

UNIVERSITÉ
LAVAL

Physique numérique PHY-3500

Physique, génie physique et optique

Hiver 2026

Université Laval

Travail Pratique 1

Identification

- Éloi Blouin : 536 999 917
- Clément Poulin : 536 994 304

In [198...]

```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
import scipy.constants as cte
import types
import pandas as pd
import scipy as sp
import math

plt.rcParams.update({'font.size': 14})

from typing import Callable
from scipy.stats import moyal
import timeit
```

Instructions pour la remise

Le travail devra être complété en trinômes sous format de cahier de bord jupyter (.ipynb) et remis dans la boîte de dépôt créée à cette fin. Ce document contiendra toutes informations

pertinentes permettant au lecteur d'apprécier vos résultats et conclusions, incluant le code Python utilisé et d'éventuelles références bibliographiques. La qualité de la présentation est très importante (utilisation de sections, de graphiques appropriés, de mise en contexte, etc.).

Prenez soin de bien indiquer votre (ou vos) nom(s) dans le cahier de bord. Pour faciliter la tâche de classification, utilisez la nomenclature suivante pour le fichier transmis (un seul) :

TPn_nom1_nom2_nom3.ipynb

Objectif

Déterminer la portée des protons dans matière par des méthodes d'intégration numérique.
Se familiariser avec la protonthérapie.

Introduction

La radiothérapie externe par faisceaux de rayons X , d'électrons ou de hadrons a pour but de détruire des cellules pathologiques (cancer par exemple) tout en épargnant les tissus sains environnants. L'éradication des cellules tumorales est effectuée par des dépôts d'énergies élevés lors du passage des photons/électrons/hadrons dans la matière (le corps). Pour effectuer des traitements optimaux, il convient d'adapter les faisceaux à la tumeur (forme, profondeur) pour administrer la dose prescrite à la structure cible tout en préservant au maximum les tissus sains. La balistique doit être adéquate et optimisée au cas clinique à traiter et le physicien médical est le garant de la dose administrée aux volumes cibles et aux organes à risque. Le physicien médical est en quelque sorte le pharmacien de la radiation, étant responsable de son dosage.

La protonthérapie est une technique avancée de radiothérapie externe qui utilise des faisceaux de protons à des fins thérapeutiques. Elle est utilisée dans quelques dizaines de centres à travers le monde ¹. TRIUMF ² à Vancouver est le centre de référence au Canada en protonthérapie. Ce centre est en mesure de traiter des tumeurs peu profondes car la ligne de faisceau médicale permet des énergies de protons relativement faibles. Néanmoins, cette technologie est très efficace pour traiter certains mélanomes oculaires car les protons ont l'avantage de mieux préserver les tissus environnants, grâce à une balistique de traitement optimale. Ainsi, les patients peuvent être soignés sans que le nerf optique reçoive une dose de rayonnement trop grande, ce qui pourrait causer la cécité.

1. <http://www.ptcog.ch/>
2. <http://www.triumf.ca/proton-therapy>

On comprend qu'il est vital de pouvoir prédire et planifier la dose reçue par le patient. Dans ce contexte, la détermination de la portée des protons dans la matière en est le premier pas.

Pouvoir d'arrêt collisionnel des protons

Le formalisme associé au pouvoir d'arrêt collisionnel (ou électronique) fut pour la première fois énoncé par Bethe en 1933 suite à un développement perturbatif en mécanique quantique sous l'approximation de Born au premier ordre. Il permet de calculer la perte d'énergie par unité de distance d'une particule chargée dans un milieu donné. Des développements additionnels ont été réalisés par Barkas et Bloch afin de tenir compte des limites de cette approximation au-delà du premier ordre³. Ces termes d'ordre supérieur deviennent significatifs pour des ions de faible vitesse. Le pouvoir d'arrêt collisionnel massique s'exprime par

$$\frac{S_{col}}{\rho} = - \left(\frac{dT}{\rho dx} \right)_{col} = NZ \int_0^{T_e^{\max}} T' \left(\frac{d\sigma}{dT'} \right)_{col} dT' \quad (1)$$

où

- $\frac{d\sigma}{dT}$ est la section efficace différentielle pour les collisions inélastiques résultant d'une énergie transférée T' ,
- N est le nombre d'atomes par gramme,
- T' est l'énergie transférée à l'électron de l'atome,
- T_e^{\max} est l'énergie maximale transférable à un électron par le proton :

$$T_e^{\max} = \frac{2m_e c^2 (\gamma^2 - 1)}{1 + 2\gamma \frac{m_e}{m_p} + \left(\frac{m_e}{m_p}\right)^2} \quad (2)$$

avec γ représentant le facteur de Lorentz, m_p et m_e étant respectivement les énergies de masse du proton et de l'électron.

Avec la correction du modèle en couches, d'effet de la polarisation et des corrections d'ordres supérieurs en Z , on obtient le pouvoir d'arrêt collisionnel pour les protons:

$$-\left(\frac{dT}{dx} \right)_{col} = 2\pi r_e^2 m_e c^2 n_e \frac{Z^2}{\beta^2} \left[\ln \left(\frac{2m_e c^2 (\gamma^2 - 1) T_e^{\max}}{I^2} \right) - 2\beta^2 - \delta - 2\frac{C}{Z} + 2ZL \right]$$

avec

- r_e le rayon classique de l'électron,
- n_e la densité électronique du matériau,
- Z la charge de la particule ($Z = 1$ pour les protons),
- I l'énergie moyenne d'excitation du matériau, accessible sur [?],
- β et γ les facteurs de Lorentz,
- δ un terme considérant les effets de la polarisation,
- $2\frac{C}{Z}$ un terme corrigéant des effets du modèle en couches du cortège électronique atomique,

- ZL_1 la correction de Barkas, souvent nommée d'ordre 1 en Z mais, qui en réalité, induit une dépendance en Z^3 ,
- Z^2L_2 la correction de Bloch, nommée d'ordre 2 en Z d'où une dépendance en Z^4 .

Il est d'usage de négliger les termes correctifs pour les protons de plus de **3 MeV** en protonthérapie, ce qui mène à

$$S_{col}(T) = 2\pi r_e^2 m_e c^2 n_e \frac{1}{\beta^2} \left[\ln\left(\frac{2m_e c^2 \beta^2 \gamma^2 T_e^{\max}}{I^2}\right) - 2\beta^2 \right] \quad (4)$$

Dans cette expression, le matériau dans lequel se propage la particule chargée est défini par les termes n_e et I seulement. Ce modèle explique bien les données expérimentales pour les protons de plus de 3 MeV ; aussi, dans ce TP, on utilisera cette valeur comme borne d'intégration. Le biais introduit dans les résultats par cette approximation ne vous sera pas reproché.

Les particules chargées se propageant dans la matière peuvent aussi engendrer des interactions nucléaires et subir des pertes radiatives (surtout pour les particules légères comme les électrons) mais dans ce TP, seul le pouvoir d'arrêt collisionnel (électronique) sera considéré.

En général, les accélérateurs dédiés à la protonthérapie produisent des faisceaux de particules dans la gamme 70 – 250 MeV. En première approximation, on peut considérer les humains comme étant constitués d'eau liquide.

Questions 1.

Exprimez la densité électronique n_e d'un milieu en fonction de sa composition atomique et de sa masse volumique ρ , et calculer n_e pour l'eau (liquide) et l'os compact (définition de l'ICRU). On s'appuiera sur les données du NIST⁴ pour les compositions atomiques de ces matériaux. Vous trouverez aussi les énergies moyennes d'excitation I de ces matériaux sur le site du NIST. Tracez les courbes des pouvoirs d'arrêt collisionnel pour ces milieux. On utilisera une échelle logarithmique en abscisse.

Le nombre d'électrons dans un échantillon est la somme du nombre d'électrons que possède chaque élément constituant l'échantillon. Un échantillon ne contenant que de l'hydrogène ne possède que N électrons, où N est aussi le nombre d'atomes d'hydrogène contenus dans l'échantillon. Dans une mole, il y a un nombre d'Avogadro N_A d'atomes.

Le nombre d'électrons par gramme d'un élément i est

$$n_i = \frac{N_A Z_i}{A_i} \left[\frac{\text{atome}}{\text{mole}} \frac{\text{électron}}{\text{atome}} \frac{\text{mole}}{\text{g}} = \frac{e^-}{\text{g}} \right]$$

avec Z_i le nombre d'électrons de l'élément i , i.e. son numéro atomique, et A_i la masse molaire de cet élément.

Dans un échantillon, le nombre d'électrons total est donc

$$n = N_A \sum_i \omega_i \frac{Z_i}{A_i}$$

où ω_i est la fraction massique de chaque élément.

La densité électronique est alors

$$n_e = \rho N_A \sum_i \omega_i \frac{Z_i}{A_i} \quad \left[\frac{e^-}{m^3} \right]$$

avec ρ la densité masse volumique en $\left[\frac{g}{m^3} \right]$

On retrouve les facteurs de Lorentz β et γ à partir de l'équation

$$\beta = \frac{v}{c} \quad \gamma = \frac{1}{\sqrt{1 - \frac{v^2}{c^2}}} = \frac{1}{\sqrt{1 - \beta^2}}$$

avec l'énergie totale d'une particule étant la somme de son énergie cinétique et de son énergie au repos en joules

$$E = E_C + E_0 = \frac{1}{2}mv^2 + mc^2 = \gamma m_0 c^2$$

En isolant le facteur de Lorentz, on obtiens

$$\gamma = \frac{E_c}{m_0 c^2} + \frac{E_0}{m_0 c^2} = \frac{E_c}{m_0 c^2} + 1$$

Si l'on veut le facteur de Lorentz pour une énergie en électronvolts T , on multiplie par la charge élémentaire e .

$$\gamma = \frac{T \cdot e}{m_0 c^2} + 1$$

Et finalement, le facteur beta se trouve avec

$$\beta = \sqrt{1 - \frac{1}{\gamma^2}}$$

<https://cds.cern.ch/record/2874048/files/PS-PA-Note-95-26.pdf>

```
In [199]: def calculate_n_e(rho: float, w_is: np.ndarray, Z_is: np.ndarray, A_is: np.ndarray):
    """
    Calculates the electronic density of a given material based on its mass density
    """
    pass
```

```

:param rho: mass density of the material [kg/m^3]
:param w_is: mass fractions of the constituant elements
:param Z_is: atomic numbers of the constituant elements
:param A_is: atomic masses of the constituant elements [g/mol]

:return: electronic density of the material [e-/m^3]
:rtype: float
"""

# asserts
try:
    assert isinstance(w_is, np.ndarray)
    assert isinstance(Z_is, np.ndarray)
    assert isinstance(A_is, np.ndarray)
except Exception as e:
    print(e)

# calculate n_e
return 1e3 * rho * cte.N_A * np.sum(w_is * Z_is / A_is)      # 1e3 -> g to kg


def calculate_lorentz_factors(T: np.ndarray) -> list[np.ndarray, np.ndarray]:
"""
calculates the lorentz beta and gamma factors for a proton with a given kinetic

:param T: kinetic energy of the proton [J]

:return: lorentz beta and gamma factors [-]
:rtype: list[np.ndarray, np.ndarray]
"""
# asserts
assert isinstance(T, np.ndarray)

# calculate lorentz factors
gamma = 1 + T / (cte.m_p * cte.c ** 2)
beta = np.sqrt(1 - 1 / gamma ** 2)

return [beta, gamma]


def calculate_S_col(n_e: float, I: float, T: float | np.ndarray) -> np.ndarray:
"""
calculates the collisional stopping power of a specified material

:param n_e: electronic density of the material [m^-3]
:param I: mean excitation energy of the material [J]
:param T: kinetic energy of the proton [J]

:return: numpy array of the collisional stopping power [J/m]
:rtype: np.ndarray
"""

# asserts
assert isinstance(n_e, float)
assert isinstance(I, float)
T = np.asarray(T)

```

```
# calculate constants
r_e = cte.physical_constants['classical electron radius'][0]
beta, gamma = calculate_lorentz_factors(T=T)
T_e_max = (2 * cte.m_e * (cte.c ** 2) * ((gamma ** 2) - 1)) / (1 + 2 * gamma * beta)

# calculate S_col [J/m]
return 2 * cte.pi * (r_e ** 2) * cte.m_e * (cte.c ** 2) * n_e * (1 / (beta ** 2))
```

In [200...]

```
water_density = 1.0e3 # [kg/m^3]
water_mean_excitation_energy = 75.00 * cte.e # [J]

water_Z_is = np.array([1, 8])
water_w_is = np.array([0.111894, 0.888106])
water_A_is = np.array([1.00794, 15.9994])

# bone data
bone_density = 1.85e3 # [kg/m^3]
bone_mean_excitation_energy = 91.90 * cte.e # [J]

bone_Z_is = np.array([1, 6, 7, 8, 12, 15, 16, 20])
bone_w_is = np.array([0.063984, 0.278000, 0.027000, 0.410016, 0.002000, 0.070000, 0.000000, 0.000000])
bone_A_is = np.array([1.00794, 12.011, 14.0067, 15.9994, 24.305, 30.97376, 32.066, 33.000000])

Ts_eV = np.geomspace(1, 1000, 100) * 1e6 # 3e6 eV -> 1e9 eV
Ts_J = cte.e * Ts_eV # même range, mais en J

conv_factor = (1 / cte.e) * 1e-2 # J/m -> eV/cm

# calculate n_e for both materials
water_n_e = calculate_n_e(
    rho=water_density,
    w_is=water_w_is,
    Z_is=water_Z_is,
    A_is=water_A_is
)
bone_n_e = calculate_n_e(
    rho=bone_density,
    w_is=bone_w_is,
    Z_is=bone_Z_is,
    A_is=bone_A_is
)
print(f"electronic density of water: {water_n_e} e^-/m^3\nelectronic density of bone: {bone_n_e} e^-/m^3")

# calculate S_col for both materials
water_Scol = calculate_S_col(n_e=water_n_e, I=water_mean_excitation_energy, T=Ts_J)
bone_Scol = calculate_S_col(n_e=bone_n_e, I=bone_mean_excitation_energy, T=Ts_J)

plt.figure(figsize=(11, 6))

plt.plot(Ts_eV, water_Scol * conv_factor, lw=2, label="Water")
plt.plot(Ts_eV, bone_Scol * conv_factor, lw=2, label="Bone")

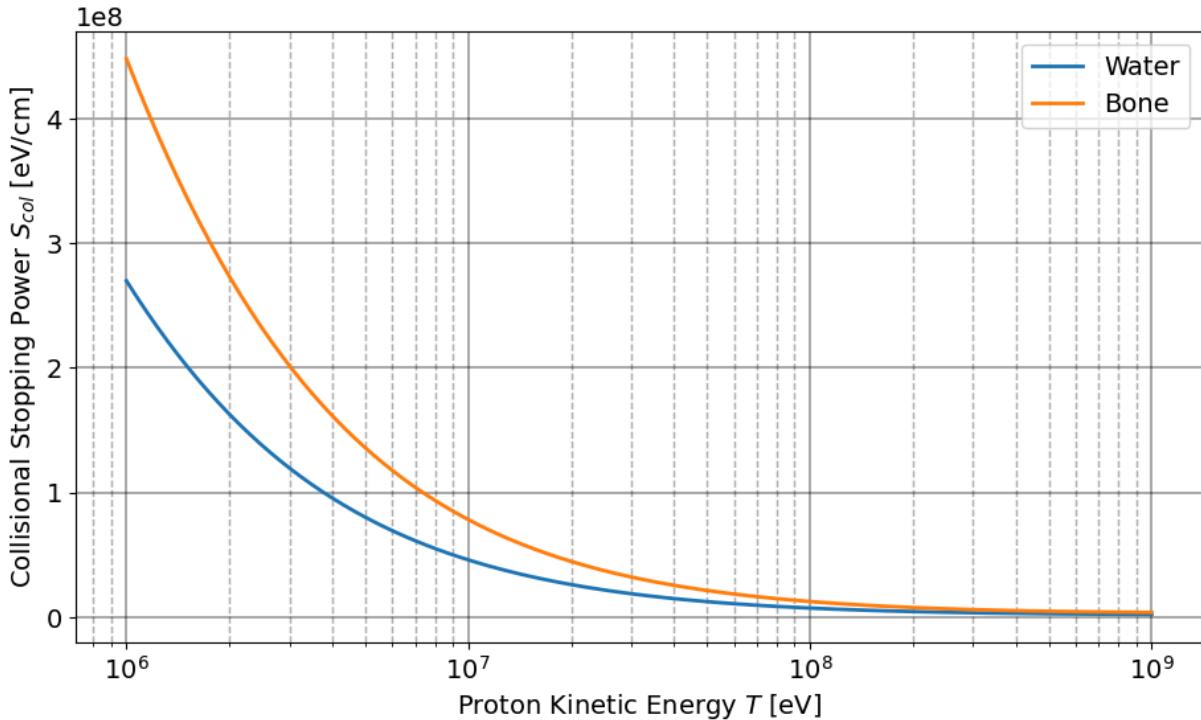
plt.xlabel(r"Proton Kinetic Energy $T$ [eV]")
plt.ylabel(r"Collisional Stopping Power $S_{\text{col}}$ [eV/cm]")
plt.xscale('log')
```

```

plt.grid(True, which='major', linestyle='-', linewidth=1.2, alpha=0.4, color='k')
plt.grid(True, which='minor', linestyle='--', linewidth=1)
plt.legend()
plt.show()

```

electronic density of water: 3.342783219356227e+29 e-/m³
electronic density of bone: 5.905866655971463e+29 e-/m³



Portée des protons dans la matière

L'approximation d'une décélération continue (CSDA) des protons dans la matière, en ligne droite, permet de calculer leur portée dans le milieu considéré. La portée par CSDA (R_{CSDA}) est obtenue en intégrant l'inverse du pouvoir d'arrêt total par rapport à l'énergie :

$$R_{CSDA} = \int_0^{T_i} \frac{dT'}{\frac{S_{col}}{\rho}} \quad (5)$$

Questions 2.

Déterminer l'homogénéité dimensionnelle de R_{CSDA} et expliquer en quelques phrases ce que représente l'équation 5 .

L'homogénéité dimensionnelle est le principe qui dicte que les deux membres d'une égalité doivent avoir la même dimension, soit les mêmes unités.

Il faut donc vérifier le terme de droite, soit l'intégrale, afin de déterminer la dimension de la portée par CSDA.

$$R_{CSDA} = \int_0^{T_i} \frac{\rho}{S_{col}} dT'$$

L'élément différentiel dT' a des unités d'énergie, en joules J . L'intégrande a des unités de

$$\frac{kg/m^3}{J/m} \Rightarrow \frac{kg}{J \cdot m^2}$$

Le terme de droite a donc la dimension kg/m^2 , ce qui fait que la portée par CSDA R_{CSDA} est en kg/m^2 .

L'équation 5 sert à universaliser les mesures de pénétration des protons dans un matériau. L'analyse dimensionnelle révèle que sa dimension est une masse par surface, ce qui revient à dire qu'un proton d'énergie T sera arrêté par n'importe quelle surface dont la masse par surface donnée est équivalente à sa portée R_{CSDA} . Ainsi, une particule ayant une portée de 5 kg/m^2 sera arrêtée par n'importe quel matériau ayant cette propriété, soit par exemple une plaque d'acier d'épaisseur x ou une couche d'air d'épaisseur y . L'air étant moins dense, il faudra que y soit bien plus élevé que x pour atteindre le 5 kg/m^2 voulu.

Questions 3.

Justifiez que pour les protons, le pouvoir d'arrêt total est bien approximé par le pouvoir d'arrêt collisionnel aux énergies de la protonthérapie. Discutez des deux autres composantes négligées ici : réactions nucléaires et pertes radiatives. Aide : prendre l'exemple de l'eau et appuyez-vous sur PSTAR du NIST⁵.

```
In [201...]: column_names = [
    "Kinetic_Energy_MeV",
    "Electronic_Stopping_Power_MeV_cm2_g",
    "Nuclear_Stopping_Power_MeV_cm2_g",
    "Total_Stopping_Power_MeV_cm2_g"
]

data_df = pd.read_csv("Q3_PSTAR Stopping Powers and Range Tab.txt", sep=r"\s+", skiprows=1, usecols=[0, 1, 2])
T, ESP, NSP, TSP = data_df[column_names[0]].to_numpy(), data_df[column_names[1]].to_numpy(), data_df[column_names[2]].to_numpy()

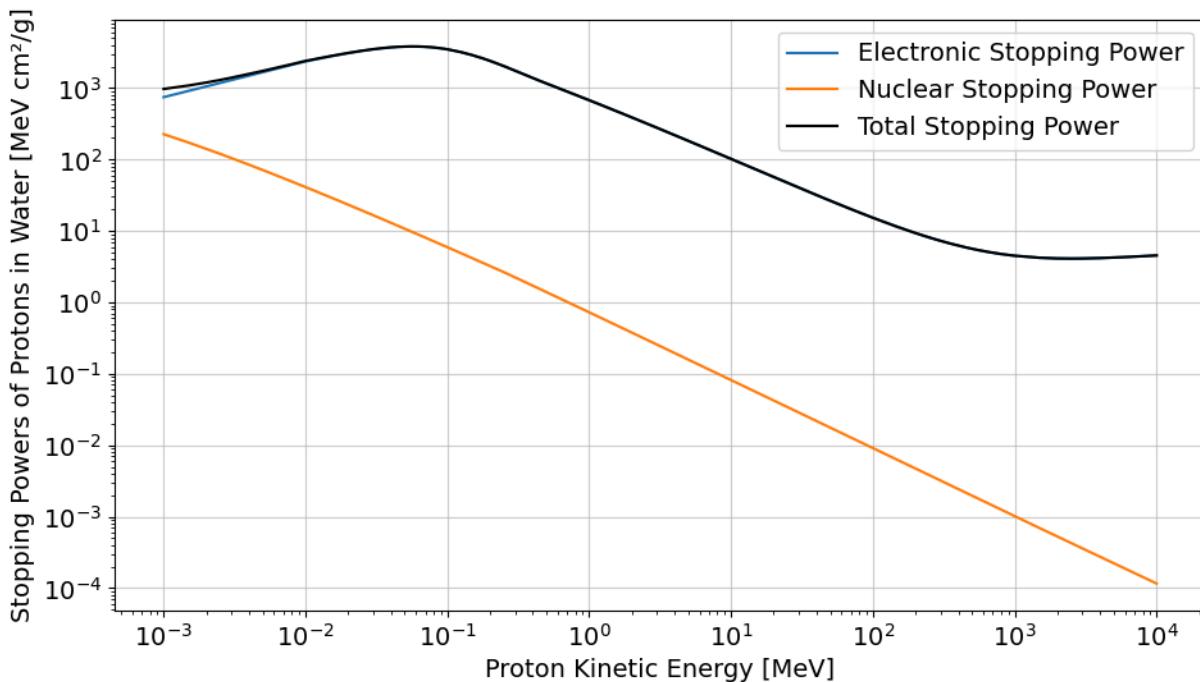
plt.figure(figsize=(11, 6))

plt.plot(T, ESP, label=f"{column_names[1]}[:-10].replace('_', ' ')")
plt.plot(T, NSP, label=f"{column_names[2]}[:-10].replace('_', ' ')")
plt.plot(T, TSP, 'k', label=f"{column_names[3]}[:-10].replace('_', ' ')")

plt.xscale('log')
plt.yscale('log')

plt.ylabel("Stopping Powers of Protons in Water [MeV cm²/g]")
plt.xlabel("Proton Kinetic Energy [MeV]")
plt.grid(alpha=0.6)
```

```
plt.legend()
plt.show()
```



In [202...]

```
# at 100MeV
idx = np.where(T == 100)[0]
print(f"Fraction du pouvoir d'arrêt total qui provient des collisions: {(ESP[idx] /
```

Fraction du pouvoir d'arrêt total qui provient des collisions: 99.935%

Fraction du pouvoir d'arrêt total qui provient du nucléaire : 0.059%

Le graphique et les données du PSTAR démontrent que pour les protons aux énergies utilisées en protonthérapie, le pouvoir d'arrêt total est presque exclusivement dicté par le pouvoir d'arrêt collisionnel. À 100 MeV, les données indiquent que le pouvoir d'arrêt nucléaire ne représente que 0,059 % de la perte d'énergie totale, ce qui rend cette composante négligeable dans le calcul de la portée. Quant aux pertes radiatives par rayonnement de freinage (bremsstrahlung), elles sont physiquement insignifiantes pour les protons, car ce mécanisme est inversement proportionnel au carré de la masse de la particule

$$S_{rad} \propto \frac{1}{m^2}$$

Le proton étant environ 1836 fois plus massif que l'électron, il ne rayonne pratiquement pas d'énergie à basse énergie. Enfin, bien que des réactions nucléaires non élastiques se produisent, elles entraînent principalement une atténuation de l'intensité du faisceau ou la production de particules secondaires plutôt qu'un changement significatif du processus de ralentissement continu, confirmant ainsi que l'approximation $S_{tot} \approx S_{col}$ est robuste.

Questions 4.

Justifiez la nécessité d'employer une méthode numérique pour calculer la portée des protons.

L'emploi d'une méthode numérique est indispensable, car la portée des protons est définie par l'intégrale de l'inverse du pouvoir d'arrêt ($1/S_E$), une grandeur physique régie par la formule de Bethe-Bloch définie plus haut et qui ne possède pas de primitive analytique simple. La complexité mathématique de cette fonction, qui inclut des variations logarithmiques et des dépendances non linéaires selon l'énergie, rend la résolution analytique impossible, ce qui force l'utilisation des méthodes d'intégration numérique.

Questions 5.

Implémenter deux algorithmes d'intégration numérique pour calculer la portée des protons dans l'eau et dans l'os compact; le premier avec la méthode des trapèzes et le second avec la méthode de Simpson. Considérez des protons de 150 MeV . Tracez un graphique de la portée calculée par chaque méthode en fonction du nombre d'échantillons (de tranches) considéré. On déterminera à l'avance le nombre de tranches nécessaires pour atteindre une erreur de l'ordre de la précision machine en Python, et on utilisera ce nombre (où un nombre de cet ordre de grandeur) comme valeur maximale (il y aura une valeur maximale pour la méthode des trapèzes et une autre pour la méthode de Simpson). Votre graphique comprendra des points choisis de façon à bien représenter le comportement de vos algorithmes (des échelles logarithmiques pourraient être nécessaires). Doubler le nombre de tranches entre chaque évaluation pourrait s'avérer judicieux pour les questions suivantes.

L'erreur totale avec la méthode des trapèzes est donnée par

$$\varepsilon = \frac{C}{N^2} + \epsilon \sqrt{N}$$

avec N le nombre de tranches et ϵ l'erreur machine, soit 10^{-16} .

Pour la méthode de simpson, elle suit l'équation

$$\varepsilon = \frac{C}{N^4} + \epsilon \sqrt{N}$$

Dans les deux cas, C est une constante qui dépend de la fonction intégrée, soit celle de la portée des protons dans ce cas-ci.

Il n'est pas très important d'en connaître la valeur puisque c'est la tendance des équations qui nous intéresse à des valeurs de N élevées. Pour $N = 2^{20} = 1048576 \approx 10^6$, ε tend vers $\epsilon \sqrt{N}$, soit $10^{-16} \cdot 10^3 = 10^{-13}$. Pour obtenir une erreur équivalente à l'erreur machine sur notre résultat d'intégrale, il faudrait utiliser un nombre de tranches inférieur à quelques

ordres de grandeur, entre 10 et 100 par exemple. Cela est impossible puisque dans un tel cas, le premier terme des équations ci-hauts domine.

```
In [203...]:
```

```
def get_material_properties(material_name: str):
    """
    returns (rho, n_e, I) for the specified material
    """
    properties = {
        "water": (water_density, water_n_e, water_mean_excitation_energy),
        "bone": (bone_density, bone_n_e, bone_mean_excitation_energy)
    }
    return properties.get(material_name.lower())

def trapeze(function: Callable[[float], float], a: float, b: float, N: int) -> float:
    """
    approximates the definite integral of function from a to b using the Trapezoidal rule

    :param function: name of the function to integrate
    :param a: lower bound of integration
    :param b: upper bound of integration
    :param N: number of subdivisions

    :returns: calculated area
    :rtype: float
    """
    h = (b - a) / N

    s = 0.5 * function(a) + 0.5 * function(b)

    for k in range(1, N):
        s += function(a + k * h)

    return h * s

def simpson(function: Callable[[float], float], a: float, b: float, N: int) -> float:
    """
    approximates the definite integral of function from a to b using the Simpson's rule

    Note: N must be an even integer for this implementation of the Simpson's rule

    :param function: name of the function to integrate
    :param a: lower bound of integration
    :param b: upper bound of integration
    :param N: number of even subdivisions

    :returns: calculated area
    :rtype: float
    """
    h = (b - a) / N
    s1 = 0.0

    for k in range(1, N, 2):
        s1 += function(a + k * h)
```

```

s2 = 0.0
for k in range(2, N, 2):
    s2 += function(a + k * h)

return (function(a) + function(b) + 4.0 * s1 + 2.0 * s2) * h / 3.0

def run_analysis(material_name, N_trap, N_simp):
    """
    runs the integration the specified material's CSDA range using both the trapeze

    Note: N must be an even integer for this implementation of the Simpson's rule

    :param material_name: used material
    :param N_trap: number of subdivisions for the trapeze method
    :param N_simp: number of subdivisions for the simpson method

    :returns: array of the number of subdivisions as axis 0 and of the calculated a
    :rtype: np.ndarray
    """
    rho, n_e, I = get_material_properties(material_name)
    f_inv_S = lambda T: (rho / calculate_S_col(n_e, I, T))      # [kg/J m^2]

    T_proton_low = 3e6 * cte.e        # MeV -> J
    T_proton_high = 150e6 * cte.e     # MeV -> J

    trap_results = np.array([trapeze(f_inv_S, T_proton_low, T_proton_high, n) for n
    simp_results = np.array([simpson(f_inv_S, T_proton_low, T_proton_high, n) for n

    return trap_results, simp_results

```

La précision machine sur un nombre flottant 64 bits `float` est de $\epsilon = 5 \times 10^{-16}$. On veut calculer l'erreur d'intégration à chaque fois que l'on double le nombre de tranches et s'arrêter lorsque ces deux erreurs sont du même ordre de grandeur.

L'erreur sur la i -ième estimation est

$$\epsilon_i = \frac{1}{3}(I_i - I_{i-1})$$

où I_i est le résultat de l'intégration et I_{i-1} est le résultat de l'itération précédente avec la moitié du nombre de tranches.

In [204...]

```

def calculate_epsilon_trap(I_2N, I_N):
    """
    Return integration error estimate for scalars or arrays
    """
    I_2N = np.asarray(I_2N)
    I_N = np.asarray(I_N)
    return (1.0 / 3.0) * (I_2N - I_N)

def calculate_epsilon_simp(I_2N, I_N):
    """

```

```
Return integration error estimate for scalars or arrays
"""
I_2N = np.asarray(I_2N)
I_N = np.asarray(I_N)
return (1.0 / 15.0) * (I_2N - I_N)
```

In [205...]

```
epsilon_limit = 5e-16

# bit shifting to have an array go from 2^1 -> 2^24 (16777216)
N = [1 << (i + 1) for i in range(20)] # set range(24) for full test length

water_R_CSDA_trap_previous, water_R_CSDA_simp_previous = run_analysis("water", [N[0]])
print(f"R_CSDA for water with 2 slices (trapèze): {water_R_CSDA_trap_previous} g/cm²")

epsilon_trap = []
epsilon_simp = []
for k, i in enumerate(N[1:]):
    water_R_CSDA_trap, water_R_CSDA_simp = run_analysis("water", [i], [i])

    epsilon_trap.append(calculate_epsilon_trap(water_R_CSDA_trap, water_R_CSDA_trap_previous))
    epsilon_simp.append(calculate_epsilon_simp(water_R_CSDA_simp, water_R_CSDA_simp_previous))

    # update previous results
    water_R_CSDA_trap_previous = water_R_CSDA_trap
    water_R_CSDA_simp_previous = water_R_CSDA_simp

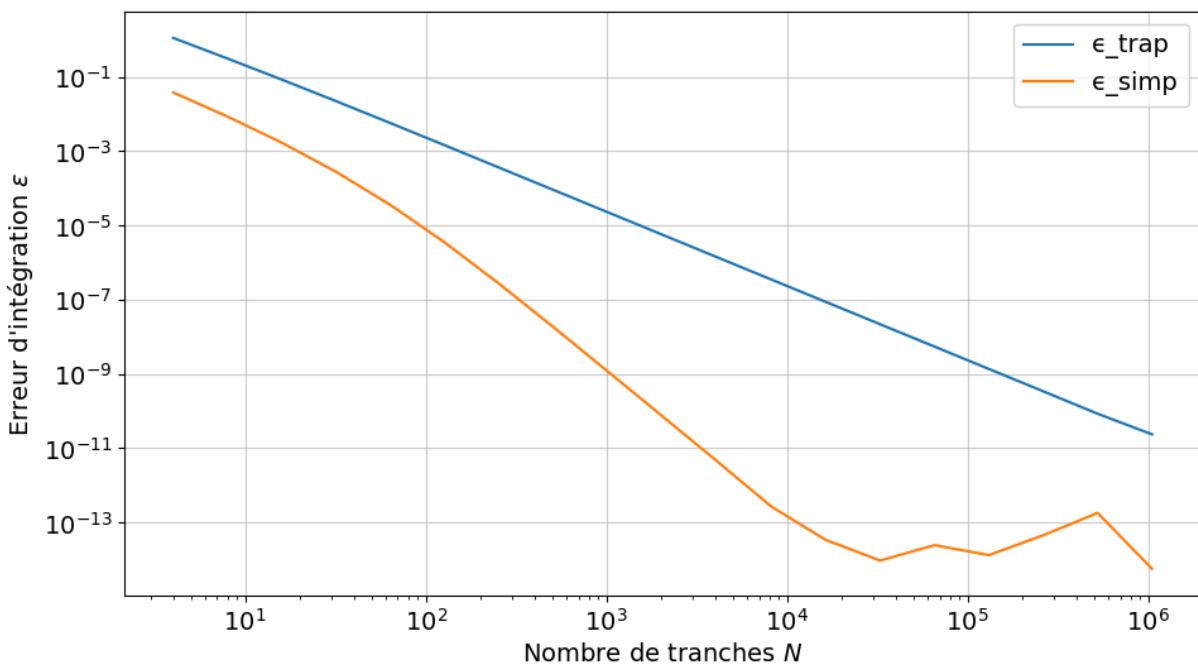
plt.figure(figsize=(11, 6))

plt.plot(N[1:], np.abs(epsilon_trap), label="epsilon_trap")
plt.plot(N[1:], np.abs(epsilon_simp), label="epsilon_simp")

plt.xscale('log')
plt.yscale('log')
plt.xlabel(r"Nombre de tranches $N$")
plt.ylabel(r"Erreur d'intégration $\epsilon$")

plt.legend()
plt.grid(alpha=0.6)
plt.show()
```

R_CSDA for water with 2 slices (trapèze): [9.54226626e+18] g/cm²



Voilà la méthode qui sera programmée. Pour atteindre la précision machine. On définit une fonction pour faire une `while` en doublant à chaque fois le nombre de rectangles. À chaque étape, on calcule l'erreur, on stocke aussi la valeur de l'intégrale dans un array pour ploter. Si l'erreur est plus grande que la limite que nous nous sommes mise, alors on augmente.

```
In [206...]: def calculate_R_CSDA_until_machine_error(material_name: str, integration_method: str):
    """
    calculates the R_CSDA for a given material until the error of the integration is
    inputs:
        material_name: name of the material to calculate the R_CSDA for
        integration_method: method to use for the integration (trapeze or simpson)
        ε_limit: error limit to stop the calculation
    outputs:
        N: number of subdivisions used for the final calculation
        R_CSDA_values: list of calculated R_CSDA values for each iteration
        ε_values: list of calculated error values for each iteration
    """
    rho, n_e, I = get_material_properties(material_name)
    f_inv_S = lambda T: (rho / calculate_S_col(n_e, I, T)) # [kg/J m^2]

    N = 2
    T_proton_low, T_proton_high = 3e6*cte.e, 150e6*cte.e # MeV -> J
    R_CSDA_values, ε_values = [], []

    R_CSDA_previous = integration_method(f_inv_S, T_proton_low, T_proton_high, N)
    R_CSDA_values.append(R_CSDA_previous)

    # first error set at -infty
    ε_values.append(np.inf)

    while True:
```

```

N *= 2
R_CSDA_current = integration_method(f_inv_S, T_proton_low, T_proton_high, N)
R_CSDA_values.append(R_CSDA_current)

if integration_method == trapeze:
    ε_current = calculate_ε_trap(R_CSDA_current, R_CSDA_previous)
elif integration_method == simpson:
    ε_current = calculate_ε_simp(R_CSDA_current, R_CSDA_previous)
else:
    raise ValueError("Invalid integration method. Choose 'trapeze' or 'simpson'")

ε_values.append(ε_current)

# Break condition when the error is less than the specified limit
if np.abs(ε_current) < ε_limit:
    break

R_CSDA_previous = R_CSDA_current

return N, R_CSDA_values, ε_values

```

In [207...]

```

ε_limit = 5e-12
ε_limit_2 = (np.finfo(float).eps) # hardware fundamental epsilon for double precision

print(f"Machine epsilon for double precision: {ε_limit}")
print(f"Machine epsilon for double precision: {ε_limit_2}")

N_trap_final, water_R_CSDA_trap_values, water_ε_trap_values = calculate_R_CSDA_until_error(ε_limit)
print(f"Final number of slices for Trapezoidal method to reach error < {ε_limit}: {N_trap_final}")

N_simp_final, water_R_CSDA_simp_values, water_ε_simp_values = calculate_R_CSDA_until_error(ε_limit)
print(f"Final number of slices for Simpson's method to reach error < {ε_limit}: {N_simp_final}")

```

Machine epsilon for double precision: 5e-12
 Machine epsilon for double precision: 2.220446049250313e-16
 Final number of slices for Trapezoidal method to reach error < 5e-12: 4194304
 Final number of slices for Simpson's method to reach error < 5e-12: 4096

In []:

```

N_simp_final = 8192
N_trap_final = 2097152
# We chose different limits so the number of slices is different from the previous

N_trap = [1 << i for i in range(1, N_trap_final.bit_length()) if (1 << i) <= N_trap]
N_simp = [1 << i for i in range(1, N_simp_final.bit_length()) if (1 << i) <= N_simp]

# Run analysis with the calculated ranges
water_R_CSDA_trap, _ = run_analysis("water", N_trap, [2])
_, water_R_CSDA_simp = run_analysis("water", [2], N_simp)

bone_R_CSDA_trap, _ = run_analysis("bone", N_trap, [2])
_, bone_R_CSDA_simp = run_analysis("bone", [2], N_simp)

conv_factor = 1e-1 # [kg/m^2] -> [g/cm^2]

plt.figure(figsize=(11, 6))

```

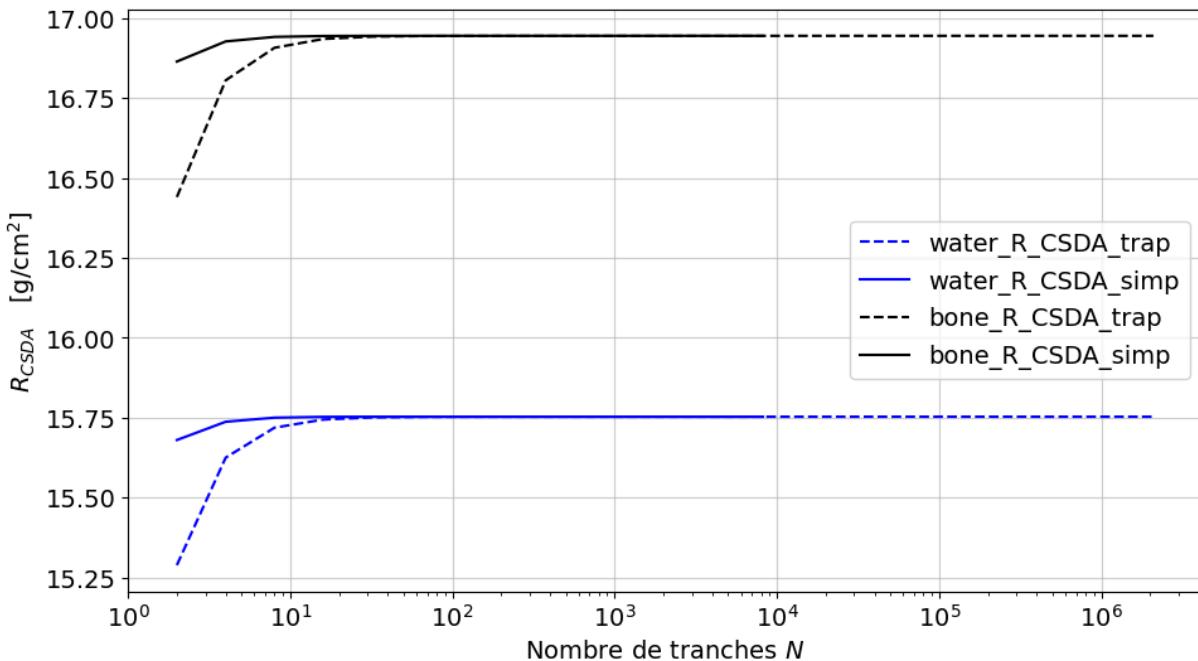
```

plt.plot(N_trap, water_R_CSDA_trap * conv_factor, 'b', ls='--', label="water_R_CSDA_trap")
plt.plot(N_simp, water_R_CSDA_simp * conv_factor, 'b', label="water_R_CSDA_simp")
plt.plot(N_trap, bone_R_CSDA_trap * conv_factor, 'k', ls='--', label="bone_R_CSDA_trap")
plt.plot(N_simp, bone_R_CSDA_simp * conv_factor, 'k', label="bone_R_CSDA_simp")

plt.xlabel(r"Nombre de tranches $N$")
plt.ylabel(r"$R_{CSDA} \quad [g/cm^2]$")

plt.xscale('log')
plt.grid(alpha=0.6)
plt.legend(loc='center right')
plt.show()

```



Questions 6.

Pour une méthode d'intégration numérique au choix, mesurez le temps de calcul en fonction du nombre de tranches N . Comparez les résultats obtenus sur au moins deux processeurs (CPU) et commentez les différences observées.

In [210...]

```

%%timeit -r 3 -n 1    # run cell 3 times with a repetition of 5

#water_R_CSDA_trap, _ = run_analysis("water", N_trap, [2])    # trapèzesLower
#pass

```

The slowest run took 5.00 times longer than the fastest. This could mean that an intermediate result is being cached.

467 ns ± 377 ns per loop (mean ± std. dev. of 3 runs, 1 loop each)

In [211...]

```

# Clément: intel® 14-core i5-13600KF @ 3.50 GHz      --> 17 s ± 263 ms
# Éloi:    Snapdragon(R) X 10-core X1P64100 @ 3.40 GHz --> 45.2 s ± 2.19 s
print(f"Augmentation de {(45.2/17 * 100):.1f} % du temps de calcul")

```

Augmentation de 265.9 % du temps de calcul

La cellule du haut s'exécutera 3 fois de suite avant d'être re-répété 5 fois d'affiler. À chaque itération, le nombre de temps requis pour faire le calcul est calculé avec son écart-type. Le nombre de trapèze intégrer à chaque fois est de 2097152. Le nombre de tranches pour Simpson est de 2 pour mesurer seulement le temps de l'intégration avec la méthode du trapèze.

On remarque une augmentation de 265% entre le X10-core et le 13600. On observe que la performance d'un calcul numérique est fortement dépendante de la machine qui execute le code, ce qui n'est pas très surprenant. La différence de performance est principalement due à la fréquence d'horloge du processeur, au nombre de coeurs et à l'efficacité de l'architecture du CPU. Le processeur Snapdragon est un processeur mobile conçu pour l'efficacité énergétique, tandis que le processeur Intel i7 est un processeur de bureau plus puissant avec une architecture optimisée pour les performances. Durant ce test, le Snapdragon n'était pas branché à l'alimentation, il était à moins de 20% de batterie en mode économie d'énergie ce qui démontre que les conditions d'exécution du code peuvent aussi influencer les résultats obtenus. Une métrique plus importante pour les ordinateurs portables serait le nombre de secondes par watt consommé, mais ce n'était pas l'objectif de cette question.

REF: <https://docs.python.org/3/library/timeit.html>

Questions 7.

Établir l'expression analytique de la dérivée du pouvoir d'arrêt en fonction de T et la tracer. On utilisera une échelle logarithme en abscisse. Aide : Exprimez le pouvoir d'arrêt en fonction de γ et utiliser le théorème de dérivation des fonctions composées. Utilisez aussi les définitions suivantes pour simplifier la notation :

$$T_e^{\max} = \frac{a(\gamma^2 - 1)}{b + \delta\gamma}, \quad \text{avec} \quad a = 2m_e c^2, \quad b = 1 + \left(\frac{m_e}{m_p}\right)^2 \quad \text{et} \quad \delta = 2\frac{m_e}{m_p}. \quad (6)$$

et

$$U = 2\pi r_e^2 m_e c^2 n_e$$

$$k = \frac{a^2}{I^2} \quad (7)$$

sachant aussi que

$$T = (\gamma - 1)m_p c^2 \Rightarrow \gamma = \frac{T}{m_p c^2} + 1 \quad (8)$$

$$\gamma = \frac{1}{\sqrt{1 - \beta^2}} \Rightarrow \gamma^2 \beta^2 = (\gamma^2 - 1) \quad (9)$$



Le pouvoir d'arrêt est défini comme

$$S_{col}(T) = 2\pi r_e^2 m_e c^2 n_e \frac{1}{\beta^2} \left[\ln \left[\frac{2m_e c^2 \beta^2 \gamma^2 T_e^{max}}{I^2} \right] - 2\beta^2 \right]$$

et

$$\gamma = \frac{1}{\sqrt{1 - \beta^2}} \Rightarrow \gamma^2 \beta^2 = \gamma^2 - 1 \Rightarrow \beta^2 = 1 - \frac{1}{\gamma^2}$$

En l'exprimant en fonction de γ , l'expression suivante est obtenue

$$\begin{aligned} S_{col}(T) &= 2\pi r_e^2 m_e c^2 n_e \frac{1}{1 - \frac{1}{\gamma^2}} \left[\ln \left[\frac{2m_e c^2 (\gamma^2 - 1) T_e^{max}}{I^2} \right] + \frac{2}{\gamma^2} - 2 \right] \\ &= \frac{U\gamma^2}{\gamma^2 - 1} \left[\ln \left[\frac{a(\gamma^2 - 1) T_e^{max}}{I^2} \right] + \frac{2}{\gamma^2} - 2 \right] = \frac{U\gamma^2}{\gamma^2 - 1} \left[\ln \left[\frac{a(\gamma^2 - 1)}{I^2} \frac{a(\gamma^2 - 1)}{b + \delta\gamma} \right] + \frac{2}{\gamma^2} - 2 \right] \\ &= \frac{U\gamma^2}{\gamma^2 - 1} \left[\ln \left[\frac{a^2(\gamma^2 - 1)^2}{I^2(b + \delta\gamma)} \right] + \frac{2}{\gamma^2} - 2 \right] = \frac{U\gamma^2}{\gamma^2 - 1} \left[\ln \left[\frac{k(\gamma^2 - 1)^2}{b + \delta\gamma} \right] + \frac{2}{\gamma^2} - 2 \right] \\ &= \frac{U\gamma^2}{\gamma^2 - 1} \ln \left[\frac{k(\gamma^2 - 1)^2}{b + \delta\gamma} \right] + \frac{2U}{\gamma^2 - 1} - \frac{2U\gamma^2}{\gamma^2 - 1} = \frac{U\gamma^2}{\gamma^2 - 1} \ln \left[\frac{k(\gamma^2 - 1)^2}{b + \delta\gamma} \right] + \frac{2U}{\gamma} \\ &= \frac{U\gamma^2}{\gamma^2 - 1} \ln \left[\frac{k(\gamma^2 - 1)^2}{b + \delta\gamma} \right] - 2U \end{aligned}$$

La dérivation en chaîne permet de trouver la dérivée analytique par rapport à T selon

$$\frac{dS_{col}(T)}{dT} = \frac{dS_{col}(T)}{d\gamma} \cdot \frac{d\gamma}{dT}$$

La seconde dérivée est facilement obtenue, soit

$$\frac{d\gamma}{dT} = \frac{d}{dT} \left[1 + \frac{T}{m_p c^2} \right] = \frac{1}{m_p c^2}$$

La première doit toutefois être décomposée comme

$$\frac{dS_{col}(T)}{d\gamma} = \frac{d}{d\gamma} \left[\frac{U\gamma^2}{\gamma^2 - 1} \ln \left[\frac{k(\gamma^2 - 1)^2}{b + \delta\gamma} \right] \right] - \frac{d}{d\gamma} 2U$$

d'où

$$\frac{d}{d\gamma} \left[\frac{U\gamma^2}{\gamma^2 - 1} \ln \left[\frac{k(\gamma^2 - 1)^2}{b + \delta\gamma} \right] \right] = \frac{U\gamma}{(\gamma^2 - 1)^2(b + \delta\gamma)} \left[\gamma[4b\gamma + \delta + 3\delta\gamma^2] - 2[b + \delta\gamma] \ln \left[\frac{k(\gamma^2 - 1)^2}{b + \delta\gamma} \right] \right]$$

$$\frac{d}{d\gamma} 2U = 0$$

et

$$\frac{dS_{col}(T)}{d\gamma} = \frac{U\gamma}{(\gamma^2 - 1)^2(b + \delta\gamma)} \left[\gamma[4b\gamma + \delta + 3\delta\gamma^2] - 2[b + \delta\gamma] \ln \left[\frac{k(\gamma^2 - 1)^2}{b + \delta\gamma} \right] \right]$$

De là, la dérivée recherchée est

$$\begin{aligned} \frac{dS_{col}(T)}{dT} &= \frac{U\gamma}{m_p c^2 (\gamma^2 - 1)^2 (b + \delta\gamma)} \left[\gamma[4b\gamma + \delta + 3\delta\gamma^2] - 2[b + \delta\gamma] \ln \left[\frac{k(\gamma^2 - 1)^2}{b + \delta\gamma} \right] \right] \\ &= \frac{2\pi r_e^2 m_e n_e \gamma}{m_p (\gamma^2 - 1)^2 \left(\frac{m_p^2}{m_p^2 + m_e^2} + \frac{2m_e}{m_p} \gamma \right)} \left[\gamma \left[\frac{4m_p^2 \gamma}{m_p^2 + m_e^2} + \frac{2m_e}{m_p} [1 + 3\gamma^2] \right] - 2 \left[\right. \right. \end{aligned}$$

où tous les termes ont été remplacés par leur valeur. Ce n'est pas la formule qui sera programmé, ce sera plutôt l'avant dernière.

```
In [212]: def Scol_derivative(n_e: float, I: float, T: float | np.ndarray) -> float | np.ndarray
"""
calculates the derivative of the collisional stopping power of a specified mate

:param n_e: electronic density of the material [m^-3]
:param I: mean excitation energy of the material [J]
:param T: kinetic energy of the proton [J]

:returns: derivative value or array for a specific energy or an energy array [m^-3]
:rtype: float | np.ndarray
"""

# constants
pi, m_e, c, m_p = cte.pi, cte.m_e, cte.c, cte.m_p
r_e = cte.physical_constants['classical electron radius'][0]
_, gamma = calculate_lorentz_factors(T)

# useful definitions
a = 2 * m_e * c ** 2
b = 1 + (m_e / m_p) ** 2
delta = 2 * (m_e / m_p)
U = 2 * pi * r_e ** 2 * m_e * c ** 2 * n_e
k = a ** 2 / I ** 2

fac1 = (U * gamma) / (m_p * c ** 2 * (gamma ** 2 - 1) ** 2 * (b + delta * gamma))
fac2 = gamma * (4 * b * gamma + delta + 3 * delta * gamma ** 2)
```

```

    fac3 = -2 * (b + delta * gamma) * np.log((k * (gamma ** 2 - 1) ** 2) / (b + del
    return fac1 * (fac2 + fac3)

```

In [213...]

```

water_density = 1.0e3                                # [kg/m^3]
water_mean_excitation_energy = 75.00 * cte.e        # [J]

water_Z_is = np.array([1, 8])
water_w_is = np.array([0.111894, 0.888106])
water_A_is = np.array([1.00794, 15.9994])

# bone data
bone_density = 1.85e3                                # [kg/m^3]
bone_mean_excitation_energy = 91.90 * cte.e          # [J]

bone_Z_is = np.array([1, 6, 7, 8, 12, 15, 16, 20])
bone_w_is = np.array([0.063984, 0.278000, 0.027000, 0.410016, 0.002000, 0.070000, 0
bone_A_is = np.array([1.00794, 12.011, 14.0067, 15.9994, 24.305, 30.97376, 32.066, 0

Ts_eV = np.geomspace(1, 1000, 100) * 1e6      # 3e6 eV -> 1e9 eV
Ts_J = cte.e * Ts_eV                           # même range, mais en J

conv_factor = 1e-2     # m^-1 -> cm^-1

# calculate n_e for both materials
water_n_e = calculate_n_e(
    rho=water_density,
    w_is=water_w_is,
    Z_is=water_Z_is,
    A_is=water_A_is
)
bone_n_e = calculate_n_e(
    rho=bone_density,
    w_is=bone_w_is,
    Z_is=bone_Z_is,
    A_is=bone_A_is
)
print(f"electronic density of water: {water_n_e} e^-/m^3\nelectronic density of bone: {bone_n_e} e^-/m^3")

# calculate Scol_dervtve for both materials
water_Scol_dervtve = Scol_derivative(n_e=water_n_e, I=water_mean_excitation_energy,
bone_Scol_dervtve = Scol_derivative(n_e=bone_n_e, I=bone_mean_excitation_energy, T=T)

plt.figure(figsize=(11, 6))

plt.plot(Ts_eV, water_Scol_dervtve * conv_factor, lw=2, label="Water")
plt.plot(Ts_eV, bone_Scol_dervtve * conv_factor, lw=2, label="Bone")

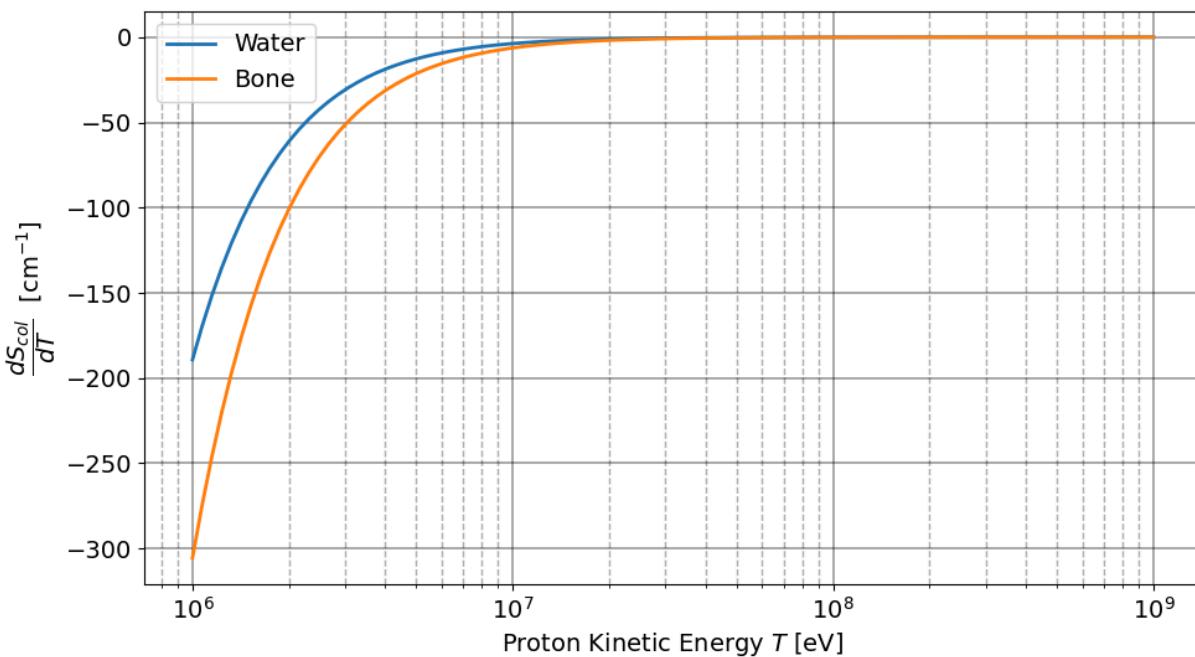
plt.xlabel(r"Proton Kinetic Energy $T$ [eV]")
plt.ylabel(r"$\frac{dS_{col}}{dT}$ [cm$^{-1}$]")
plt.xscale('log')

plt.grid(True, which='major', linestyle='-', linewidth=1.2, alpha=0.4, color='k')
plt.grid(True, which='minor', linestyle='--', linewidth=1)

```

```
plt.legend()
plt.show()
```

electronic density of water: 3.342783219356227e+29 e⁻/m³
 electronic density of bone: 5.905866655971463e+29 e⁻/m³



Questions 8.

Rapportez vos portées calculées dans un tableau, qui comprendra aussi les erreurs d'approximation calculées (pour la méthode des trapèzes) et évaluées de façon pratique (pour les deux méthodes). Si vous êtes vraiment courageuse ou courageux, vous pourriez aussi calculer les erreurs d'approximation pour Simpson, mais restons-en à l'évaluation pratique de l'erreur pour ce cas. Commentez vos observations.

Pour la méthode des trapèzes, on peut donner une erreur d'approximation calculée à partir de $\max |f''(T)|$. Pour les deux méthodes, on évalue aussi une erreur pratique par comparaison de deux intégrales successives (Richardson) :

- Trapèzes : $\epsilon_{\text{pratique}} \approx |I_N - I_{N/2}|/3$
- Simpson : $\epsilon_{\text{pratique}} \approx |I_N - I_{N/2}|/15$

Le tableau suivant récapitule les portées R_{CSDA} et les erreurs associées pour l'eau et l'os compact, avec les valeurs finales de N utilisées plus haut.

In [214...]

```
def trapeze_error_bound(f, a, b, N, num=20000):
    """
        Limite supérieur for trapezoidal error using a maximum |f''(T)|.
    """
    T = np.linspace(a, b, num)
    fvals = f(T)
    d2 = np.gradient(np.gradient(fvals, T), T)
```

```

max_d2 = np.max(np.abs(d2))
h = (b - a) / N
return (b - a) * h ** 2 * max_d2 / 12.0

def simp_error_bound(f, a, b, N, num=200000):
    """
    Limite supérieur for Simpson's error using max |f''''(T)|.
    """
    T = np.linspace(a, b, num)
    fvals = f(T)
    # Compute 4th derivative using successive np.gradient calls
    d1 = np.gradient(fvals, T)
    d2 = np.gradient(d1, T)
    d3 = np.gradient(d2, T)
    d4 = np.gradient(d3, T)
    max_d4 = np.max(np.abs(d4))
    h = (b - a) / N
    return (b - a) * h ** 4 * max_d4 / 180.0

# Nouvelle fonction ε avec N et N/2 pour une estimation plus pratique de l'erreur
def calculate_practical_error_(I_N, I_half, method):
    """Erreur pratique avec multiplication par deux du nombre de tranches"""
    if method == "trap":
        return np.abs(I_N - I_half) / 3.0
    if method == "simp":
        return np.abs(I_N - I_half) / 15.0
    raise ValueError("Error !!!")

def build_table(material_name, N_trap, N_simp):
    rho, n_e, I = get_material_properties(material_name)
    f_inv_S = lambda T: (rho / calculate_S_col(n_e, I, T))

    T_low = 3e6 * cte.e
    T_high = 150e6 * cte.e

    # Trap results at N and N/2
    I_trap = trapeze(f_inv_S, T_low, T_high, N_trap)
    I_trap_half = trapeze(f_inv_S, T_low, T_high, N_trap // 2)

    # Simpson results at N and N/2 (must be even)
    I_simp = simpson(f_inv_S, T_low, T_high, N_simp)
    I_simp_half = simpson(f_inv_S, T_low, T_high, N_simp // 2)

    # Errors
    err_trap_calc = trapeze_error_bound(f_inv_S, T_low, T_high, N_trap)
    err_simp_calc = simp_error_bound(f_inv_S, T_low, T_high, N_simp)
    err_trap_prac = calculate_practical_error_(I_trap, I_trap_half, "trap")
    err_simp_prac = calculate_practical_error_(I_simp, I_simp_half, "simp")

    conv = 1e-1 # kg/m^2 -> g/cm^2
    return [
        {
            "Materiau": material_name,

```

```

        "Methode": "Trapezes",
        "N": N_trap,
        "R_CSDA [g/cm^2)": I_trap * conv,
        "Erreur approx (calc) [g/cm^2)": err_trap_calc * conv,
        "Erreur pratique [g/cm^2)": err_trap_prac * conv,
    },
    {
        "Materiau": material_name,
        "Methode": "Simpson",
        "N": N_simp,
        "R_CSDA [g/cm^2)": I_simp * conv,
        "Erreur approx (calc) [g/cm^2)": err_simp_calc * conv,
        "Erreur pratique [g/cm^2)": err_simp_prac * conv,
    },
]

```

In [215...]: # Valeur finale de N pour les deux méthodes, obtenue précédemment

```

N_trap_final = 2097152
N_simp_final = 8192

rows = []
rows += build_table("water", N_trap_final, N_simp_final)
rows += build_table("bone", N_trap_final, N_simp_final)

q8_table = pd.DataFrame(rows)
q8_table

```

Out[215...]:

	Materiau	Methode	N	R_CSDA [g/cm^2]	Erreur approx (calc) [g/cm^2]	Erreur pratique [g/cm^2]
0	water	Trapezes	2097152	15.752946	7.224901e-12	5.002221e-13
1	water	Simpson	8192	15.752946	2.366587e-06	2.690588e-14
2	bone	Trapezes	2097152	16.944959	7.973360e-12	2.955858e-13
3	bone	Simpson	8192	16.944959	2.611714e-06	3.069545e-14

Optimisation

Supposons maintenant que l'on cherche à réduire au maximum le temps de calcul, disons pour évaluer en temps quasi-réel la portée de chaque proton individuel émanant de l'accélérateur et dont on connaît précisément l'énergie. Ces protons ont une distribution en énergie pouvant être approximée par une distribution de Moyal, que vous pourrez générer avec `scipy.stats.moyal` avec les paramètres `loc = 150, scale = 4` (unités en MeV).

Une distribution de Moyal possède la forme présentée à la question 9. On cherche à calculer la portée R_{CDSA} que chaque proton d'énergie cinétique T . Nous calculons ensuite le pouvoir d'arrêt pour ensuite l'intégrer avec une méthode rapide.

Nous calculons ici la portée dans l'eau , mais la même méthode pourrait être appliquée pour l'os compact ou n'importe quel autre matériau.

Nous allons utiliser la méthode de Simpson car elle nécessite moins de compute time pour arriver à une même précision. Nous allons utiliser un nombre de tranches qui garantit un calcul plus petit que 1 ms.

In [216...]

```
moyal_loc, moyal_scale, moyal_size = 150.0, 4.0, 10000

# Single proton kinetic energy (MeV)
T_moyal_single_value_MeV = sp.stats.moyal.rvs(loc=moyal_loc, scale=moyal_scale)
#T_moyal_single_value_MeV = 150.0 # to fix the value for testing
T_moyal_single_value_J = T_moyal_single_value_MeV * 1e6 * cte.e

rho_bone, n_e_bone, bone_mean_excitation_energy = get_material_properties("water")
N_simp_8_optimisation = 2**4

print("water mean excitation energy (I):", bone_mean_excitation_energy / cte.e, "eV")
print(f"water density (rho): {rho_bone} kg/m³")
print("water electronic density", n_e_bone, "e⁻/m³\n")
print(f"Kinetic energy T {T_moyal_single_value_MeV} MeV")
print("Kinetic energy T ", T_moyal_single_value_J, " J" )
print(f"Number of slices {N_simp_8_optimisation}")

T_proton_low = 3e6 * cte.e      # MeV -> J
T_proton_high = T_moyal_single_value_J

if T_proton_high <= T_proton_low:
    raise ValueError("T must be > 3 MeV for the CSDA integral bounds.")

f_inv_S = lambda T: (rho_bone / calculate_S_col(n_e=n_e_bone, I=bone_mean_excitation_energy)) * conv_factor_range

# Integrate with Simpson
conv_factor_range = 1e-1 # kg/m² -> g/cm²
R_CSDA_moyal_single_value = simpson(f_inv_S, T_proton_low, T_proton_high, N_simp_8_optimisation)
print(f"\nR_CSDA for a single T value: {R_CSDA_moyal_single_value} g/cm²")
```

water mean excitation energy (I): 75.0 eV
 water density (rho): 1000.0 kg/m³
 water electronic density 3.342783219356227e+29 e⁻/m³

Kinetic energy T 150.55892964122896 MeV
 Kinetic energy T 2.41221999111227e-11 J
 Number of slices 16

R_CSDA for a single T value: 15.855251405967403 g/cm²

In [217...]

```
# Speed test: 1000 single-proton calculations
number_of_moyal_Q8 = 1000
T_samples_MeV_Q8 = sp.stats.moyal.rvs(loc=moyal_loc, scale=moyal_scale, size=number_of_moyal_Q8)
T_samples_J_Q8 = T_samples_MeV_Q8 * 1e6 * cte.e

start_time = timeit.default_timer()
R_CSDA_samples = []
for T_J in T_samples_J_Q8:
```

```

f_inv_S = lambda T: (rho_bone / calculate_S_col(n_e=n_e_bone, I=bone_mean_excit
R_CSDA_val = simpson(f_inv_S, T_proton_low, T_J, N_simp_8_optimisation) * conv_
R_CSDA_samples.append(R_CSDA_val)
end_time = timeit.default_timer()

elapsed = end_time - start_time
count = len(R_CSDA_samples)
time_per_calc = elapsed / count if count > 0 else float('nan')

print(f"\nComputed {count} CSDA ranges in {elapsed:.6f} s")
print(f"Average time per calculation: {time_per_calc*1e6:.2f} µs")

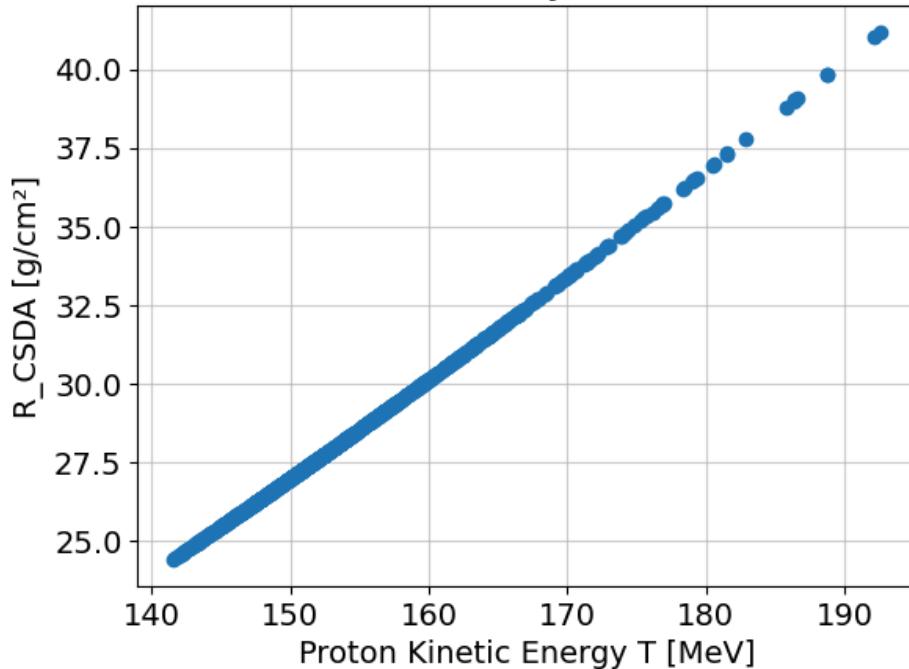
plt.plot(T_samples_MeV_Q8, R_CSDA_samples, 'o')
plt.xlabel("Proton Kinetic Energy T [MeV]")
plt.ylabel("R_CSDA [g/cm²]")
plt.title("R_CSDA for 1000 Protons with Moyal-distributed Kinetic Energies")
plt.grid(alpha=0.6)
plt.show()

```

Computed 1000 CSDA ranges in 0.174321 s

Average time per calculation: 174.32 µs

R_CSDA for 1000 Protons with Moyal-distributed Kinetic Energies



Une optimisation encore plus rapide serait de faire un fit polynomial entre l'énergie du proton et sa portée calculée de manière très précise. On pourrait ensuite arrivé à une évaluation de la portée en temps quasi-réel, soit en quelques microsecondes, ce qui serait idéal pour une utilisation clinique.

Questions 9.

Utilisez `moyal.rvs` pour générer aléatoirement 10000 énergies tirées de cette distribution et tracez-la.

In [218...]

```
moyal_loc, moyal_scale, moyal_size = 150.0, 4.0, 10000

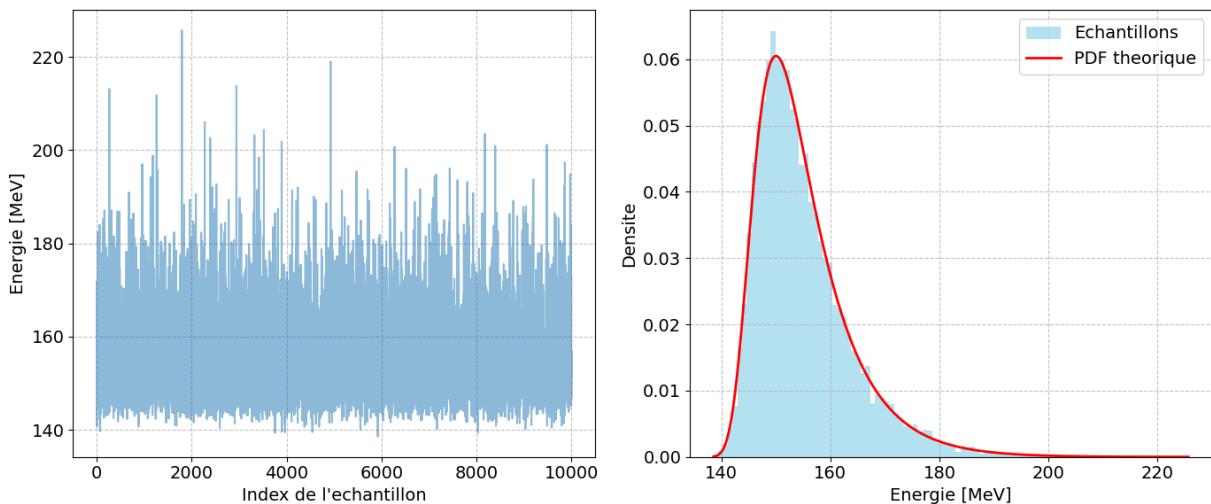
sampling_mev = moyal.rvs(loc=moyal_loc, scale=moyal_scale, size=moyal_size)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

ax1.plot(sampling_mev, markersize=1, alpha=0.5)
ax1.set_xlabel("Index de l'échantillon")
ax1.set_ylabel("Energie [MeV]")
ax1.grid(True, linestyle="--", alpha=0.7)

x = np.linspace(min(sampling_mev), max(sampling_mev), 1000)
ax2.hist(sampling_mev, bins=100, density=True, alpha=0.6, color="skyblue", label="Echantillons")
ax2.plot(x, moyal.pdf(x, loc=moyal_loc, scale=moyal_scale), "r-", lw=2, label="PDF théorique")
ax2.set_xlabel("Energie [MeV]")
ax2.set_ylabel("Densité")
ax2.legend()
ax2.grid(True, linestyle="--", alpha=0.7)

plt.tight_layout()
plt.show()
```



Questions 10.

À l'aide du module timeit, vous estimerez le nombre de protons que vous pouvez calculer par seconde selon trois méthodes : vos implémentations des méthodes par trapèzes et Simpson ainsi que la fonction `scipy.integrate.quad`. Utilisez les 10000 valeurs d'énergie générées plus haut pour faire vos tests. Afin de comparer des pommes avec des pommes, vos calculs permettront tous d'atteindre la précision par défaut de la routine `scipy.integrate.quad`.

Nous allons calculer la portée R_{CSDA} pour chaque énergie générée à partir de la distribution de Moyal, en utilisant les méthodes des trapèzes, de Simpson et la fonction `scipy.integrate.quad`. On commence par calculer la précision de chaque intégral avec

la fonction `calculate_practical_error_ε` pour trouver le nombre de tranches nécessaire pour les méthodes des trapèzes et de Simpson. Ensuite, on utilise `timeit` pour mesurer le temps d'exécution de chaque méthode sur les 10000 énergies générées.

In [219...]

```
# La précision par défaut de scipy.integrate.quad est 1.49e-8
ε_limit = 1.49e-8

N_simp_Q10, R_CSDA_simp_values_Q10, ε_simp_values_Q10 = calculate_R_CSDA_until_mach
print(f"Valeur précision {ε_simp_values_Q10[-1]:.2e} < {ε_limit} with number of sli
N_trap_Q10, R_CSDA_trap_values_Q10, ε_trap_values_Q10 = calculate_R_CSDA_until_mach
print(f"Valeur précision {ε_trap_values_Q10[-1]:.2e} < {ε_limit} with number of sli
```

Valeur précision $1.15e-09 < 1.49e-08$ with number of slice 1024

Valeur précision $5.48e-09 < 1.49e-08$ with number of slice 65536

Nous allons maintenant, programmer les 3 méthodes d'intégration.

In [220...]

```
def compute_R_CSDA_integral_timer(T_samples, N, material_name="water", method=simps

    rho, n_e, I = get_material_properties(material_name)

    start_time = timeit.default_timer()
    R_CSDA_samples = []
    for T_J in T_samples:

        f_inv_S = lambda T: (rho / calculate_S_col(n_e=n_e, I=I, T=T_J))

        R_CSDA_val = method(f_inv_S, T_low, T_J, N) * 0.1 # conv factor kg/m^2 -> g

        R_CSDA_samples.append(R_CSDA_val)
    end_time = timeit.default_timer()

    time_per_calc = (end_time - start_time) / len(T_samples) if len(T_samples) > 0
    return time_per_calc, R_CSDA_samples
```

In [221...]

```
T_samples_J_Q8 = T_samples_MeV_Q8 * 1e6 * cte.e

print("Number of samples n:", len(T_samples_J_Q8))
time_per_calc_simp, R_CSDA_samples_simp = compute_R_CSDA_integral_timer(T_samples_J_Q8)
print(f"Simpson: Average time per calculation: {time_per_calc_simp*1e3:.2f} ms")

time_per_calc_trap, R_CSDA_samples_trap = compute_R_CSDA_integral_timer(T_samples_J_Q8)
print(f"Trapeze: Average time per calculation: {time_per_calc_trap*1e3:.2f} ms")

# Compute the intergral with scipy.integrate.quad
sp_quad = lambda f, a, b, N: sp.integrate.quad(f, a, b)[0]
time_per_calc_sp_quad, R_CSDA_samples_sp_quad = compute_R_CSDA_integral_timer(T_samples_J_Q8)
print(f"scipy.integrate.quad: Average time per calculation: {time_per_calc_sp_quad*1e3:.2f} ms")
```

Number of samples n: 1000

Simpson: Average time per calculation: 16.77 ms

Trapeze: Average time per calculation: 1045.55 ms

scipy.integrate.quad: Average time per calculation: 0.29 ms

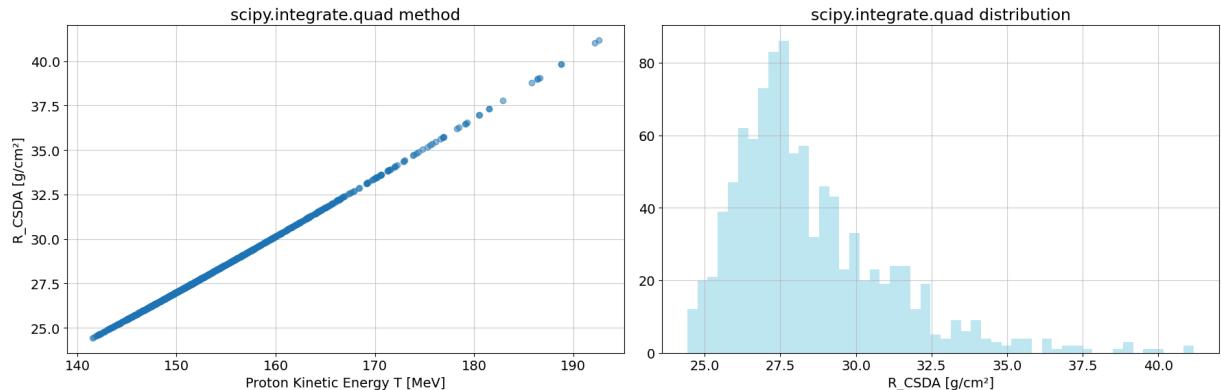
Questions 11.

Faites aussi un histogramme des portées obtenues pour ces 10000 protons et commentez la distribution obtenue.

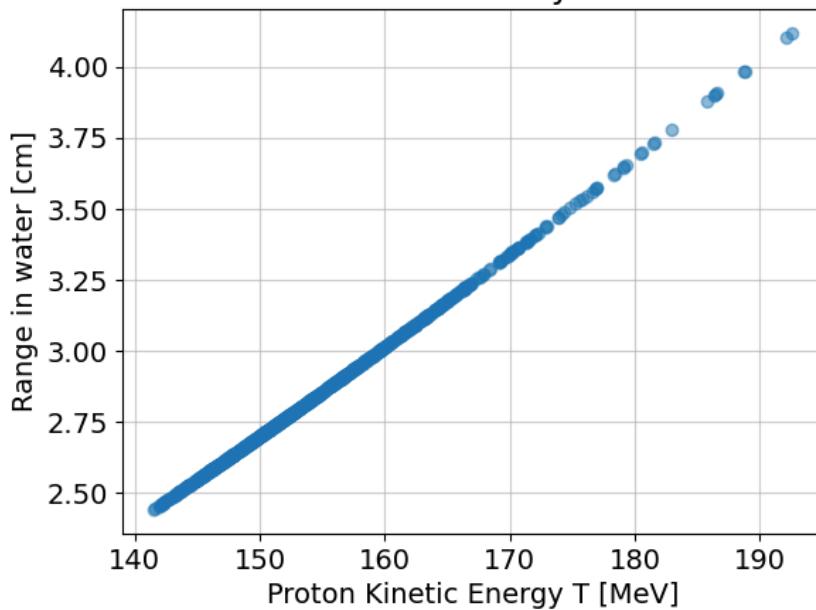
```
In [ ]:
fig, axes = plt.subplots(1, 2, figsize=(18, 6))
axes[0].plot(T_samples_MeV_Q8, R_CSDA_samples_sp_quad, "o", alpha=0.5)
axes[0].set_xlabel("Proton Kinetic Energy T [MeV]")
axes[0].set_ylabel("R_CSDA [g/cm2]")
axes[0].set_title("scipy.integrate.quad method")
axes[0].grid(alpha=0.6)

# histo
axes[1].hist(R_CSDA_samples_sp_quad, bins=50, alpha=0.5, color="skyblue")
axes[1].set_xlabel("R_CSDA [g/cm2]")
axes[1].set_title("scipy.integrate.quad distribution")
axes[1].grid(alpha=0.6)
plt.tight_layout()
plt.show()

R_CSDA_samples_simp_cm = np.array(R_CSDA_samples_simp) / water_density * 1e2    # g
# plot of the range with R_CSDA / rho to get the range in cm
R_CSDA_samples_sp_quad_cm = np.array(R_CSDA_samples_sp_quad) / water_density * 1e2
plt.plot(T_samples_MeV_Q8, R_CSDA_samples_simp_cm, "o", alpha=0.5)
plt.xlabel("Proton Kinetic Energy T [MeV]")
plt.ylabel("Range in water [cm]")
plt.title("Range in water for 1000 Protons with Moyal-distributed Kinetic Energies")
plt.grid(alpha=0.6)
plt.show()
```



Range in water for 1000 Protons with Moyal-distributed Kinetic Energies



Comme montré plus haut, les énergies des protons suivent une distribution de Moyal, qui est une distribution asymétrique avec une longue queue à droite. La portée est très linéaire par rapport à l'énergie cinétique, ainsi, la distribution de T ressemble à celle de R_{CSDA} .

Énergie déposée

Il est possible de calculer l'énergie déposée pour un pas de déplacement du proton s dans un milieu comme suit :

$$s = \int_{T_f}^{T_i} \frac{dT'}{S_{col}} \quad (10)$$

où T_i et T_f sont les énergies cinétiques respectivement avant et après que le proton ait subi l'atténuation d'épaisseur s du matériau.

On cherche à calculer l'énergie déposée E_{dep} par un proton qui traverse une épaisseur s d'un matériau donné. En réarrangeant l'équation 10, on trouve que

$$\int_{T_f}^{T_i} \frac{dT'}{S_{col}} = s \quad \Rightarrow \quad \int_{T_f}^{T_i} \frac{dT'}{S_{col}} - s = 0$$

En définissant la fonction

$$f(T_f) = \int_{T_f}^{T_i} \frac{dT'}{S_{col}} - s$$

on peut trouver la valeur de T_f qui annule cette fonction, soit $f(T_f) = 0$. L'énergie déposée est alors donnée par

$$ds = \frac{dT}{s_{col}}$$

On sépare en une quantité finie de discréétisation de l'énergie cinétique et de la profondeur pour obtenir une approximation numérique de l'énergie déposée.

$$\Delta s = \frac{\Delta T}{s_{col}}$$

Nous partons de l'énergie initial, par exemple $T_i = 150\text{MeV}$. Pour cette énergie, nous calculons le pouvoir d'arrêt S_{col} . Ensuite, nous choisissons une petite valeur de Δs , par exemple 0.1mm , et nous calculons l'énergie déposée pour ce pas de déplacement en utilisant l'équation ci-haute. On répète ce processus en mettant à jour l'énergie cinétique du proton à chaque étape jusqu'à ce que le proton soit complètement arrêté (c'est-à-dire lorsque T_f atteint zéro).

En accumulant l'énergie déposée à chaque étape, nous pouvons obtenir la distribution de l'énergie déposée le long du trajet du proton dans le matériau. Lorsque T est petit, S_{col} devient grand alors Δs devient petit, ce qui correspond à une augmentation de l'énergie déposée, ce qui est caractéristique du pic de Bragg.

In [234...]

```
def transport_proton(T_initial_MeV, material_name, delta_s_m):
    """
    Simulates proton transport through matter

    :param T_initial_MeV: Initial proton kinetic energy [MeV]
    :param material_name: 'water' or 'bone'
    :param delta_s_m: Distance step size [m]

    :returns: (distances [m], stopping powers [MeV/m], energies [MeV])
    """
    rho, n_e, I = get_material_properties(material_name)

    # Convert units
    T_initial_J = T_initial_MeV * 1e6 * cte.e
    T_min_J = 3e6 * cte.e # Lower validity limit (1 MeV)

    # Initialize arrays
    s_values = [0.0] # Position [cm]
    S_col_values = [] # Stopping power [MeV/cm]
    T_values = [T_initial_MeV] # Energy [MeV]

    T_current = T_initial_J
    s_current = 0.0 # [cm]
    delta_s_cm = delta_s_m * 1e2 # Convert step size to cm

    while T_current > T_min_J:
        # Calculate stoping power at current energy
        S_col_J_m = calculate_S_col(n_e, I, T_current)

        # Convert to MeV/cm for plotting
        S_col_MeV_cm = S_col_J_m * (1e-2 / (1e6 * cte.e))
```

```

S_col_values.append(S_col_MeV_cm)

# Calculate energy loss for this step: ΔT = S_col × Δs
delta_T_J = S_col_J_m * delta_s_m

# Update energy and position
T_current -= delta_T_J
s_current += delta_s_m

# stock values in arrays
s_values.append(s_current)
T_values.append(T_current / (1e6 * cte.e)) # Convert back to MeV, sorry for

# Remove last position (where T < T_min)
s_values = s_values[:-1]
T_values = T_values[:-1]

# Add trailing points with zero stopping power and energy
for _ in range(1):
    s_values.append(s_values[-1] + delta_s_m)
    S_col_values.append(0.0)
    T_values.append(0.0)

return np.array(s_values), np.array(S_col_values), np.array(T_values)

```

Questions 12.

Écrire un algorithme capable de réaliser le transport des protons subissant une décélération continue dans le milieu et tracer le dépôt d'énergie en fonction de la profondeur pour l'eau et l'os pour des proton d'énergie cinétique 150 MeV (faisceau monoénergétique). Votre courbe comportera un point où l'énergie déposée est nulle. La position de ce point est-elle conforme à vos résultats antérieurs sur la portée? Qu'est-ce qui influence sa valeur?

On choisit ici l'eau et l'os compact pour calculer le transport des protons. Voir ci-dessous pour l'algorithme de transport.

In [240...]

```

# Initial energy for both materials
T_initial = 150 # MeV

# Transport protons through water and bone
print(f"Transporting {T_initial} MeV proton through water and bone...\n")

delta_s = 1e-6 # step size in m
s_water, S_col_water, T_water = transport_proton(T_initial, "water", delta_s_m=delta_s)
s_bone, S_col_bone, T_bone = transport_proton(T_initial, "bone", delta_s_m=delta_s)

# R_CSDA form Q5
R_CSDA_water = water_R_CSDA_simp[-1] # Kg/m²
R_CSDA_bone = bone_R_CSDA_simp[-1]

```

```

range_Q5_water = R_CSDA_water / water_density # [m]
range_Q5_bone = R_CSDA_bone / bone_density

# print to debug and analyse
print(f"Water: Calculated range = {s_water[-1]*100:.3f} cm")
print(f"          R_CSDA from Q5 = {R_CSDA_water * conv_factor:.3f} g/cm²")
print(f"          R_CSDA / ρ = {range_Q5_water * 1e2:.3f} cm")
print(f"          Difference: {abs((s_water[-1] - range_Q5_water)*100):.3f} cm\n")

print(f"Bone:   Calculated range = {s_bone[-1]*100:.3f} cm")
print(f"          R_CSDA from Q5 = {R_CSDA_bone * conv_factor:.3f} g/cm²")
print(f"          R_CSDA / ρ = {range_Q5_bone * 1e2:.3f} cm")
print(f"          Difference: {abs((s_bone[-1] - range_Q5_bone)*100):.3f} cm\n")

```

Transporting 150 MeV proton through water and bone...

Water: Calculated range = 15.753 cm
 R_CSDA from Q5 = 1.575 g/cm²
 R_CSDA / ρ = 15.753 cm
 Difference: 0.000 cm

Bone: Calculated range = 8.917 cm
 R_CSDA from Q5 = 1.694 g/cm²
 R_CSDA / ρ = 9.159 cm
 Difference: 0.243 cm

La profondeur trouvé à la question 5 est de 15.753 cm pour l'eau et de 9.159 cm pour l'os compact. La différence entre la portée calculée à la question 5 et celle trouvée avec l'algorithme de transport est de 0.000 cm pour l'eau et de 0.243 cm pour l'os compact, ce qui est très proche, surtout pour l'eau.

In [241...]

```

# Plot 1 & 2: Individual Bragg peaks for water and bone
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Water Bragg peak
axes[0].plot(s_water*100, S_col_water, 'b-', lw=2, label='Water')
axes[0].set_xlabel(r"Depth in Water $s$ [cm]", fontsize=13)
axes[0].set_ylabel(r"Stopping Power $S_{\text{col}}$ [MeV/cm]", fontsize=13)
axes[0].set_title("Bragg Peak in Water", fontsize=14)
axes[0].grid(True, alpha=0.3)
axes[0].legend(fontsize=11)

# Bone Bragg peak
axes[1].plot(s_bone*100, S_col_bone, 'k-', lw=2, label="Bone")
axes[1].set_xlabel(r"Depth in Bone $s$ [cm]", fontsize=13)
axes[1].set_ylabel(r"Stopping Power $S_{\text{col}}$ [MeV/cm]", fontsize=13)
axes[1].set_title("Bragg Peak in Bone", fontsize=14)
axes[1].grid(True, alpha=0.3)
axes[1].legend(fontsize=11)

plt.tight_layout()
plt.show()

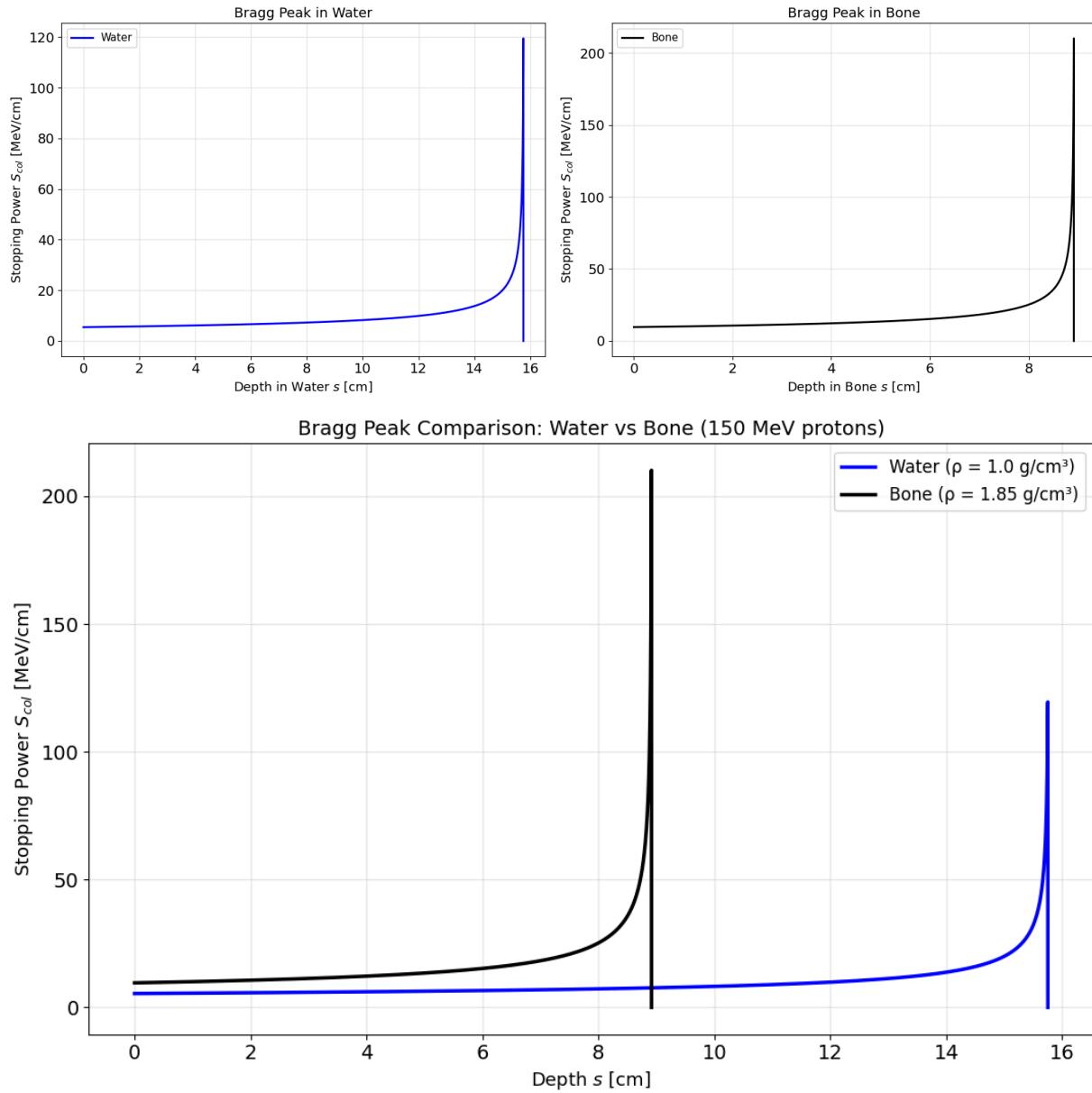
# Plot 3: Overlay comparison

```

```

fig, ax = plt.subplots(1, 1, figsize=(11, 7))
ax.plot(s_water*100, S_col_water, 'b-', lw=2.5, label='Water ( $\rho = 1.0 \text{ g/cm}^3$ )')
ax.plot(s_bone*100, S_col_bone, 'k-', lw=2.5, label='Bone ( $\rho = 1.85 \text{ g/cm}^3$ )')
ax.set_xlabel(r"Depth $s$ [cm]", fontsize=13)
ax.set_ylabel(r"Stopping Power $S_{\text{col}}$ [MeV/cm]", fontsize=13)
ax.set_title("Bragg Peak Comparison: Water vs Bone (150 MeV protons)", fontsize=14)
ax.grid(True, alpha=0.3)
ax.legend(fontsize=12)
plt.tight_layout()
plt.show()

```



On remarque que le point où l'énergie déposée est nulle correspond à la portée R_{CSDA} calculée précédemment, ce qui est conforme à nos résultats antérieurs. La valeur de ce point est influencée par plusieurs facteurs, notamment l'énergie initiale du proton, les propriétés du matériau. Nous devons bien connaître la composition du corps humain pour faire des calculs précis.

En général, une énergie initiale plus élevée conduira à une portée plus longue, tandis que des matériaux plus denses ou avec une énergie d'excitation plus élevée réduiront la portée du proton. Ce qui n'est pas surprenant.

Questions 13.

On nomme cette courbe le pic de Bragg. En déduire l'intérêt des protons pour la radiothérapie.

Le pic de Bragg permet de cibler des régions très spécifiques dans le corps humain lors d'une opération. Cela signifie que la majorité des protons sont absorbés à une certaine distance de pénétration, ce qui permet de protéger des régions alentour saines qui ne doivent pas être touchées. Comme la radiothérapie cherche à tuer des cellules cancéreuses, le pic de Bragg est l'outil idéal pour atteindre la précision chirurgicale requise.

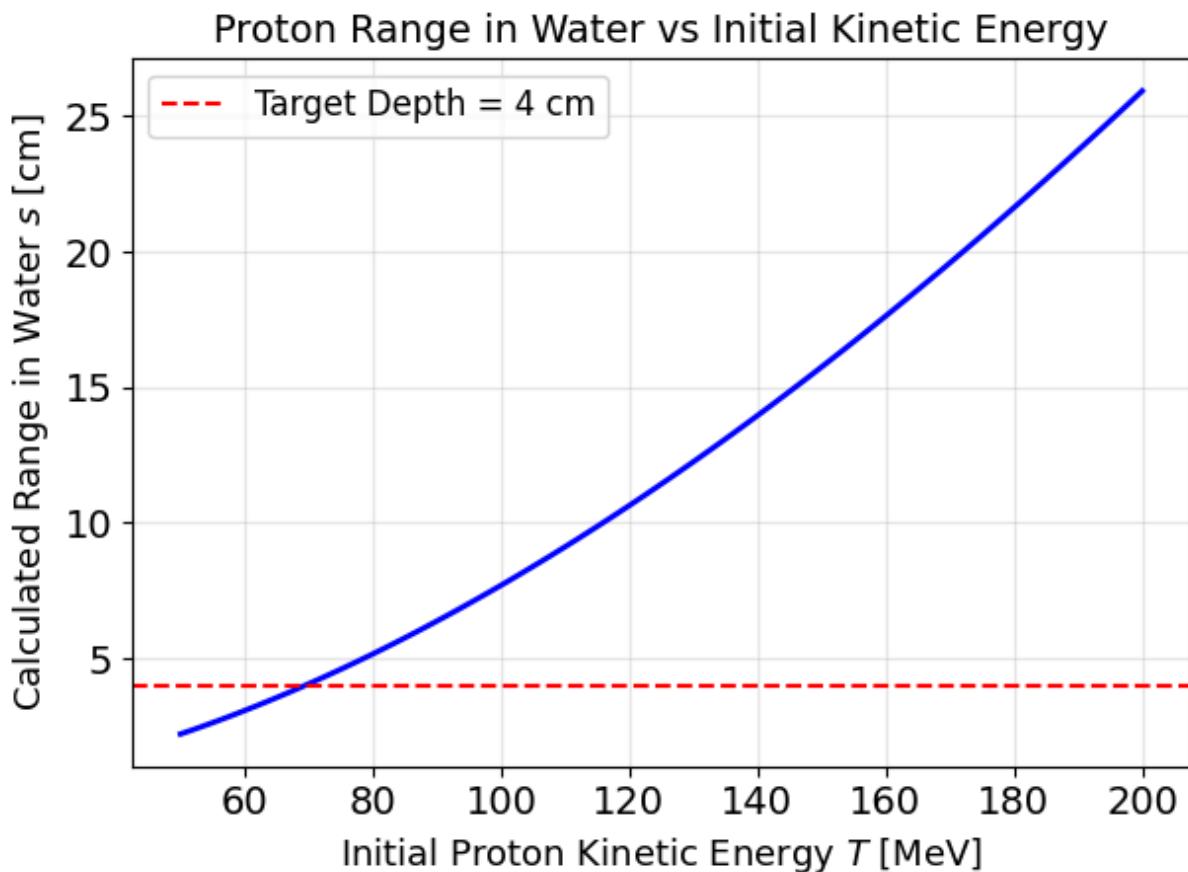
Questions 14.

Tracez la portée R_{CSDA} des protons dans l'eau en fonction de l'énergie cinétique T pour $50 \leq T \leq 200\text{MeV}$. Indiquez sur le graphique la profondeur $D = 4\text{ cm}$ et déterminez graphiquement l'énergie du faisceau nécessaire pour atteindre cette profondeur.

Ici, l'on doit faire varier l'énergie cinétique T initial des protons pour ensuite calculer la profondeur dans l'eau. On peut ensuite tracer la portée R_{CSDA} en fonction de T et trouver graphiquement l'énergie nécessaire pour atteindre une profondeur de 4 cm.

```
In [243...]: T_lin_space_MeV = np.linspace(50, 200, 100) # MeV
s_water_for_T = []
for T_MeV in T_lin_space_MeV:
    s_water, S_col_water, T_water = transport_proton(T_MeV, "water", delta_s_m=delt
    s_water_for_T.append(s_water[-1])
    # Stocker la portée finale pour chaque énergie
s_water_cm = np.array(s_water_for_T) * 1e2 # Convertir en cm
plt.plot(T_lin_space_MeV, s_water_cm, 'b-', lw=2)
plt.axhline(4.0, color='r', linestyle='--', label='Target Depth = 4 cm')
plt.xlabel(r"Initial Proton Kinetic Energy $T$ [MeV]", fontsize=13)
plt.ylabel(r"Calculated Range in Water $s$ [cm]", fontsize=13)
plt.title("Proton Range in Water vs Initial Kinetic Energy", fontsize=14)
plt.grid(True, alpha=0.3)
plt.legend(fontsize=12)
plt.tight_layout()
plt.show()
```

```
# print the energy needed to reach 4 cm in water
target_depth = 4.0*1e-2 # m
# numpy interpolation function
energy_for_target_depth = np.interp(target_depth, s_water_for_T, T_lin_space_MeV)
print(f"Energy needed to reach {target_depth} cm in water: {energy_for_target_depth}")
```



Energy needed to reach 0.04 cm in water: 69.42 MeV

Questions 15.

En quoi les protons sont-ils préférables aux photons pour traiter un mélanome oculaire?

Les protons sont préférables aux photons grâce au pic de Bragg, qui permet au faisceau de s'arrêter net après la tumeur sans irradier les tissus sains situés derrière. Cette précision chirurgicale est cruciale pour l'œil, car elle protège des structures vitales comme le nerf optique ou le cerveau, augmentant ainsi les chances de préserver la vision.

Questions 16.

Dans l'approche développée ici, les protons vont essentiellement en ligne droite dans la matière. Est-ce réaliste? Que devra-t-on éventuellement ajouter à notre modèle?

L'approximation est valide, mais un meilleur modèle pourrait être programmé connaissant la géométrie du corps humain ainsi qu'une meilleure modélisation entre le protons et les atomes du corps humain. Lors de collisions entre les protons et les particules. Il pourrait y avoir déviation de l'angle de propagation.

De nouveaux matériaux comme la peau et le muscle pourraient être ajoutés. Il faudrait prendre en compte l'épaisseur de chaque couche ainsi que l'angle d'arrivée. Il pourrait aussi y avoir un modèle statistique qui prendrait en compte les différentes interactions possibles entre les protons et les atomes du corps humain, comme les interactions nucléaires ou les pertes radiatives, même si elles sont négligeables pour les protons à basse énergie. Il serait ainsi possible d'obtenir une distribution de position en 3D et non seulement en 1D. La distance totale parcourue pourrait aussi être modélisée par une probabilité au lieu d'une valeur unique.

Cette approche plus réaliste permettrait de mieux prédire la distribution de l'énergie déposée, ce qui est crucial pour optimiser les traitements de radiothérapie et minimiser les effets secondaires.

Références

3. voir détails à https://en.wikipedia.org/wiki/Bethe_formula
4. voir <https://physics.nist.gov/cgi-bin/Star/compos.pl?ap>
5. <http://physics.nist.gov/PhysRefData/Star/Text/PSTAR.html>