

Rapport de stage de 3ème année

Première période

Elève: Rolland Tristan

Entreprise : Laboratoire de Physique et Chimie des Nano-Objets

Tuteurs de stage : Mr Cayez Simon et Mr Ratel-Ramond Nicolas

Périodes de stage : 12/06/2024 au 06/09/2024 (puis 05/07/2024 au 25/08/2024 puis ??/??/2025)

Ecole: Institut National des Sciences Appliquées (INSA) de Toulouse



Rolland.T

Remerciements

Avant de commencer le développement de cette expérience professionnelle, il me paraît tout naturel de commencer par remercier les personnes qui m'ont permis d'effectuer ce travail ainsi que ceux qui m'ont permis d'en faire un moment agréable et profitable.

Je remercie donc Mr Viau, directeur du LPCNO, pour m'avoir fait confiance et ainsi m'offrir la possibilité de vivre cette expérience professionnelle, ainsi que Mr Cayez et Mr Ratel-Ramond, chercheurs au LPCNO, pour m'avoir formé avec rigueur et patience. Et, pour terminer, j'aimerais remercier l'ensemble des employés pour toute aide qu'ils ont pu me prodiguer au cours de cette expérience.

Introduction

Le Laboratoire de Physique et Chimie des Nano-Objets (LPCNO) situé à Toulouse, en France, est reconnu pour ses recherches innovantes dans le domaine des nanotechnologies. Durant mon stage au sein de ce laboratoire, j'ai eu l'opportunité de travailler sur un projet captivant portant sur la caractérisation de nano-objets par diffusion des rayons X assistée par machine learning.

L'objectif principal de ce stage était de développer une base de données sous la forme d'un Pandas DataFrame dont chaque ligne devait correspondre aux caractéristiques d'une particule ainsi qu'aux tableaux d'intensité et des normes des vecteurs (de diffusion) du réseau réciproque, résultant de la diffusion aux rayons X théorique de cette particule. La diversité des particules étant obtenue grâce à l'exploitation de toutes les combinaisons possibles de taille, forme et composition différentes. Cette base de données sera utilisée par la suite (lors de la période suivante de mon stage et du projet multidisciplinaire) pour un travail de machine

learning, qui permettra à terme la caractérisation automatique de particules à partir de leurs courbes de diffusion.

Ce rapport est structuré en plusieurs parties. La première section présentera le laboratoire LPCNO et son domaine de recherche. La deuxième partie détaillera les missions effectuées durant le stage, allant de la génération des particules et le calcul des intensités de diffusion à la création du Pandas DataFrame et l'exploitation de ces données. Enfin, la troisième section analysera les résultats obtenus et annoncera les prochaines étapes à réaliser durant les périodes suivantes.

Présentation de l'entreprise

Le Laboratoire de Physique et Chimie des Nano-Objets (LPCNO) est un centre de recherche de premier plan situé à Toulouse, en France. Fondé en 2001, le LPCNO résulte d'une collaboration entre l'Institut National des Sciences Appliquées (INSA) de Toulouse, le Centre National de la Recherche Scientifique (CNRS), et l'Université Toulouse III - Paul Sabatier. Ce laboratoire se consacre à l'étude et au développement de nanomatériaux innovants, en combinant les expertises en physique, chimie et science des matériaux. Le LPCNO est organisé en plusieurs équipes de recherche, chacune spécialisée dans un aspect particulier des nanotechnologies tel que les Nanoparticules et Nanostructures, la physique des Nano-Objets ou encore la chimie des Nanomatériaux.

En résumé, le LPCNO est une institution dynamique et innovante, dédiée à l'exploration et à l'exploitation des propriétés uniques des nano-objets pour développer des solutions technologiques de pointe.

Description du stage

Dans cette partie j'aborderai les différentes activités et missions que j'ai pu effectuées durant ce stage. Toutes les fonctions évoquées seront dans l'annexe 1.

Premier jours

Lors de mon arrivée, mes tuteurs de stage m'ont accueilli au LPCNO et m'ont fait une visite de celui-ci. J'ai pu observer les différentes branches du laboratoire (et donc mieux comprendre sa composition) ainsi que les salles de manipulation avec quelques explications des machines présentes.

Ensuite, mes tuteurs m'ont expliqué plus en détails le déroulement et l'objectif du stage notamment en me donnant certains documents à regarder. J'ai pu par exemple lire un ancien rapport de stage dont le sujet était avoisinant afin de mieux cerner le sujet de mon propre stage.

Pour terminer ce premier jour, mes tuteurs de stage et moi-même avons créé mon environnement de travail. En effet, tout le travail de programmation de mon stage a été réalisé en langage Python sur la plateforme Jupyter Lab. Il était donc nécessaire de me créer un environnement anaconda afin d'importer les fichiers et bibliothèques nécessaires et de me créer les dossiers où mes futurs codes seront enregistrés. Cette étape était pour moi une première, car je n'avais jamais utilisé Anaconda auparavant.

Milieu du stage

Une fois l'objectif du stage compris et mon environnement de travail près, j'ai pu commencer à coder en manipulant pour la première fois la future bibliothèque pyNanoMatBuilder de Mr Poteau Romuald. Cette futur bibliothèque à comme capacité de générer et d'afficher des particules divers et variées (des classes Johnson, Catalan, Platonic, Crystal, Archimedean et autre par exemple) tout en fournissant les caractéristiques de celles-ci (comme la taille, le nombre d'atomes, le moment d'inertie ou encore le fichier xyz associé à la particule, qui est un fichier contenant les coordonnées dans l'espace, des atomes contenus dans la particule, et serviront pour calculer la diffusion aux rayons X théorique avec DebyeCalculator par la suite).

Pour prendre en main cette future bibliothèque, j'ai réalisé la génération et l'affichage de mes premières particules au cas par cas (c'est-à-dire particule par particule) puis grâce à une boucle qui parcourait quelques caractéristiques que je voulais sur mes particules.

Ensuite, je me suis penché sur l'enregistrement de mes particules créées en codant un programme qui enregistrerait le fichier xyz de chaque particule générée dans la boucle. Une fois que j'avais les fichiers xyz des particules j'ai cherché à obtenir les courbes de diffusion théorique associées à chaque particules en utilisant la fonction DebyeCalculator. Néanmoins il avait un problème de lecture de fichier car mes fichiers xyz contenaient une colonne en trop. Il a donc été nécessaire d'utiliser la fonction "clean" (cf Annexe 1) afin de supprimer cette colonne en trop et que la fonction DebyeCalculator lise correctement les fichiers.

Maintenant que je pouvais avoir mes tableaux numpy des Intensités et des normes des vecteurs (de diffusion) du réseau réciproque, résultant de la diffusion aux

rayons X théorique de cette particule, je voulais également avoir les courbes de diffusion de mes particules. J'ai donc créé un programme qui récupérait ces tableaux numpy pour le convertir en listes et afficher les tracés avec les caractéristiques des particules.

A ce stade là du stage, j'avais les caractéristiques de mes particules et les courbes de diffusions associées. Il fallait donc commencer à enregistrer ces informations pour pouvoir construire une base de données. Pour cela mes tuteurs de stage m'ont demandé de garder ces informations sous le format d'un Pandas DataFrame car c'est un outil complet et facile d'utilisation, qui permettra par la suite de manipuler et modifier la base de données. Je me suis donc formé à l'utilisation de ce format que je n'avais jamais utilisé.

J'ai ensuite créé une fonction qui générerait des DataFrame de une ligne contenant les caractéristiques principales (éléments composant la particule, la distance inter-atomique la plus proche de la maille élémentaire de l'élément en question, la taille de la particule et la catégorie du solide) ainsi que les listes d'intensités et des normes des vecteurs de diffusion. J'ai ensuite automatisé la création de ces DataFrame d'une ligne via une fonction qui prenait en arguments des listes de caractéristiques de particules et qui générerait toutes les combinaisons possibles de particules avec ces caractéristiques. Cette fonction concaténait au fur et à mesure les DataFrame créés pour ne sortir qu'un seul DataFrame. J'ai effectué ce type de code pour chaque catégorie de solide pour obtenir des parties de la base de données (qui je concatènerai à la fin) afin de simplifier et de séquencer les tâches.

Fin du stage

Maintenant que nous avons nos premiers résultats de base de données et que le format nous paraissait bien, nous avons comme objectif d'être plus précis sur notre base de données , en y ajoutant des colonnes et donc des informations sur les particules, ainsi que d'en créer une beaucoup plus grande (avec plus de lignes). J'ai donc rajouter des colonnes supplémentaires tel que le rayon d'une sphère fictive englobant la particule (ce qui rajoutait des données sur la taille qui jusqu'à lors se limitait au nombre de couche dans la plupart des cas), le nombre d'atomes dans la particule, les q_{min} , q_{max} , q_{step} qui définissent la gammes des normes des vecteurs de diffusion qu'on utilise, le b_{iso} (qui modélise l'agitation thermique des atomes et donc rapproche nos particule de la réalité) ainsi que la polydispersité (que nous n'avons pas encore exploité pour le moment).

Nous avons alors notre format et notre forme de DataFrame finale. Il restait donc à générer un grand nombre de lignes en parcourant de très grandes boucles afin de générer une base de données conséquente. Pour cela, comme la capacité de calcul de l'ordinateur sur lequel je travaillais n'était pas suffisante, nous avons utilisé le GPU, qui est un autre ordinateur possédant une capacité de calcul plus importante. J'ai ainsi pu généré pour chaque classe de solide un DataFrame, et grâce à une fonction que j'ai créée permettant de concaténer les DataFrame entre eux et de supprimer les lignes en doubles, j'ai donc pu obtenir un DataFrame final de plus de 5000 lignes.

Enfin , pour vérifier nos résultats, j'ai tracé des diagrammes de dispersion de notre DataFrame final (permettant de voir quelle forme ou quelle taille apparaissait le plus) ainsi que créé une fonction qui prenait une ligne de manière aléatoire dans le DataFrame final et affichait la courbe de diffusion (permettant d'avoir un aperçu de la qualité de nos génération).

En complément

En complément de mon travail de programmation, j'étais chargé de faire un feed back de mon expérience sur la future bibliothèque pyNanoMatBuilder, en tant que premier utilisateur. Cela permettra de corriger certaines fonctionnalités pas encore totalement au point.

De plus, j'ai eu l'opportunité d'assister à des séminaires au sein du laboratoire. Ces séminaires sont organisés par des chercheurs externes ou non au laboratoire, et sont des présentations sur des sujets de recherches qui sont en cours, afin que les chercheurs du LPCNO apprennent de nouvelles choses sur des sujets de recherche similaires aux leurs.

Enfin, j'ai eu la chance d'observer des manipulations WAXS (Wide-angle X-ray scattering) et SAXS (Small-angle X-ray scattering) qui sont des techniques expérimentales permettant d'étudier la diffusion de nanoparticules. Cela m'a permis de voir plus concrètement en quoi une intelligence artificielle serait d'une grande aide pour la caractérisation des nanoparticules et de mettre plus de concret sur l'objectif de mon stage.

Analyse des résultat obtenus

Actuellement, notre base de données est incomplète dû à certains problèmes techniques venant de la bibliothèque. Nous n'avons pas toutes les formes souhaitées et pour certaines formes déjà générées, des 0 figure dans les colonnes spécifiant le rayon de la sphère imaginaire, ainsi que pour le nombre d'atomes. Ici encore ce sont des problèmes techniques liés à la bibliothèque en construction. Les problèmes ont été remontés et il restera à générer les formes manquantes ou incomplètes après résolution des problèmes.

Annexes:

Annexe 1 : Guide d'utilisation :

Dans cette annexe je détaillerai les différentes étapes à effectuer pour un utilisateur lambda qui veut générer sa propre base de données , et l'analyser.

PS: Afin de vous simplifier les choses je vous conseil de garder les mêmes noms de fichiers que moi, sinon vous aurez à modifier certains chemins ou nom de fichier dans les codes (je vous indiquerai les endroits à changer dans le cas échéant)

PS: A partir d'ici zoomer ce document jusqu'à 150% pour bien voir les codes.

Etape 1 : préambule

- Pour commencer, télécharger Anaconda et créer un nouvel environnement (choisir la version 3.1.1 de python). Ouvrir le terminal de anaconda (anaconda prompt) et se placer dans l'environnement créé avec la commande "conda activate"

ex si mon environnement s'appel Tristan : conda activate tristan

- Importer ensuite les bibliothèques suivantes (car nous les utiliseront dans les programmes) en utilisant la commande "pip install" :

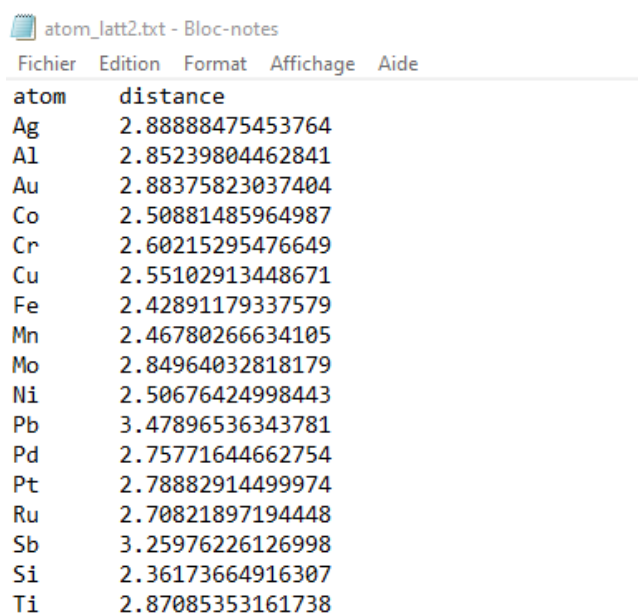
ase, torch, debyecalculator, matplotlib, pyNanoMatBuilder

ex : pip install ase

- Ouvrir jupyter avec la commande "jupyter lab" et créer le dossier ou il y aura tous vos codes.

Etape 2 : Importations de fichier

Afin d'utiliser les futurs programmes, il est nécessaire de créer un fichier en format txt comme ci dessous :



atom	distance
Ag	2.88888475453764
Al	2.85239804462841
Au	2.88375823037404
Co	2.50881485964987
Cr	2.60215295476649
Cu	2.55102913448671
Fe	2.42891179337579
Mn	2.46780266634105
Mo	2.84964032818179
Ni	2.50676424998443
Pb	3.47896536343781
Pd	2.75771644662754
Pt	2.78882914499974
Ru	2.70821897194448
Sb	3.25976226126998
Si	2.36173664916307
Ti	2.87085353161738

Ce fichier doit contenir uniquement deux colonnes, la première correspond aux atomes qui composeront les particules de la base de données, et la seconde correspond à la distance entre plus proche voisin dans la maille élémentaire.

Vous pouvez utiliser excel pour rentrer les valeurs puis convertir le fichier excel en fichier txt.

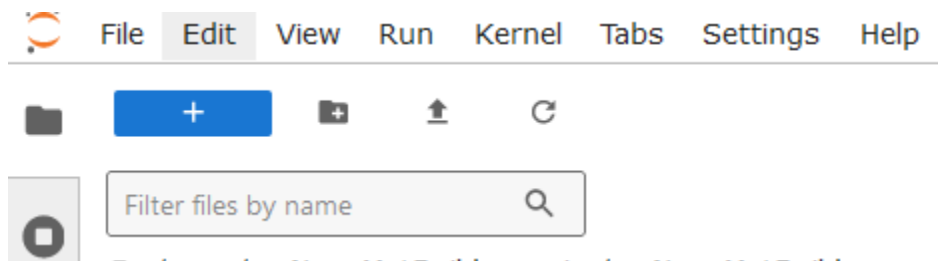
Remarque : vous pouvez également rajouter les éléments que vous voulez en vous aidant de la Crystallography Open Database: <http://www.crystallography.net/cod/> en regardant le paramètre de maille et la structure de l'élément que vous voulez ajouter.

Enfin, mettez ce fichier dans le dossier que vous avez créé en étape 1 (directement avec un copier coller sur l'interface de jupyter lab)

Etape 3 : création de dossiers

Avant de commencer les codes des programmes il est nécessaire de créer les dossiers où on enregistrera certains fichiers créés pendant l'exécution des programmes.

- Créer un nouveau dossier "stage" directement dans le dossier que vous avez créé étape 1. Cliquer sur l'icône à droite de l'icône "+" en bleu:



- Entrer dans le dossier "stage" et créer les 4 dossiers suivants :
 - Base_de_donnees
 - Base_de_donnees_finale
 - Base_de_donnees_sauvegarde

- xyz_tempo

Je vous expliquerai au fur et à mesure l'utilité de ces dossiers .

Etape 4 : les codes à effectuer

Maintenant que les bibliothèques et le fichier txt sont importées dans l'environnement et que nous sommes sur jupyter lab avec tous les dossiers créés au préalable , nous pouvons commencer à coder.

Nous allons coder plusieurs notebook, chacun correspondant à la génération d'un Pandas DataFrame pour une catégorie de solide. Nous aurons donc 5 notebook (correspondant aux classes Johnson, Catalan, Archimedean, Platonic et Other. La classe crystal n'est pour le moment pas fonctionnelle). La finalité de chaque notebook est de créer un fichier pk dont chaque ligne correspond aux caractéristiques d'une particule ainsi qu'aux tableaux d'intensité et des normes des vecteurs (de diffusion) du réseau réciproque, résultant de la diffusion aux rayons X théorique de cette particule.

Ce fichier pk sera enregistré dans Base_de_donnees et pourra ensuite être lu sous la format Pandas DataFrame. On créera également un 7ème notebook qui permettra de fusionner tous les fichiers pk du dossier Base_de_donnees pour n'avoir qu'un seul fichier pk (qu'on enregistrera dans Base_de_donnees_finale) et donc d'avoir qu'un seul Pandas DataFrame final, qui sera finalement notre base de données.

Nous allons voir ensemble le premier notebook pour expliquer les choses de manière plus concrète.

Notebook 1 : Catalan

- importation des bibliothèques :

```
#####
```

```
# Initialization #
```

```
#####
```

```
import os
```

```
import sys
```

```
print(os.getcwd())
```

```
cwd0 = './styles/'
```

```
sys.path.append(cwd0)
```

```
import visualID as vID
```

```
from visualID import fg, hl, bg
```

```
vID.init(cwd0)
```

```
import numpy as np
```

```
import ase
```

```
from ase.io import write
```

```
from ase.visualize import view
```

```
import pyNanoMatBuilder.utils as pNMBu
```

```
import importlib
```

```
import pandas as pd
```

```
from pyNanoMatBuilder import platonicNPs as pNP
```

```
import importlib
```

```
importlib.reload(pNP)
```

```
importlib.reload(pNMBu)
```

```
from pyNanoMatBuilder import catalanNPs as cNP
```

-
- création de la fonction Clean :

```
: def clean_xyz(filename):
    dtype = {'names': ('element', 'x', 'y', 'z'),
             'formats': ('U2', float, float, float)}
    element,x,y,z=np.loadtxt(filename,unpack=True,skiprows=2 , usecols=(0, 1, 2, 3), dtype=dtype)
    outfile=os.path.dirname(filename)+'/'+os.path.basename(filename).split('.')[0]+'xyz'
    nb_atoms=len(x)
    line2write=str(nb_atoms)+'\n'
    line2write+='\n'
    for i in range(nb_atoms):
        line2write+=str(element[i])+'\t %.8f'%x[i] + '\t %.8f'%y[i] + '\t %.8f'%z[i] +'\n'
    with open(outfile,'w') as file:
        file.write(line2write)
    return outfile
```

Voila le code pour copier coller, il faudra faire attention aux indentation:

```
def clean_xyz(filename):
```

```
    dtype = {'names': ('element', 'x', 'y', 'z'),
```

```
            'formats': ('U2', float, float, float)}
```

```
    element,x,y,z=np.loadtxt(filename,unpack=True,skiprows=2 , usecols=(0, 1, 2, 3), dtype=dtype)
```

```
    outfile=os.path.dirname(filename)+'/'+os.path.basename(filename).split('.')[0]+'xyz'
```

```
    nb_atoms=len(x)
```

```
    line2write=str(nb_atoms)+'\n'
```

```
    line2write+='\n'
```

```
    for i in range(nb_atoms):
```

```
        line2write+=str(element[i])+'\t %.8f'%x[i] + '\t %.8f'%y[i] + '\t %.8f'%z[i] +'\n'
```

```
    with open(outfile,'w') as file:
```

```
        file.write(line2write)
```

```
    return outfile
```

Cette fonction a pour but de supprimer une colonne en trop dans les fichiers xyz (qui sont des fichiers contenant les coordonnées dans l'espace, des atomes contenu dans la particule, et serviront pour calculer la diffusion aux rayons X théorique avec DebyeCalculator).

Avant la fonction clean :

```

1 240
2 Lattice="24.469499999999996 0.0 0.0 0.0 24.469499999999996 0.0 0.0
3 Au -10.19562500 -4.07825000 -2.03912500 0
4 Au -10.19562500 -2.03912500 -4.07825000 0
5 Au -10.19562500 -2.03912500 -0.00000000 0
6 Au -10.19562500 -0.00000000 -2.03912500 0
7 Au -10.19562500 -0.00000000 2.03912500 0
8 Au -10.19562500 2.03912500 -0.00000000 0
9 Au -8.15650000 -6.11737500 -2.03912500 0
10 Au -6.11737500 -8.15650000 -2.03912500 0
11 Au -6.11737500 -6.11737500 -4.07825000 0
12 Au -8.15650000 -6.11737500 2.03912500 0
13 Au -6.11737500 -8.15650000 2.03912500 0
14 Au -6.11737500 -6.11737500 -0.00000000 0
15 Au -6.11737500 -6.11737500 4.07825000 0
16 Au -8.15650000 -2.03912500 -6.11737500 0
17 Au -6.11737500 -4.07825000 -6.11737500 0
18 Au -6.11737500 -2.03912500 -8.15650000 0
19 Au -8.15650000 -4.07825000 -4.07825000 0
20 Au -8.15650000 -2.03912500 -2.03912500 0
21 Au -6.11737500 -4.07825000 -2.03912500 0
22 Au -6.11737500 -2.03912500 -4.07825000 0
23 Au -8.15650000 -4.07825000 -0.00000000 0
24 Au -8.15650000 -2.03912500 2.03912500 0
25 Au -6.11737500 -4.07825000 2.03912500 0
26 Au -6.11737500 -2.03912500 -0.00000000 0
27 Au -8.15650000 -4.07825000 4.07825000 0
28 Au -6.11737500 -4.07825000 6.11737500 0

```

Après la fonction clean :

```

1 240
2 Lattice="24.469499999999996 0.0 0.0 0.0 24.469499999999996
3 Au -10.19562500 -4.07825000 -2.03912500
4 Au -10.19562500 -2.03912500 -4.07825000
5 Au -10.19562500 -2.03912500 -0.00000000
6 Au -10.19562500 -0.00000000 -2.03912500
7 Au -10.19562500 -0.00000000 2.03912500
8 Au -10.19562500 2.03912500 -0.00000000
9 Au -8.15650000 -6.11737500 -2.03912500
10 Au -6.11737500 -8.15650000 -2.03912500
11 Au -6.11737500 -6.11737500 -4.07825000
12 Au -8.15650000 -6.11737500 2.03912500
13 Au -6.11737500 -8.15650000 2.03912500
14 Au -6.11737500 -6.11737500 -0.00000000
15 Au -6.11737500 -6.11737500 4.07825000
16 Au -8.15650000 -2.03912500 -6.11737500
17 Au -6.11737500 -4.07825000 -6.11737500
18 Au -6.11737500 -2.03912500 -8.15650000
19 Au -8.15650000 -4.07825000 -4.07825000
20 Au -8.15650000 -2.03912500 -2.03912500
21 Au -6.11737500 -4.07825000 -2.03912500
22 Au -6.11737500 -2.03912500 -4.07825000
23 Au -8.15650000 -4.07825000 -0.00000000
24 Au -8.15650000 -2.03912500 2.03912500
25 Au -6.11737500 -4.07825000 2.03912500
26 Au -6.11737500 -2.03912500 -0.00000000
27 Au -8.15650000 -4.07825000 4.07825000
28 Au -6.11737500 -4.07825000 6.11737500

```


-
- Création de la fonction de gestion des datas frame pour la catégorie catalan:

AJOUTER PHOTO CODE ICI

Voila le code pour copier coller, il faudra faire attention aux indentation:

```
def fonction_gestion_dataframe_shapes_catalan(element,forme,distance,couche,qmin=0.01,qmax=20,qstep=0.01,biso=0.01, endroit_fich_xyz='tempxyz.xyz'):
    """
    forme (rhombic_dodecahedron,dihedral_rhombic_dodecahedron) -- string
    element (ex Au )-- string
    distance = distance entre plus proche voisin -- float
    couche = liste de longueur 3 => [nbr couche voulu,0,0] qui donne le nombre de couche pour former le solide , equivalent de taille --list integer
    qmin -- list
    qmax -- list
    qstep -- list
    biso : pour simuler la vibration atomique et avoir des modèles plus réalistes -- list
    endroit_fich_xyz = l'endroit ou on veut enregistrer notre fichier xyz qui sera créé puis supprimé durant l'exécution de la fonction --string
    """

    #on importe les bibliothèques utiles

    from pyNanoMatBuilder import crystalNPs as cyNP

    import numpy as np

    import importlib
    from pyNanoMatBuilder import archimedeanNPs as aNP
    from pyNanoMatBuilder import platonicNPs as pNP
    from pyNanoMatBuilder import catalanNPs as cNP
    from pyNanoMatBuilder import utils as pNMBu

    # Entrer l'endroit ou on veut enregistrer notre fichier xyz
    fich_xyz=f"{endroit_fich_xyz}"

    # Création du DataFrame vide (que on viendra remplir dans la suite ) avec les colonnes spécifiées
    df = pd.DataFrame(columns=["Solid_class", "Element", "Shape", "Distance", "Size_parameters", "Eq_Radius", "n_Atoms", "polydispersity", "qmin", "qmax", "qstep", "b_iso", "q", "Intensities"])

    #Disjonction de cas selon les arguments (plus précisément la forme) que ont a rentés
    if forme != "rhombic_dodecahedron" and forme != "dihedral_rhombic_dodecahedron":
        print("La forme donnée n'est pas dans la classe des solides catalan")

    #cas pour le rhombic_dodecahedron
    if forme == "rhombic_dodecahedron":
```

```

#si c'est bien cette forme alors on creer un fichier xyz qui prends en compte les arguments de la fonction
bccrdd = cNP.bccrDD(f"{element}",distance,couche[0],aseView=False,thresholdCoreSurface = 0.,skipSymmetryAnalyzis = True,noOutput = True)
write(fich_xyz, bccrdd.NP)

#On garde le rayon tq toute la particule rentre dans une sphere de ce rayon

try :
    r=bccrdd.radiusCircumscribedSphere()
except :
    r=0

n_Atoms=bccrdd.nAtoms


#cas pour le dihedral_rhombic_dodecahedron
if forme == "dihedral_rhombic_dodecahedron" :

    #si c'est bien cette forme alors on creer un fichier xyz qui prends en compte les arguments de la fonction
    fccdrdd = cNP.fccdrDD(f"{element}",distance,couche[0],aseView=False,thresholdCoreSurface = 0.,skipSymmetryAnalyzis = True,noOutput = True)
    write(fich_xyz, fccdrdd.NP)

    #On garde le rayon tq toute la particule rentre dans une sphere de ce rayon

    try :
        r=fccdrdd.radiusCircumscribedSphere()
    except :
        r=0

    n_Atoms=fccdrdd.nAtoms


#On utilise la fonction Clean sur le fichier xyz que on vient de creer
xyz_file=fich_xyz
xyz_file=clean_xyz(xyz_file)


# On calcul I(q) avec la fonction debyecalculator et on rentre les tableau numpy générés dans les variable Q et I
from debyecalculator import DebyeCalculator

calc = DebyeCalculator(qmin=qmin,qmax=qmax,qstep=qstep,device='cuda',biso=biso)
Q, I = calc.iq(structure_source=xyz_file)


##### supression du fichier xyz pour garder seulement le nom
os.remove(fich_xyz)


# On transforme nos deux tableau numpy Q et I en tableau et on rentrer nos arguments de fonction dans des variables
Q=list(Q)

```

```

Intensite=list(I)
Distance =distance
Couche=couche
Element= f"{element}"
Forme=f"{forme}"
Type_solide= "Catalan"
Rayon=r
Qmin=qmin
Qmax=qmax
Qstep=qstep
Biso=biso
N_Atoms=n_Atoms
Polydispersity=0

# On ajout une ligne au DataFrame (avec ce qu'on voulait dedans comme données) en utilisant loc
df.loc[len(df)] = [Type_solide,Element, Forme, Distance, Couche,Rayon,N_Atoms,Polydispersity,Qmin,Qmax,Qstep,Biso, Q,Intensite]

return df

```

Explications : cette fonction va créer un fichier xyz (qui s'appellera fich_xyz) de la particule voulue (il faut renseigner en argument de la fonction les caractéristique de la particule, les arguments étant expliqués en commentaire dans la fonction) avec les fonctions “write”.

Ce fichier xyz créé va ensuite être mit sous la bonne forme avec la fonction “clean” et va servir d’argument à la fonction “DebyeCalculator” qui va sortir des tableau numpy (convertit ensuite en liste) des intensités et des normes des vecteurs (de diffusion) du réseau réciproque, résultant de la diffusion aux rayons X théorique de cette particule. Puisque le fichier xyz est maintenant inutile nous pouvons le supprimer avec la fonction “os.remove”.

Enfin, la liste des normes des vecteurs de diffusion, la liste des intensités et les paramètres de la particule entrer en arguments (ainsi que quelques paramètres récupérés dans la fonction comme le nombre d’atomes et le rayon de la sphère qui

contiendrait toute la particule) sont mit dans une ligne du DataFrame créé au début de la fonction). La fonction ressort ainsi ce DataFrame de une ligne.

- Création de la fonction qui génère les fichier pk pour la classe catalan

```
def generation_fichier_pk_catalan(forme,couche,nbr_elem,List_qmax,List_qmin,List_qstep,List_biso,lieu_fich_xyz,fichier_intermediaire,fichier_sortie):
```

```
# Création du data frame vide (df_concatenated) que on va remplir au fur et à mesure
```

```
df_concatenated = pd.DataFrame(columns=["Solid_class","Element", "Shape", "Distance","Size_parameters","Eq_Radius","n_Atoms","polydispersity","qmin","qmax","qstep","b_iso","q","Intensities"])
```

```
# On effectue donc les boucles imbriquées
```

```
for f in forme :
```

```
    for s in couche:
```

```
        for i in range (0,nbr_elem):
```

```
            for qmin in List_qmin:
```

```
                for qmax in List_qmax:
```

```
                    for qstep in List_qstep:
```

```
                        for biso in List_biso:
```

```
                            #try:
```

```
                                # Création du data frame d'une ligne avec la fonction de gestion des datas frame , et contenant les éléments de la boucle en cours
```

```
                                df_ligne = fonction_gestion_dataframe_shapes_catalan(liste_element[i],f,list_distance[i],s,qmin,qmax,qstep,biso,lieu_fich_xyz)
```

```
                                # Concaténation du data frame globale avec la data frame d'une ligne
```

```
                                df_concatenated = pd.concat([df_concatenated , df_ligne])
```

```
                            # except:
```

```
                            # pass
```

```
# Enregistrement forme par forme lors de grosses boucle pour garder certaines choses meme si cela crash
```

```
df_concatenated.to_pickle(fichier_intermediaire%f)
```

```
#Sauvegarder le DataFrame en CSV avec tabulation
```

```
df_concatenated.to_pickle(fichier_sortie)
```

On utilise des boucles afin de parcourir toutes les formes, les éléments, le nombre de couches, les b iso, les qmin, qmax et qstep souhaitées. On peut remarquer que à chaque élément il y a une seule distance associée, donc on ne parcourt que les éléments dans la boucle et on leurs associe leurs distance respective

On effectue ici la concaténation des DataFrame générés par la fonction de gestion des DataFrame pour ne donner qu'un seul DataFrame final qu' on enregistre au format pk.

- Création de liste qui serviront d'arguments pour generation_fichier_pk_catalan

```
# Création de la liste des couches (à changer en fonction de ce que l'on veut jusqu'a 9 apres c'est plus gros que 5nm de dia
size = [[4,3,2],[5,3,2],[6,3,2],[7,3,2],[8,3,2],[9,3,2]]

# Création de la liste des formes, avec liste_de_toutes_les_formes= ["trigonal_bipyramid", "pentagonal_bipyramid"]
forme= ["pentagonal_bipyramid"] #( à changer en fonction de ce que l'on veut (en prenant dans les elements de la liste_de_t

# Création des listes d'éléments (liste_element) et de distances (liste_distance
dtype=({"names":("atom","distance"),"formats":('U2',float)}) #juste pour avoir la liste des elements et des distances de dessous
liste_element,liste_distance=np.loadtxt("atom_latt2.txt",unpack=True,skiprows=1,usecols=[0,1],dtype=dtype)

# Création d'une variable nbr_elem pour parcourir la liste des éléments que de 0 à un certain nombre.
# Si on veut parcourir toute la liste des éléments, nbr_elem =len(liste_element)
nbr_elem=len(liste_element)

#création des listes qmin, qmax, qstep et biso
List_qmin=[0.01]
List_qmax=[20]
List_qstep=[0.01]
List_biso=[0.1,0.3,0.5,0.9]

# endroit ou on veut enregistrer notre fichier xyz qui sera créé puis supprimé durant l'exécution de la fonction
lieu_fich_xyz="stage/xyz_tempo/tempo.xyz"

#Nom du fichier qui enregistre forme par forme dans la boucle pour sauvegarder les datas frame petit a petit au cas ou en cas
fichier_intermediaire="stage/Base_de_donnees_sauvegarde/%s.pk"

# Nom du fichier csv de sortie
fichier_sortie="stage/Base_de_donnees/pentagonal_bipyramid_432a932_tt_elem_biso_01_03_05_09.pk"

print(len(forme)*nbr_elem*len(size)*len(List_qmin)*len(List_qmax)*len(List_qstep)*len(List_biso),"lignes attendues dans le c
```

```
# Création de la liste des couches (à changer en fonction de ce que l'on veut jusqu'a ? apres c'est plus gros que ?nm de diamètre)
```

```
couche = [[2,0,0],[3,0,0],[4,0,0],[5,0,0],[6,0,0],[7,0,0],[8,0,0],[9,0,0],[10,0,0]]
```

```
# Création de la liste des formes, avec liste_de_toutes_les_formes= ["rhombic_dodecahedron","dihedral_rhombic_dodecahedron"]
```

```
formes= ["rhombic_dodecahedron","dihedral_rhombic_dodecahedron"] #( à changer en fonction de ce que l'on veut (en prenant dans les elements de la
liste_de_toutes_les_formes)
```

```
# Création des listes d'éléments (liste_element) et de distances (liste_distance
```

```
dtype=({"names":("atom","distance"),"formats":('U2',float)}) #juste pour avoir la liste des elements et des distances de dessous
```

```
liste_element,liste_distance=np.loadtxt("atom_latt2.txt",unpack=True,skiprows=1,usecols=[0,1],dtype=dtype)
```

```
# Création d'une variable nbr_elem pour parcourir la liste des éléments que de 0 à un certain nombre.
```

```

# Si on veut parcourir toute la liste des éléments, nbr_elem =len(liste_element)
nbr_elem=6

#création des listes qmin, qmax, qstep et biso
List_qmin=[0.01]
List_qmax=[20]
List_qstep=[0.01]
List_biso=[0.1,0.3,0.5,0.9]

# endroit ou on veut enregistrer notre fichier xyz qui sera créé puis supprimé durant l'exécution de la fonction
lieu_fich_xyz="stage/xyz_tempo/tempo.xyz"

#Nom du fichier qui enregistre forme par forme dans le boucle pour sauvegarder les datas frame petit a petit au cas ou en cas de bug
fichier_intermediaire='stage/Base_de_donnees_sauvegarde/%s.pk'

# Nom du fichier csv de sortie
fichier_sortie='stage/Base_de_donnees/test.pk'

print(len(forme)*nbr_elem*len(couche)*len(List_qmin)*len(List_qmax)*len(List_qstep)*len(List_biso),"lignes attendues dans le data frame")

```

Remarque : attention à ne prendre en élément que ceux qui sont dans le fichier txt que vous avez créé en étape 2.

A l'endroit "*Création des listes d'éléments (liste_element) et de distances (liste_distance)*"

Le nom du fichier créé en étape 2 (atom_latt2.txt) doit être changé si vous n'avez pas mis le même que moi. Idem pour les variables "lieu_fich_xyz" qui est le dossier où l'on crée puis supprime les fichiers xyz de la fonction de gestion de DataFrame (il est normalement toujours vide) , "fichier_intermediaire" qui est le dossier où les DataFrame sont enregistrés formes par formes dans le cas où il y a un bug (cf programme ci-dessus, "fichier_sorti" qui est le dossier où l'on enregistre les DataFrame créés en fin de boucle (cf programme ci-dessus également).

- Appel de la fonction

```
%time generation_fichier_pk_catalan(forme,couche,nbr_elem,List_qmax,List_qmin,List_qstep  
,List_biso,lieu_fich_xyz,fichier_intermediaire,fichier_sortie)
```

Remarque : le %time sert à vous donner le temps que la fonction a mis pour s'exécuter et vous donne donc un ordre de grandeur pour les prochaines exécutions.

- Affichage du DataFrame créer

```
# On rentre la lecture du fichier dans la variable data
```

```
data = pd.read_pickle(fichier_sortie)
```

```
# On indexe la data frame comme il faut
```

```
data.index=np.arange(data.shape[0])
```

```
# On affiche le résultat final
```

```
data.shape
```

```
data
```

Ce code permet de lire le fichier pk que on a enregistré plus haut et permet de l'afficher.

Le code pour générer le fichier final de la classe Catalan est maintenant terminé. Nous pouvons analyser nos résultats.

- Analyse de nos résultats

On va creer un graph qui permet de forme la distribution des formes de notre data frame

```
import matplotlib.pyplot as plt
```

Créer la figure et les axes

```
fig, ax = plt.subplots()
```

Tracer l'histogramme

```
ax.hist(data['Shape'], bins=9,edgecolor='black')
```

Ajouter des labels aux axes

```
ax.set_xlabel('Shape')
```

```
ax.set_ylabel('Frequency')
```

Personnaliser les intervalles de l'axe des x

```
x_ticks = range(0, 3) # Plage de 0 à 3
```

```
plt.xticks(x_ticks)
```

Afficher la figure

```
plt.show()
```

```
#-----
```

On va creer un graph qui permet de forme la distribution des éléments de notre data frame

```
import matplotlib.pyplot as plt
```

Créer la figure et les axes

```
fig, ax = plt.subplots(figsize=(15,6))
```

Tracer l'histogramme

```
ax.hist(data['Element'], bins=100,edgecolor='black')
```

```
# Ajouter des labels aux axes

ax.set_xlabel('Element')

ax.set_ylabel('Frequency')


# Personnaliser les intervalles de l'axe des x

x_ticks = range(0, 19) # Plage de 0 à 50

plt.xticks(x_ticks)


# Afficher la figure

plt.show()
```

Ce code trace la distribution des éléments et des formes de notre DataFrame.

Les prochains notebook fonctionnant exactement comme celui expliqué, je mettrai uniquement les codes à copier avec des explications plus brèves.

Notebook 2 : archimedean

Importation des bibliothèques

```
#####
```

```
# Initialization #
```

```
#####
```

```
import os
```

```
import sys
```

```
print(os.getcwd())
```

```
cwd0 = './styles/'
```

```
sys.path.append(cwd0)
```

```
import visualID as vID
```

```
from visualID import fg, hl, bg
```

```
vID.init(cwd0)
```

```
import numpy as np
```

```
import ase
```

```
from ase.io import write
```

```
from ase.visualize import view
```

```
import pyNanoMatBuilder.utils as pNMBu
```

```
import importlib
```

```
import pandas as pd
```

```
from pyNanoMatBuilder import platonicNPs as pNP
```

```
import importlib

importlib.reload(pNP)

importlib.reload(pNMBu)


from pyNanoMatBuilder import archimedeanNPs as aNP
```

Définition des fonctions nécessaires à la création du data frame

Définition de la fonction clean

Cette fonction a pour but de supprimer la ligne en trop des fichiers xyz que on va créer

```
def clean_xyz(filename):

    dtype = {'names': ('element', 'x', 'y', 'z'),

             'formats': ('U2', float, float, float)}

    element,x,y,z=np.loadtxt(filename,unpack=True,skiprows=2 , usecols=(0, 1, 2, 3), dtype=dtype)

    outfile=os.path.dirname(filename)+'/'+os.path.basename(filename).split('.')[0]+'.'xyz'

    nb_atoms=len(x)

    line2write=str(nb_atoms)+'\n'

    line2write+='\n'

    for i in range(nb_atoms):

        line2write+=str(element[i])+'\t %.8f%x[i] + '\t %.8f%y[i] + '\t %.8f%z[i] +'\n'

    with open(outfile,'w') as file:

        file.write(line2write)

    return outfile
```

Définition de la fonction de gestion des data frame pour la catégorie des solides archimedean

Cette fonction a pour but de créer un data frame de une ligne contenant les arguments que on a donné

```
def fonction_gestion_dataframe_shapes_archimedean(element,forme,distance,couche,qmin=0.01,qmax=20,qstep=0.01,biso=0.01, endroit_fich_xyz='tempxyz.xyz'):
```

```
    """
```

```
    forme (cuboctahedron, truncated-tetrahedron, truncated-octahedron, truncated-cube bientot dispo mais pas encore) -- string
```

```
    element (ex Au )-- string
```

```
    distance = distance entre plus proche voisin -- float
```

```
    couche = liste de longueur 3 => [nbr couche voulu,0,0] qui donne le nombre de couche pour former le solide , equivalent de taille --list integer
```

```
    qmin -- list
```

```
    qmax -- list
```

```
    qstep -- list
```

```
    biso : pour simuler la vibration atomique et avoir des modèles plus réalistes -- list
```

```
    endroit_fich_xyz = l'endroit ou on veut enregistrer notre fichier xyz qui sera créé puis supprimé durant l'exécution de la fonction --string
```

```
    """
```

```
    #on importe les bibliothèques utiles
```

```
    from pyNanoMatBuilder import crystalNPs as cyNP
```

```
    import numpy as np
```

```
    import importlib
```

```
    from pyNanoMatBuilder import archimedeanNPs as aNP
```

```
    from pyNanoMatBuilder import platonicNPs as pNP
```

```
    from pyNanoMatBuilder import utils as pNMBu
```

```
    # Entrer l'endroit ou on veut enregistrer notre fichier xyz
```

```
    fich_xyz=f"{endroit_fich_xyz}"
```

```
    # Création du DataFrame vide (que on viendra remplir dans la suite ) avec les colonnes spécifiées
```

```
    df = pd.DataFrame(columns=["Solid_class", "Element", "Shape", "Distance", "Size_parameters", "Eq_Radius", "n_Atoms", "polydispersity", "qmin", "qmax", "qstep", "b_iso", "q", "Intensities"])
```

#Disjonction de cas selon les arguments (plus précisément la forme) qu'on a rentrés

if forme != "cuboctahedron" **and** forme != "truncated_tetrahedron" **and** forme != "truncated_octahedron" **and** forme != "truncated_cube" :

print("La forme donnée n'est pas dans la classe des solides archimedean")

#cas pour le cuboctahedron

if forme == "cuboctahedron" :

#si c'est bien cette forme alors on creer un fichier xyz qui prends en compte les arguments de la fonction

cubo = aNP.fccCubo(f"{element}",distance,couche[0],aseView=**False**,thresholdCoreSurface = 0.,skipSymmetryAnalyzis = **True**,noOutput = **True**)

write(fich_xyz, cubo.NP)

#On garde le rayon tq toute la particule rentre dans une sphere de ce rayon

try :

r=cubo.radiusCircumscribedSphere()

except :

r=0

n_Atoms=cubo.nAtoms

#cas pour le truncated_tetrahedron

if forme == "truncated_tetrahedron" :

#si c'est bien cette forme alors on creer un fichier xyz qui prends en compte les arguments de la fonction

trTd = aNP.fccTrTd(f"{element}",distance,couche[0],aseView=**False**,thresholdCoreSurface = 0.,skipSymmetryAnalyzis = **True**,noOutput = **True**)

write(fich_xyz, trTd.NP)

#On garde le rayon tq toute la particule rentre dans une sphere de ce rayon

try :

r=trTd.radiusCircumscribedSphere()

except :

r=0

```

n_Atoms=trTd.nAtoms

#cas pour le truncated_octahedron

if forme == "truncated_octahedron" :

    #si c'est bien cette forme alors on creer un fichier xyz qui prends en compte les arguments de la fonction

    trOh = aNP.fccTrOh(f"{element}",distance,couche[0],aseView=False,thresholdCoreSurface = 0.,skipSymmetryAnalyzis = True,noOutput = True)

    write(fich_xyz, trOh.NP)

    #On garde le rayon tq toute la particule rentre dans une sphere de ce rayon

    try :

        r=trOh.radiusCircumscribedSphere()

    except :

        r=0

    n_Atoms=trOh.nAtoms

#cas pour le truncated_cube, pour le moment pas dispo

if forme == "truncated_cube" :

    #si c'est bien cette forme alors on creer un fichier xyz qui prends en compte les arguments de la fonction

    trCube = aNP.fccTrCube(f"{element}",distance,couche[0],aseView=False,thresholdCoreSurface = 0.,skipSymmetryAnalyzis = True,noOutput = True)

    write(fich_xyz, trCube.NP)

    #On garde le rayon tq toute la particule rentre dans une sphere de ce rayon

    try :

        r=trCube.radiusCircumscribedSphere()

    except :

        r=0

    n_Atoms=trCube.nAtoms

#On utilise la fonction Clean sur le fichier xyz que on vient de creer

xyz_file=fich_xyz

xyz_file=clean_xyz(xyz_file)

```

On calcul I(q) avec la fonction debyecalculator et on rentre les tableau numpy générés dans les variable Q et I

from debyecalculator **import** DebyeCalculator

calc = DebyeCalculator(qmin=qmin,qmax=qmax,qstep=qstep,device='cuda',biso=biso)

Q, I = calc.iq(structure_source=xyz_file)

suppression du fichier xyz pour garder seulement le nom

os.remove(fich_xyz)

#On rentre nos valeurs dans des variables

Q=Q

Intensite=I

Distance =distance

Couche=couche

Element= f'{{element}}'

Forme=f'{{forme}}'

Type_solide= "Archimedean"

Rayon=r

Qmin=qmin

Qmax=qmax

Qstep=qstep

Biso=biso

N_Atoms=n_Atoms

Polydispersity=0

On ajout une ligne au DataFrame (avec ce qu'on voulait dedans comme données) en utilisant loc

df.loc[len(df)] = [Type_solide,Element, Forme, Distance, Couche,Rayon,N_Atoms,Polydispersity,Qmin,Qmax,Qstep,Biso, Q,Intensite]

return df

Définition de la fonction qui génère les fichier pk pour la classe Archimedean

On utilise des boucles afin de parcourir toutes les formes, les éléments et le nombre de couches souhaitées. On peut remarquer que à chaque élément il y a une seule distance associée, donc on ne parcourt que les éléments dans la boucle et on leur associe leurs distance respective

On effectue ici la concaténation des data frame générés par la fonction d'au-dessus

```
def generation_fichier_pk_archimedean(forme,couche,nbr_elem,List_qmax,List_qmin,List_qstep,List_biso,lieu_fich_xyz,fichier_intermediaire,fichier_sortie):
```

```
    # Création du data frame vide (df_concatenated) que on va remplir au fur et à mesure
```

```
    df_concatenated = pd.DataFrame(columns=["Solid_class", "Element", "Shape", "Distance", "Size_parameters", "Eq_Radius", "n_Atoms", "polydispersity", "qmin", "qmax", "qstep", "b_iso", "q", "Intensities"])
```

```
    # On effectue donc les boucles imbriquées
```

```
    for f in forme :
```

```
        for s in couche:
```

```
            for i in range (0,nbr_elem):
```

```
                for qmin in List_qmin:
```

```
                    for qmax in List_qmax:
```

```
                        for qstep in List_qstep:
```

```
                            for biso in List_biso:
```

```
                                try:
```

```
                                    # Création du data frame d'une ligne avec la fonction de gestion des datas frame , et contenant les éléments de la boucle en cours
```

```
                                    df_ligne = fonction_gestion_dataframe_shapes_archimedean(liste_element[i],f,list_distance[i],s,qmin,qmax,qstep,biso,lieu_fich_xyz)
```

```
                                    # Concaténation du data frame globale avec la data frame d'une ligne
```

```
                                    df_concatenated = pd.concat([df_concatenated , df_ligne])
```

```
                                except:
```

```
                                    pass
```

```
    # Enregistrement forme par forme lors de grosses boucle pour garder certaines choses meme si cela crash
```

```
    df_concatenated.to_pickle(fichier_intermediaire%f)
```

```
    #Sauvegarder le DataFrame en CSV avec tabulation
```

```
    df_concatenated.to_pickle(fichier_sortie)
```

Création du data frame

Variables à changer pour les utilisateurs

```
# Création de la liste des couches (à changer en fonction de ce que l'on veut jusqu'à ? apres c'est plus gros que ?nm de diamètre)

couche = [[3,0,0]]

#[3,0,0],[4,0,0],[5,0,0],[6,0,0],[7,0,0],[8,0,0],[9,0,0],[10,0,0],[11,0,0],[12,0,0],[13,0,0],[14,0,0],[15,0,0],[16,0,0],[17,0,0],[18,0,0],[19,0,0],[20,0,0],[21,0,0],[22,0,0]

# Création de la liste des formes, avec liste_de_toutes_les_formes= ["cuboctahedron", "truncated_tetrahedron", "truncated_octahedron", "truncated_cube"]

forme= [ "truncated_cube"] #( à changer en fonction de ce que l'on veut (en prenant dans les elements de la liste_de_toutes_les_formes)


# Création des listes d'éléments (liste_element) et de distances (liste_distance

dtype={"names":("atom","distance"), "formats":('U2',float)} #juste pour avoir la liste des elements et des distances de dessous

liste_element,liste_distance=np.loadtxt("atom_latt2.txt",unpack=True,skiprows=1,usecols=[0,1],dtype=dtype)

# Création d'une variable nbr_elem pour parcourir la liste des éléments que de 0 à un certain nombre.

# Si on veut parcourir toute la liste des éléments, nbr_elem =len(liste_element)

nbr_elem=3

#création des listes qmin, qmax, qstep et biso

List_qmin=[0.01]

List_qmax=[20]

List_qstep=[0.01]

List_biso=[0.1,0.3,0.5,0.9]

# endroit ou on veut enregistrer notre fichier xyz qui sera créé puis supprimé durant l'exécution de la fonction

lieu_fich_xyz="stage/xyz_tempo/tempo.xyz"

#Nom du fichier qui enregistre forme par forme dans le boucle pour sauvegarder les datas frame petit a petit au cas ou en cas de bug

fichier_intermediaire='stage/Base_de_donnees_sauvegarde/%s.pk'

# Nom du fichier csv de sortie

fichier_sortie='stage/Base_de_donnees/truncated_tetrahedron_2a22_tt_elem_biso_01_03_05_09.pk'

print(len(forme)*nbr_elem*len(couche)*len(List_qmin)*len(List_qmax)*len(List_qstep)*len(List_biso),"lignes attendues dans le data frame")
```

Appel de la fonction

```
%time generation_fichier_pk_archimedean(forme,couche,nbr_elem,List_qmax,List_qmin,List_qstep,List_biso,lieu_fich_xyz,fichier_intermediaire,fichier_sortie)
```

Analyse de nos résultats

```
# On va creer un graph qui permet de forme la distribution des formes de notre data frame
```

```
import matplotlib.pyplot as plt
```

```
# Créer la figure et les axes
```

```
fig, ax = plt.subplots()
```

```
# Tracer l'histogramme
```

```
ax.hist(data['Shape'], bins=9,edgecolor='black')
```

```
# Ajouter des labels aux axes
```

```
ax.set_xlabel('Shape')
```

```
ax.set_ylabel('Frequency')
```

```
# Personnaliser les intervalles de l'axe des x
```

```
x_ticks = range(0, 3) # Plage de 0 à 3
```

```
plt.xticks(x_ticks)
```

```
# Afficher la figure
```

```
plt.show()
```

```
#-----
```

```
# On va creer un graph qui permet de forme la distribution des éléments de notre data frame
```

```
import matplotlib.pyplot as plt
```

```
# Créer la figure et les axes

fig, ax = plt.subplots(figsize=(15,6))

# Tracer l'histogramme

ax.hist(data['Element'], bins=100,edgecolor='black')

# Ajouter des labels aux axes

ax.set_xlabel('Element')

ax.set_ylabel('Frequency')

# Personnaliser les intervalles de l'axe des x

x_ticks = range(0, 19) # Plage de 0 à 50

plt.xticks(x_ticks)

# Afficher la figure

plt.show()
```

Ce code trace la distribution des éléments et des formes de notre DataFrame.

Notebook 3 : johson

I) Importation des bibliothèques

In [2]:

```
#####
```

```
# Initialization #
```

```
#####
```

```
import os
```

```
import sys
```

```
print(os.getcwd())
```

```
cwd0 = './styles/'
```

```
sys.path.append(cwd0)
```

```
import visualID as vID
```

```
from visualID import fg, hl, bg
```

```
vID.init(cwd0)
```

```
import numpy as np
```

```
import ase
```

```
from ase.io import write
```

```
from ase.visualize import view
```

```
import pyNanoMatBuilder.utils as pNMBu
```

```
import importlib
```

```
import pandas as pd
```

```
from pyNanoMatBuilder import platonicNPs as pNP
```

```
import importlib
```

```
importlib.reload(pNP)
```

```
importlib.reload(pNMBu)
```

II) Définition des fonctions nécessaires à la création du data frame

Définition de la fonction clean

Cette fonction a pour but de supprimer la ligne en trop des fichiers xyz qu' on va créer

```
def clean_xyz(filename):
```

```
    dtype = {'names': ('element', 'x', 'y', 'z'),
```

```
              'formats': ('U2', float, float, float)}
```

```
    element,x,y,z=np.loadtxt(filename,unpack=True,skiprows=2 , usecols=(0, 1, 2, 3), dtype=dtype)
```

```
    outfile=os.path.dirname(filename)+'/'+os.path.basename(filename).split('.')[0]+'xyz'
```

```
    nb_atoms=len(x)
```

```
    line2write=str(nb_atoms)+'\n'
```

```
    line2write+='\n'
```

```
    for i in range(nb_atoms):
```

```
        line2write+=str(element[i])+'\t %.8f%x[i] + '\t %.8f%y[i] + '\t %.8f%z[i] +'\n'
```

```
    with open(outfile,'w') as file:
```

```
        file.write(line2write)
```

```
    return outfile
```

Définition de la fonction de gestion des datas frame pour la catégorie des solides johnson

Cette fonction a pour but de créer un data frame de une ligne contenant les arguments que on a donné

```
def fonction_gestion_dataframe_shapes_johnson(element,forme,distance,size,qmin=0.01,qmax=20,qstep=0.01,biso=0.01, endroit_fich_xyz='tempxyz.xyz'):

    """

    forme (trigonal_bipyramid, pentagonal_bipyramid ) -- string

    element (ex Au )-- string

    distance = distance entre plus proche voisin -- float

    size: liste de 3 éléments, le premier est le nombre d'atome par bords, le deuxième est la taille de la partie allongée et le troisième est la marque de troncature --list
    integer

    qmin -- list

    qmax -- list

    qstep -- list

    biso : pour simuler la vibration atomique et avoir des modèles plus réalistes -- list

    endroit_fich_xyz = l'endroit ou on veut enregistrer notre fichier xyz qui sera créé et supprimé durant l'exécution de la fonction --string

    """

    #on importe les bibliothèques utiles


    from pyNanoMatBuilder import crystalNPs as cyNP

    import numpy as np

    from pyNanoMatBuilder import platonicNPs as pNP

    from pyNanoMatBuilder import johnsonNPs as jNP

    from pyNanoMatBuilder import utils as pNMBu

    import importlib


    # Entrer l'endroit ou on veut enregistrer notre fichier xyz

    fich_xyz=f'{endroit_fich_xyz}'
```

Création du DataFrame vide (qu' on viendra remplir dans la suite) avec les colonnes spécifiées

```
df = pd.DataFrame(columns=["Solid_class", "Element", "Shape", "Distance", "Size_parameters", "Eq_Radius", "n_Atoms", "polydispersity", "qmin", "qmax", "qstep", "b_iso", "q", "Intensities"])
```

#Disjonction de cas selon les arguments (plus précisément la forme) qu'on a rentrés

```
if forme != "trigonal_bipyramid" and forme != "pentagonal_bipyramid" :
```

```
    print("La forme donnée n'est pas dans la classe des solides johnson")
```

#cas pour le trigonal_bipyramid

```
if forme == "trigonal_bipyramid" :
```

#si c'est bien cette forme alors on creer un fichier xyz qui prends en compte les arguments de la fonction

```
    tbp = jNP.fcctbp(f"{element}", distance, size[0], aseView=False, thresholdCoreSurface = 0., skipSymmetryAnalyzis = True, noOutput = True)
```

```
    write(fich_xyz, tbp.NP)
```

```
    try:
```

```
        r=tbp.radiusCircumscribedSphere()
```

```
    except:
```

```
        r=0 #pas encore de fonction radius pour cette classe, donc on rentre 0 par défaut
```

```
    n_Atoms=tbp.nAtoms
```

#cas pour le pentagonal_bipyramid

```
if forme == "pentagonal_bipyramid" :
```

#si c'est bien cette forme alors on creer un fichier xyz qui prends en compte les arguments de la fonction

```
    ppy1 = jNP.epbpyM(f"{element}", distance, size[0], size[1], size[2], aseView=False, thresholdCoreSurface = 0., skipSymmetryAnalyzis = True, noOutput = True)
```

```
    write(fich_xyz, ppy1.NP)
```

```
    try:
```

```

r=ppy1.radiusCircumscribedSphere()

except:

    r=0 #pas encore de fonction radius pour cette classe, donc on rentre 0 par défaut

n_Atoms=ppy1.nAtoms

#On utilise la fonction Clean sur le fichier xyz que on vient de creer

xyz_file=fich_xyz

xyz_file=clean_xyz(xyz_file)

# On calcul I(q) avec la fonction debyecalculator et on rentre les tableau numpy générés dans les variable Q et I

from debyecalculator import DebyeCalculator

calc = DebyeCalculator(qmin=qmin,qmax=qmax,qstep=qstep,device='cuda',biso=biso)

Q, I = calc.iq(structure_source=xyz_file)

##### suppression du fichier xyz pour garder seulement le nom

os.remove(fich_xyz)

# On transforme nos deux tableau numpy Q et I en tableau et on rentre nos arguments de fonction dans des variables

Q=Q

Intensite=I

Distance =distance

Size=size

Element= f"{element}"

Forme=f"{forme}"

Type_solide= "Johnson"

Rayon=r

Qmin=qmin

Qmax=qmax

Qstep=qstep

```

```
Biso=biso
```

```
N_Atoms=n_Atoms
```

```
Polydispersity=0
```

```
# On ajout une ligne au DataFrame (avec ce qu'on voulait dedans comme données) en utilisant loc
```

```
df.loc[len(df)] = [Type_solide,Element, Forme, Distance, Size,Rayon,N_Atoms,Polydispersity,Qmin,Qmax,Qstep,Biso, Q,Intensite]
```

```
return df
```

Définition de la fonction qui génère les fichier pk pour la classe johnson

On utilise des boucles afin de parcourir toutes les formes, les éléments et le nombre de couches souhaitées. On peut remarquer que à chaque élément il y a une seule distance associée, donc on ne parcourt que les éléments dans la boucle et on leurs associe leurs distance respective

On effectue ici la concaténation des data frame générés par la fonction d'haut dessus

```
def generation_fichier_pk_johnson(forme,size,nbr_elem,List_qmax,List_qmin,List_qstep,List_biso,lieu_fich_xyz,fichier_intermediaire,fichier_sortie):
```

```
# Création du data frame vide (df_concatenated) que on va remplir au fur et à mesure
```

```
df_concatenated = pd.DataFrame(columns=["Solid_class","Element", "Shape", "Distance","Size_parameters","Eq_Radius","n_Atoms","polydispersity","qmin","qmax","qstep","b_iso","q","Intensities"])
```

```
# On effectue donc les boucles imbriquées
```

```
for f in forme :
```

```
    for s in size:
```

```
        for i in range (0,nbr_elem):
```

```
            for qmin in List_qmin:
```

```
                for qmax in List_qmax:
```

```
                    for qstep in List_qstep:
```

```
                        for biso in List_biso:
```

```
# Création du data frame d'une ligne avec la fonction de gestion des datas frame , et contenant les éléments de la boucle en cours
```

```
df_ligne = fonction_gestion_dataframe_shapes_johnson(liste_element[i],f,list_distance[i],s,qmin,qmax,qstep,biso,lieu_fich_xyz)
```

```
# Concaténation du data frame globale avec la data frame d'une ligne
```

```
df_concatenated = pd.concat([df_concatenated , df_ligne])
```

```
# Enregistrement forme par forme lors de grosses boucle pour garder certaines choses meme si cela crash
```

```
df_concatenated.to_pickle(fichier_intermediaire%f)
```

```
#Sauvegarder le DataFrame en CSV avec tabulation
```

```
df_concatenated.to_pickle(fichier_sortie)
```

III) Création du data frame

Variables à changer pour les utilisateurs

```
# Création de la liste des couches (à changer en fonction de ce que l'on veut jusqu'à 9 après c'est plus gros que 5nm de diamètre)
```

```
size = [[4,3,2],[5,3,2],[6,3,2],[7,3,2],[8,3,2],[9,3,2]]
```

```
# Création de la liste des formes, avec liste_de_toutes_les_formes= ["trigonal_bipyramid", "pentagonal_bipyramid"]
```

```
forme= ["pentagonal_bipyramid"] #( à changer en fonction de ce que l'on veut (en prenant dans les elements de la liste_de_toutes_les_formes)
```

```
# Création des listes d'éléments (liste_element) et de distances (liste_distance)
```

```
dtype={"names":("atom","distance"),"formats":('U2',float)} #juste pour avoir la liste des elements et des distances de dessous
```

```
liste_element,liste_distance=np.loadtxt("atom_latt2.txt",unpack=True,skiprows=1,usecols=[0,1],dtype=dtype)
```

```
# Création d'une variable nbr_elem pour parcourir la liste des éléments que de 0 à un certain nombre.
```

```
# Si on veut parcourir toute la liste des éléments, nbr_elem =len(liste_element)
```

```
nbr_elem=len(liste_element)
```

```
#création des listes qmin, qmax, qstep et biso
```

```
List_qmin=[0.01]
```

```
List_qmax=[20]
```

```
List_qstep=[0.01]
```

```
List_biso=[0.1,0.3,0.5,0.9]
```

```
# endroit ou on veut enregistrer notre fichier xyz qui sera créé puis supprimé durant l'exécution de la fonction
```

```
lieu_fich_xyz="stage/xyz_tempo/tempo.xyz"
```

```
#Nom du fichier qui enregistre forme par forme dans le boucle pour sauvegarder les datas frame petit à petit au cas ou en cas de bug
```

```
fichier_intermediaire='stage/Base_de_donnees_sauvegarde/%s.pk'
```

```
# Nom du fichier csv de sortie
```

```
fichier_sortie='stage/Base_de_donnees/pentagonal_bipyramid_432a932_tt_elem_biso_01_03_05_09.pk'
```

```
print(len(forme)*nbr_elem*len(size)*len(List_qmin)*len(List_qmax)*len(List_qstep)*len(List_biso),"lignes attendues dans le data frame")
```

Appel de la fonction

```
%time generation_fichier_pk_johnson(forme,size,nbr_elem,List_qmax,List_qmin,List_qstep,List_biso,lieu_fich_xyz,fichier_intermediaire,fichier_sortie)
```

IV) Analyse de nos résultats

```
# On va créer un graph qui permet de voir la distribution des formes de notre data frame
```

```
import matplotlib.pyplot as plt
```

```
# Créer la figure et les axes
```

```
fig, ax = plt.subplots()
```

```
# Tracer l'histogramme
```

```
ax.hist(data["Shape"], bins=9, edgecolor='black')
```

```
# Ajouter des labels aux axes
```

```
ax.set_xlabel('Shape')
```

```
ax.set_ylabel("Frequency")
```

```
# Personnaliser les intervalles de l'axe des x
```

```
x_ticks = range(0, 3) # Plage de 0 à 3
```

```
plt.xticks(x_ticks)
```

```
# Afficher la figure
```

```
plt.show()
```

```
#-----
```

```
# On va creer un graph qui permet de forme la distribution des éléments de notre data frame
```

```
import matplotlib.pyplot as plt
```

```
# Créer la figure et les axes
```

```
fig, ax = plt.subplots(figsize=(15,6))
```

```
# Tracer l'histogramme
```

```
ax.hist(data["Element"], bins=100,edgecolor='black')
```

```
# Ajouter des labels aux axes
```

```
ax.set_xlabel("Element")
```

```
ax.set_ylabel("Frequency")
```

```
# Personnaliser les intervalles de l'axe des x
```

```
x_ticks = range(0, 19) # Plage de 0 à 50
```

```
plt.xticks(x_ticks)
```

```
# Afficher la figure
```

```
plt.show()
```

Ce code trace la distribution des éléments et des formes de notre DataFrame.

Notebook 4 : platonic

I) Importation des bibliothèques

```
#####
```

```
# Initialization #
```

```
#####
```

```
import os
```

```
import sys
```

```
print(os.getcwd())
```

```
cwd0 = './styles/'
```

```
sys.path.append(cwd0)
```

```
import visualID as vID
```

```
from visualID import fg, hl, bg
```

```
vID.init(cwd0)
```

```
import numpy as np
```

```
import ase
```

```
from ase.io import write
```

```
from ase.visualize import view
```

```
import pyNanoMatBuilder.utils as pNMBu
```

```
import importlib
```

```
import pandas as pd
```

```
from pyNanoMatBuilder import platonicNPs as pNP
```

```
import importlib
```

```
importlib.reload(pNP)
```

```
importlib.reload(pNMBu)
```

II) Définition des fonctions nécessaires à la création du data frame

Définition de la fonction clean

Cette fonction a pour but de supprimer la ligne en trop des fichiers xyz qu' on va créer

```
def clean_xyz(filename):  
  
    dtype = {'names': ('element', 'x', 'y', 'z'),  
             'formats': ('U2', float, float, float)}  
  
    element,x,y,z=np.loadtxt(filename,unpack=True,skiprows=2 , usecols=(0, 1, 2, 3), dtype=dtype)  
  
    outfile=os.path.dirname(filename)+'/'+os.path.basename(filename).split('.')[0]+''.xyz'  
  
    nb_atoms=len(x)  
  
    line2write=str(nb_atoms)+'\n'  
  
    line2write+='\n'  
  
    for i in range(nb_atoms):  
  
        line2write+=str(element[i])+'\t %.8f%x[i] + '\t %.8f%y[i] + '\t %.8f%z[i] + '\n'  
  
    with open(outfile,'w') as file:  
  
        file.write(line2write)  
  
    return outfile
```

Définition de la fonction de gestion des datas frame pour la classe des solides platoniques

Cette fonction a pour but de créer un DataFrame de une ligne contenant les arguments que on a donnés

```
def fonction_gestion_dataframe_shapes_platonic(element,forme,distance,couche,qmin=0.01,qmax=20,qstep=0.01,biso=0.01, endroit_fich_xyz='tempxyz.xyz'):
```

```
    """
```

```
    forme (icosahedron , octahedron, dodecahedron , tetrahedron, cubefcc, cubebcc ) -- string
```

```
    element (ex Au )-- string
```

```
    distance = distance entre plus proche voisin -- float
```

```
    couche = liste de longueur 3 => [nbr couche voulu,0,0] qui donne le nombre de couche pour former le solide , equivalent de taille --list integer
```

```
    qmin -- list
```

```
    qmax -- list
```

```
    qstep -- list
```

```
    biso : pour simuler la vibration atomique et avoir des modèles plus réalistes -- list
```

```
    endroit_fich_xyz = l'endroit ou on veut enregistrer notre fichier xyz qui sera créé et supprimé durant l'exécution de la fonction --string
```

```
    """
```

```
    #on importe les bibliothèques utiles
```

```
    from pyNanoMatBuilder import crystalNPs as cyNP
```

```
    import numpy as np
```

```
    from pyNanoMatBuilder import platonicNPs as pNP
```

```
    import importlib
```

```
    # endroit ou on veut enregistrer notre fichier xyz
```

```
    fich_xyz=f'{endroit_fich_xyz}'
```

```
    # Création du DataFrame vide (qu' on viendra remplir dans la suite ) avec les colonnes spécifiées
```

```
    df = pd.DataFrame(columns=["Solid_class", "Element", "Shape",  
"Distance", "Size_parameters", "Eq_Radius", "n_Atoms", "polydispersity", "qmin", "qmax", "qstep", "b_iso", "q", "Intensities"])
```

```
    #Disjonction de cas selon les arguments (plus précisément la forme) qu'on a rentés
```

```
    if forme != "icosahedron" and forme != "octahedron" and forme != "dodecahedron" and forme != "tetrahedron" and forme != "cubefcc" and forme != "cubebcc":
```

```

print("La forme donnée n'est pas dans la classe des solides platonic")

#cas pour l'icosahedron

if forme == "icosahedron" :

    #si c'est bien cette forme alors on creer un fichier xyz qui prends en compte les arguments de la fonction

    ico = pNP.regIco(f"{element}",distance,couche[0],aseView=False,thresholdCoreSurface = 0.,skipSymmetryAnalyzis = True,noOutput = True)

    write(fich_xyz, ico.NP)

    #On garde le rayon tq toute la particule rentre dans une sphère de ce rayon

    try :

        r=ico.radiusCircumscribedSphere()

    except :

        r=0

    n_Atoms=ico.nAtoms

#cas pour le tetrahedron

if forme == "tetrahedron" :

    #si c'est bien cette forme alors on creer un fichier xyz qui prends en compte les arguments de la fonction

    Td = pNP.regfccTd(f"{element}",distance,couche[0],aseView=False,thresholdCoreSurface = 0.,skipSymmetryAnalyzis = True,noOutput = True)

    write(fich_xyz, Td.NP)

    #On garde le rayon tq toute la particule rentre dans une sphère de ce rayon

    try :

        r=Td.radiusCircumscribedSphere()

    except :

        r=0

    n_Atoms=Td.nAtoms

#cas pour le dodecahedron

if forme == "dodecahedron" :

    #si c'est bien cette forme alors on creer un fichier xyz qui prends en compte les arguments de la fonction

```

```

rdd = pNP.regDD(f"{element}",distance,couche[0],aseView=False,thresholdCoreSurface = 0.,skipSymmetryAnalyzis = True,noOutput = True)

write(fich_xyz, rdd.NP)

#On garde le rayon tq toute la particule rentre dans une sphere de ce rayon

try :

    r=rdd.radiusCircumscribedSphere()

except :

    r=0

    n_Atoms=rdd.nAtoms

#cas pour l'octahedron

if forme == "octahedron" :

    #si c'est bien cette forme alors on creer un fichier xyz qui prends en compte les arguments de la fonction

    fccOh = pNP.regfccOh(f"{element}",distance,couche[0],aseView=False,thresholdCoreSurface = 0.,skipSymmetryAnalyzis = True,noOutput = True)

    write(fich_xyz,fccOh.NP)

    #On garde le rayon tq toute la particule rentre dans une sphere de ce rayon

    try:

        r=fccOh.radiusCircumscribedSphere()

    except:

        r=0

        n_Atoms=fccOh.nAtoms

#cas pour le cube fcc

if forme == "cubefcc":

    #si c'est bien cette forme alors on creer un fichier xyz qui prends en compte les arguments de la fonction

    fcc = pNP.cube('fcc',f"{element}",distance,couche[0],aseView=False,thresholdCoreSurface = 0.,skipSymmetryAnalyzis = True,noOutput = True)

    write(fich_xyz, fcc.NP)

    #On garde le rayon tq toute la particule rentre dans une sphere de ce rayon

    try:

        r=fcc.radiusCircumscribedSphere()

    except:

        r=0

```

```

n_Atoms=fcc.nAtoms

#cas pour le cube bcc

if forme == "cubebcc":

    #si c'est bien cette forme alors on creer un fichier xyz qui prends en compte les arguments de la fonction

    bcc = pNP.cube('bcc',f'{element}',distance,couche[0],aseView=False,thresholdCoreSurface = 0.,skipSymmetryAnalyzis = True,noOutput = True)

    write(fich_xyz, bcc.NP)

    #On garde le rayon tq toute la particule rentre dans une sphere de ce rayon

    try:

        r=bcc.radiusCircumscribedSphere()

    except:

        r=0

    n_Atoms=bcc.nAtoms

    #On utilise la fonction Clean sur le fichier xyz que on vient de creer

    xyz_file=fich_xyz

    xyz_file=clean_xyz(xyz_file)

    # On calcul I(q) avec la fonction debyecalculator et on rentre les tableau numpy générés dans les variable Q et I

    from debyecalculator import DebyeCalculator

    calc = DebyeCalculator(qmin=qmin,qmax=qmax,qstep=qstep,device='cuda',biso=biso)

    Q, I = calc.iq(structure_source=xyz_file)

    ##### supression du fichier xyz pour garder seulement le nom

    os.remove(fich_xyz)

    # On transforme nos deux tableau numpy Q et I en tableau et on rentrer nos arguments de fonction dans des variables

    Q=Q

    Intensite=I

    Distance =distance

    Couche=couche

    Element= f'{element}'

    Forme=f'{forme}'

```

```

Type_solide= "Platonic"

Rayon=r

Qmin=qmin

Qmax=qmax

Qstep=qstep

Biso=biso

N_Atoms=n_Atoms

Polydispersity=0


# On ajout une ligne au DataFrame (avec ce qu'on voulait dedans comme données) en utilisant loc

df.loc[len(df)] = [Type_solide,Element, Forme, Distance, Couche,Rayon,N_Atoms,Polydispersity,Qmin,Qmax,Qstep,Biso, Q,Intensite]


return df

```

Définition de la fonction qui génère les fichier pk pour la classe Platonic

On utilise des boucles afin de parcourir toutes les formes, les éléments et le nombre de couches souhaitées. On peut remarquer que à chaque élément il y a une seule distance associée, donc on ne parcourt que les éléments dans la boucle et on leurs associes leurs distance respective

On effectue ici la concaténation des data frame générés par la fonction d'haut dessus

```
def generation_fichier_pk_platonic(forme,couche,nbr_elem,List_qmax,List_qmin,List_qstep,List_biso,lieu_fich_xyz,fichier_intermediaire,fichier_sortie):
```

```

# Création du data frame vide (df_concatenated) que on va remplir au fur et à mesure

```

```

df_concatenated = pd.DataFrame(columns=["Solid_class", "Element", "Shape",
"Distance", "Size_parameters", "Eq_Radius", "n_Atoms", "polydispersity", "qmin", "qmax", "qstep", "b_iso", "q", "Intensities"])

```

```

# On effectue donc les boucles imbriquées

```

```

for f in forme :

    for s in couche:

        for i in range (0,nbr_elem):

            for qmin in List_qmin:

                for qmax in List_qmax:

                    for qstep in List_qstep:

                        for biso in List_biso:

                            # Création du data frame d'une ligne avec la fonction de gestion des datas frame , et contenant les éléments de la boucle en cours

                            df_ligne = fonction_gestion_dataframe_shapes_platonic(liste_element[i],f,liste_distance[i],s,qmin,qmax,qstep,biso,lieu_fich_xyz)

                            # Concaténation du data frame globale avec la data frame d'une ligne

                            df_concatenated = pd.concat([df_concatenated , df_ligne])

                        # Enregistrement forme par forme lors de grosses boucle pour garder certaines choses meme si cela crash

                        df_concatenated.to_pickle(fichier_intermediaire%f)

                    #Sauvegarder le DataFrame en CSV avec tabulation

                    df_concatenated.to_pickle(fichier_sortie)

```

III) Création du data frame

Variables à changer pour les utilisateurs

```

# Création de la liste des couches (à changer en fonction de ce que l'on veut jusqu'à ? après c'est plus gros que ?nm de diamètre)

couche = [[2,0,0],[3,0,0],[4,0,0],[5,0,0],[6,0,0],[7,0,0],[8,0,0],[9,0,0],[10,0,0]]

#[3,0,0],[4,0,0],[5,0,0],[6,0,0],[7,0,0],[8,0,0],[9,0,0],[10,0,0]

# Création de la liste des formes, avec liste_de_toutes_les_formes= ["icosahedron", "octahedron", "dodecahedron", "tetrahedron", "cubefcc", "cubebcc"]

forme= ["cubebcc"] #( à changer en fonction de ce que l'on veut (en prenant dans les elements de la liste_de_toutes_les_formes)

# Création des listes d'éléments (liste_element) et de distances (liste_distance)

```

```
dtype={"names":("atom","distance"),"formats":('U2',float)} #juste pour avoir la liste des elements et des distances de dessous
```

```
liste_element,liste_distance=np.loadtxt("atom_latt2.txt",unpack=True,skiprows=1,usecols=[0,1],dtype=dtype)
```

```
# Création d'une variable nbr_elem pour parcourir la liste des éléments que de 0 à un certain nombre.
```

```
# Si on veut parcourir toute la liste des éléments, nbr_elem =len(liste_element)
```

```
nbr_elem=len(liste_element)
```

```
#création des listes qmin, qmax, qstep et biso
```

```
List_qmin=[0.01]
```

```
List_qmax=[20]
```

```
List_qstep=[0.01]
```

```
List_biso=[0.1,0.3,0.5,0.9]
```

```
# endroit ou on veut enregistrer notre fichier xyz qui sera créé puis supprimé durant l'exécution de la fonction
```

```
lieu_fich_xyz="stage/xyz_tempo/tempo.xyz"
```

```
#Nom du fichier qui enregistre forme par forme dans le boucle pour sauvegarder les datas frame petit a petit au cas ou en cas de bug
```

```
fichier_intermediaire='stage/Base_de_donnees_sauvegarde/%s.pk'
```

```
# Nom du fichier csv de sortie
```

```
fichier_sortie='stage/Base_de_donnees/asup.pk'
```

```
print(len(forme)*nbr_elem*len(couche)*len(List_qmin)*len(List_qmax)*len(List_qstep)*len(List_biso),"lignes attendues dans le data frame")
```

Appel de la fonction

```
%time generation_fichier_pk_platonic(forme,couche,nbr_elem,List_qmax,List_qmin,List_qstep,List_biso,lieu_fich_xyz,fichier_intermediaire,fichier_sortie)
```

IV) Analyse de nos résultats

```
# On va créer un graph qui permet de forme la distribution des formes de notre data frame
```

```
import matplotlib.pyplot as plt
```

```
# Créer la figure et les axes
```

```
fig, ax = plt.subplots()
```

```
# Tracer l'histogramme
```

```
ax.hist(data['Shape'], bins=9,edgecolor='black')
```

```
# Ajouter des labels aux axes
```

```
ax.set_xlabel('Shape')
```

```
ax.set_ylabel('Frequency')
```

```
# Personnaliser les intervalles de l'axe des x
```

```
x_ticks = range(0, 3) # Plage de 0 à 3
```

```
plt.xticks(x_ticks)
```

```
# Afficher la figure
```

```
plt.show()
```

```
#-----
```

```
# On va creer un graph qui permet de forme la distribution des éléments de notre data frame
```

```
import matplotlib.pyplot as plt
```

```
# Créer la figure et les axes
```

```
fig, ax = plt.subplots(figsize=(15,6))

# Tracer l'histogramme

ax.hist(data['Element'], bins=100,edgecolor='black')

# Ajouter des labels aux axes

ax.set_xlabel('Element')

ax.set_ylabel('Frequency')

# Personnaliser les intervalles de l'axe des x

x_ticks = range(0, 19) # Plage de 0 à 50

plt.xticks(x_ticks)

# Afficher la figure

plt.show()
```

Ce code trace la distribution des éléments et des formes de notre DataFrame.

Notebook 5: Other

I) Importation des bibliothèques

```
#####
```

```
# Initialization #
```

```
#####
```

```
import os
```

```
import sys
```

```
print(os.getcwd())
```

```
cwd0 = './styles/'
```

```
sys.path.append(cwd0)
```

```
import visualID as vID
```

```
from visualID import fg, hl, bg
```

```
vID.init(cwd0)
```

```
import numpy as np
```

```
import ase
```

```
from ase.io import write
```

```
from ase.visualize import view
```

```
import pyNanoMatBuilder.utils as pNMBu
```

```
import importlib
```

```
import pandas as pd
```

```
from pyNanoMatBuilder import platonicNPs as pNP
```

```
import importlib
```

```
importlib.reload(pNP)

importlib.reload(pNMBu)
```

II) Définition des fonctions nécessaires à la creation du data frame

Définition de la fonction clean

Cette fonction a pour but de supprimer la ligne en trop des fichiers xyz qu' on va créer

```
def clean_xyz(filename):

    dtype = {'names': ('element', 'x', 'y', 'z'),
             'formats': ('U2', float, float, float)}

    element,x,y,z=np.loadtxt(filename,unpack=True,skiprows=2 , usecols=(0, 1, 2, 3), dtype=dtype)

    outfile=os.path.dirname(filename)+'/'+os.path.basename(filename).split('.')[0]+''.xyz'

    nb_atoms=len(x)

    line2write=str(nb_atoms)+'\n'

    line2write+='\n'

    for i in range(nb_atoms):

        line2write+=str(element[i])+'\t %.8f%%x[i] + '\t %.8f%%y[i] + '\t %.8f%%z[i] '+'\n'

    with open(outfile,'w') as file:

        file.write(line2write)

    return outfile
```

Définition de la fonction de gestion des datas frame pour la catégorie des solides autres

Cette fonction a pour but de créer un data frame de une ligne contenant les arguments que on a donnés

```
def fonction_gestion_dataframe_shapes_other(element, forme, distance,nombre_couches,qmin=0.01,qmax=20,qstep=0.01,biso=0.01,endroit_fich_xyz='tempxyz.xyz'):

    """

    forme (trigonal_platelet ) -- string

    element (ex Au )-- string

    distance = distance entre plus proche voisin -- float

    nombre_couches: liste de trois éléments, le premier est le nombre de couche, le second est la longueur des bords et le dernier est nul --list integer

    qmin -- list

    qmax -- list

    qstep -- list
```

```

biso : pour simuler la vibration atomique et avoir des modèles plus réalistes -- list

endroit_fich_xyz = l'endroit ou on veut enregistrer notre fichier xyz qui sera créé puis supprimé durant l'exécution de la fonction --string
"""

#on importe les bibliothèques utiles

from pyNanoMatBuilder import crystalNPs as cyNP

import numpy as np

from pyNanoMatBuilder import platonicNPs as pNP

from pyNanoMatBuilder import johnsonNPs as jNP

import importlib

import sys

import pyNanoMatBuilder.utils as pNMBu

from pyNanoMatBuilder import otherNPs as oNP

import importlib

#endroit ou on veut enregistrer notre fichier xyz

fich_xyz=f"{endroit_fich_xyz}"

# Création du DataFrame vide (que on viendra remplir dans la suite ) avec les colonnes spécifiées

df = pd.DataFrame(columns=["Solid_class", "Element", "Shape",
"Distance", "Size_parameters", "Eq_Radius", "n_Atoms", "polydispersity", "qmin", "qmax", "qstep", "b_iso", "q", "Intensities"])

#Disjonction de cas selon les arguments (plus précisément la forme) que ont a rentés

if forme != "trigonal_platelet" :

    print("La forme donnée n'est pas dans la classe des solides others")

#cas pour le trigonal_platelet

if forme == "trigonal_platelet" :

    #si c'est bien cette forme alors on creer un fichier xyz qui prends en compte les arguments de la fonction

```

```

tpt = oNP.fcctpt(f"{element}",distance,nombre_couches[0],nombre_couches[1],aseView=False,thresholdCoreSurface = 0.,skipSymmetryAnalysis = True,noOutput =
True)

write(fich_xyz, tpt.NP)

try :

    r=tpt.radiusCircumscribedSphere()

except :

    r=0 #pas encore de fonction radius pour cette classe, donc on rentre 0 par défaut

n_Atoms=tpt.nAtoms


#On utilise la fonction Clean sur le fichier xyz que on vient de creer

xyz_file=fich_xyz

xyz_file=clean_xyz(xyz_file)

# On calcul I(q) avec la fonction debyecalculator et on rentre les tableau numpy générés dans les variable Q et I

from debyecalculator import DebyeCalculator

calc = DebyeCalculator(qmin=0.01,qmax=20,qstep=0.01,device='cuda',biso=0.01)

Q, I = calc.iq(structure_source=xyz_file)

##### supression du fichier xyz pour garder seulement le nom

os.remove(fich_xyz)

# on rentrer nos arguments de fonction dans des variables

Q=Q

Intensite=I

Distance =distance

Nombre_couche=nombre_couches

Element= f"{element}"

Forme=f"{forme}"

Type_solide= "Other"

Rayon=r

Qmin=qmin

Qmax=qmax

Qstep=qstep

```

```
Biso=biso
```

```
N_Atoms=n_Atoms
```

```
Polydispersity=0
```

```
# On ajoute une ligne au DataFrame (avec ce qu'on voulait dedans comme données) en utilisant loc
```

```
df.loc[len(df)] = [Type_solide,Element, Forme, Distance, Nombre_couche,Polydispersity,Rayon,N_Atoms,Qmin,Qmax,Qstep,Biso, Q,Intensite]
```

```
return df
```

Définition de la fonction qui creer un fichier pk avec les paramètres qu'on veut

On utilise des boucles afin de parcourir toutes les formes, les éléments et le nombre de couches et d'atomes par bords souhaitées. On va donc créer plusieurs data frame d'une ligne et on va les concatener au fur et à mesure pour obtenir un seul data frame global

On peut remarquer que à chaque élément il y a une seule distance associée, donc on ne parcourt que les éléments dans la boucle et on leurs associes leurs distance respective

```
def generation_fichier_pk_other(forme,couche,nbr_elem,List_qmax,List_qmin,List_qstep,List_biso,lieu_fich_xyz,fichier_intermediaire,fichier_sortie):
```

```
# Création du data frame vide (df_concatenated) que on va remplir au fur et à mesure
```

```
df_concatenated = pd.DataFrame(columns=["Solid_class","Element", "Shape",  
"Distance","Size_parameters","Eq_Radius","n_Atoms","polydispersity","qmin","qmax","qstep","b_iso","q","Intensities"])
```

```
# On effectue donc les boucles imbriquées
```

```
for f in forme :
```

```
    for s in couche:
```

```
        for i in range (0,nbr_elem):
```

```
for qmin in List_qmin:
```

```
    for qmax in List_qmax:
```

```
        for qstep in List_qstep:
```

```
            for biso in List_biso:
```

```
                try :
```

```
                    # Création du data frame d'une ligne avec la fonction de gestion des datas frame , et contenant les éléments de la boucle en cours
```

```
                    df_ligne = fonction_gestion_dataframe_shapes_other(liste_element[i],f,liste_distance[i],s,qmin,qmax,qstep,biso,lieu_fich_xyz)
```

```
                    # Concaténation du data frame globale avec la data frame d'une ligne
```

```
                    df_concatenated = pd.concat([df_concatenated , df_ligne])
```

```
                except :
```

```
                    pass
```

```
            # Enregistrement forme par forme lors de grosses boucle pour garder certaines choses meme si cela crash
```

```
            df_concatenated.to_pickle(fichier_intermediaire%f)
```

```
        #Sauvegarder le DataFrame en CSV avec tabulation
```

```
        df_concatenated.to_pickle(fichier_sortie)
```

III) Création du data frame

Variables à changer pour les utilisateurs

```
# Création de la liste des couches avec un premier la largeur d'un côté et le deuxième le nombre de couche (à changer en fonction de ce que l'on veut jusqu'à ? apres c'est plus  
#gros que ?nm de diamètre)
```

```
couche = [[2,2,0],[3,2,0],[4,2,0],[5,2,0],[6,2,0],[7,2,0],[8,2,0],[9,2,0],[10,2,0],[2,3,0],[3,3,0],[4,3,0],[5,3,0],[6,3,0],[7,3,0],[8,3,0],[9,3,0],[10,3,0]]
```

```
# Création de la liste des formes, avec liste_de_toutes_les_formes= ["trigonal_platelet"]
```

```
forme= ["trigonal_platelet"] #( à changer en fonction de ce que l'on veut (en prenant dans les elements de la liste_de_toutes_les_formes)
```

```
# Création des listes d'éléments (liste_element) et de distances (liste_distance)

dtype={"names":("atom","distance"),"formats":('U2',float)} #juste pour avoir la liste des elements et des distances de dessous

liste_element,liste_distance=np.loadtxt("atom_latt2.txt",unpack=True,skiprows=1,usecols=[0,1],dtype=dtype)


# Création d'une variable nbr_elem pour parcourir la liste des éléments que de 0 à un certain nombre.

# Si on veut parcourir toute la liste des éléments, nbr_elem =len(liste_element)

nbr_elem=len(liste_element)


#création des listes qmin, qmax, qstep et biso

List_qmin=[0.01]

List_qmax=[20]

List_qstep=[0.01]

List_biso=[0.1,0.3,0.5,0.9]


#Nom du fichier qui enregistre forme par forme dans le boucle pour sauvegarder les datas frame petit a petit au cas ou en cas de bug

fichier_intermediaire='stage/Base_de_donnees_sauvegarde/%s.pk'


# endroit ou on veut enregistrer notre fichier xyz qui sera créé puis supprimé durant l'exécution de la fonction

lieu_fich_xyz="stage/xyz_tempo/tempo.xyz"


# Nom du fichier csv de sortie

fichier_sortie='stage/Base_de_donnees/trigonal_platelet_220a1020_et_230a1030_tt_elem_biso_01_03_05_09.pk'


print(len(forme)*nbr_elem*len(couche)*len(List_qmin)*len(List_qmax)*len(List_qstep)*len(List_biso),"lignes attendues dans le data frame")
```

Appel de la fonction

```
%time generation_fichier_pk_other(forme,couche,nbr_elem,List_qmax,List_qmin,List_qstep,List_biso,lieu_fich_xyz,fichier_intermediaire,fichier_sortie)
```

IV) Analyse de nos résultats

On va créer un graph qui permet de forme la distribution des formes de notre data frame

```
import matplotlib.pyplot as plt
```

Créer la figure et les axes

```
fig, ax = plt.subplots()
```

Tracer l'histogramme

```
ax.hist(data['Shape'], bins=9,edgecolor='black')
```

Ajouter des labels aux axes

```
ax.set_xlabel('Shape')
```

```
ax.set_ylabel('Frequency')
```

Personnaliser les intervalles de l'axe des x

```
x_ticks = range(0, 3) # Plage de 0 à 3
```

```
plt.xticks(x_ticks)
```

Afficher la figure

```
plt.show()
```

```
#-----
```

On va creer un graph qui permet de forme la distribution des éléments de notre data frame

```
import matplotlib.pyplot as plt
```

```
# Créer la figure et les axes

fig, ax = plt.subplots(figsize=(15,6))


# Tracer l'histogramme

ax.hist(data['Element'], bins=100,edgecolor='black')


# Ajouter des labels aux axes

ax.set_xlabel('Element')

ax.set_ylabel('Frequency')


# Personnaliser les intervalles de l'axe des x

x_ticks = range(0, 19) # Plage de 0 à 50

plt.xticks(x_ticks)


# Afficher la figure

plt.show()
```

Ce code trace la distribution des éléments et des formes de notre DataFrame.

I) Importation des bibliothèques

```
importlib.reload(pNP)

importlib.reload(pNMBu)
```

```
import glob
```

II) Fusion des fichiers pk

On creer la liste de tous les fichiers pk à fusionner, on les ouvre un par un (dans une boucle) au format DataFrame, puis on concatène les DataFrame entre eux pour au final ne former qu'un seul DataFrame qu'on enregistre pour finir au format pk.

```
# Création de la liste de tous les fichiers pk (que on a creer et qui sont dans le dossier Base de données) que ont va fusionner
```

```
liste_tout_fichiers_pk = glob.glob('stage/Base_de_donnees/*.pk')
```

```
# Création d'un variable portant le chemin d'accès au fichier pk ou il y aura tout ce qu'on a creer
```

```
pk_fichier_global = "stage/Base_de_donnees_finale/fichier_global.pk"
```

```
# Création du data frame vide (df_global) que on va remplir au fur et à mesure
```

```
df_global = pd.DataFrame(columns=["Solid_class", "Element", "Shape",  
"Distance", "Size_parameters", "Eq_Radius", "n_Atoms", "qmin", "qmax", "qstep", "b_iso", "q", "Intensities"])
```

```
for fich in liste_tout_fichiers_pk :
```

```
# Lecture des fichiers pk dans des DataFrames
```

```
df_fichier_de_la_boucle = pd.read_pickle(fich)
```

```
# Fusionner les DataFrames
```

```
df_global = pd.concat([df_global, df_fichier_de_la_boucle], ignore_index=True)
```

```
# Sauvegarder le DataFrame fusionné en pk
```

```
df_global.to_pickle(pk_fichier_global)
```

III) Suppression des doublons dans le Data frame global

Définition de la fonction qui supprime les doublons

Attention, pour information la colonne size_parameters passe de listes a tuples

```
def supprimer_doublons(df):  
    """  
    Supprime les doublons d'un DataFrame.  
  
    Paramètres:  
  
    df (pd.DataFrame): Le DataFrame à traiter.  
  
    Retour:  
  
    pd.DataFrame: Un nouveau DataFrame sans doublons.  
    """  
  
    # Suppression des lignes dont les 5 premières colonnes sont identiques  
  
    df['Size_parameters'] = df['Size_parameters'].apply(tuple)  
  
    # df['qmin'] = df['qmin'].apply(tuple)  
  
    # df['qmax'] = df['qmax'].apply(tuple)  
  
    # df['qstep'] = df['qstep'].apply(tuple)  
  
    # df['b_iso'] = df['b_iso'].apply(tuple)  
  
    df_sans_doublons = df[~df[["Solid_class", "Element", "Shape", "Distance", "Size_parameters", "Eq_Radius", "n_Atoms", "qmin", "qmax", "qstep", "b_iso"]].duplicated()]  
  
    return df_sans_doublons
```

IV) Application de cette fonction sur le DataFrame qu'on a créé

```
df_sans_doublons=supprimer_doublons(pd.read_pickle(pk_fichier_global))
```

V) On remet les bons indexs sur chaque lignes et on ré enregistre le data frame

```
df_sans_doublons=df_sans_doublons.reset_index(drop=True)
```

```
df_sans_doublons.to_pickle(pk_fichier_global)
```

VI) Ré Affichage du DataFrame fusionné

Cela permet de vérifier l'opération

```
# Afficher le DataFrame fusionné
```

```
df_global = pd.read_pickle(pk_fichier_global)
```

```
print("\nContenu du fichier PK fusionné (df_global):")
```

Puis dans une autre cellule du notebook :

```
df_global
```

4	Platonic	Al	cubefcc	2.852398	(2, 0, 0)	6.98692	63	0.01	20	0.01	0.1	[0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.0...	[670041.8, 668334.06, 665496.56, 661542.0, 656...	0
5583	Archimedean	Pb	truncated_tetrahedron	3.478965	(7, 0, 0)	8.158897	0	0.01	20	0.01	0.9	[0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.0...	[31048628.0, 30925274.0, 30720662.0, 30436172...	0
5584	Archimedean	Pt	truncated_tetrahedron	2.788829	(7, 0, 0)	6.540384	0	0.01	20	0.01	0.5	[0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.0...	[28082944.0, 28011072.0, 27891634.0, 27725172...	0
5585	Archimedean	Ru	truncated_tetrahedron	2.708219	(7, 0, 0)	6.351336	0	0.01	20	0.01	0.1	[0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.0...	[8944763.0, 8923054.0, 8886978.0, 8836681.0, 8...	0
5586	Archimedean	Ru	truncated_tetrahedron	2.708219	(7, 0, 0)	6.351336	0	0.01	20	0.01	0.9	[0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.0...	[3944754.0, 3923019.0, 3886699.0, 3836540.0, 8...	0

Voilà un exemple de ce qui devrait s'afficher

Etape 6 : Tracé aléatoire de $I(q)$ de notre base de données

Pour vérifier la cohérence de notre base de données, il est important de prendre une ligne dans notre DataFrame et de vérifier la forme du tracé $I(q)$ pour voir rapidement si celle-ci semble correcte ou non. Cela ne permet pas de vérifier la cohérence de manière précise mais seulement d'avoir une idée globale et de détecter rapidement certaines erreurs.

Code de création d'une fonction qui donne le tracé $I(q)$ d'une ligne de data frame choisie au hasard :

1) Importation des bibliothèques

```
#####
```

```
# Initialization #
```

```
#####
```

```
import os
```

```
import sys
```

```
print(os.getcwd())
```

```
cwd0 = './styles/'
```

```
sys.path.append(cwd0)
```

```
import visualID as vID
```

```
from visualID import fg, hl, bg
```

```
vID.init(cwd0)
```

```
import numpy as np
```

```
import ase
```

```
from ase.io import write
```

```
from ase.visualize import view
```

```
import pyNanoMatBuilder.utils as pNMBu
```

```
import importlib
```

```
import pandas as pd
```

```
from pyNanoMatBuilder import platonicNPs as pNP
```

```
import importlib
```

```
importlib.reload(pNP)
```

```
importlib.reload(pNMBu)
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
import numpy as np
```

```
import ase
```

```
from ase.io import write
```

```
from ase.visualize import view
```

```
import pyNanoMatBuilder.utils as pNMBu
```

```
import importlib
```

```
from matplotlib import pyplot as plt
```

```
import torch
```

```
import random
```

II) Définition de la fonctions nécessaire pour le tracer random d'un $I(q)$

```
def tracer_ligne_aleatoire(df, colonne_x="q", colonne_y="Intensities"):

    # Choisir une ligne au hasard

    nbr_alea=int(random.uniform(0, len(df)))

    # Extraire les listes des colonnes spécifiées

    x = df[colonne_x].iloc[nbr_alea]

    y = df[colonne_y].iloc[nbr_alea]

    # Tracer les données

    titre='I(q)'

    label = f'distance : {df["Distance"][nbr_alea]:.2f}, shape : {df["Shape"][nbr_alea]}, class : {df["Solid_class"][nbr_alea]}, size : {df["Size_parameters"][nbr_alea]}'

    plt.figure(figsize=(8, 6))

    plt.loglog(x, y,label=label)

    plt.legend(loc=1)

    plt.xlabel(colonne_x)

    plt.ylabel(colonne_y)

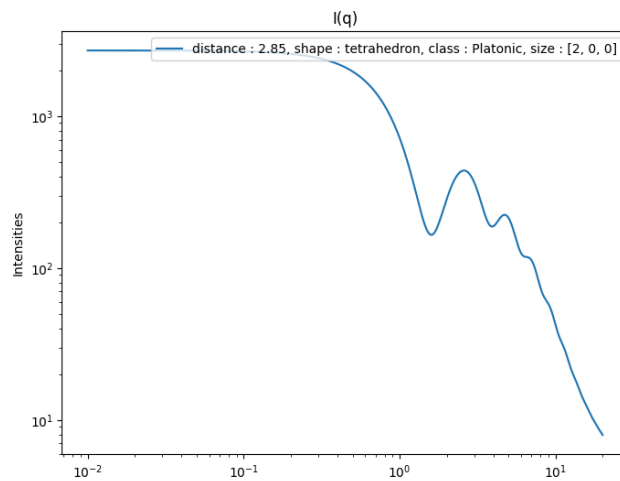
    plt.title(titre)

    plt.show()
```

III) Appel de la fonction

```
tracer_ligne_aleatoire(pd.read_pickle("stage/Base_de_données_finale/fichier_global.pk"), "q", "Intensities")
```

exemple de tracé que l'on peut obtenir :



A finir bien préciser aussi pour les tailles dans les tableaux quel indice correspond a quoi etc, super important pour l'utilisateur!!

Si temps : faire une table des matières.