs de q_x et q_y sont fournies sous forme de deux listes contenant deux éléments et non pas un tableau numpy. Pour les autres c'est beaucoup plus simple, on peut toujours prendre l'intervalle $[-180, 180]$ pour χ sans avoir de problème car les valeurs d'un angle sont toujours incluses dans cet intervalle, à l'inverse de q_r qui peut prendre des valeurs dans l'intervalle $[0, \infty]$.

Quant au nombre de points on choisit 1000s points pour les 2 intervalles, c'est un nombre qui a déjà été pensé pour ne pas fausser les résultats. Il est basé sur 2 critères :

1. Il ne faut pas avoir plus de points qu'il y a de pixel dans une direction. Subdiviser l'image en plus de petit carré qu'il n'y a de pixel n'aurait aucun sens physiquement et reviendrait à faire du sur-échantillonnage.
2. Il faut avoir suffisamment de points pour faire le binning sans perte d'information. En effet, s'il y a trop peu de points, le binning ne sera pas significatif de la réalité.

Avec ces 4 paramètres (`[azi_min, azi_max]` et `[rad_min, rad_max]`), on définit un arc de cercle dont la longueur est définie par l'intervalle azimuthal et dont l'épaisseur est définie par l'intervalle radial. C'est dans cet arc de cercle que l'intégration est effectuée.

Une fois les paramètres remplis, on assigne les valeurs par défaut si nécessaires, puis on utilise la méthode `caking` de la librairie `smi_analysis`. Une fois le `caking` effectué, on récupère la liste des q_r et des χ afin de construire un `meshgrid` de q_r et de χ afin de s'en servir pour l'affichage de `caking` et l'enregistrer. Bien entendu on récupère aussi le tableau d'intensité du `caking` que l'on peut afficher et/ou enregistrer dans un groupe de données approprié du fichier `hdf5`.

process radial/azimuthal average

Les paramètres de ces fonctions sont essentiellement les mêmes que pour le `caking`. La seule différence est qu'on ne peut pas choisir le nombre de point dans l'intervalle azimuthal, ce paramètre est imposé du côté de la librairie `smi_analysis`.

Les paramètres nous permettent de choisir la zone de l'image sur laquelle on veut faire l'intégration pour arriver à un signal 1D. L'utilité de définir une zone et de ne pas intégrer sur toute l'image est d'isoler des zones d'intérêts, surtout dans les cas où l'image est asymétrique.

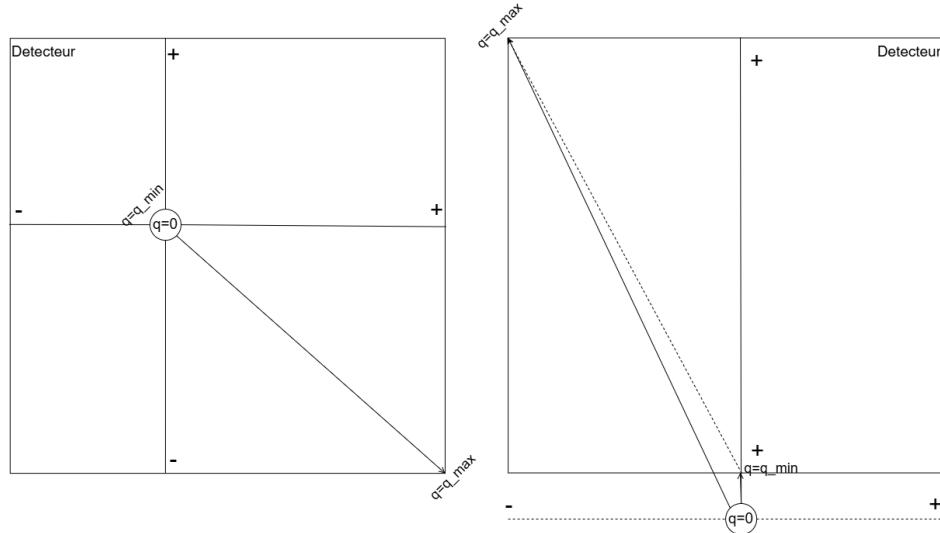


FIGURE 29 – Schéma expliquant le choix de $q_{r,min}$. Sur le premier schéma, le centre du faisceau est dans la région du détecteur, ce qui permet de prendre $q_{r,min} = 0$. Dans le deuxième schéma, le faisceau est hors de la région du détecteur, on prend donc le coin le moins éloigné du faisceau, ce qui correspond aux q_x , q_y les plus petits possibles.

Encore une fois, on passe les paramètres à la librairie `smi_analysis` qui s'occupe de faire la moyenne radiale du signal dans la zone en arc de cercle spécifiée. Cela nous retourne une liste de q_r et une liste d'intensité associée qu'on peut tout simplement enregistrer dans le fichier `hdf5`.

process horizontal/vertical intégration

Cette fois la zone définie pour faire l'intégration est une région rectangulaire. On utilise donc 4 paramètres pour identifier les 4 coins de la zone rectangulaire sur laquelle intégrer le signal. Par contre on ne choisit pas le nombre de point pour l'intégration, c'est la librairie `smi_analysis` qui s'en occupe là aussi.

process absolute intensity

Ce processus n'est pas basé sur la bibliothèque `smi_analysis` et a été codé entièrement dans le package `saxs_nxformat`. On commence par fournir les paramètres pour le calcul de l'intensité absolue qui ne sont pas déjà présents dans le fichier et on les utilise pour calculer le facteur d'échelle, cf. partie conversion en intensité absolue [4.1.5](#).

On fait la multiplication sur les données brutes pour qu'on ait aussi l'intensité absolue sur les traitements qui suivent.

process 2 param intensity

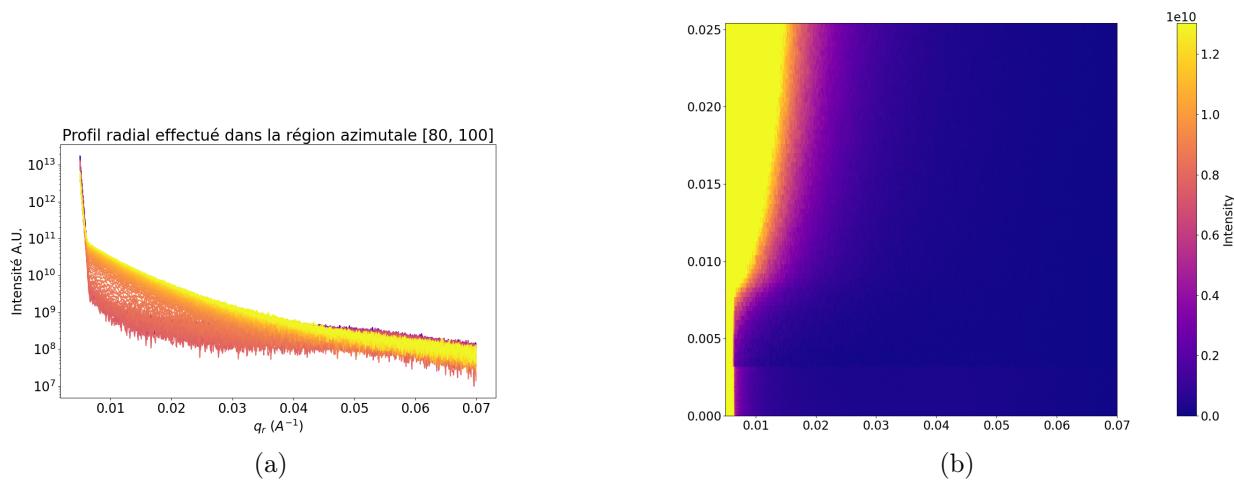


FIGURE 30 – Exemple de représentation d'un grand nombre de données ou un paramètre change entre chaque mesure (ici l'étirement de l'échantillon). On voit bien qu'il y a une évolution sur la figure (a) mais on ne peut pas savoir à quelle mesure correspond quel graphe et on ne peut pas mettre de légende car elle serait beaucoup trop longue (il y a environ 150 courbes). En passant en carte de chaleur 2D (b), où on a sur l'axe des x la valeur de q_r et en y la valeur de l'étirement, on voit plus clairement l'évolution de l'intensité en fonction de l'étirement.

Ce processus n'a rien de physique, c'est purement un exercice de visualisation de données. L'idée vient de la situation suivante : un grand nombre de mesures a été effectué avec un paramètre qui change, le stress appliqué à l'échantillon. Il est intéressant d'observer comment le stress appliqué influe sur la figure de diffraction des rayons X, notamment le profil radial de celle-ci.

On peut visualiser ce changement en traçant toutes les courbes sur le même graphique, ou alors en traçant le profil radial de l'intensité en fonction, non seulement de q_r , mais aussi en fonction du stress. Pour ce faire, on peut utiliser une carte de chaleur.

La difficulté est de savoir quel "autre" paramètre est valide. Afin de détecter les paramètres qui varient parmi un ensemble de fichier j'ai créé une méthode privée "detect variables" qui va explorer l'ensemble des fichiers ouverts et relever les paramètres qui changent entre chaque fichier. La fonction va ainsi retourner le chemin hdf5 du paramètre qui varie. Il est cependant toujours possible d'utiliser l'index du fichier (à savoir sa place dans la liste des fichiers) pour tracer la carte de chaleur.

Une fois cet autre paramètre fourni, on fait plusieurs vérifications pour être sûr qu'il n'y ait pas de problème d'affichage. Une fois tous les paramètres extraits on fabrique "à la main" un meshgrid des deux paramètres ainsi que du tableau des intensités. De cette manière, on peut tracer une carte de chaleur de l'intensité en fonction des deux paramètres.

La méthode demande encore un peu de travail, on ne peut pas encore choisir le titres et les axes de la figure comme on le voir sur la figure 30.

process display

4 MISE EN PLACE DU TRAITEMENT DES DONNÉES AUTOMATIQUES

Ce processus permet tout simplement d'afficher ce que contient un groupe de données sous forme de graphique. Il est possible de régler l'échelle du graphique ainsi que les différents labels et légendes possibles.

Ce processus utilise tout simplement la méthode `_display` que j'ai codé. Cette méthode permet de détecter si les données qu'on essaie d'afficher sont 1D ou 2D et affiche donc un graphique ou une carte de chaleur en fonction. Elle permet aussi de rassembler tous les graphiques d'un groupe de données de tous les fichiers sélectionnés dans un seul et même graphique.

Pour résumer, cette méthode ne sert qu'à visualiser les données présentes dans le fichier hdf5. Il est aussi possible de le faire grâce à HDF view mais l'interface est un peu complexe et peu intuitive.

Je n'ai malheureusement pas eu le temps de me plonger dans le fonctionnement détaillé du code de la librairie `smi_analysis` dans le cadre de mon alternance car j'avais déjà toute ma partie à coder. De plus, je n'avais pas besoin de refaire le code qui permettait de faire tous ces processus étant donné que celui qui existait était déjà à jour et parfaitement fonctionnel.

4.4 Perspectives d'amélioration de la classe `NexusFile`

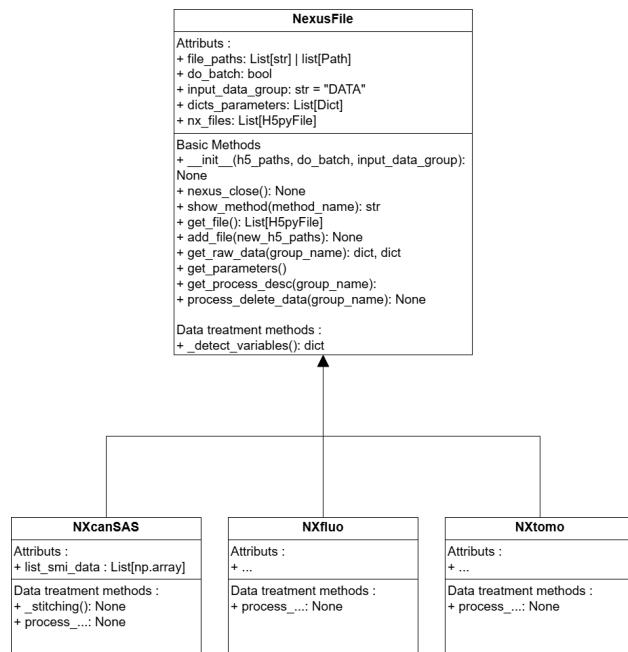


FIGURE 31 – Modélisation UML de l'amélioration de la classe `NexusFile` et de ses sous classes.

La classe `NexusFile` reste très simple et est très spécifique à la définition `NXcanSAS`, ce qui rend son nom un peu caduque. On peut imaginer quelques améliorations.

Pour commencer, plutôt que de se limiter au standard `NXcanSAS` on pourrait étendre le programme à d'autres standards comme `NXfluo` pour les expériences de fluorescence X ou `NXtomo` pour les expériences de tomographie X.

Pour se faire on pourrait garder la classe `NexusFile` avec les méthodes de base et créer des sous-classes nommées `NXcanSAS`, `NXfluo`, `NXtomo` avec leurs méthodes bien spécifiques. La classe parent `Nexus-`

File accueillerait l'ensemble des attributs et méthodes de base. Les sous-classes accueilleraient donc les méthodes de traitement ainsi que d'autres méthodes spécifiques au type d'expériences qu'elles encapsulent.

Par exemple, dans la classe NXcanSAS on aurait les méthodes process_q_space ou process_radial_average, dans la classe NXtomo et NXfluo on aurait des méthodes "process_" différentes qui sont caractéristiques de ces expériences.

Ainsi, dans le cas où on voudrait étendre le programme à plusieurs standards NeXus, il serait préférable d'adopter cette structure que j'ai modélisé dans la figure 31.

Cette structure est en effet plus flexible et adaptée dans le cas où plusieurs standards sont utilisés. Je n'ai malheureusement pas pu la mettre en place avant le rendu de ce rapport car nous avons commencer à réfléchir à la possibilité d'exporter le programme sur d'autres machines assez tard dans l'alternance.

4.5 Mise en place du GUI

Après avoir développé la classe Nexus File, qui peut être utilisée via un script Python, il a fallu que je mette en place un GUI. L'avantage principal d'une interface graphique est qu'elle permet aux utilisateurs sans connaissance en Python d'utiliser le programme. C'est aussi plus facile d'utilisation avec une approche "click and go".

L'interface graphique pour l'analyse de données est composé de 3 panneaux principaux :

1. Le panneau d'entrée
2. Le panneau des paramètres
3. Le panneau console

Nous développerons le fonctionnement de ces panneaux dans les parties suivantes.

4.5.1 Construction du panneau d'entrée

Il est constitué de deux parties principales. La première partie concerne la sélection des fichiers qui doivent être traités. Il est possible de charger des fichiers dans l'application à l'aide du bouton "Upload".

Les fichiers sont ensuite mis dans une liste dans laquelle il est possible de sélectionner les fichiers qui doivent subir le traitement, il y a aussi des boutons pour sélectionner tous les fichiers ou tous les enlever de la sélection. Enfin il y a une option pour que tous les graphiques résultant du traitement s'affichent dans la même image ou dans une image séparée.

Une perspective d'amélioration serait d'ajouter des boutons pour ajouter ou retirer des fichiers à une sélection qui est déjà faite. Pour le moment, en appuyant sur le bouton chargement on supprime la sélection précédente au profit de celle qui vient d'être sélectionnée.

Pour créer cette partie, rien de bien compliqué, on utilise tout simplement les widgets tkinter bouton pour créer les boutons et le widget Listbox pour la liste des fichiers sélectionnables.

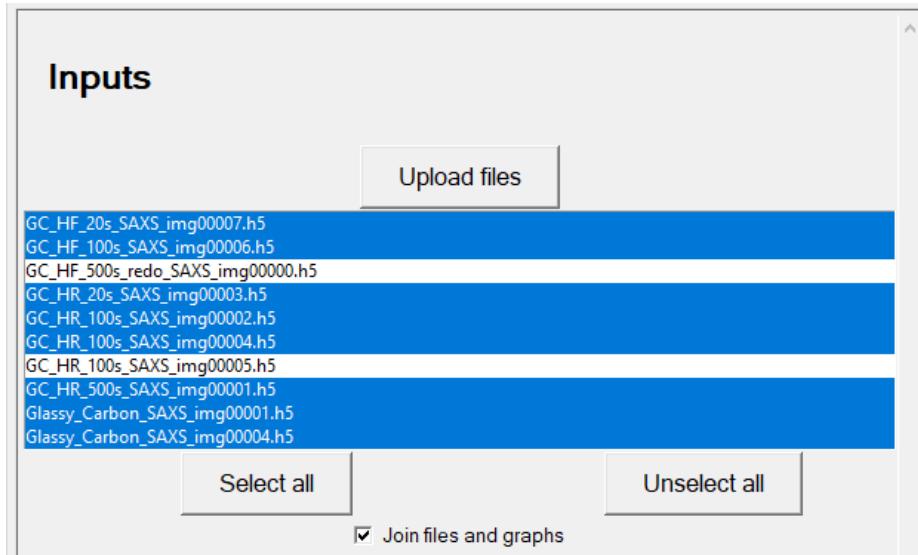


FIGURE 32 – La partie du GUI qui permet de sélectionner les fichiers à traiter. Plusieurs fichiers sont présents mais ils ne sont pas tous traités, seuls ceux en surbrillance le sont.

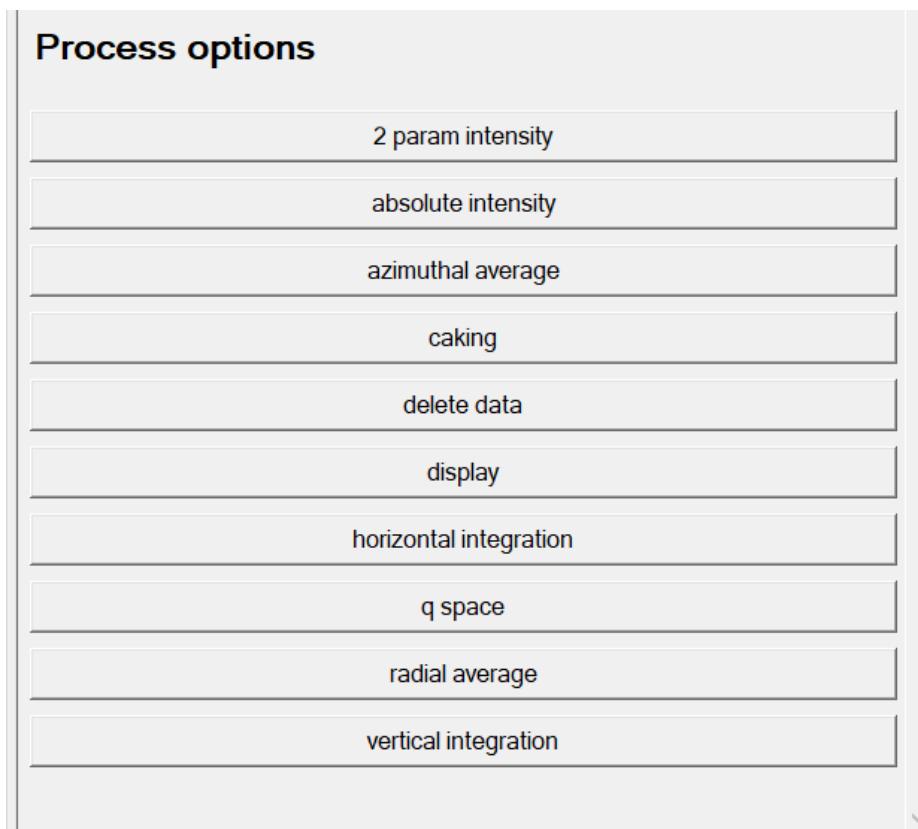


FIGURE 33 – La partie du GUI qui permet de sélectionner le traitement à effectuer. Seul un traitement à la fois peut être appliqué.

La deuxième partie permet de sélectionner le processus que l'on veut appliquer aux données. Ce sont tous simplement des boutons tkinter qui sont liés à une méthode "create param" qui permet d'afficher les paramètres de la méthode correspondante dans le panneau des paramètres. Afin de créer ces boutons, et de leur donner le bon label, on utilise une astuce liée à la bibliothèque

native à Python : inspect. Cette bibliothèque nous donne la possibilité d'inspecter une classe afin d'en relever toutes les méthodes qui la constituent. On ne retient que les méthodes qui commencent par "process_" et on crée un bouton relié à la fonction "create param" avec un paramètre par défaut qui est le nom complet de la méthode.

Le plus important, c'est l'inspection de la classe. Ainsi, si une nouvelle méthode "process_" est ajoutée dans la classe NexusFile, elle sera dynamiquement détectée et ajoutée dans le GUI.

Le tout est englobé dans un cadre défilant. De cette manière, on peut faire défiler les éléments pour voir toutes les opérations existantes.

4.5.2 Le panneau des paramètres

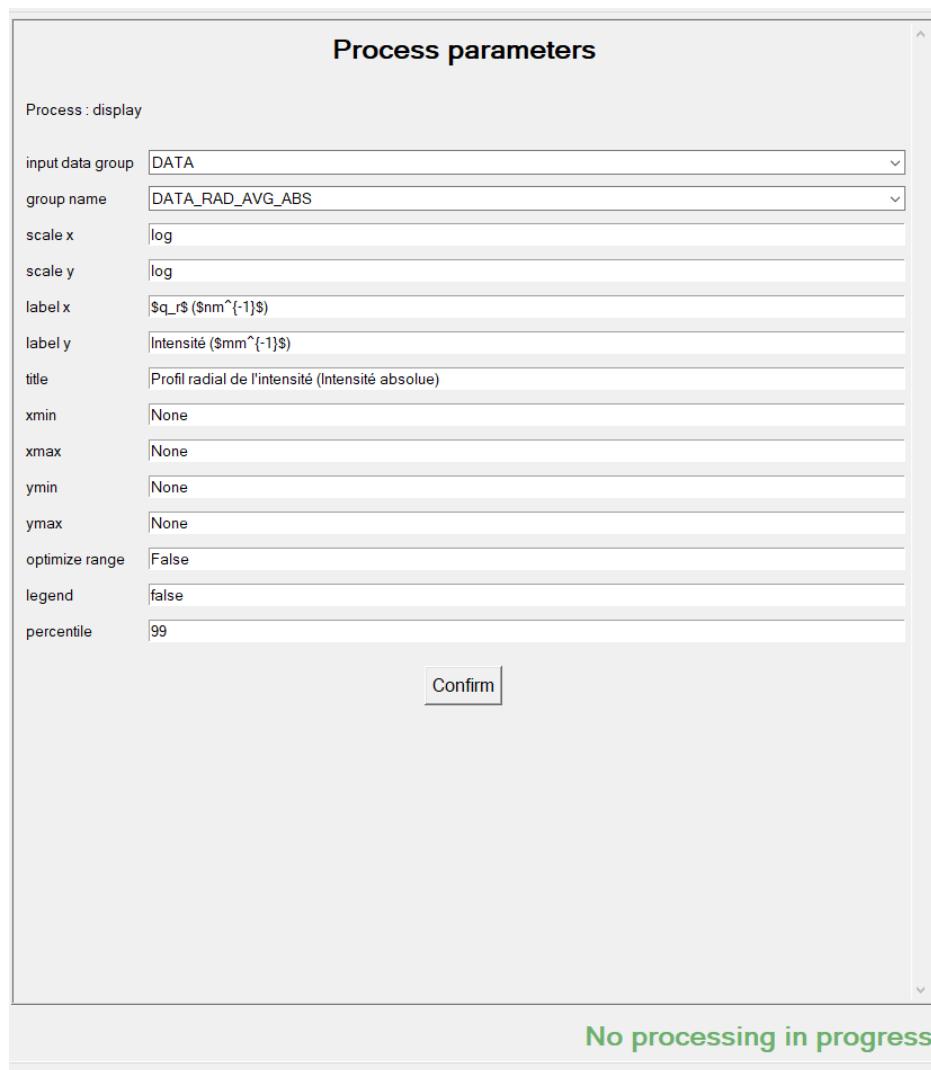


FIGURE 34 – La partie du GUI qui permet de choisir les paramètres du traitement correspondant, ici nous avons choisi le traitement qui permet d'afficher les données.

Ce panneau permet de renseigner les paramètres liés à la méthode qui a été sélectionnée par l'utilisateur. Il est vide par défaut. Pour le remplir, il faut que l'utilisateur charge et sélectionne au moins un fichier avant de cliquer sur un des boutons de traitement.

Suite à cela, on crée un seul paramètre "à la main", c'est le paramètre qui va permettre de sélectionner les données d'entrée, le paramètre "input data" de la classe NexusFile. Ce paramètre est tout simplement un menu déroulant qui contient tous les groupes de données communs aux fichiers sélectionnés.

Pour le reste des paramètres, tout est 100% dynamique :

1. On récupère la méthode et sa signature
2. On parcourt tous les paramètres de la signature
3. Ensuite on sépare le paramètre en trois morceaux : le nom, le type et la valeur par défaut. Le type n'est pas encore utilisé mais une bonne perspective serait de l'utiliser pour créer un certain type de champs remplissables. Par exemple si le type est booléen, on pourrait utiliser une boîte à cocher au lieu d'une boîte où on saisit du texte qui accepte uniquement "true" ou "false" en paramètre.
4. On crée ensuite le widget qui va accueillir la valeur du paramètre. C'est par défaut une boîte où on peut saisir du texte, mais il y a des exceptions. J'ai choisi d'utiliser un menu déroulant pour les paramètres qui font intervenir le nom de groupe de données. De cette manière, ceux qui sont déjà disponibles ou occupés sont visibles. Il est cependant toujours possible de choisir un nom de groupe qui n'existe pas déjà ou qui est occupé.
5. Enfin, on insère la valeur par défaut au widget créé et on lui donne un nom qui est le nom du paramètre complet avant de le faire apparaître à l'écran.

Pour finir, on crée un bouton qui permet de lancer le processus. Ce bouton est associé à une autre méthode, similaire à celle qui construit les paramètres mais qui permet de lancer le traitement. Ce bouton prend en paramètre le processus qui a été sélectionné.

4.5.3 Lancement du traitement des données

Une fois le bouton de confirmation lancé, la méthode "start processing" est appelée, avec en paramètre, le processus sélectionné. Cette fois, on parcourt l'ensemble des widgets de la fenêtre des paramètres :

1. Dès que l'on trouve un widget avec l'attribut "tag", on le récupère
2. Ensuite, une fois le widget récupéré, on prend sa valeur, son contenu et on l'ajoute à un dictionnaire qui a pour paire "clé-valeur" nom_paramètre / valeur_paramètre.

Une fois les paramètres récupérés, on ouvre les fichiers sélectionnés avec la classe NexusFile. De là, on appelle la méthode process. Mais ce n'est pas aussi simple.

Le premier obstacle que j'ai rencontré vient du fait que j'ai récupéré la méthode de traitement telle qu'elle est codée dans le script, c'est-à-dire dans la classe NexusFile, donc ce traitement est un "objet méthode" de la classe NexusFile. Je ne pouvais donc pas utiliser la syntaxe objet.méthode() car j'avais directement la méthode sous forme d'objet. Il a donc fallu que je revienne à ce que fait l'appel d'une méthode dans le contexte des classes. Une méthode contient forcément le paramètre "self" qui représente l'instance de l'objet. Donc quand on fait objet.méthode([ensemble de paramètres]), en terme de code (et de manière très simplifiée) on fait méthode(objet, [ensemble de paramètre]).

Donc plutôt que d'appeler la méthode de manière classique, j'ai fait passer l'instance de l'objet que je venais de créer à la méthode directement.

Le deuxième obstacle a été la manière dont je devais passer les paramètres. Certains traitements ont des paramètres en commun, mais certains sont spécifiques à chaque méthode de traitement. Il se trouve que, par un gros coup de chance, la manière dont je stockais les valeurs des paramètres (dans un dictionnaire avec pour paire nom_paramètre / valeur_paramètre) était une excellente manière de faire. En utilisant l'opérateur double astérisque (**) sur un dictionnaire qu'on passe en paramètre de fonction, il s'avère que le dictionnaire va être unpacké afin de remplir les paramètres de la fonction.

Par exemple, pour un dictionnaire `dict = {param1 : valeur1, param2 : valeur2}`, lorsqu'on lui applique l'opérateur `**` dans une fonction nommée `func`, on a `func(**dict)` qui devient `func(param1=valeur1, param2=valeur2)`.

C'est donc cette technique d'unpacking des paramètres que j'ai utilisé pour remédier à ce deuxième problème. Le dictionnaire que je construis en récupérant les paramètres ayant précisément la structure nom_paramètre / valeur_paramètre, je peux utiliser cette technique sans avoir peur d'avoir des arguments manquants car tout est basé sur la signature de la méthode. J'arrive ainsi à boucler la boucle comme représenté sur la figure 35

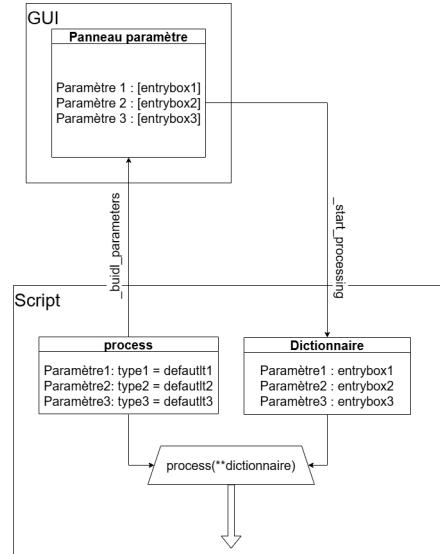


FIGURE 35 – Appel de méthode via le GUI

4.5.4 Le panneau de console

Ce panneau contient tout simplement une zone de texte qui affiche diverses informations sur les différentes étapes de traitement des données. Les informations restent très simples, par exemple la console affiche quand le processus est démarré et fait remonter des erreurs si nécessaire. Il n'y a pas grand chose de plus, c'est surtout un panneau qui a pour objectif d'informer l'utilisateur du bon déroulement du traitement des données.



FIGURE 36 – La partie du GUI qui permet d'afficher ou en est le traitement des données ou s'il y a eu une erreur.

5 Finitions et automatisation complète du traitement

Afin de terminer l'automatisation du traitement des données, il ne reste qu'une seule chose à faire : automatiser le lancement du programme. En effet, une fois que le thread de traitement des données est lancé, tout peut se faire automatiquement. Mais dans l'objectif d'éviter tous les problèmes, il est important de mettre en place pour le programme un moyen de s'arrêter et de redémarrer sans aucune intervention humaine.

5.1 Jenkins

5.1.1 Introduction

Jenkins est un serveur d'intégration continue. Il permet de régulièrement lancer des "tâches". Lorsqu'une "tâche" est lancée on dit qu'elle a été "build". Jenkins conserve l'historique des builds et les résultats de la console, ainsi, en cas de problème, on peut facilement diagnostiquer le problème. Jenkins est extrêmement flexible et peut même voir ses fonctionnalités étendues à l'aide de différents plugins, ce qui permet de mettre en place des choses qui n'auraient pas été possible sans leur présence. La plupart du temps ces plugins sont développés par la communauté.

Jenkins peut prendre le rôle d'un utilisateur et exécuter certaines choses comme si c'était l'utilisateur qui l'avait exécuter. L'exécution se fait sur une machine choisie à l'avance, dans le cas spécifique du CEA il s'agit d'une machine virtuelle sur laquelle tous les builds Jenkins se trouvent. Un peu de la même manière que cron, Jenkins demande un "programme" c'est-à-dire qu'il doit savoir quand lancer le programme, pour se faire on utilise la même syntaxe que cron. Cependant, il existe une spécificité supplémentaire : en mettant H au début de l'expression qui donne le timing d'exécution à Jenkins, on assure que plusieurs jobs ne sont pas build en simultané, ce qui peut poser

problème dans certains cas.

5.1.2 Mise à jour du programme

La première chose que j'ai mis en place avec Jenkins est un job qui permet de faire une mise à jour du programme tous les jours. Pour se faire, nous avons créé un répertoire github qui accueille le programme avec deux branches, une branche main sur laquelle je mets toutes les petites modifications que j'effectue, et une branche Jenkins qui va être mise à jour lorsque suffisamment de modifications ont été apportées à la branche principale.

Du côté de Jenkins, il est possible d'installer un programme depuis github à l'aide de la commande "pip install" étant donné que Jenkins peut tout simplement lancer des lignes de commande dans une console windows. On peut aussi cloner un répertoire github sur la machine.

J'ai donc décidé de utiliser la commande pip, ce qui s'avère un peu plus pratique et plus simple d'utilisation. La mise à jour n'ayant pas besoin d'être faite très régulièrement, on se contente de la faire toutes les nuits du lundi au vendredi. Ainsi, j'ai utilisé l'expression suivante : H 1 * * 1-5. Ce qui permet de lancer un build toutes les nuits à 1h du matin du lundi au vendredi.

J'ai également ajouté une nouvelle étape d'automatisation, la mise à jour automatique du programme d'analyse et de réduction de données va tourner.

5.1.3 Lancement du programme

La seconde chose que je devais mettre en place était un job qui puisse build le programme lui-même toutes les heures environ. Avant de faire cela, il a juste fallu apporter quelques modifications au programme. J'ai ajouté un paramètre "-jenkins" qu'il est possible d'ajouter au moment de l'exécution de programme.

Si ce paramètre est :

1. true : le programme ne lance aucun gui et lance directement le processus de conversion des fichiers en hdf5 pendant 3500 secondes. De cette manière, un peut avant 1h d'exécution le programme s'arrête tout seul et peut être redémarré
2. false : le programme lance le gui et tout se passe comme si on utilisait l'exécutable du programme (cet exécutable est généré par le .toml nous en reparlerons)

Ainsi, toutes les heures (grâce à l'expression H * */1 * *) jenkins va build le programme en activant l'environnement Python et en lançant le module avec le paramètre -jenkins "true" ce qui converti les fichiers pendant un peu moins d'un heure avant de refaire la même chose après une heure. J'ai mis ce système en place car j'avais un peu peur que le programme soit trop gourmand en ressource s'il tournait en continu même avec les mesures que j'avais déjà pris. De cette manière, il redémarre et repart de zéro toutes les heures, assurant un bon fonctionnement.

De cette manière on a un programme qui se lance automatiquement et qui redémarre toutes les heures, ce qui permet un traitement continu des fichiers présents sur le serveur.

5.2 Le fichier .toml

Un fichier toml permet de facilement gérer un environnement Python en installant les dépendances nécessaires ainsi qu'en installant un programme à partir d'un répertoire github.

Le fichier .toml qui se trouve dans la racine du répertoire github du projet permet de pouvoir utiliser la commande pip install pour installer le module. Ce fichier toml permet beaucoup de choses mais voici les points clés :

1. Comme mentionné, il permet d'utiliser la commande pip install pour installer le programme dans un environnement Python déjà existant. Pour se faire il suffit tout simplement d'utiliser pip instal git+(lien de la branche github). Cela permet d'installer le programme dans l'environnement Python où pip a été utilisé.
2. En même temps que l'installation, il est possible de créer un exécutable qui va lancer le programme. Cet exécutable est créé en fournissant ainsi que la fonction Python nécessaire à l'exécuter. De cette manière, en lançant l'exécutable, on fait comme si on utilisait la fonction qui a été fournie.
3. Le toml gère aussi les dépendances. Pour que mon programme fonctionne correctement, il a besoin de plusieurs autres bibliothèques non natives à Python, à savoir numpy, matplotlib, h5py, Fabio et memory_profiler. En renseignant ces bibliothèques dans le fichier, il les installera en même temps que mon module afin d'assurer son bon fonctionnement.

6 Conclusion

Au cours de cette alternance nous avons exposé la mise en place d'une "pipeline" qui permet la conversion de données dans un nouveau format qui est plus à jour et permet un partage plus facilité d'informations entre membres de la communauté scientifique. Suite à cette conversion, il y a un traitement minime ou plus avancé des données qui est effectué selon les besoins de l'utilisateur. Le tout se faisant de manière 100% automatique et sans intervention humaine. L'objectif de cette alternance était de rendre l'analyse de données sortant d'un équipement accessible à tous. On peut dire que cet objectif est largement atteint.

De plus, nous avons aussi développé un aspect orienté "application" qui contient des interfaces graphiques afin de permettre à un utilisateur de manipuler ses données lui-même. Nous ne nous sommes cependant pas contentés de faire ceci, nous avons aussi permis à des utilisateurs plus expérimentés en matière de script Python de manipuler leurs données dans un script directement en développant un aspect plus orienté "bibliothèque" du projet. Il peut ainsi être directement importé sous forme de classe dans un fichier Python après avoir été installé à l'aide de la commande pip install.

Pour parvenir à faire tout ceci, j'ai eu le plaisir de mettre en œuvre une grande partie de mes connaissances acquises à l'université, ce qui m'a fourni un grand sentiment d'accomplissement et de satisfaction. J'ai aussi eu le plaisir et la joie d'apprendre énormément de nouvelles compétences grâce à des collègues de travail qui ont toujours été là lorsque j'étais dans le besoin. L'atmosphère bienveillante et accueillante de l'entreprise était aussi un énorme plus dans mon épanouissement dans le milieu du travail.

7 Annexes

7.1 Bibliothèques Python utilisées

Afin de pouvoir implémenter la conversion de edf à hdf5 ainsi que les différents processus applicables sur les mesures, il m'a été demandé d'utiliser Python. Python étant un langage très utilisé dans la communauté scientifique, il existe de nombreuses bibliothèques libres que nous pouvons utiliser. Certaines sont installées de base dans Python, mais la plupart sont à installer.

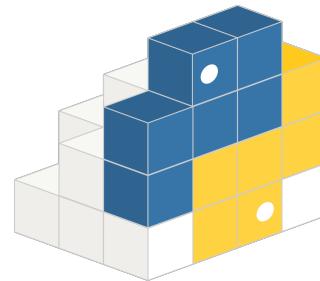


FIGURE 37 – Logo du site PyPI

Fabio [noauthor_Fabio_nodate]

C'est cette bibliothèque qui va nous permettre de lire les données qui sont sous edf. Elle permet d'extraire l'en-tête du fichier ainsi que les données brutes. l'en-tête va contenir des métadonnées, c'est-à-dire des informations sur la mesure qui sont plus ou moins pertinentes. C'est notamment dans l'en-tête qu'on trouvera des informations comme la distance échantillon-détecteur ou encore la longueur d'onde utilisée pour la mesure. Les données brutes contiennent bien sûr la mesure d'intensité effectuée.

H5py [5]

H5py est la bibliothèque qui va permettre d'ouvrir, et surtout d'écrire, des fichiers en hdf5. Grâce à elle on peut accéder à toutes les fonctionnalités du hdf5, on peut créer des groupes, dataset et leurs affecter des attributs à l'aide de méthodes qui ressemblent à celles utilisées sur les dictionnaires natifs à Python.

Tkinter [13]

C'est la principale bibliothèque Python utilisée pour faire des interfaces graphiques. Cette bibliothèque étant présente dans les versions récentes de Python par défaut, il était plus naturel de l'utiliser. Elle donne une interface assez brute et peu esthétique mais elle est tout de même très flexible et adaptable. En cas de besoin, on peut toujours ajouter un attribut ou un élément d'interface graphique "fait maison" ce qui est très pratique comme nous le verrons plus tard.

pyFAI [10]

La bibliothèque pyFAI (Python Fast Azimuthal Integration) est une bibliothèque beaucoup moins générale et très axée analyse de données scientifiques, plus précisément les données de diffusion aux petits angles issue de source X-ray / neutronique / électroniques. Cette bibliothèque se spécialise dans l'intégration azimuthale des données (bien qu'elle soit capable de bien plus) et utilise le calcul parallèle pour faire ces opérations très coûteuses en très peu de temps.

Une particularité de pyFAI est que cette bibliothèque fait l'usage de fichier appelé PONI qui veut dire Point Of Normal Incidence, c'est le point où le faisceau aurait frappé le détecteur en l'absence d'échantillon et de rotation. Ce fichier permet de faire la conversion de "l'espace pixel" à "l'espace

réel" en stockant 6 variables dans le système SI :

1. L : La distance entre l'échantillon et le PONI
2. $poni_1$: Offset en y du pixel $(0, 0)$ par rapport au PONI
3. $poni_2$: Offset en x du pixel $(0, 0)$ par rapport au PONI
4. rot_1 : rotation du détecteur par rapport à l'échantillon selon l'axe vertical
5. rot_2 : rotation du détecteur par rapport à l'échantillon selon l'axe horizontal
6. rot_3 : rotation du détecteur par rapport à l'échantillon selon l'axe porté par le faisceau

Les axes formant un repère direct et le pixel $(0, 0)$ se trouvant en bas à gauche de l'image.

A l'aide de ces 3 variables, on peut transformer la position $(\frac{d_H}{d_V})$ d'un pixel dans l'espace pixel (les positions étant des nombres entiers) en position $(\frac{p_1}{p_3})$ d'un pixel dans l'espace réel grâce à la formule matricielle suivante :

$$\begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = Rot \cdot \left(\begin{bmatrix} 0 & psize_x \\ psize_y & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} d_H \\ d_V \end{bmatrix} + \begin{bmatrix} -poni_1 \\ -poni_2 \\ L \end{bmatrix} \right) \quad (10)$$

On fait d'abord la conversion des coordonnées en pixel en coordonnées réelles sur le détecteur, puis on applique l'offset par rapport au PONI à chaque pixel et on finit par appliquer les 3 rotations (la matrice Rot dépend en fait de rot_1 , rot_2 et rot_3). De cette manière on obtient des coordonnées de chaque pixel dans l'espace par rapport à l'échantillon.

pyGIX

Cette deuxième bibliothèque scientifique se spécialise dans l'analyse de données de réflexion en incidence rasante (Python Grazing Incidence X-ray)

smi_analysis [11]

Une bibliothèque encore plus spécialisée qui permet de faire la plupart des traitements cités un peu plus haut en utilisant les fonctionnalités de pyFAI et pyGIX. C'est la bibliothèque que j'utilise principalement car elle m'a permis d'utiliser pyFAI et pyGIX sans avoir à me plonger dedans à 100% ce qui m'a fait gagner beaucoup de temps. De plus, elle permet d'éviter la création et la lecture d'un PONI car au lieu d'utiliser un PONI on injecte directement les paramètre que le PONI stocke dans une fonction dédiée.

C'est aussi cette bibliothèque qui est utilisé pour faire le traitement des données et qui existait déjà avant que j'arrive.

Références

- [1] Andrew J. ALLEN et al. “NIST Standard Reference Material 3600 : Absolute Intensity Calibration Standard for Small-Angle X-ray Scattering”. In : (6 fév. 2017). URL : <https://www.nist.gov/publications/nist-standard-reference-material-3600-absolute-intensity-calibration-standard-small>.
- [2] *Carte centres CEA*. URL : <https://www.cea.fr/Pages/le-cea/les-centres-cea.aspx> (visité le 12/06/2025).
- [3] *Cron*. URL : <https://cron-job.org/en/>.
- [4] *European Synchrotron Facility*. URL : <https://www.esrf.fr/Instrumentation/software/data-analysis/OurSoftware/SAXS/SaxsHeader-1>.
- [5] *H5Py*. URL : <https://www.h5py.org/>.
- [6] *HDF5*. URL : <https://www.hdfgroup.org/solutions/hdf5/>.
- [7] *NeXus*. URL : <https://www.nexusformat.org/>.
- [8] *NXcanSAS*. URL : <https://manual.nexusformat.org/classes/applications/NXcanSAS.html#nxcansas>.
- [9] *POSIX*. URL : <https://en.wikipedia.org/wiki/POSIX#:~:text=The%20Portable%20Operating%20System%20Interface,maintaining%20compatibility%20between%20operating%20systems..>
- [10] *PyFAI*. URL : <https://pyfai.readthedocs.io/en/stable/>.
- [11] *SMI Analysis*. URL : https://github.com/gfreychet/smi-analysis/tree/master/smi_analysis.
- [12] “The ESRF Data Format”. In : (20 mai 1994). URL : https://www.esrf.fr/computing/scientific/data_format/esrf_format.pdf.
- [13] *Tkinter*. URL : <https://docs.python.org/3/library/tkinter.html>.
- [14] *XCOM*. URL : <https://physics.nist.gov/PhysRefData/Xcom/html/xcom1.html>.
- [15] *Xenocs : XEUS*. URL : https://www.xenocs.com/saxs-products/saxs-equipment-xeuss/?gad_source=1&gad_campaignid=19651805743&gbraid=0AAAAAC-qWrVrJbyUq24KeH0ccu00_Jbvz&gclid=CjwKCAjwgb_CBhBMEiwA0p3o0KI7h63311DKLU7vspg5SLEcvHXtzkFWcpdqfKSSeMKW43108Cho2BwE.

Triée par ordre alphabétique