**Projet Python**

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

**photons traversant différents milieux**

Table des matières

[I. Introduction 1](#_Toc162458019)

[II. Étape 1 1](#_Toc162458020)

[a) Travail demandé 1](#_Toc162458021)

[b) Explication du code 1](#_Toc162458022)

[c) Problèmes rencontrés 2](#_Toc162458023)

[III. Étape 2 3](#_Toc162458024)

[a) Travail demandé 3](#_Toc162458025)

[b) Explication du code 3](#_Toc162458026)

[c) Problèmes rencontrés 4](#_Toc162458027)

[IV. Étape 3 4](#_Toc162458028)

[a) Travail demandé 4](#_Toc162458029)

[b) Explication du code 4](#_Toc162458030)

[c) Problèmes rencontrés 6](#_Toc162458031)

[V. Étape 4 6](#_Toc162458032)

[VI. Conclusion 9](#_Toc162458033)

[VII. Références 10](#_Toc162458034)

# 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’évolution de τ.

Pour ce faire, il nous est demandé de réaliser des scripts 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 premier appel. De plus, un certain nombre de commentaires devrait permettre une lecture compréhensible du code.

L’intégralité du suivi des versions 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 Institute 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 à l’aide de la bibliothèque matplotlib, tracer l’évolution des τ dans l’interpréteur ainsi que vérifier le contenu des listes pour les comparé visuellement à la base de données et ainsi garantir avec un certain degré de confiance la conformité du code à nos attentes  
La base de données 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 pair 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 bibliothèques 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 à 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 entrées d’utilisateur avec une boucle while qui est True tant que l’input ne correspond 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 a rentré un élément valide.

Une fois que l’on a pu choisir quelle 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 à la dernière ligne remplie. On dispose maintenant de listes 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 contenu 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 à une clef une certaine valeur, ce qui 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’affichent 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 à 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’outil fourni sur le site fournissait un fichier texte avec 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 dans Excel la base de données préliminaire était complète.

Les autres problèmes viennent plutôt de l’effort d’automatisation qui a forcé l’usage de certaine fonctions 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 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 à 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 note qu’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 variables 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 appelle 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 à gauche.

Une fois notre UI créée, il faut gérer les fonctions que l’on appelle. 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 à 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 à 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 implémentation maladroite du test des variables 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 à 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 à 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 reprend 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 cases a 1 c’est-à-dire les pré cocher pour des raison d’optimisation des tests ou l’on aura pas besoin de tout cocher à chaque fois et en plus peut cela sembler utile pour l’utilisateur qui peut directement tracer un graphe à 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 quitter comme à 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 à 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 rajouté de nouvelles colonnes de données allant avec nos nouveaux checkbox (une nouvelle base de données a donc été créée et nommée bdd\_photons\_all\_datas), 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 fonctions scale\_to\_real. En effet, elles servent juste à contourner une limitation des curseurs 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 peut 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 à 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 arguments command suivant sont identiques :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Maintenant que l’on a un moyen de passer deux arguments à notre fonction dans le curseur, on peut se pencher sur le corps des fonctions 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, le premier : « value » signifie que la fonction va récupérer la valeur du curseur, le 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 à jour le label correspondant à la borne supérieure ou à la borne inférieure. La fonction va commencer par convertir la valeur du curseur en « real\_value » en appelant la fonction scale\_to\_real définie plus tôt. Ensuite elle va mettre à jour le label avec la méthode .config. Notons le formatage du texte qui utilise la lettre f pour indiquer que la valeur entre les accolades est une variable. Un petit peu à la manière d’un %d en C. On va dans la foulé écrire la nouvelle valeur en écriture scientifique pour des raison de lisibilité. Pour cela on utilise :.4e après la valeur de notre variable ce qui indique à python d’écrire notre valeur en écriture scientifique avec 4 chiffres après la virgule.  
Voici ce que l’on obtient comme résultat :Une image contenant texte, capture d’écran, diagramme, ligne

Description générée automatiquement

## Problèmes rencontrés

Un des premiers problèmes qui s’est posé est en lien avec l’utilisation des frames. En effet, dans le but de s’inspirer au mieux de l’exemple donnée dans le cahier des charges, nous voulions donner des titres à nos frame pour indiquer ce qu’ils contenaient, or cela n’est pas possible par passage d’un argument au widget frame, il faut donc créer un label que l’on place dans la fenêtre au même endroit que le frame et que l’on va bouger avec du padding. Ce qui n’est pas idéale puisque cela implique de devoir modifier le padding à la moindre modification de la géométrie de la fenêtre. L’autre petit point qui a posé problème a été le passage de deux arguments en fonction et l’emploi de la fonction anonyme lambda. Bien que le concept ne soit pas très compliqué à saisir, il a d’abord fallu trouver des ressources faisant référence à cette méthode et bien comprendre comment s’en servir.

# Étape 4

Le traitement de l’étape 4 va être un peu différents, en effet, on ne demande pas explicitement de faire des choses particulières mais plutôt d’améliorer le programme comme on le souhaite. Des lors, plutôt que de se jeter dans le code sans réfléchir, il a fallu penser à l’avance à ce que l’on voudrait faire en définissant des objectifs. Permettant ainsi de se focaliser en premier lieu sur une disposition incluant tous les éléments graphiques que l’on souhaitait utiliser et donc éviter de modifier l’interface à chaque ajout. De plus, bien que tkinter soit redoutable d’efficacité pour déployer rapidement une interface fonctionnelle, nous avons voulu regarder s’il était possible de la rendre un peu moins austère et plus au gout du jour. Dans un premier temps on remarque qu’il est possible de créer des thèmes tkinter qui permettent de définir les couleurs de fond des boutons, le rayon de courbure des angles, la police, etc. Et bien, qu’il existe des thèmes existants assez satisfaisant, au travers des recherches, une autre bibliothèque, fille de tkinter est apparu et a tout de suite su nous séduire. Il s’agit de CustomTkinter[[5]](#endnote-5), c’est un peu comme un thème directement intégré à tkinter mais en plus on peut dynamiquement modifier les couleurs, les angles, etc avec des arguments. La syntaxe est extrêmement similaire à tkinter, typiquement un CTk avant le nom du widget suffit pour faire la conversion.

Nous avons donc porté notre dévolu sur cette nouvelle bibliothèque et le premier travail a été de convertir le script de la partie trois en script CustomTkinter. Une fois fais, il a fallu définir quelles seraient les fonctions que l’on voudrait implementer. Pour cela, on a commencé par s’inspirer des exemples donnés dans le cahier des charges, c’est-à-dire des fonctions permettant d’obtenir la valeur de τ pour une énergie donnée, transformer τ en section efficace, changer les unités du plot (de τ a µ) et une dernière fonction qui permet d’enregistrer les informations dans un fichier texte. Ensuite, il a fallu réfléchir aux autres fonctions que l’on voudrait intégrer. La première qui nous est apparu comme une évidence a été l’ajout de bouton de saisie manuelle des bornes car bien que les sliders permettent de faire varier facilement les bornes, si l’on veut s’affranchir de leur limite ou saisir des valeurs précises cela n’était pas possible. Les autres fonctions que l’on a implémentées sont principalement des fonctions ayant pour but d’essayer et d’utiliser différents widgets de CustomTkinter. On a donc décider de rajouter des radiobutton, des Textbox, des segmentedbutton, etc pour apprivoiser la bibliothèque.

Une fois que l’on a eu défini nos objectifs, il a fallu les placer dans notre interface, or il se trouve que la disposition de l’étape 3 laisse un espace vide entre le frame pour choisir les interactions et la zone de plot, on a donc choisi judicieusement cet emplacement pour placer nos nouveaux boutons de fonctions. On commence donc comme pour les étapes 2 et 3 par créer les boutons et les variables correspondantes pour les placer et fixer notre interface. On obtient donc l’interface suivante avec les boutons placés mais non fonctionnel Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

On a choisi d’utiliser non pas un frame standard, mais un Tabview, d’une part pour experimenter avec ce widget mais aussi car cela va permettre de rajouter plus de fonction dans le même espace en pouvant alterner entre deux onglets dans le frame.  
  
 On va maintenant traiter chaque bouton de haut en bas.

* Le premier bouton nomé « Aide » appele la fonction Documentation. Cette fonction va créer une fenêtre dans laquelle on va mettre le widget CTkTextbox qui va permettre de faire une fenetre que l’on peut faire defiler. Le but étant d’afficher un petit texte qui explique le fonctionnement de l’interface. Afin de modifier le contenu de la Textbox, on definit une variable qui contient le texte puis on utilise la méthode .insert() qui prend comme premier argument un couple de chiffre ici « 0.0 » qui indique la colonne et la ligne ou l’on insert le deuxieme argument, ici notre variable qui contient le texte.
* Une image contenant texte, capture d’écran, Police, logiciel

  Description générée automatiquementLe bouton « Extraction Tau » va appeler la fonction Val\_Tau, cette dernière va ouvrir la fenêtre suivante :

On commence par choisir le type d’interaction avec un menu déroulant, on choisit l’énergie puis on déclenche l’extraction avec le bouton correspondant. Le résultat est renvoyé à l’utilisateur.

Lors de l’activation du bouton « extraction », on appel la fonction Extraction, cette fonction commence par ouvrir la database avec le matériau choisi, elle récupère la valeur d’énergie donnée par l’utilisateur et va parcourir la database en cherchant la valeur la plus proche de l’énergie donnée par l’utilisateur. Pour ce faire, on va parcourir la liste en calculant la différence entre la valeur de la liste et la valeur de l’utilisateur et on la compare à une variable différence que l’on modifie si la valeur obtenue est inférieure à celle déjà dans différence, de plus si la différence est plus petite on va récupérer la valeur d’énergie et extraire la valeur de tau correspondante. Ainsi à la fin de la boucle, la dernière modification de différence correspond à la plus faible différence et donc on a bien extrait le τ correspondant à l’énergie la plus proche de celle entrée par l’utilisateur.

* Une image contenant texte, capture d’écran, Police

  Description générée automatiquementLe bouton suivant se charge de convertir la valeur de tau en section efficace. Il va appeler la fonction section\_eff qui va se charger de créer une fenêtre de retour comme celle-ci :

Elle va récupérer les variables correspondant au matériau choisi, à l’interaction choisi et à l’énergie entrée puis va convertir τ en section efficace avec la formule suivante =>

* Le bouton « changement unité » va se charger de convertir les listes de données entre τ et µ. Cependant, on veut non pas que ce soit un toggle mais que quand on clique sur le bouton plusieurs fois à la suite, on alterne en τ et µ. On aurait pu utiliser un switch mais le bouton étant déjà créé on a utilisé une variable de contrôle qui va imiter le comportement d’un interrupteur. Ainsi, si la variable est a 0 on modifie les valeurs et on passe la variable a 1 et donc a la prochaine pression, on va modifier les variables dans le sens inverse et modifier la variable a 0.
* Une image contenant texte, capture d’écran, Police, nombre

  Description générée automatiquementLe dernier bouton de cet onglet permet à l’utilisateur de sauvegarder les données obtenues avec les boutons précédents. Il appel la fonction fenetre\_donnees qui va ouvrir la fenêtre suivante :

On donne à l’utilisateur la possibilité de choisir le nom des fichiers à sauvegarder et on appelle les fonctions sauvegarder\_donnees ou sauvegarder\_images en fonction du bouton pressé. La première va utiliser la fonction open pour ouvrir un fichier et la méthode .write pour modifier le fichier[[6]](#endnote-6). La seconde utilise la méthode .savefig pour enregistrer le plot.

* Finalement, on a rajouté deux radio button qui permettent de sélectionner toutes les interactions ou de les désélectionner en un seul clique. Le principe des radio button est de ne pouvoir sélectionner qu’un seul des boutons à la fois. Une fois que l’on sélectionne un des boutons, cela modifie la valeur d’une variable de contrôle qui permet de choisir le comportement de la fonction Selection\_radioButton. Celle-ci va simplement set les variables de contrôle des checkbox pour les cocher ou décocher.
* Il faut noter que l’on a souhaiter rajouter un texte de survol lorsque la souris passe au-dessus d’un bouton afin d’expliquer son utilité, pour cela, on utilise le widget CTkToolTip[[7]](#endnote-7) qui permet simplement d’avoir ce comportement. Nous n’avons implémenté un tooltip que pour un seul bouton pour démontrer son utilisation.

Une image contenant texte, capture d’écran, multimédia, logiciel

Description générée automatiquementOn peut maintenant passer sur le deuxième onglet intitulé Options :

Le premier menu déroulant permet de sélectionner le mode d’apparence, « dark », « light », ou « system » (s’adapte au paramètre de la machine). Puis appel une fonction intégrée à CustomTkinter qui change l’apparence. Le deuxième menu permet de sélectionner le niveau de zoom de l’interface en appelant aussi une fonction de CustomTkinter. Le dernier menu déroulant n’est pas fonctionnel malheureusement.

Une image contenant texte, capture d’écran, Police

Description générée automatiquementLe bouton « Crédits » ouvre la fenêtre suivante :

Elle comporte un simple label qui explique l’origine de l’application, un bouton désactivé simplement pour l’esthétisme, un segmentedbutton qui permet d’ouvrir les pages internet des différentes ressources utiles. Pour ce faire, lorsque l’on clique sur un des segments, une variable va stocker le texte du segment et appeler une fonction qui utilise la bibliothèque webbrowser pour ouvrir les liens correspondants.  
Le dernier widget utilisé est une progressbar qui n’a aucune réelle utilité à par expérimenter avec un widget de plus.

La toute dernière fonction que l’on a implémenté dans le programme est la saisie manuelle des bornes pour le plot. Les deux boutons de saisie manuelle ouvrent des fenêtres similaires qui permettent de choisir les valeurs que l’on veut pour les bornes. La seule subtilité étant qu’il faut faire le distinguo entre la valeur choisi manuellement et celle choisi par les sliders. Pour cela on utilise une nouvelle fois une variable de contrôle qui va dicter le comportement de la fonction trace pour changer les valeurs des bornes en adéquation avec ce que l’on a choisi.

# Conclusion

Malgré, plusieurs années d’utilisation de python, je n’avais jamais utilisé python pour faire des GUI donc jamais utilisé tkinter qui se révèle être extrêmement pratique pour faire fonctionner rapidement des petites applications. CustomTkinter permet même de rendre nos applications quasi professionnelles, du moins beaucoup plus moderne. Bien que j’ai dû mener ce projet seul, je ne suis pour autant pas resté seul dans mon coin, au contraire, dès que j’eus fini les trois premières parties avant les vacances de février j’ai pu échanger avec plusieurs de mes camarades pour les aider, et échanger sur des idées de fonction à faire ou simplement ce que devrait faire une fonction. Par exemple faut-il réinitialiser le choix du matériau ou pas. De plus, je me suis efforcé de documenter toute la progression du projet sur la page GitHub que j’ai créé. Me permettant ainsi de manipuler GitHub mais aussi d’instaurer un environnement assez proche d’un projet professionnel je pense, ou je ne faisais pas le projet que pour moi mais où je devais détailler les ajouts à chaque commit sur GitHub. J’ai pu aussi découvrir plusieurs fonctionnalités de Python, comme la fonction lambda ou la bibliothèque webbrowser.

# 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. CustomTkinter : <https://customtkinter.tomschimansky.com> [↑](#endnote-ref-5)
6. Ecriture fichier : <https://python.doctor/page-lire-ecrire-creer-fichier-python> [↑](#endnote-ref-6)
7. CTkToolTip : <https://github.com/Akascape/CTkToolTip> [↑](#endnote-ref-7)