**Projet Python**

**Modélisation des coefficients d’atténuation massiques des**

**photons traversant différents milieux**

Table des matières

[I. Introduction 1](#_Toc159679757)

[II. Étape 1 1](#_Toc159679758)

[a) Travail demandé 1](#_Toc159679759)

[b) Explication du code 1](#_Toc159679760)

[c) Problèmes rencontrés 2](#_Toc159679761)

[III. Étape 2 3](#_Toc159679762)

[a) Travail demandé 3](#_Toc159679763)

[b) Explication du code 3](#_Toc159679764)

[c) Problèmes rencontrés 4](#_Toc159679765)

[IV. Étape 3 4](#_Toc159679766)

[a) Travail demandé 4](#_Toc159679767)

[b) Explication du code 4](#_Toc159679768)

[c) Problèmes rencontrés 5](#_Toc159679769)

[V. Étape 4 5](#_Toc159679770)

[VI. Références 5](#_Toc159679771)

# Introduction

Le but de ce projet est de réaliser une interface permettant à un utilisateur de modéliser l’évolution des coefficients d’atténuation massique pour des photons qui traversent différents milieux, plus précisément, de l’aluminium, du plomb, du cobalt et du cuivre. De plus il faudra rendre possible la sélection des différents effets physiques pour lesquels modéliser l’évolutions de τ.

Pour ce faire il nous est demandé de réaliser des script étape par étape pour arriver à la fin au programme final

Ce projet a été réalisé avec la version 3.11.6 de Python et sous l’IDE Spyder en version 5.5.0

Plutôt que d’ajouter des captures d’écran à répétition, on se propose de faire une explication du code linéaire du programme principale avec une explication des fonctions lors de leur première appel. De plus un certain nombre de commentaire devrait permettre une lecture compréhensible du code.

L’intégralité du suivi des version et leurs implémentations est disponible sur le GitHub suivant : <https://github.com/matmat201/Python_Project_M1>

# Étape 1

## Travail demandé

La première étape consiste à un tracé des différentes courbes directement dans l’interpréteur. Pour cela il est nécessaire au préalable de constituer une base de données des différentes valeurs de τ pour les différents effets physiques. On nous propose de récupérer les données depuis le site du National Institue of Standards and Technology (N.I.S.T)[[1]](#endnote-1). On nous demande de la nommer bdd\_photons.xlsx cependant ce ne sera pas la base de données final comme ce sera développé dans l’étape 3. Une fois la base de données construite on va a l’aide de la bibliothèque matplotlib, tracer l’évolution des τ dans l’interpréteur ainsi que vérifier le contenue des listes pour les comparé visuellement a la base de donnée et ainsi garantir avec un certain degré de confiance la conformité du code a nos attentes  
La base de donnée va contenir les valeurs d’énergie, et de τ pour la diffusion Compton (incoherent scattering), l’effet photoélectrique (Photoelectric absorbtion), ainsi que la création de paire à la fois dans le champ du noyaux et des électrons. Respectivement Nuclear pair production et Electron pair production. Chaque page du tableur sera consacré à un des quatre éléments

## Explication du code

La première chose à faire est d’importer les bibliothèque que l’on va utiliser ici openpyxl pour la lecture de document Excel et la sous-bibliothèque pyplot de matplotlib pour le tracé de courbe  
Ensuite, on utilise les fonctions de openpyxl pour ouvrir la database  
On sait que l’on dispose de 4 pages dans notre database, donc au lieu de faire un code qui récupère chaque page, on demande a l’utilisateur la page qu’il souhaite récupérer et on utilise get\_sheet\_by\_name avec l’input de l’utilisateur  
On ajoute aussi la vérification des entrée d’utilisateur avec une boucle while qui est True tant que l’input ne match pas un élément d’une liste contenant les noms des pages. On sort de la boucle avec l’instruction break qui se déclenche si l’utilisateur à rentrer un élément valide.

Une fois que l’on a pu choisir qu’elle page ouvrir, on va créer des listes que l’on va remplir à l’aide de la méthode .append() en parcourant chaque colonne du tableur de la ligne 4 a la dernière ligne remplie. On dispose maintenant de liste que l’on peut utiliser pour tracer l’évolution de τ en fonction de l’énergie. Pour le traçage, on réutilise la syntaxe et les méthodes vue en cours, notamment le passage en échelle log.

Enfin on va afficher le contenue des listes créées par le programme dans la console. Encore une fois dans un souci d’économie de l’énergie du développeur, on va se passer de boucles intempestives et utiliser un dictionnaire, ce dernier permet d’associer a une clef une certaine valeur, cela est surtout utile ici pour automatiser l’affichage en console, ainsi en bouclant sur dico.item(), on récupère un couple clef, valeur que l’on a associé intelligemment. En effet, la clef nous permet de changer la chaine de caractère pour coller au nom de la liste que l’on affiche, et la valeur contient la liste. Finalement, pour améliorer la lisibilité on va changer l’affichage de nos listes, en effet, nos listes sont au format « vecteur » et donc s’affiche en colonne, on remédie à ce problème grâce à la fonction join qui va concaténer chaque valeur de la liste en les séparant par une virgule, cependant on doit donner a join des strings, on va donc utiliser la fonction map qui permet de changer toutes les valeurs de la liste en string sans faire de boucle.

Voici un exemple de ce que retourne le script pour l’aluminium

Une image contenant texte, capture d’écran, nombre, ligne

Description générée automatiquement

## Problèmes rencontrés

L’un des premiers problèmes rencontrés a été la création de la base de données, en effet, l’extraction des données avec l’outils fourni sur le site fournissait un fichier texte uniquement les titres, après avoir essayé plusieurs combinaisons, la méthode retenue a été de copier-coller les données depuis l’affichage en tableau du site et après un formatage des données en écriture scientifique la base de données préliminaire été complète.

Les autres problèmes viennent plutôt de l’effort d’automatisation qui a forcé l’usage de certaine fonction et méthodes. En commençant par la fonction join puisque ayant l’habitude de travailler avec des array numpy ou l’on peut transformer une vecteur ligne en vecteur colonne avec des compréhension, ici il a fallu chercher une si une fonction pouvait faire une opération similaire. Certes avec join on ne change pas la liste mais l’affichage permis est plus que satisfaisant.

Un autre petit défi a été l’implémentation d’une variable dans un titre de plot. Mais la syntaxe apparait facilement après une recherche et est facile à utiliser.

# Étape 2

## Travail demandé

Pour la partie 2, le but est cette fois-ci d’implémenter le tracé à l’intérieur d’une fenêtre tkinter. Contrairement a la partie 1, il n’est question que de l’aluminium pour cette partie, cependant on demande de pouvoir donner à l’utilisateur la possibilité de choisir quel type d’effet il souhaite modéliser. Pour ce faire on va utiliser le widget CheckButton de tkinter. De plus pour simplifier le processus « créatif », on se propose de reprendre la disposition présentée dans le cahier des charges. On peut encore utiliser la base de données crée dans l’étape 1

## Explication du code

Comme pour chaque script on importe les bibliothèques, ici la nouveauté est tkinter, ainsi que différents modules de matplotlib pour inclure les plots dans les fenêtres tkinter.  
On commence par la mise en place de l’interface utilisateur (UI), après la création de la fenêtre et son titre, on va gérer les cases à cocher, pour cela on va créer des variable de « contrôle » qui vont servir à stocker l’état de la case, un 0 signifiant non coché et un 1 coché, elles sont donc de classe IntVar(). Ensuite, on utilise la syntaxe de création des Checkbutton, pour ça on s’inspire de l’exemple donné sur le site pythonbasics.org[[2]](#endnote-2)

Avec cet exemple, on voit que la méthode .get sur une variable de contrôle permet de récupérer son contenu, on va donc s’en servir dans la fonction trace\_alu que l’on appel quand on clique sur chaque bouton. Ensuite, il ne reste plus qu’as gérer la création de la zone de graphe et la mise en page pour ce qui est de l’UI. Pour la zone de graphe, on utilise le code vu en cours, on va aussi ajouter un bouton qui efface le plot et un bouton qui ferme la fenêtre. Pour ce qui est de la mise en page, on utilise la méthode .grid et on place nos widgets en indiquant la ligne et optionnellement la colonne, l’argument sticky= ‘w’ permet de coller la case a gauche.

Une fois notre UI créée, il faut gérer les fonctions que l’on appel. La fonction la plus simple est effacer car elle se content de nettoyer le graphe et de recréer un espace de plot vierge. La fonction trace\_alu quant a elle va reprendre la logique établie dans la partie 2 en extrayant les données de la base de données à l’aide de openpyxl mais cette fois sans s’embêter avec la gestion des différentes pages, on récupère uniquement les τ de l’aluminium. Par la suite, la création de la fenêtre de plot ne change quasiment pas de la version en console, si ce n’est la création de l’alias ax. C’est au moment de choisir quelles données tracer que l’on va faire appel a nos variables de contrôle, ainsi en testant si la variable de contrôle vaut 1 (i.e. : la case est cochée) on plot avec les axes correspondant, en plaçant la commande fig.canvas.draw() après tous les tests, on assure la possibilité de plot 2 ou plus sets de données en même temps.

Une image contenant texte, capture d’écran, diagramme, ligne

Description générée automatiquementOn obtient alors la fenêtre suivante :

## Problèmes rencontrés

Il n’y a pas eu de problèmes majeurs, puisque le plus gros du travail avait déjà été effectué en cours et pendant l’étape 1, si ce n’est au début une implémentations maladroite du test des variable de contrôle ou on ne pouvait pas plot plusieurs courbes en même temps et où il fallait faire 9 tests pour toutes les combinaisons possibles de case.

# Étape 3

## Travail demandé

Le but de cette partie est de reprendre le script de l’étape 2 mais d’ajouter de nouvelles fonctionnalités, augmentant la versatilité et les interactions avec l’utilisateur, il s’agit d’implémenter un menu déroulant permettant de choisir le type de milieu traversé, augmenter le nombre d’interaction possible a cocher et l’ajout de borne dynamique pour les plots via des curseurs. Une nouvelle fois, on se propose de s’inspirer grandement de la disposition fournit dans le cahier des charges incluant l’utilisation de cadre pour isoler les différentes fonctions dans la fenêtre.

## Explication du code

Encore une fois, tout comme la partie 1 et 2 on importe les bibliothèques, ici la nouveauté est ttk, un module de tkinter pour gérer les Combobox (menu déroulant).  
On recommence une nouvelle fois par la mise en place de l’interface utilisateur (UI), après la création de la fenêtre et son titre, on va gérer les modules de haut en bas, commençant donc par le menu déroulant. Pour ce faire, on crée d’abord un Label contenant l’instruction, ensuite on a besoin d’une liste qui va contenir les valeurs du menu déroulant, on crée le menu déroulant avec la syntaxe que l’on peut trouver en exemple ici[[3]](#endnote-3) (cela servira aussi à reprendre la logique pour récupérer la sortie du menu). On initialise le menu a la valeur 0 soit la première valeur de la liste ici « Aluminium ». Maintenant on va utiliser la méthode .bind() pour relier une certaine action a une fonction à réaliser ici on relie <<ComboboxSelected>> ce qui signifie que l’utilisateur à sélectionner un élément du menu à la fonction choix\_mat\_fct que l’on détaillera plus tard.

Il faut maintenant rajouter les cases des nouvelles interactions mais on reprendre la logique exposée en partie 2 c’est-à-dire d’abord une variable de contrôle puis la création de la case. Cependant, il faut maintenant les placer à l’intérieur d’un Frame, en soit la création d’un frame et son placement sont relativement simple mais c’était sans compter des futurs problèmes purement esthétiques sur lequel nous reviendrons dans le c). Pour placer les cases dans le frame, il suffit de remplacer le premier argument (« master ») de Checkbutton avec le nom du cadre, ensuite on utilise .grid() pour réaliser le placement dans le frame. De plus on va initialiser certaine case a 1 c’est-à-dire les pré cocher pour des raison d’optimisation des test ou l’on a pas besoin de tout cocher a chaque fois et en plus peut sembler utile pour l’utilisateur qui peut directement tracer un graphe a la pression d’un bouton

Maintenant il faut implementer les curseurs de sélections, que l’on va la encore placer dans un Frame, là encore avec la même logique et placement. Ensuite, on s’occupe des curseurs, la syntaxe est tirée des exemples que l’on trouve ici[[4]](#endnote-4). On commence par choisir le master (la zone où l’on va le placer donc ici le cadre), puis on indique les bornes, la résolution (le pas minimum), l’orientation, et finalement la commande (fonction) à appeler. On reviendra sur cet aspect qui a su montrer son lot de rebondissement. Ensuite, on peut l’initialiser et le placer, cette fois un simple pack suffit. C’est ensuite la même logique pour tous les curseurs de sélections de bornes en plaçant juste le deuxième cadre sur la colonne 1 et le premier sur la colonne 0.

Il ne reste plus qu’as ajouter la zone de graphe ainsi que trois boutons de fonction, qui vont respectivement tracer la courbe avec les paramètres choisi, effacer la fenêtre et quitte comme pour l’étape 2.

Pour ce qui est des fonctions, la fonction qui efface est la même qu’à l’étape 2, ensuite, la première fonction que l’on va appeler est celle qui va gérer le choix du matériau traversé c’est la fonction qui a été liée à l’action <<ComboboxSelected>>. Pour cette fonction, on va avoir besoin d’une variable qui va récupérer la sortie du menu déroulant, de plus on va initialiser cette variable a la valeur Aluminium ainsi même si l’utilisateur ne sélectionne aucune valeur, il y aura quand même une valeur pour les futures fonctions qui vont s’en servir. Ensuite, au sein même de la fonction, on va utiliser le terme global qui va assurer la modification de notre variable en dehors de la fonction. Ensuit le fonctionnement est relativement simple, on utilise la méthode .get() pour récupérer la valeur du menu que l’on stocke dans notre variable. Optionnellement on a rajouté un print dans la console pour s’assurer du bon fonctionnement du retour

La prochaine fonction est quasiment identique à la fonction trace\_alu, la différence étant que l’on va utiliser notre variable pour choisir la page du tableur à extraire et que l’on a rajouter de nouveau set de données allant avec nos nouveau checkbox, il faut donc ajouter tout autant de test pour plot les bons sets de données sélectionnées. La dernière subtilité intervient avec les bornes des axes mais on va détailler ce point dans nos fonctions qui gère les curseurs.

Les premières fonctions et les plus simples pour nos curseurs sont les fonction scale\_to\_real. En effet, elles servent juste à contourner une limitation des curseur qui est que une borne de 10-4 est trop petite pour les bornes du curseurs, on va donc faire un curseur qui va de 1 à 100 et multiplier ce résultat par 10-4 . Maintenant qu’on a une fonction pour convertir dans la bonne échelle le retour d’un curseur, il faut une fonction qui va permettre de visualiser sous le curseur, la valeur réelle qui lui correspond, c’est là que vont intervenir quelques petites astuces. Tout d’abord au lieu de faire une fonction qui gère la borne supérieure et une qui gère la borne inferieur, on a choisi de faire tout dans une fonction qui va prendre en argument la valeur du curseur pour l’afficher dans un label ainsi qu’une valeur booléenne qui indique si on est dans le cas supérieur ou inférieur. Cependant une telle fonction qui a deux arguments ne peux pas directement être donnée en tant qu’action à effectuer par un widget après l’argument command=. Il va falloir donc soit créer une fonction intermédiaire a un seul argument et qui appelle la vraie fonction avec les deux arguments. Or en python, on a la possibilité d’utiliser le terme lambda qui agit comme une fonction « anonyme » qui va renvoyer vers la vraie fonction avec les deux arguments de manière plus condensée et donc sans créer différentes fonctions intermédiaires. Ainsi les deux argument command suivant sont identique :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Maintenant que l’on a un moyen de passer deux arguments a notre fonction dans le curseur, on peut se pencher sur le corps des fonction qui gère l’affichage de la valeur réelle du curseur après conversion. Comme on peut le voir notre fonction update\_value va prendre deux arguments, la première value signifie que la fonction va récupérer la valeur du curseur, la deuxième argument est une valeur booléenne qui va permettre de choisir qu’elle partie de la fonction on va exécuter et donc si on va mettre a jour la

## Problèmes rencontrés

# Étape 4

Ecriture dans un fichier[[5]](#endnote-5)

# Références

1. N.I.S.T website: <https://physics.nist.gov/PhysRefData/Xcom/html/xcom1.html> [↑](#endnote-ref-1)
2. Checkbutton exemple: <https://pythonbasics.org/tkinter-checkbox/> [↑](#endnote-ref-2)
3. Combobox exemple : <https://www.tresfacile.net/liste-combobox-tkinter/> [↑](#endnote-ref-3)
4. Scale exemple : <https://python-course.eu/tkinter/sliders-in-tkinter.php> [↑](#endnote-ref-4)
5. Ecriture fichier : <https://python.doctor/page-lire-ecrire-creer-fichier-python> [↑](#endnote-ref-5)