Projet Numérique n°6

Simulation instationnaire de l'équation de Schrödinger - Matrices Creuses

Ewan Petit, Éléonore Huet

Novembre 2024

Table des matières

1 Introduction

Le comportement de la fonction d'onde $\psi(x,t)$ qui décrit une particule de masse m au sein d'un espace à une dimension sur un intervalle $[\alpha,\beta]$, pendant un temps T plongée dans un potentiel $V(x,t)$ est régie par l'équation de Schrödinger :

$$i\frac{\partial\psi}{\partial t}(x,t) = -\frac{\hbar^2}{2m}\frac{\partial^2\psi}{\partial x^2}(x,t) + V(x,t)\psi(x,t) \tag{1}$$

Du fait de sa complexité, cette équation différentielle linéaire aux dérivées partielles n'admet pas, dans la grande majorité des cas, de solution générales analytiques.

Notre objectif est de créer et d'utiliser, en python et à l'aide des modules à disposition, des outils numériques afin d'obtenir des solutions approximatives mais pertinentes de l'équation de Schrödinger dans des cas instationnaires.

Pour ce faire, nous utiliserons la méthode des différences finies afin d'intégrer numériquement Schrödinger, et nous implémenterons différents solveurs basés sur des schémas d'intégrations différents, que nos comparerons du point de vu de la performance.

2 Modélisation : Méthode des différences finies

On peut tout d'abord adimensionner l'équation de Schrödinger :

$$i\frac{\partial\tilde{\psi}}{\partial\tilde{t}}(\tilde{x},\tilde{t}) = -\frac{1}{2}\frac{\partial^2\tilde{\psi}}{\partial\tilde{x}^2}(\tilde{x},\tilde{t}) + V(x,t)\tilde{\psi}(\tilde{x},\tilde{t})\,^{1} \tag{2}$$

Après le changement de variables suivant

$$\tilde{x} = \frac{x}{L_*} \quad \tilde{t} = \frac{t}{T_*} \quad \tilde{\psi} = \frac{\psi}{L_*^{-\frac{1}{2}}}$$

Où L_*, T_* et E_* sont des échelles dimensionnelles "naturelles" choisies arbitrairement afin de pouvoir prendre en compte numériquement l'effet de chaque terme de l'équation (1). En effet, dans l'équation initiale, certains termes diffèrent radicalement en terme d'ordre de grandeur et risques de ne pas tous être pris en compte.

Dans la suite, par abus de notation, nous considérerons nos grandeurs adimensionnées sans tilde.

On impose la condition initiale :

$$\forall x \in [a,b] \quad \psi(x,0) = \psi_0(x)$$

Où ψ_0 est une fonction connu que nous nous devons de définir.

On impose de plus des conditions aux limites dites de "Dirichlet" qui resterons les mêmes dans tous nos problèmes.

$$\forall t \in [0,T] \quad \begin{cases} \psi(a,t) &= 0 \\ \psi(b,t) &= 0 \end{cases} \tag{3}$$

De plus, on peut réécrire (2) :

$$\frac{\partial\psi}{\partial t} = \mathbf{F}(t,\psi), \quad \mathbf{F}(t,\psi) = \frac{i}{2}\frac{\partial^2\psi}{\partial x^2}(x,t) - iV(x,t)\psi(x,t) \tag{4}$$

2.1 Discrétisation

La méthode des différences finies consiste dans un premier temps à choisir un maillage spatiale et temporel pour discrétiser la situation. Dans la suite, on noteras M+1 (resp. N+1) le nombre de points que l'on choisi pour discrétiser notre intervalle spatial (resp. temporel). Ainsi, on notera :

$$x = (x_0, x_1, \ldots, x_{M-1}, x_M) \quad \text{avec} \quad x_0, x_M = a, b$$
$$t = (t_0, t_1, \ldots, t_{N-1}, t_N) \quad \text{avec} \quad t_0, t_N = 0, T$$

1. $i^2 = -1$

Et :

$$\delta x = \frac{L}{M}, \quad L = b - a$$
$$t = \frac{T}{N}$$

ψ peut alors être exprimée sous la forme d'un array (ie. un tableau numpy) $(M+1) \times (N+1)$.

$$\psi \hat{=} \begin{pmatrix} \psi(x_0,t_0) & \psi(x_0,t_1) & \psi(x_0,t_2) & \dots & \psi(x_0,t_{N-1}) & \psi(x_0,t_N) \\ \psi(x_1,t_0) & \psi(x_1,t_1) & \psi(x_0,t_2) & \dots & \psi(x_1,t_{N-1}) & \psi(x_1,t_N) \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ \psi(x_{M-1},t_0) & \psi(x_{M-1},t_1) & \psi(x_{M-1},t_2) & \dots & \psi(x_{M-1},t_{N-1}) & \psi(x_{M-1},t_N) \\ \psi(x_M,t_0) & \psi(x_M,t_1) & \psi(x_M,t_2) & \dots & \psi(x_M,t_{N-1}) & \psi(x_M,t_N) \end{pmatrix}$$

Et pour plus de commodité, on définira $\forall (j,n) \in [0,M] \times [0,T] \quad \psi(x_j,t_n) = \psi_{j,n}$ et

$$\psi_n \hat{=} \psi(t_n) = \begin{pmatrix} \psi_{0,n} \\ \vdots \\ \psi_{M,n} \end{pmatrix} \quad \text{le n-ième vecteur colonne de } \psi$$

On peut maintenant approximer $\frac{\partial^2 \psi}{\partial x^2}(x,t)$:

$$\frac{\partial^2 \psi}{\partial x^2}(x,t) \approx \frac{\psi_{j-1} - 2\psi_j + \psi_{j+1}}{\delta x^2} \tag{5}$$

Ainsi, d'après (4) : $\forall (j,n) \in [1, M-1] \times [0, N-1]$

$$F(tn, \psi_{j,n}) = \frac{i}{2}\left(\frac{\psi_{j-1,n} - 2\psi_{j,n} + \psi_{j+1,n}}{\delta x^2}\right) - iV_j\psi_{j,n} \tag{6}$$

Les conditions aux limites se traduisent par :

$$\forall n \in [0,N] \begin{cases} \psi(x_0,t_n) = 0 \\ \psi(x_M,t_n) = 0 \end{cases} \tag{7}$$

De plus, dans la suite de notre projet, nous travaillerons exclusivement avec des potentiels stationnaires $V(x,\cancel{t})$. Ainsi peut-on modéliser le potentiel comme un vecteur colonne de dimension M+1 :

$$V = \begin{pmatrix} V_0 \\ V_1 \\ \vdots \\ V_{M-1} \\ V_M \end{pmatrix}$$

2.2 Schémas d'intégration

2.2.1 Méthode explicite : Runge-Kutta 4

Parmi les différentes méthodes explicites, on choisit d'utiliser celle de Runge-Kutta 4. Elle est en effet l'une des plus utilisées en physique car l'erreur commise est de l'ordre $\mathcal{O}(\delta t^5)$. Il s'agit de la plus précise. En comparaison, on a pour Euler explicite $\mathcal{O}(\delta t^2)$.

Pour obtenir la formule de récurrence sur ψ, il faut auparavant calculer quatre autres grandeurs :

$$\begin{aligned} \mathbf{K}_1 &= \mathbf{F}\left(t_n, \psi^n\right) \\ \mathbf{K}_2 &= \mathbf{F}\left(t_n + \frac{\delta t}{2}, \psi_n + \frac{\delta t}{2}\mathbf{K}_1\right) \\ \mathbf{K}_3 &= \mathbf{F}\left(t_n + \frac{\delta t}{2}, \psi_n + \frac{\delta t}{2}\mathbf{K}_2\right) \\ \mathbf{K}_4 &= \mathbf{F}\left(t_n + \delta t, \psi_n + \delta t \mathbf{K}_3\right) \end{aligned}$$

Finalement on a :

$$\psi_{n+1} = \psi_n + \frac{\delta t}{6}\left(\mathbf{K}_1 + 2\mathbf{K}_2 + 2\mathbf{K}_3 + \mathbf{K}_4\right)$$

En connaissant les conditions aux limites, on peut déduire la valeur de ψ à chaque instant n.

2.2.2 Méthode implicite : Crank-Nicolson

Pour assurer la convergence - ie. la stabilité - de l'intégration de l'équation de Schrödinger, on va également utiliser la méthode de Crank-Nicolson, qui est une méthode de résolution **implicite**. On considère l'égalité suivante :

$$\psi_{n+1}(x) - \frac{\delta t}{2}\mathbf{F}\left[t_{n+1}, \psi_{n+1}\right] = \psi_n(x) + \frac{\delta t}{2}\mathbf{F}\left[\ t_n, \psi_n\right]$$

Il vient, d'après [(4)] :

$$\psi_{n+1} - \frac{\delta t}{2}\left(\frac{i}{2}\frac{\partial^2\psi_{n+1}}{\partial x^2}(x) - iV_{n+1}(x)\psi_{n+1}(x)\right) = \psi_n(x) + \frac{\delta t}{2}\left(\frac{i}{2}\frac{\partial^2\psi_{n+1}}{\partial x^2}(x) - iV_n(x)\psi_n(x)\right) \tag{8}$$

En considérant la discrétisation spatiale, on a :
$\forall (n, j) \in [0, N-1] \times [1, M-1]$,

$$-S\psi_{j-1,n+1} + \psi_{j,n+1}\left(1 + 2S + i\frac{\delta t}{2}V_{j,n+1}\right) - S\psi_{j+1,n+1} = S\psi_{j-1,n} + \psi_{j,n}\left(1 - 2S - i\frac{\delta t}{2}V_{j,n}\right) + S\psi_{j+1,n} \tag{9}$$

En posant $S = i\frac{\delta t}{4\delta x^2}$.
Avec les conditions aux limites , on peut définir l'opérateur gauche (resp. droite) O_g (resp. O_d) :

$$O_g = \begin{pmatrix} 1 & 0 & \cdot & \cdot & \cdot & & 0 \\ -S & (1+2S+i\frac{\delta t}{2}V_1) & -S & 0 & & & \\ & \ddots & \ddots & \ddots & & & \\ \vdots & 0 & -S & (1+2S+i\frac{\delta t}{2}V_j) & -S & 0 & \vdots \\ & & & \ddots & \ddots & \ddots & \\ & & & 0 & -S & (1+2S+i\frac{\delta t}{2}V_{M-1}) & -S \\ 0 & & \cdot & \cdot & \cdot & 0 & 1 \end{pmatrix}$$

$$O_d = \begin{pmatrix} 1 & 0 & \cdot & \cdot & \cdot & & 0 \\ S & (1-2S-i\frac{\delta t}{2}V_1) & S & 0 & & & \\ & \ddots & \ddots & \ddots & & & \\ \vdots & 0 & S & (1-2S-i\frac{\delta t}{2}V_j) & S & 0 & \vdots \\ & & & \ddots & \ddots & \ddots & \\ & & & 0 & S & (1-2S-i\frac{\delta t}{2}V_{M-1}) & S \\ 0 & \cdot & \cdot & \cdot & & 0 & 1 \end{pmatrix}$$

Et ainsi formaliser l'égalité (9) sous forme matricielle :

$$\forall n \in [0, N-1], \quad O_g\psi_{n+1} = O_d\psi_n \tag{10}$$

3 Implémentation

Cette partie concerne le document **code_librairie.pdf** (référencé par mylib.py dans les autres fichiers)

Pour programmer un solveur de l'équation de Schrödinger basé sur les schémas de résolution suivants, on implémente dans tous les cas des fonctions dont les arguments sont : l'intervalle temporel [a, b], le temps T, M, N ainsi que des fonctions qui prennent en argument un maillage spatial et retourne respectivement un potentiel $V(x)$ et la fonction $\psi_0(x)$ sur ce maillage. Nos solveurs renvoient le maillage spatial (resp. temporel) x (resp. t) ainsi que le tableau ψ (f_num dans le programme) de taille $(M+1) \times (N+1)$.

3.1 Runge-Kutta 4

Pour implémenter **solve_RK4** basé sur le schéma d'intégration explicite Runge-Kutta 4, on commence par établir le maillage spatio-temporel. Pour Runge-Kutta, il faut avoir :

$$dt \leq \frac{dx^2}{2} \tag{11}$$

C'est la **condition de stabilité** qu'il est essentiel de respecter si l'on veut de bons résultats. Cette formule découle de l'analyse de Neumann et explicite bien la relation entre le pas de temps et d'espace dans l'équation de Schrödinger : les termes en dérivée seconde ont un effet qui est quadratique par rapport à dx, on voit ainsi qu'il est nécessaire de réduire dt de manière proportionnelle au carré de dx. Par conséquent une petite variation dans l'espace a un impact important sur la variation temporelle.

On définit les pas d'espace dx et de temps dt par :

$$dx = \frac{L}{M} \text{ et } dt = \frac{T}{N}$$

Il faut ensuite initialiser la fonction d'onde ψ en définissant sa valeur aux bords du domaine :

$$\forall n \in [0, N], \quad \psi(x_0, t_n) = 0 \text{ et } \psi(x_M, t_n) = 0$$

car en 0 et en M on a un potentiel infini donc ψ y est forcément nulle.

On créé un tableau psi_num qui servira a stocker les valeurs de ψ.

On définit maintenant la fonction F(ψ) discrétisée exprimée en (6).

Il suffit ensuite d'écrire les équations paramétrant l'intégration explicitées en 2.2.1 puis d'enregistrer chaque valeur ψ_{n+1} calculée dans le tableau des résultats psi_num.

Finalement, on doit s'assurer que le calcul se déroule bien et qu'il n'y a pas d'instabilité numérique. Une instabilité peut apparaître dans plusieurs conditions, notamment si on choisit un pas de temps trop grand. On insère dans le code une condition d'arrêt : à chaque itération, on cherche le maximum du module de la fonction d'onde et on le compare à une valeur arbitraire très grande (10^6). Si l'amplitude de ψ dépasse ce nombre, on arrête le calcul car on le considère comme instable.

Pour suivre l'évolution du calcul on affiche sur l'écran où en est le calcul tous les 100 pas et on affiche un message à la fin si le calcul s'est déroulé correctement.

3.2 Crank-Nicolson

On commence par initialiser f_num le tableau de ψ et (d'après la CI) définir f_num[:0] (ie. ψ_0).

Pour ce qui est de la résolution effective, on utilise deux méthodes, détaillées ci-après.

3.2.1 Matrices denses

Dans un premier temps, nous avons programmé nos solveur en utilisant des matrices denses pour manipuler les opérateurs O_g et O_d, c'est-à-dire des arrays numpy de taille $(M+1)\times(M+1)$ (fonction **solve_dense_CN**). On les initialise, puis on les remplit selon 2.2.2 par itération en parcourant les matrices.

Ensuite, on introduit l'opérateur

$$O = O_g^{-1} O_d$$

Ainsi, à chaque étape, plutôt que de résoudre une équation aux éléments propres pour trouver successivement les ψ_{n+1}, le programme effectuera simplement une multiplication matricielle :

$$\forall n \in [0, N-1], \quad \psi_{n+1} = O\psi_n$$

De la même manière que pour la fonction d'intégration basée sur le schéma RK4, on insère une condition d'arrêt pour arrêter la boucle en cas d'instabilité.

On relève un inconvénient majeur : si ce programme semble résoudre les problèmes de stabilité rencontrés avec la méthode explicite, le programme est assez coûteux en temps d'exécution (cf chapitre suivant). Et pour

cause : l'opération produit de matrice carré de taille $(M+1)\times(M+1)$ et de complexité $\mathcal{O}(M^2)$. De plus si nos matrices opérateurs O_g et O_d majoritairement constituées de 0, ceux-ci sont quand même stockés dans la mémoire vive de l'ordinateur pendant les calculs. Or, d'après l'outil .getsizeof du module sys, le coût mémoire d'un float en double précision et de 16 octets. Alors, le coût mémoire d'une matrice de taille $(M+1)^2$ remplie de float en double précision et de $(M+1)^2\times 16$ octets, soit, pour M = 10000, 1.6 Gigaoctets !

3.2.2 Matrices creuses

Nos matrices respectives O_g et O_d possèdent $3M-1$ éléments non-nuls ($3M-3$ pour O_d en réalité, ce qui ne change pas grand chose en terme de coût mémoire).

On utilise le module sparse de la bibliothèque scipy afin de ne ne stocker que les éléments non-nuls. Pour chaque matrice, on initialise 3 vecteurs val, row, col, puis on les remplit respectivement par les valeurs et les coordonnées (ligne et colonne) des éléments non-nuls de nos opérateurs (cf 2.2.2). Afin de stocker notre matrices "creuses", on utilise la fonction .coo_matrix() de sparse.

En reprenant notre exemple précedent, le coût mémoire de notre matrice est maintenant de $(9M-3)\times 16$ octets , soit 1.4 Megaoctets ; plus de 1000 fois moins lourd qu'une matrice dense.

On utilise de plus tocsr() pour stocker nos matrices de manière encore plus efficace : le format csr (Compressed Sparse Row) est un format où l'array row est compressé (occupe moins de mémoire) et trié, ce qui fluidifie encore les calculs.

Dans un premier temps (fonction **solve_sparse_CN**), on réintroduit l'opérateur O définit précédemment. Cependant, O est une matrice pleine, comme on peut le voir sur la Figure 1. La complexité en temps de ce programme sera donc en $O(M^2)$ (cf. **Complexité**). On perd donc tout l'intérêt de l'utilisation des matrices creuses.

FIGURE 1 – Matrice de O lors de l'exécution du programme **solve_sparse_CN** pour M=10 (cas du puits de potentiel infini). On remarque que O est pleine.

Dans un deuxième programme (solve_sparse_CN1), on utilise la fonction spsolve() du sous-module linalg de sparse et on résout (10) $\forall n \in [0, N-1]$.

Dans un troisième programme (**solve_sparse_CN2**), on utilise la fonction splu() du sous-module linalg de sparse et on résout (10) : cette fonction effectue une factorisation "lower-upper". Comme O_g reste la même pour tout n, en "factorisant" (10) par O_g, on accélère le calcul.

4 Complexité opérationnelle

Pour déterminer lequel des algorithmes et la plus pertinent, un aspect clé à étudier est sa complexité temporelle, ie. le temps d'exécution (cf. **code_complexite.pdf**). Compte tenu de la forme des algorithme, à M fixé on se doute que tout les algorithmes seront en $\mathcal{O}(N)$. Ici, on va plutôt étudier la complexité temporelle à N fixé en fonction de M. Cette étude va exclure d'office l'algorithme de résolution explicite, du fait de la condition (11) :

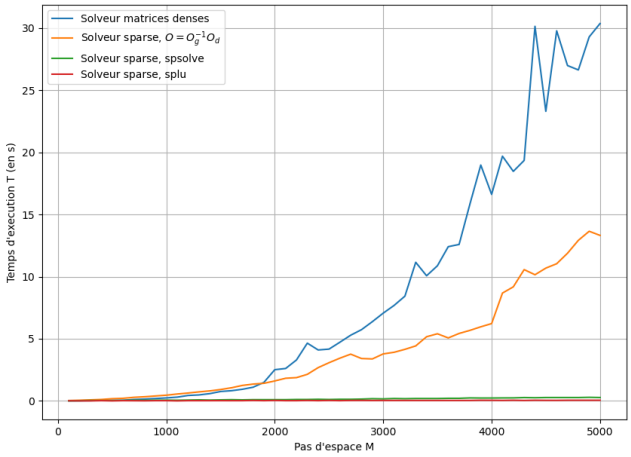

FIGURE 2 – Comparaison des complexités en fonction de M : algorithmes basés sur le schéma d'intégration explicite.

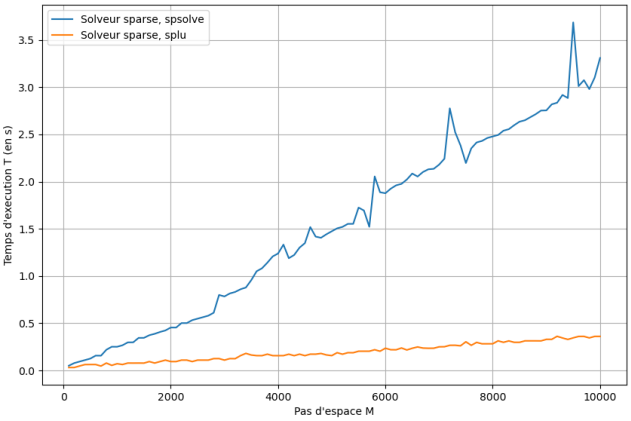

FIGURE 3 – Comparaison des complexités en fonction de M : **solve_sparse_CN2** et **solve_sparse_CN1.**

La Figure 2 montre que la complexité des algorithmes aux matrices denses et **solve_sparse_CN** (qui utilise des matrices creuses mais en inversant la matrice O_g) possèdent une complexité exponentielle $\mathcal{O}(M^\alpha)$; le temps d'exécution devient vraiment significatif à partir d'un certain temps. Les algorithmes **solve_sparse_CN2** et **solve_sparse_CN1** quant à eux, semblent bien plus efficaces.

Sur la Figure 3, qui "zoom" sur le comportement du temps d'exécution des algorithmes utilisant des matrices creuses (sans inverser O_g), on remarque que la complexité de ceux-ci est linéaire ($\mathcal{O}(M)$), comme on avait pu le prévoir. En outre, l'algorithme **solve_sparse_CN2** (splu()), se révèle plus efficace.

Ainsi, si l'on veut un maillage spatial de haute résolution lors des applications aux problèmes instationnaire, on va opter pour l'algorithme de résolution utilisant des matrices creuses, et particulièrement celui faisant intervenir splu().

5 Application

Cette partie concerne le document **code_applications.pdf**

5.1 Validation : Puits infini de potentiel

Pour tester notre code, on propose dans un premier temps le cas simple du puits de potentiel infiniment profond. Il est défini tel que :

$$V(x) = \begin{cases} 0 & \text{si } x \in [0, L] \\ +\infty & \text{ailleurs} \end{cases}$$

Le domaine de calcul est $x \in [0, 1]$ et $t \in [0, T]$
Comme condition initiale, on choisit le paquet d'onde parabolique normalisé :

$$\psi(x, 0) = \sqrt{30}x(1 - x)$$

On fixes les limites des domaines spaciaux et temporels :

$$a, b = 0, 1$$
$$T = 1$$

Pour RK4, on prend 10000 points sur le maillage temporel et 100 ponts de temps afin de satisfaire à la condition de stabilité. Après de nombreux essais, nous avons jugé que 10000 était une valeur adéquate qui permettait à coup sûr de pouvoir réaliser tous les calculs sans instabilité.
On peut visualiser la fonction d'onde (cf. Animation) :

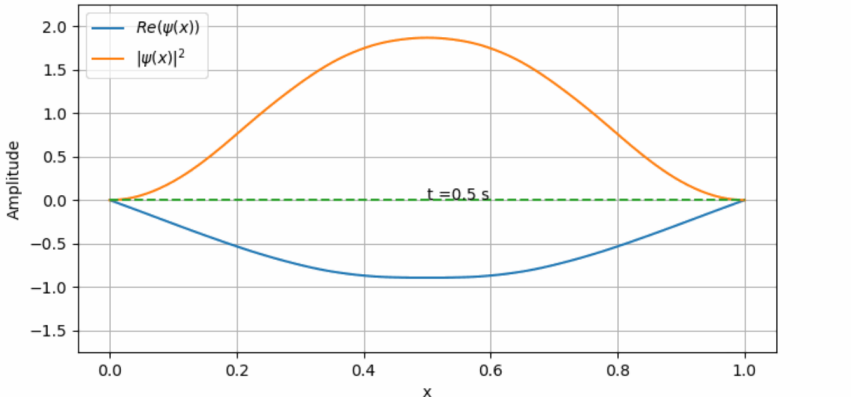

FIGURE 4 – ψ par **solve_RK4**, N=100, M=10000

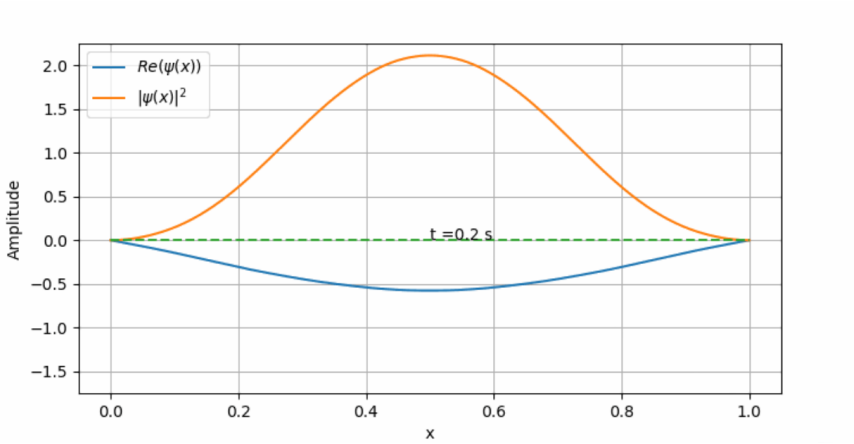

FIGURE 5 – ψ par **solve_sparse_CN**, N=1000, M=1000

Toutes nos fonctions de résolutions nous donnent la même animation (cf. animations 5.1), semblables à celles qu'on peut entrevoir sur les Figures 4 et 5.

Dorénavant, pour les raisons de stabilité et de complexité détaillées avant, nous n'utiliserons pour traiter les cas suivant que la fonction solve_sparse_CN2, (schéma implicite, matrice creuse, splu()).

5.2 Effet Tunnel

L'objectif maintenant est d'étudier l'équation d'onde dans l'effet tunnel. Il s'agit du phénomène observé lorsqu'une fonction d'onde rencontre une barrière de potentielle et n'est pas entièrement réfléchie ; une petite partie est transmise et arrive à passer de l'autre côté.

On définit une barrière de potentiel V(x) :

$$V(x) = \begin{cases} 0, & \text{si } 0 < x < \frac{1-\alpha}{2} \\ V, & \text{si } \frac{1-\alpha}{2} < x < \frac{1+\alpha}{2} \\ 0, & \text{si } \frac{1+\alpha}{2} < x < 1 \\ +\infty, & \text{ailleurs} \end{cases}$$

Ici 0 < a < 1 définit la largeur adimensionnée de la barrière et V > 0 sa hauteur. On souhaite simuler l'évolution temporelle d'un paquet d'onde initiale $\psi(x,0) = \psi_0(x)$ de la forme :

$$\psi_0(x) = Ce^{ikx} \exp\left(-\frac{(x-x_0)^2}{2d^2}\right) \tag{12}$$

avec :

- $C = \left[\frac{1}{\pi d^2}\right]^{1/4}$ le coefficient de normalisation qui garantit $\langle\psi_0|\psi_0\rangle = 1$.
- x_0 : Centre de la gaussienne.
- d : Largeur de la gaussienne à mis-hauteur.
- $k = \frac{2\pi}{\lambda}$: Nombre d'onde.

On doit vérifier quelques conditions : Le paquet d'onde doit se propager vers la droite donc on doit avoir $U_0 > 0$. Or, analytiquement $U_0 = k$, donc on fixe $k > 0$.

Plus l'énergie initiale H_0 est grand par rapport à V, moins la barrière de potentiel agit comme une barrière sur le mouvement de la particule. Analytiquement,

$$H_0 = \frac{1}{2}k^2 + \frac{1}{4}\frac{1}{d^2}$$

On reconnaît dans le terme en k^2 un terme cinétique : plus on "lance" notre particule avec une vitesse élevée, plus son énergie initiale et grande, donc plus elle a de chance de franchir la barrière.

On peut ainsi mettre une condition dans le code, qui indique selon la valeur de V rentrée si H_0 va dépasser ou non. Cela va nous être utile pour interpréter les résultats.

On paramètre la fonction du potentiel $fV\_bar(x)$ en choisissant une valeur de α arbitraire entre 0 et 1. Plus a sera proche de 1, plus la barrière sera épaisse et difficile à traverser.

On définit finalement la fonction ψ comme l'expression (12). On a donc on paquet d'onde initial qui se déplace vers la barrière de potentiel, comme représenté dans la Figure 6.

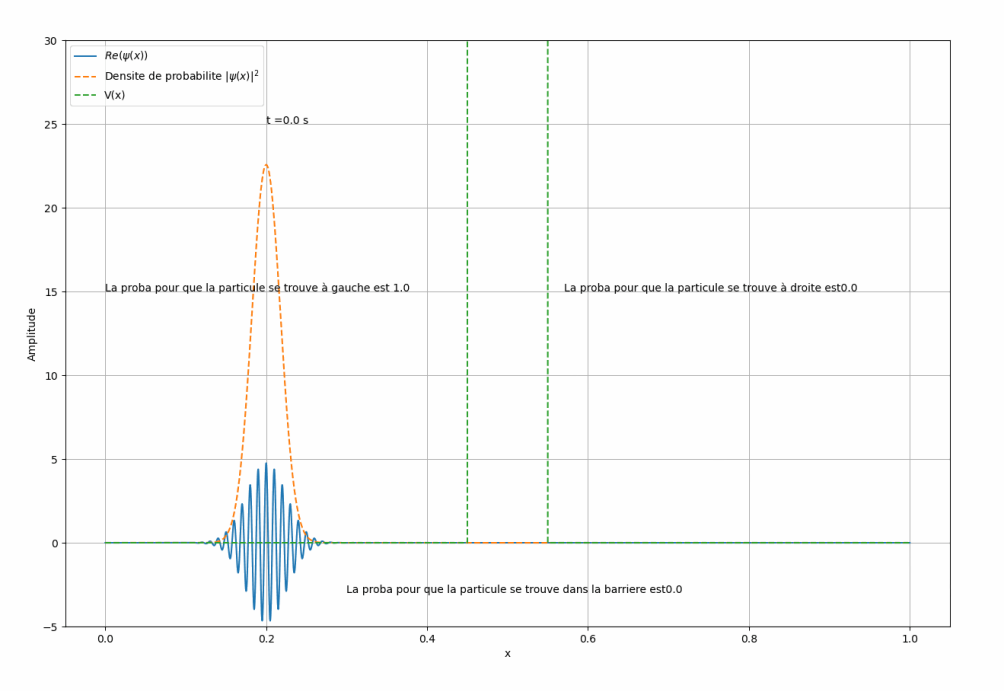

FIGURE 6 – Paquet d'onde initial : amplitude de $\psi(x)$ à l'instant initial. k = 628, d=0.025 N = 10000, M=10000, V= 250000.

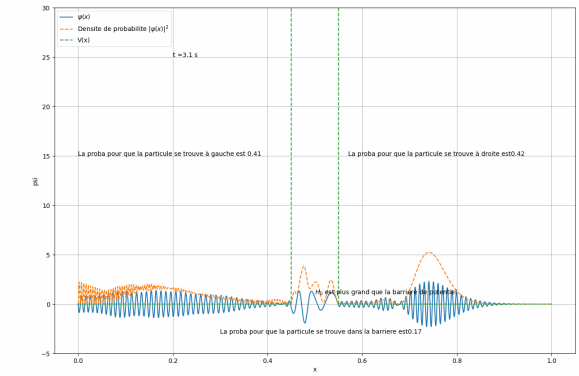

FIGURE 7 – $Re(\psi(x)$ à un instant t médian. k = 628, d = 0.025 V= 190000, N = 10000, M=10000.

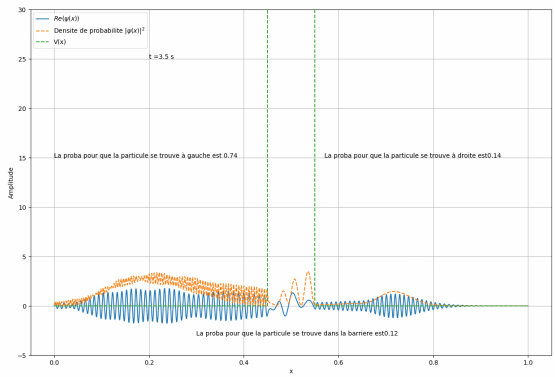

FIGURE 8 – $Re(\psi(x)$ à un instant t médian. k = 628, d = 0.025 V= 200000, N = 10000, M=10000.

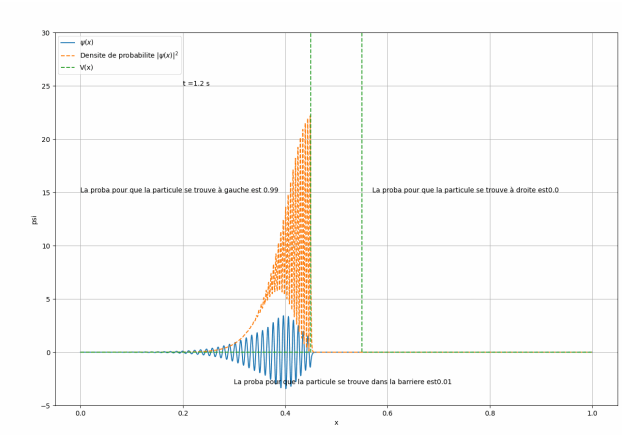

FIGURE 9 – $Re(\psi(x)$ à un instant t médian. k = 628, d = 0.025 V= 400000, N = 10000, M=10000.

Les Figures 7, 8 et 9 représentent l'amplitude de la partie réelle le la fonction d'onde à un temps plus ou moins médian dans la simulation, pour des barrières de potentiel de plus en plus hautes mais de même largeur. On a $H_0 \approx 193000$. Un des constat que l'on peut établir, c'est que le potentiel se comporte effectivement comme une barrière. Plus elle est haute, moins la fonction d'onde passe. Ainsi, plus la barrière est haute, moins la particule aura de chance de traverser la barrière. Les calculs de probabilités pour les différentes parties sont détaillés sur les figures. Dans la Figure 7, où $H_0 \approx V$ la particule a peu près autant de chance d'être à gauche que d'être à droite. Dans la Figure 9, la probabilité pour que la particule soit dans la partie droite et quasiment nulle, même à la fin de la simulation (comparer les animations **5.2_1.gif** et **5.2_1ter.gif**).

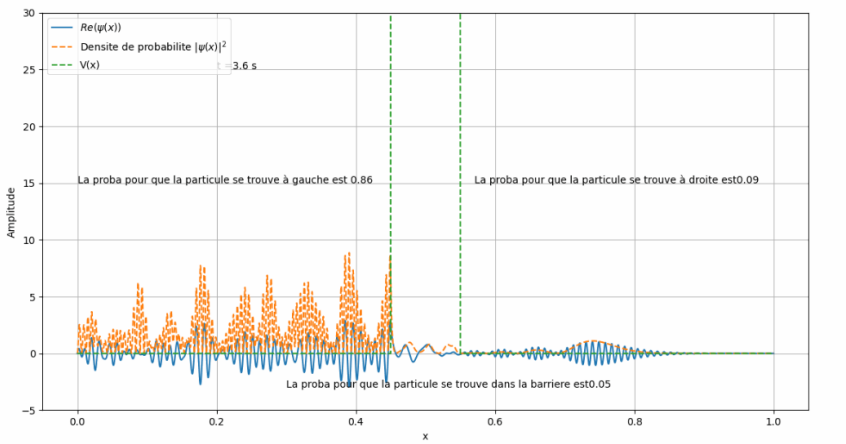

FIGURE 10 – $Re(\psi(x)$ à un instant t médian. k = 571, d = 0.025 V= 200000, N = 10000, M=10000.

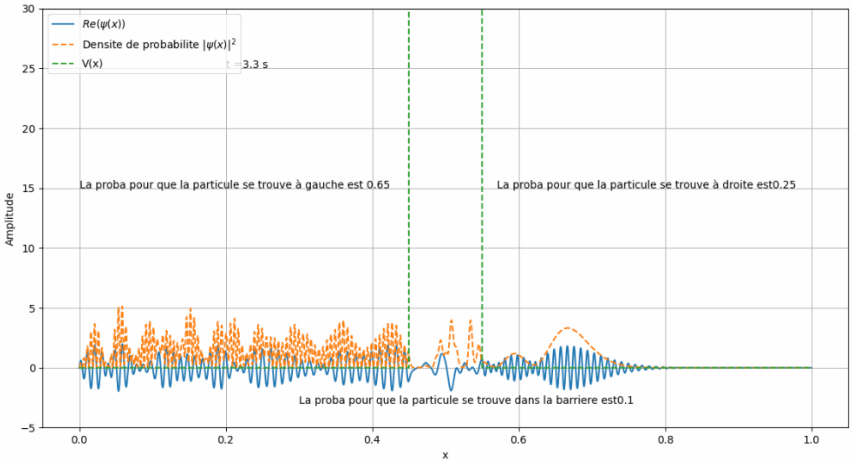

FIGURE 11 – $Re(\psi(x)$ à un instant t médian. k = 628, d = 0.025 V= 200000, N = 10000, M=10000.

Dans les Figures 10 (resp. 11), on impose un $k = 571$ (resp. 628). La hauteur ainsi que la largeur de la barrière de potentiel restent constants. Or, on observe que dans la cas où k est le plus petit, la probabilité de présence de la particule dans la partie droite et plus faible que dans le cas où k est la plus grand. Ainsi, on vérifie que l'énergie cinétique de la particule affecte positivement la probabilité qu'elle traverse la barrière de potentiel (comparer à ce titre les animations **5.2_1.gif** et **5.2_1bis.gif**).

De plus, les animations **5.2_2.gif** et **5.2_2bis.gif** mettent en avant l'effet de la largeur de la barrière sur la probabilité de présence de la particule à droite.

On peut remarquer sur ces simulation qu'en additionnant les probabilités de présence dans les 3 parties, on retrouve à peu près 1 (l'erreur peut-être attribuée au coefficient de normalisation C, que nous avons calculer en supposant que notre espace était sans bords ou infini).

Ce sont ces phénomènes, par lesquels une particule semble pouvoir traverser une barrière infranchissable avec un certaines probabilités, qui constituent l'effet tunnel.

5.3 Rebond Quantique

On étudie ici les rebonds sur un miroir d'un neutron plongé dans le champ de pesanteur terrestre, lâché à une altitude z_0.

Dans la partie 2 on avait défini des échelles de grandeurs pour adimensionner l'équation de Schrödinger. Ces grandeurs sont reliée par (avec L_* fixé :)

$$E_* = \frac{\hbar^2}{m_p L_*^2} \quad T_* = \frac{m_p L_*^2}{\hbar} \tag{13}$$

Où $m_p = 1.7 \times 10^{-27} kg$ est la masse du protons.

Avec $g = 9.81 m.s^{-1}$ l'accélération de pesanteur, on peut définir l'échelle de

$$L_* = \left[\frac{\hbar^2}{m_p^2 g}\right]^{1/3}$$

Ainsi :

$$V(z) = z$$

Ici, on va discrétiser le temps $t \in [0, T]$ et l'axe vertical $z \in [0, D]$.

On choisit (CI) :

$$\psi_0(z) = Ce^{ikx} \exp\left(-\frac{(z - z_0)^2}{2d^2}\right) \tag{14}$$

Avec

- $C = \left[\frac{1}{\pi d^2}\right]^{1/4}$ le coefficient de normalisation qui garantit $\langle \psi_0 | \psi_0 \rangle = 1$.
- z_0 : Centre de la gaussienne.
- d : Largeur de la gaussienne à mis-hauteur.

On a toujours les même conditions aux limites : en 0, on a le miroir, une surface solide que ne peut traverser la particule. On considère de plus que $z_0 \ll D \approx +\infty$. On prend $D = 100$, $z_0 = 50$.

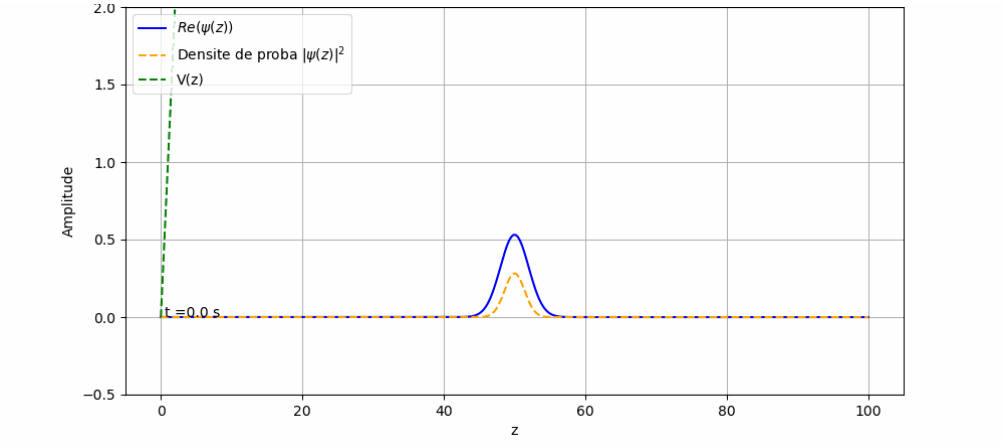

FIGURE 12 – Paquet d'onde gaussien à t_0 sans vitesse initiale.

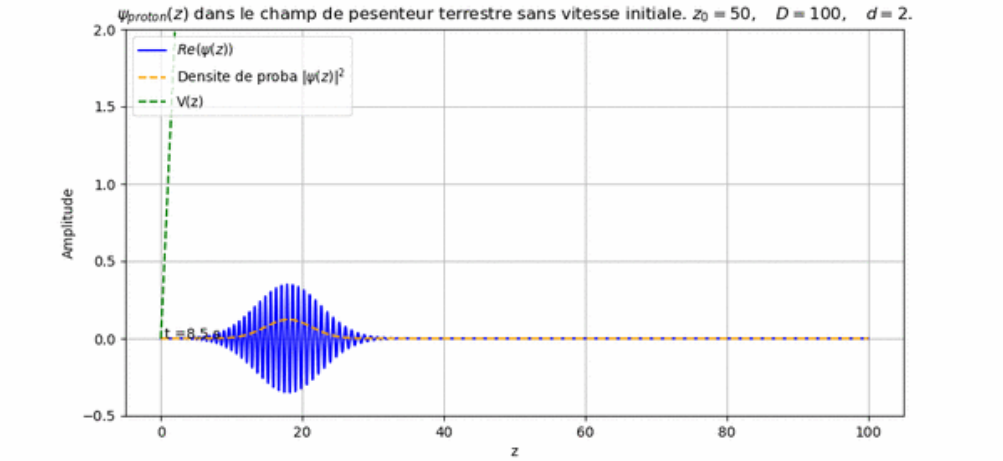

FIGURE 13 – Paquet d'onde gaussien à un temps t_n peu après la début de la simulation. $k < 0$

On a un paquet d'onde initial sans vitesse (ψ_0 étant réelle, $\langle \hat{p} \rangle = 0$). C'est ce que l'on observe sur la Figure 12. Ensuite, sous l'action du champ de pesanteur, la particule se met en mouvement vers les potentiel décroissants, cf. **5.3_k=0.gif**.

On peut également imposer une impulsion vers la droite $k > 0$ (cf. animation **5.3_pulse.gif**).

6 Conclusion

Afin de résoudre l'équation de Schrödinger dans des cas instationnaires, le recours aux méthodes numériques dites "des différences finies", est nécessaire. En première approche, on peut utiliser un schéma d'intégration numérique explicite, simple et rapide à implémenter. Seulement, la condition d'instabilité ainsi que la complexité temporelle des algorithmes explicites sont contraignant. Avec un algorithme basé sur un schéma de résolution implicite, on pallie les problèmes d'instabilité, mais son implémentation avec des matrices denses pose toujours un problème de complexité temporelle. En remarquant que les opérateurs utilisés sont "creux", on peut se servir des différentes fonctions du module scipy.sparse afin d'implémenter un algorithme de complexité linéaire en la taille des opérateur, améliorant ainsi le temps d'exécution de l'intégration numérique. Ce dernier outil nous a permis de modéliser de manière pertinente et en haute résolution des problèmes instationnaires.