Universite Paris-Saclay - L3 Physique
Simulation numérique de lois d'évolution
Projets de mécanique quantique

Wietze Herreman - Caroline Nore

Année 2023-2024

Introduction

1 Informations et modalités de l'UE

Bienvenu dans le descriptif des projets de mécanique quantique donné aux étudiants de la L3 Physique de l'université Paris-Saclay. Avant de se lancer dans l'équation de Schrödinger, on rappelle quelques informations utiles concernant l'organisation de cette UE.

- Projets par groupes de 3 (ou 2).
  On déconseille de faire tout le travail tout seul. Apprendre à collaborer fait partie du projet et il y a suffisamment de choses à faire. Chaque groupe fera un autre projet (7 ici sur la mécanique quantique). Les difficultés de chaque projet sont comparables et ils ont souvent des blocs en commun.

- Rapport écrit + code (40% de la note)
  Un rapport de 20 pages maximum est à rendre pour chaque groupe. Vous allez également avoir à déposer vos codes dans un répertoire en ligne, à la fin du projet.

- Soutenance orale (40% de la note)
  Une soutenance orale est organisée en décembre. Durant cette présentation de 15 minutes, chaque membre du groupe devra préciser sa contribution au travail collectif et présenter une partie (5mn par membre). La présentation sera suivie de 15mn de questions.

- Travail en salle (20% de la note)
  Votre présence en salle info est obligatoire et mesurée par une liste d'émargement. Profiter de ces moments pour travaillez sur votre projet et pour nous poser toutes vos questions. Si vous ne faites rien, ça ne sert à rien de venir.

- Collaborations et influences externes (jusqu'à 100% de la note)
  Les discussions entre groupes ainsi que tout autre aide extérieure sera tolérée mais attention, nous attendons de chacun de vous d'être "expert" de sa partie. Il est très facile pour nous de reconnaître des codes qui sont des copié-collés de vos voisins ou des codes que vous n'avez pas écrits vous-même. Lorsqu'on constate de forts déséquilibres dans le travail fourni à l'intérieur d'un groupe, ça vous sera signalé et ça sera aussi pris en compte dans la note finale.

La réussite du projet sera évaluée sur les critères suivants :

- Compréhension et présentation du problème physique dans le rapport et la soutenance
- Soin apporté au code : organisation, clarté, espacement, commentaires, ...
- Qualité de la présentation des résultats : unités des axes des graphes, légendes, codes couleurs, ...
- Discussion des résultats.

Les années passées ont démontrées que la plupart des étudiants réalise de très beaux projets. Ca demande un certain investissement de temps mais vous allez développer des compétences très utiles pour vos futurs stages en entreprise ou dans des laboratoires.

2 Eléments théoriques essentiels

Dans cette série de projets, on s'intéresse à la **mécanique quantique ondulatoire**. On apprend à trouver numériquement des solutions de l'équation de Schrödinger et à simuler comment la fonction d'onde évolue au cours du temps. Les projets se déclinent en deux genres

- projets sur les états propres et leur utilisation
- projets où on réalise des simulations instationnaires

Cette section a comme but de faire quelques rappels essentiels de mécanique quantique ondulatoire.

2.1 Equation de Schrödinger en 1D

Dans tous les projets, on étudie le mouvement d'une particule quantique en 1D. La fonction d'onde $\psi(x,t)$ satisfait l'équation de Schrödinger :

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

Ici $\hbar = h/(2\pi) = 1.05457\ 10^{-34} Js$, m la masse de la particule. $V(x,t)$ est l'énergie potentielle de la particule à position x et à temps t. La fonction d'onde $\psi(x,t)$ est complexe et sa module au carré définit la densité de probabilité pour la présence de la particule

$$\text{probabilité de trouver la particule entre } x \text{ et } x+dx = |\psi(x,t)|^2 dx \tag{2}$$

Les distributions de probabilité sont toujours normalisées et on doit donc avoir

$$\langle\psi|\psi\rangle = \int_D |\psi(x,t)|^2 dx = 1 \tag{3}$$

à tout instant t. Ici D est l'intervalle ou domaine d'existence de la particule sur x.

En mécanique quantique, on utilise de nombreux opérateurs. Dans certains projets, on aura besoin des opérateurs de position $\hat{x}$, de quantité de mouvement (impulsion) $\hat{p}$ et d'énergie (Hamiltonien) $\hat{H}$. Si on utilise la fonction d'onde $\psi(x,t)$ pour représenter l'état quantique, on définit l'action de ces opérateurs comme

$$\hat{x}\,\psi(x,t) = x\,\psi(x,t) \quad , \quad \hat{p}\,\psi(x,t) = -i\hbar\frac{\partial\psi(x,t)}{\partial x} \quad , \quad \hat{H}\,\psi(x,t) = -\frac{\hbar^2}{2m}\frac{\partial^2\psi(x,t)}{\partial x^2} + V(x,t)\,\psi(x,t) \tag{4}$$

L'Hamiltonien apparait dans l'équation de Schrödinger, qui se laisse écrire comme $i\hbar\,\partial\psi/\partial t = \hat{H}\psi$. On calcule la position moyenne et la quantité de mouvement moyenne de la particule avec ces opérateurs et le produit Hermitien

$$\langle x\rangle(t) = \langle\psi|\hat{x}|\psi\rangle = \int_D \overline{\psi}(x,t)\ x\ \psi(x,t)dx \tag{5}$$

$$\langle p\rangle(t) = \langle\psi|\hat{p}|\psi\rangle = \int_D \overline{\psi}(x,t)\ \left(-i\hbar\frac{\partial\psi(x,t)}{\partial x}\right) dx \tag{6}$$

Les écarts type sur la position et l'impulsion sont définis par

$$\Delta x(t) = \sqrt{\langle\psi|(\hat{x}-\langle x\rangle)^2|\psi\rangle} = \sqrt{\int_D \overline{\psi}(x,t)\ (x-\langle x\rangle(t))^2\psi(x,t)dx} \tag{7}$$

$$\Delta p(t) = \sqrt{\langle\psi|(\hat{p}-\langle p\rangle)^2|\psi\rangle} = \sqrt{\int_D \overline{\psi}(x,t)\ \left(-i\hbar\frac{\partial}{\partial x}-\langle p\rangle(t)\right)^2\psi(x,t)dx} \tag{8}$$

Ces fonctions mesurent l'incertitude sur la position et la quantité de mouvement. Selon le principe d'incertitude de Heisenberg, on a

$$\Delta x\Delta p \geq \frac{\hbar}{2} \tag{9}$$

à tout instant. Ceci nous dit qu'on ne peut jamais connaitre parfaitement la position et la quantité de mouvement (ou la vitesse) d'une particule en même temps.

2.2 Equation de Schrödinger adimensionnée

Les constantes $\hbar$ et m sont très petites dans les unités MKSA et il convient de choisir un autre système d'unité, plus naturel pour les systèmes quantiques. Imaginons E_*, L_* et T_* trois échelles typiques pour l'énergie, l'espace et le temps. Avec ces grandeurs, on peut définir un changement de variables

$$V = \widetilde{V}\, E_*, \quad \vec{r} = \widetilde{\vec{r}}\, L_*, \quad T = \widetilde{t}\, T_*, \quad \psi = \widetilde{\psi} L_*^{-1/2} \tag{10}$$

Les grandeurs avec tilde sont adimensionnées et on dit qu'on travaille dans un système d'unités relatif à E_*, L_* et T_*. Si on injecte ces changements de variables dans l'équation de Schrödinger, alors on peut arriver après quelques divisions sur

$$i\frac{\partial\widetilde{\psi}}{\partial\widetilde{t}} = -\underbrace{\frac{\hbar T_*}{mL_*^2}}\frac{1}{2}\frac{\partial^2\widetilde{\psi}}{\partial\widetilde{x}^2} + \underbrace{\frac{T_*E_*}{\hbar}}\widetilde{V}\,\widetilde{\psi} \tag{11}$$

Les groupes sous-lignés sont des nombres sans dimension. Un **système d'unités naturel** de l'équation de Schrödinger correspond à un choix particulier des échelles E_*, L_* et T_* qui vont mettre ces nombres à 1

$$\frac{\hbar T_*}{mL_*^2} = \frac{T_*E_*}{\hbar} = 1 \tag{12}$$

En pratique on choisit **une seule échelle typique**, puis on déduit les deux autres échelles naturelles. Voici les trois situations qui peuvent donc arriver :

- Soit on choisit E_* et alors on fixe $T_* = \hbar/E_*$ et $L_* = \hbar/\sqrt{mE_*}$
- Soit on choisit L_* et alors on fixe $E_* = \hbar^2/mL_*^2$ et $T_* = mL_*^2/\hbar$
- Soit on choisit T_* et alors on fixe $L_* = \sqrt{\hbar T_*/m}$ et $E_* = \hbar/T_*$

Les échelles naturelles E_*, L_* et T_* ainsi trouvés ne sont pas privées de sens physique. Elle nous informent sur **quelles distances et temps typiques** on peut s'attendra à voir des effets quantiques. L'équation de Schrödinger adimensionnée dans des échelles naturelles ne dépend plus d'aucun paramètre et cela est une simplification considérable

$$i\frac{\partial\widetilde{\psi}}{\partial\widetilde{t}} = -\frac{1}{2}\frac{\partial^2\widetilde{\psi}}{\partial\widetilde{x}^2} + \widetilde{V}\,\widetilde{\psi} \tag{13}$$

C'est cette équation qu'on utilisera **dans tous nos calculs numériques**. Afin d'alléger la notation, on n'utilisera plus ces tildes pour indiquer que les variables sont adimensionnées. Le problème qu'on résolve sera donc noté

$$i\frac{\partial\psi}{\partial t} = -\frac{1}{2}\frac{\partial^2\psi}{\partial x^2} + V\,\psi \tag{14}$$

sans tildes. Par contre, n'oublier pas que tous nos résultats numériques sont relatifs à un certain choix d'unités naturelles. Si cela est souhaitable, il est toujours possible d'exprimer les résultats à posteriori sous une forme dimensionnée. Quelques exemples :

- on réalise des simulations adimensionnées sur un domaine $x \in [-4, 4] \longrightarrow$ domain réel de $x \in [-4L_*, 4L_*]$
- on trouve un niveau d'énergie adimensionné $E_0 = 2.47 \longrightarrow$ énergie réelle $E_0 = 2.47E_*$
- on réalise une simulation sur l'intervalle de temps adimensionnée $t = 0$ à $10 \longrightarrow$ temps réel $t = 0$ à $10T_*$

Dans les échelles naturelles, la normalisation se fait toujours avec l'intégrale (3), sauf que l'intégrale couvre le domaine adimensionné. Les opérateurs $\hat{x}$, $\hat{p}$ et $\hat{H}$ sont simplifiés comme

$$\hat{x} = x \quad , \quad \hat{p} = -i\frac{\partial}{\partial x} \quad , \quad \hat{H} = -\frac{1}{2}\frac{\partial^2}{\partial x^2} + V(x,t) \tag{15}$$

C'est ces opérateurs qu'on utilise dans les formules pour $\langle x\rangle(t)$, $\langle p\rangle(t)$, $\Delta x(t)$ et $\Delta p(t)$. Le principe d'incertitude de Heisenberg prend la forme

$$\Delta x \Delta p \geq \frac{1}{2} \tag{16}$$

dans les unités naturelles.

2.3 Etats propres & décomposition spectrale

Si le potentiel $V(x)$ est stationnaire, on peut chercher des états propres. Il s'agit de solutions séparables de la forme

$$\psi(x,t) = \phi(x)\, \exp(-iEt) \tag{17}$$

On injecte cette proposition de solution dans l'équation de Schrödinger adimensionnée pour trouver que la fonction $\phi(x)$ est solution de

$$E\phi(x) = -\frac{1}{2}\frac{d^2\phi(x)}{dx^2} + V(x)\ \phi(x) \tag{18}$$

L'intervalle sur lequel on résout ce problème peut être $x \in]-\infty, +\infty[$, mais dans nos calculs numériques cet intervalle sera toujours fini, par exemple $x \in [a,b]$. On utilisera pratiquement toujours des conditions limites de Dirichlet homogène

$$CL\ :\ \phi(a) = 0 \quad , \quad \phi(b) = 0 \tag{19}$$

sur les bords. Ceci imite l'effet de barrières de potentiel infiniment hautes ($V(x<a) \to +\infty$ et $V(x>b) \to +\infty$) qui repoussent les particules. L'équation (18) et les conditions aux limites (19) définissent un problème aux valeurs propres. Le but y est de trouver pour quelles valeurs de E il existe des solutions ϕ non-nulles. Il existe une infinité de solutions de ce problème et chacune de ces **fonctions propres** a son **énergie propre**. On notera

$$E_n \quad \text{énergie propre avec sa fonction propre} \quad \phi_n(x) \tag{20}$$

Les fonctions propres $\phi(x) \in \mathbb{C}$. Grâce à la structure particulière de l'équation de Schrödinger (l'opérateur Hamiltonien est hermitien), on a la garantie que toutes les valeurs propres sont réels ($E \in \mathbb{R}$), ordonnés ($E_0 \leq E_1 \leq E_2 \leq \ldots$) et dénombrables par un entier n positif. On appelle **spectre**, l'ensemble des énergies propres admissibles. On peut toujours normaliser les fonctions propres vis à vis du produit hermitien et les utiliser comme une base orthonormée de l'espace de Hilbert des fonctions carré intégrables (sur D). Nous pouvons alors utiliser

$$\langle \phi_m | \phi_n \rangle = \int_a^b \overline{\phi}_m(x)\ \phi_n(x) dx = \delta_{mn} \tag{21}$$

ainsi que la relation de complétude

$$\mathcal{I} = \sum_{n=0}^{\infty} |\phi_n\rangle\langle\phi_n| \tag{22}$$

L'existence de cette relation d'orthonormalité et la complétude de la base des fonctions propres nous permet de réaliser une **décomposition spectrale** de la fonction d'onde. On écrit la fonction d'onde alors comme

$$\psi(x,t) = \sum_{n=0}^{\infty} c_n\, \phi_n(x)\, \exp(-iE_n t) \tag{23}$$

Cette fonction d'onde est solution de l'équation de Schrödinger, quelle que soit la valeur des coefficients c_n. Disposant de tous les E_n et ϕ_n, il suffit de préciser ces coefficients c_n pour prédire l'avenir du paquet d'onde à chaque instant ultérieur. En pratique, pour fixer c_n, il suffit de donner une condition initiale

$$CI\ :\ \psi(x,0) = \psi_0(x) \tag{24}$$

qui fixe la fonction d'onde à l'instant $t = 0$. Si on exprime la décomposition spectrale à cet instant $t = 0$, on a

$$\psi_0(x) = \sum_{n=0}^{\infty} c_n\, \phi_n(x) \tag{25}$$

Les coefficients c_n sont les coefficients d'expansion de $\psi_0(x)$ dans la base des fonctions propres. Utilisant l'orthonormalité de cette base, on a la formule

$$c_n = \langle \phi_n | \psi_0 \rangle \tag{26}$$

pour calculer c_n. Pour retrouver cette formule (vue dans votre cours d'algèbre), il suffit de multiplier (25) avec $\overline{\phi}_m(x)$ puis d'intégrer sur le domaine. Dans le membre de droite, nous utilisons ensuite $\langle \phi_m | \phi_n \rangle = \delta_{mn}$ et cela permet de réduire la somme à un seul terme :

$$\int_a^b \overline{\phi}_m(x)\psi_0(x)dx \quad = \quad \sum_{n=0}^{\infty} c_n \langle \phi_m | \phi_n \rangle = \sum_{n=0}^{\infty} c_n \delta_{mn} = c_m \tag{27}$$

Pour résumer, si on connait tous les états et énergies propres, $\phi_n(x)$ et E_n ainsi qu'une condition initiale $\psi_0(x)$, alors on peut calculer des coefficients d'expansion $c_n = \langle \phi_n | \psi_0 \rangle$ et calculer l'évolution spatio-temporelle de la fonction d'onde initial en évaluant la formule (23).

3 Tutoriels & programmation

3.1 Tutoriels sur la méthode des différences finies : jupyter notebook

Avant de vous lancez sur les projets, il sera nécessaire d'apprendre la méthode des différences à l'aide de tutoriels qui prennent la forme de notebooks Jupyter. Nos tutoriels et leurs corrigés sont à télécharger sur eCampus. Nous vous conseillons de travailler sur votre ordinateur personnel (téléchargez anaconda) ou sur une machine locale. Pour travailler sur une machine locale

- démarrer l'ordinateur puis sélectionner une séance linux
- ouvrir Terminal (petit bouton écran noir en bas à gacuhe).
- jupyter notebook + enter
- naviguer vers le répertoire où vous avez placé les notebooks + sélectionner un .ipynb
- travailler sur ces notebooks et sauvegarder régulièrement

Votre répertoire perso est toujours déporté (enregistré sur un serveur) : si vous changez d'ordinateur vous allez retrouvez tous vos fichiers. L'avantage de travailler sur la machine locale est que vous utilisez la pleine puissance de cet ordinateur. Sur JupyterHub, certaines fonctionnalités ne fonctionnent pas toujours très bien, notamment les animations.

Dans les tutoriels, on explique la méthode des différences finies qui est la méthode la plus simple pour discrétiser et résoudre des équations différentielles. Nous proposons 3 notebooks Jupyter

- Tutoriel 1 : problèmes stationnaires
  Apprendre à discrétiser et à résoudre numériquement un problème à conditions aux limites stationnaire pour une fonction $f(x)$. Problème de Poisson (1D) avec de différentes types de conditions aux limites. Autres équations différentielles d'ordre 2 linéaire.

- Tutoriel 2 : problèmes aux valeurs propres
  Apprendre à discrétiser un problème au valeurs propres défini par une équation différentielle d'ordre 2.

- Tutoriel 3 : problèmes instationnaires
  Apprendre à réaliser une simulation instationnaire pour une fonction $f(x, t)$ évoluant dans le temps. Notion de schéma temporel explicite et implicite. Exemples sur l'équation de diffusion, l'équation de d'advection, l'équation de Burgers. Génération d'animations des solutions.

A la fin de chaque notebook, on donne quelques exercices pour vous entrainer. Les programmes demandés dans les projets diffèrent parfois assez peu des exemples donnés dans les tutoriels.

3.2 Projets : Spyder

Pour réaliser les projets, nous vous conseillons de travailler sur Spyder. Il s'agit d'un programme assez pratique qui permet à la fois d'éditer et de débugger les codes. On peut y lancer les programmes avec de simples boutons 'play' et on peut y faire tout le post-traitement (figures, animation, analyse de données). Si vous travaillez sur votre machine personnelle, il suffit de lancer spyder à partir de l'interface anaconda. Pour lancer spyder sur une machine locale

- ouvrir Terminal
- spyder + enter

Une introduction au spyder sera donnée en salle de cours. Il y a notamment quelques options de visualisation à configurer afin d'assurer un meilleur affichage des animations.

Vous êtes libres de programmer comme vous le voulez, mais voici quand même quelques règles de bonne conduite. De manière générale, il est préférable de séparer les fonctions utilisées pour réaliser les calculs ou les visualisations (le code, la boite à outils), des scripts qui lancent les calculs (les vrai travail et le traitement des résultats (le post-processing). Ce découpage naturel suggère de travailler avec trois fichiers Python qui font chacun une partie. Dans les projets, tous les calculs restent assez légers (pas besoin d'un cluster). On conseille un minimum de 2 fichiers Python avec dans le cas idéal, le contenu suivant :

1. Le premier fichier rassemblera toutes vos fonctions pour pouvoir réaliser les calculs, les fonctions pour visualiser, les fonctions pour projeter, etc. En les mettant dans un seul fichier Python, celui-ci fera office de librairie de fonctions. Nous somme ici du coté programmeur. L'idée est qu'un utilisateur λ ne doit pas forcément tout comprendre de ce qui se passe dans ces fonctions, même si λ, ça sera vous en pratique.

2. Dans un deuxième fichier, nous sommes du coté utilisateur. On y définit les paramètres ainsi que les fonctions locales qui nous intéressent plus particulièrement (potentiel et fonction d'onde initiale, ...) et on y lance les calculs. On y fait également le traitement des résultats (création de graphes, animation, messages à l'écran). Pour pour pouvoir utiliser les fonctions définies dans le premier fichier, il suffit d'importer ce fichier comme une librairie de fonctions avec l'instruction

   `import ''le nom de ce premier fichier Python qui contient toutes ces fonctions''`.

   Vous pouvez alors accéder, coté utilisateur, à toutes les fonctions de la librairie personnalisée. L'appel a ces fonctions est un peu différent. Exemple pour utiliser une fonction `etats_propres`, programmée dans un fichier sauvegardé sous le nom "malibrairie.py", on aurait

   ```
   import malibrairie
   E,phi=malibrairie.etats_propres(x,V_puit_infini)
   ```

   dans le fichier Python, coté utilisateur. On utilise le nom de la librairie, un point et le nom de la fonction. Tout, exactement comme vous avez déjà l'habitude de le faire avec les librairies Numpy, Math ou Scipy. Vous pouvez même utiliser des alias, genre

   `import malibrairie as MQ`, puis utiliser cet alias.

   Enfin, dans le fichier utilisateur, on conseille d'utiliser des "cellules". Sous Spyder, on crée des cellules de calcul avec l'instruction `#%% nom de la cellule`. Spyder interprète cette ligne comme une ligne de commentaire, mais sait aussi que ça démarque une cellule. Toutes les instructions dans une cellule peuvent être lancées en une fois comme en Jupyter (shift + enter). Vous pouvez aussi lancer l'exécution d'une cellule avec un des boutons de l'éditeur (bouton play dans rectangle). L'utilisation de ces cellules est vivement conseillée car cela permet d'avancer progressivement dans le projet. Il ne faut pas relancer de longs calculs chaque fois !

Chapitre 1

Etats propres & décomposition spectrale

Dans ces projets, on s'intéresse à la mécanique quantique ondulatoire 1D, d'une particule placée dans un potentiel $V(x)$ **stationnaire**. Un objectif commun à tous les projets de ce chapitre sera de :

- réaliser un programme qui, pour un potentiel $V(x)$ donné, **calcule les états et énergies propres**
- créer des fonctions permettant de **faire plusieurs types de graphiques**
- réaliser une **décomposition spectrale sur les modes propres**
- faire une animation qui montre **l'évolution d'un paquet d'onde initial**

Le programme principal à réaliser est très proche de celui donné dans le tutoriel sur les valeurs propres. Il est donc normal qu'on demande un peu plus de choses à coté dans ces projets. Généralement, nous allons un peu plus loin sur les questions physiques.

1 Première partie du projet : tronc commun

1.1 Programme principal : fonction `etats_propres`

Programmer une fonction Python qui calcule et renvoie valeurs et fonctions propres de l'équation de Schrödinger. Les valeurs et vecteurs propres sont triées dans un ordre croissant et on ne veut pas des valeurs propres infini. L'utilisateur de ce programme doit pouvoir utiliser cette fonction avec un potentiel $V(x)$ de son choix et un maillage de calcul x de son choix. On supposera par contre que ce maillage reste toujours uniforme. Votre fonction doit donc avoir au minimum deux arguments d'entrée : `V_loc`, le nom de la fonction qui définit le potentiel et `x`, le maillage de calcul. Ce programme produit au minimum deux sorties, un vecteur `E` contenant les énergies propres et une matrice `phi` contenant, sur ses colonnes, les vecteurs propres.

Préparer bien tout sur une feuille de papier, suivant les pas indiqués dans les tutoriels. Démarrer la programmation en partant des programmes "valeurs propres" du tutoriel.

Validation 1 : puit de potentiel infiniment profond

Sous forme dimensionnée, le puit de potentiel infiniment profond est défini par

$$V(x) = \begin{cases} 0 & , \quad x \in [0, L] \\ +\infty & , \quad \text{ailleurs} \end{cases} \tag{1.1}$$

La seule grandeur naturelle disponible est L, la largeur du puit et on fixe donc l'échelle naturelle $L_* = L$. En découlent que $E_* = \hbar^2/mL_*^2$ et $T_* = mL_*^2/\hbar$. Les fonctions et énergies propres du problème adimensionné satisfont le problème

$$E\phi(x) = -\frac{1}{2}\frac{d^2\phi(x)}{dx^2} \quad , \forall x \in [0,1] \quad , \quad CL \ : \ \phi(0) = 0 \quad , \quad \phi(1) = 0 \tag{1.2}$$

Trouver à la main les solutions de ce problème, c.a.d. tous les E_n et $\phi_n(x)$ admissibles. Mener ensuite les expériences numériques suivantes :

1. Définir une fonction `V_puit_infini` qui accepte ce maillage comme argument et renvoie un vecteur contenant des zéros et de la même taille de ce maillage. Définir un maillage uniforme de 101 points qui couvre l'intervalle $x \in [0, 1]$. Lancer votre programme qui calcule les énergies et états propres et avec ces deux arguments `x` et `V_puit_infini`. Vérifier que vous trouvez bien les mêmes énergies et fonctions propres que dans votre calcul analytique. Faire aussi quelques graphiques qui montrent quelques unes des premières fonctions propres.
2. Etudier comment la précision sur la première valeur propre s'améliore en faisant le calcul avec un maillage de plus en plus fin. Nous voulons un tableau qui montre la convergence des calculs numériques vers les bonnes énergies propres E_0, E_1, E_2, E_3 exactes et pour δx, le pas d'espace, décroissant. Faire un graphe pour illustrer cette convergence.

Validation 2 : oscillateur harmonique quantique

Sous forme dimensionnée, le potentiel de l'oscillateur harmonique est défini par

$$V(x) = \frac{kx^2}{2} = \frac{m\omega^2 x^2}{2} \quad , \quad x \in]-\infty, \infty[\tag{1.3}$$

Ici $\omega = \sqrt{k/m}$ est la fréquence naturelle d'oscillation d'une masse m attachée à un ressort de raideur k. Il est donc naturel de choisir $T_* = \omega^{-1}$ comme échelle naturelle de temps. En conséquence, nous fixons $L_* = \sqrt{\hbar/m\omega}$ et $E_* = \hbar\omega$ comme échelles naturelles d'énergie et d'espace dans ce problème. Les fonctions et énergies propres du problème adimensionné satisfont

$$E\phi(x) = -\frac{1}{2}\frac{d^2\phi(x)}{dx^2} - \frac{x^2}{2}\phi(x) \quad , \forall x \in]-\infty, \infty[\quad , \quad CL \ : \ \phi(-\infty) = 0 \quad , \quad \phi(+\infty) = 0 \tag{1.4}$$

Dans votre cours de mécanique quantique, les énergies propres E_n (adimensionnées) sont trouvées comme

$$E_n = n + 1/2 \tag{1.5}$$

pour $n = 0, 1, 2 \ldots$. Les fonction propres sont des fonctions d'Hermite. Vous pouvez trouver les expressions adimensionnées dans le notebook `Rappel2.ipynb`. Mener les expériences numériques suivantes :

1. Définir une fonction `V_harmonique` qui accepte le maillage comme argument et renvoie le vecteur $x^2/2$ sur les points de ce maillage. Définir un maillage uniforme de 101 points qui couvre l'intervalle $x \in [-5, 5]$. Lancer votre programme qui calcule les énergies et états propres. Vérifier que vous trouvez bien les mêmes énergies propres que dans le calcul analytique.
2. Pourquoi le calcul marche-t-il moins bien avec un domaine de calcul $x \in [-1, 1]$. Pourquoi doit-on prendre un domain de calcul suffisamment grand ?

1.2 Amélioration : normalisation des fonctions propres

Nous voulons des fonctions propres normalisées telles que

$$< \phi_n | \phi_n >= \int_a^b |\phi_n(x)|^2 dx = 1 \tag{1.6}$$

Le calcul numérique brut ne nous donne que des valeurs nodales des vecteurs propres et la normalisation n'a pas eu lieu. Si $\phi_{n,brut}$ est le n-ième vecteur propre trouvé par le calcul numérique alors on peut calculer l'intégrale du produit hermitien de manière approchée par une formule de quadrature numérique. Cette formule sera de la forme

$$< \phi_{n,brut} | \phi_{n,brut} > \approx \sum_{j=0}^{M} |\phi_{n,brut}(x_j)|^2 w_j \tag{1.7}$$

Ici w_j sont des poids d'intégration qui dépendent de la méthode choisie (trapèze, simpson, ...). A vous de vous rappeler quels sont ces poids ou plus généralement, comment intégrer numériquement une fonction définie par des valeurs nodales. Pour calculer la somme, utiliser `np.sum` de Numpy. Une fois que vous savez comment calculer cette intégrale de manière apprcohée, il suffit de renormaliser les vecteurs propres comme

$$\phi_n = \frac{\phi_{n,brut}}{\sqrt{< \phi_{n,brut} | \phi_{n,brut} >}} \tag{1.8}$$

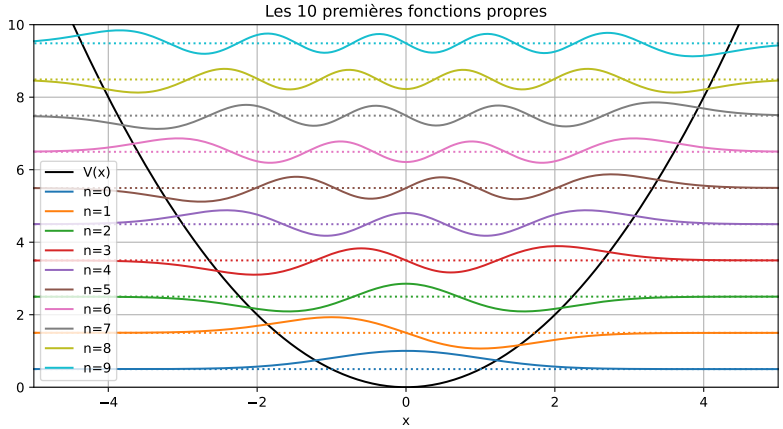


FIGURE 1.1 – Représentation graphique des fonctions et énergies propres. Ici on montre la partie Re($\phi(x)$)

Pour chaque vecteur propre $\phi_{n,brut}$ on calcule le produit $< \phi_{n,brut} | \phi_{n,brut} >$ et on l'utilise pour remet à échelle ce vecteur, c'est tout. A la fin de votre programme validé, ajouter une boucle qui parcourt donc tous les vecteurs propres et leur applique cette procédure de normalisation.

Vous pouvez tester votre code amélioré sur les fonctions propres du puit de potentiel infiniment profond car dans ce cas, on peut calculer les fonctions propres normalisées de manière analytique.

1.3 Figures : quelques fonctions pour automatiser

Afin de faciliter l'interpretation des résultats numériques, vous devez programmer quelques fonctions qui reçoivent les résultats calculées et permettent de les visualiser.

1. Programmer une fonction capable de montrer la n-ième fonction d'onde $\phi_n(x)$ en fonction x, partie réelle, partie imaginaire et module au carré sur trois sous-figures (3 lignes) d'une même figure.
2. Programmer une fonction qui produit un spectrogramme : les énergies discrètes comme des lignes horizontales et la valeur de cette énergie (ou l'indice n) à coté de ces lignes. Il est préférable de prévoir un argument pour ne que monter les N premières énergies.
3. En mécanique quantique, on représente souvent les fonctions propres ensemble avec les énergies propres sur un diagramme tel que celui de la figure 1.1. On y montre le potentiel $V(x)$ (ligne noir), ensemble avec les énergies discrètes (lignes pointillés), ensemble avec les fonctions d'ondes $C\phi_n(x)$ (partie réelle ou imaginaire), multiplié par une constante C bien choisie (les courbes ne doivent pas se croiser, mais on doit pouvoir "voir" la fonction). Dans le graphe, les fonctions d'ondes sont également décalées verticalement par la quantité E_n afin de suggérer leur lien avec le niveau d'énergie E_n. Ici aussi, il est préférable de prévoir un argument pour ne que monter les N premières énergies avec leur fonctions. Le coefficient C est un autre argument d'entrée utile car selon le problème étudié, C sera très différent (le but est de voir les fonctions propres)

Tester ces fonctions de visualisation sur le cas de l'oscillateur harmonique quantique.

1.4 Projection & animation : évolution temporelle d'un paquet d'onde

Au-dessus, nous avons expliqué comment réaliser une décomposition spectrale, qui permet de savoir comment un paquet d'onde initial $\psi_0(x)$ évolue dans le temps. Dans notre représentation discrète, sur un maillage de $M+1$ points, le programme permet de calculer M_{max} fonctions propres et énergies propres (à vous de identifier M_{max}). On ne peut donc écrire qu'une décomposition spectrale tronquée

$$\psi(x,t) \approx \sum_{n=0}^{M_{max}-1} c_n \phi_n(x) e^{-iE_n t} \tag{1.9}$$

Les coefficients c_n se calculent toujours comme

$$c_n = \langle \phi_n | \psi_0 \rangle = \int_a^b \overline{\phi}_n(x)\psi_0(x)\,dx \tag{1.10}$$

c'est à dire avec un produit hermitien. Ces intégrales peuvent être évaluées de manière approchée avec une formule de quadrature qui prend la forme

$$c_n \approx \sum_{j=0}^{M_{max}-1} \overline{\phi}_n(x_j)\psi_0(x_j)w_j \tag{1.11}$$

Ici w_j sont les mêmes poids d'intégration que dans la discussion sur la normalisation. Les valeurs $\psi_0(x_j)$ sont les valeurs nodales de la fonction d'onde initiale et doivent donc être calculé au préalable avant de pouvoir évaluer cette formule. Par contre, si vous avez donc réussi à normaliser les fonctions propres, vous allez sans doute réussir ce calcul approchée de ces coefficients c_n.

Avec c_n connus, on peut enfin évaluer la somme (1.9) à tout instant voulu et partout dans le domaine. On peut donc calculer l'évolution temporelle d'un paquet d'onde initial arbitraire et produire des animations qui montrent comment ce paquet d'onde évolue dans le temps.

En pratique, nous vous demandons de créer deux programmes supplémentaires :

1. Le premier programme `projection` réalise la projection de la fonction d'onde initiale $\psi_0(x)$ sur les fonctions propres. Ce programme reçoit le maillage `x`, la matrice `phi` contenant sur ses colonnes les fonctions propres normalisées et `psi0_loc` le nom de la fonction Python qui contient la définition de la condition initiale. On imagine ici, que cette fonction sera définie plus tard par l'utilisateur et que celui-ci passera le nom de sa fonction à ce programme `projection`. Le programme doit retourner un vecteur `c` contenant les coefficients $c_0, c_1, \ldots, c_{M_{max}-1}$.
   Afin de tester la qualité et la justesse de la projection, il est utile de calculer dans ce programme, la fonction

   $$\psi_{0,approx}(x) = \sum_{n=0}^{M_{max}-1} c_n\phi_n(x) \tag{1.12}$$

   sur les points du maillage, ainsi que l'erreur relatif

   $$\epsilon = \frac{\max_x |\psi_0(x) - \psi_{0,approx}(x)|}{\max_x |\psi_0(x)|} \tag{1.13}$$

   La valeur de ϵ sera imprimée à l'écran et donne une indication quantitative de l'erreur de projection. La somme tronquée (1.12) étant une approximation, on ne retrouve jamais exactement la même fonction ψ_0 après la projection. Il est aussi conseillé de faire un graphe au sein de cette fonction, qui montre $\psi_{0,approx}(x)$ et $\psi_0(x)$ (partie réelle et partie imaginaire) en fonction de x. Cela permet de constater visuellement les différences entre les deux fonctions et de valider ou non la projection. La projection ne sera que bien réussi si les deux fonctions sont très proches dans ces graphes.
2. Faire un deuxième programme `psi_eval` qui donne le vecteur d'état du système à temps t selon la formule (1.9). L'utilisateur doit donner les coefficients d'expansion, c'est à dire le tableau `c`, ainsi que le maillage `x` et la matrice des vecteurs propres `phi`. Il doit aussi donner soit un "vecteur" temps $t = [t_0, t_1, t_2, \ldots, t_N]$ contenant les instants discrets sur lesquelles il souhaite la solution. La fonction retourne une matrice `psi` contenant sur ses $N+1$ colonnes les vecteurs d'état aux instants $t_0, t_1, t_2, \ldots, t_N$.

Attention : penser à initialiser les vecteurs/matrices (`c`, `psi0_approx`, `psi0`, `psi`, ...) contenant les fonctions d'ondes comme des vecteurs/matrices complexes. Si vous ne le faites pas, la projection, la reconstruction de la fonction $\psi_{0,approx}$ et les animations ne seront pas bonnes car vous ignorez la partie imaginaire. Nous travaillons sur $\mathbb{C}$ et non sur $\mathbb{R}$. Numpy travaille sur $\mathbb{R}$ par défaut.

Une fois ces programmes crées, l'utilisateur devra procéder comme dans la validation ci-dessous pour pouvoir réaliser des animations de l'évolution temporelle d'un paquet d'onde.

Validation : puit de potentiel infini

On utilise le potentiel du puit de potentiel infiniment profond. En adimensionné, le domaine sera donc $x \in [0, 1]$.

1. Définir un maillage `x` et une fonction Python pour le potentiel (on peut réutiliser la fonction du puit infini)
2. Calculer les états et énergies propres, `E` et `phi`.
3. On suppose un paquet d'onde initial de la forme $\psi_0(x) = \sqrt{30}\,x(1-x)$. Définir une fonction Python `psi0(x)` adéquate qui renvoie un vecteur `psi_init` qui contient cet état initial.
4. Lancer le programme qui fait la projection pour obtenir le vecteur des coefficients `c`.
5. Définir un tableau `t` qui contient tous les instants discrets de 0 à 2 en $N = 200$ pas de temps. Lancer le programme qui calcule la fonction d'onde à ces les instants. Cela produit une matrice `psi`.
6. Réaliser une animation du paquet d'onde, comme vous l'avez appris dans le tutoriel 3 sur les problèmes instationnaires. Il suffira de parcourir les colonnes de la matrice `psi`. L'animation ressemble à une corde de guitare tenue par les bords et qui oscille entre les positions haute et basse. Il faut fixer l'étendu de l'axe y. Vous pouvez comparer votre animation à celle produite par vos collègues qui sont sur les projets instationnaires.

2 Deuxième partie du projet : choisir 1 sujet parmi 5

Après avoir réalisé ce premier volet du projet, chaque groupe continuera sur l'un des 5 sujets qui sont proposés ici. L'idée est que les différents groupes travaillent sur des sujets différents.

2.1 Projet 1 : Programme plus performant avec des matrices creuses

Les matrices qui définissent le problème au valeurs propres discret sont très **creuses, c'est à dire plein de zéros**. Ces zéros n'influencent pas vraiment le calcul mais on les charge néanmoins en mémoire. Puis, on effectue de nombreuses multiplications par zéro ce qui ne fait que ralentir le calcul numérique des vecteurs propres. L'utilisation de matrices creuses (librairie `scipy.sparse`) permet une économie en mémoire et accélère fortement les calculs des valeurs et vecteurs propres. C'est aussi le seul moyen pour faire des calculs sur des maillages fins avec de très nombreux points. Le but de ce projet est **de proposer un nouveau programme de calcul des valeurs propres plus performant et utilisant les matrices creuses.** Ce projet s'adresse aux étudiants qui aiment programmer et qui n'ont pas peur d'utiliser des librairies un peu plus pro.

Pour mener ce projet à terme, procéder par les étapes suivantes.

1. Dans les tutoriels, on vous a appris à coder un problème à valeurs propres généralisé $\mathbf{Af} = \lambda\mathbf{Bf}$. La matrice $\mathbf{B}$ était presque une matrice identité. Seuls les lignes 0 et M étaient différent et permettent d'imposer la condition aux limites, $f_0 = 0$ et $f_M = 0$ en l'occurence. Dans la version creuse du programme, on va tout d'abord éliminer ces lignes correspondants aux points de bords, car ces lignes ne contiennent pas vraiment d'information utile. De cette manière, on arrive à une formulation différente du problème, qui a la structure d'un problème aux valeurs propres ordinaire

$$\underbrace{\begin{bmatrix} \times & \times & & & \\ \times & \times & \times & & \\ & \ddots & \ddots & \ddots & \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix}}_{\mathbf{AA}} \underbrace{\begin{bmatrix} f_1 \\ f_2 \\ \dots \\ f_{M-2} \\ f_{M-1} \end{bmatrix}}_{\mathbf{ff}} = \lambda \underbrace{\begin{bmatrix} f_1 \\ f_2 \\ \dots \\ f_{M-2} \\ f_{M-1} \end{bmatrix}}_{\mathbf{ff}} \tag{1.14}$$

   Dans notre cas de l'équation de Schrödinger, la matrice **AA** sera tri-diagonale et même symétrique pour un maillage uniforme. Identifier cette matrice symétrique. Cette élimination des points de bords est intéressant et nécessaire car on ne dispose pas de méthode spécifique "creuse" pour trouver les valeurs propres de problèmes aux valeurs propres généralisées. L'élimination des points de bords enlève par contre une certaine facilité dans la gestion des indices : l'élément [0] du vecteur **ff** en Python ne sera plus f_0 mais f_1. **Il faudra en prendre note dans vos programmes et dans vos boucles qui définissent la matrice.**

2. Nous devons maintenant définir la matrice **AA** comme une matrice creuse. L'idée est de ne plus mettre en mémoire les très nombreux zéros de la matrice. A la place, on mettra en mémoire uniquement les éléments

non-nulles (valeurs) et leur coordonnées (indices de colonne et indices de ligne). Donnons un exemple comment procéder en Python. On imagine 3 vecteurs,

$$\texttt{row} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 4 \end{bmatrix} \quad , \quad \texttt{col} = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 4 \end{bmatrix} \quad , \quad \texttt{val} = \begin{bmatrix} -0.1 \\ 3.2 \\ -1.1 \\ 0.9 \end{bmatrix} \tag{1.15}$$

et on imagine que ceux-ci contiennent les coordonnées (indices de ligne dans `row`, indices de colonne dans `col`) et les valeurs dans (`val`). Pour être encore plus clair, on veut ici une matrice avec

- à la ligne 0 et la colonne 1, la valeur -0.1,
- à la ligne 1 et la colonne 2, la valeur 3.2
- à la ligne 2 et la colonne 0, la valeur -1.1
- à la ligne 4 et la colonne 4, la valeur 0.9

Ensuite, on doit encore préciser la taille totale de la matrice. Dans notre exemple, on veut créer une matrice `A` de taille 5×6, avec ces éléments non-nuls. En Python, on crée tout d'abord les vecteurs `row`, `col`, `val`. La longueur de ces vecteurs correspond exactement au nombre d'éléments non-nuls dans la matrice creuse. Toute la difficulté de la programmation des matrices est déplacée dans la programmation de ces 3 vecteurs. Une fois ces vecteurs créés, on utilise la fonction `coo_matrix` de la librairie `scipy.sparse`). L'instruction `A=coo_matrix((val,(row,col)),shape=(5,6))` crée la matrice

$$\texttt{A} = \begin{bmatrix} 0 & -0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3.2 & 0 & 0 & 0 \\ -1.1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.9 & 0 \end{bmatrix} \tag{1.16}$$

dans un format creux, c'est à dire, sans mettre en mémoire les très nombreux zéros. Il existe plein d'autres formats de matrices creuses, mais ce format dite de coordonnées est un des plus simples selon nous. Créer en tant qu'exercice une matrice diagonale creuse de taille $M+1 \times M+1$ qui contient sous la diagonale les éléments M à 1. Exemple pour $M = 9999$:

$$\texttt{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 9999 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9998 & 0 & 0 & 0 & 0 \\ 0 & 0 & 9997 & 0 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{1.17}$$

Il faudra donc trouver un moyen de créer les vecteurs `row`, `col`, `val` qui permettent de localiser et de définir les entrées non-nulles.

3. Estimer le nombre d'octets nécessaires pour mettre en mémoire la matrice de l'exercice précédente en format plein (avec les zéros) et en format creux (sans les zéros mais avec les coordonnées). Un nombre réel = 16 octets en précision double.
4. Tenter pour un cas particulier de potentiel $V(x)$ de programmer la matrice **AA** sous format creux. Le mieux est de passer par une boucle, mais cela n'est pas forcément nécessaire.
5. Dans la librairie `scipy.sparse.linalg` il existe une fonction `eigsh` qui permet de calculer de manière très efficace des valeurs propres de matrices symétriques réelles. Consulter les documentations de cette fonction. On voudra utiliser cette fonction pour trouver les `Nval` valeurs propres les plus petites et les vecteurs propres associés. Identifier l'instruction Python qui permettra de le faire.
6. Créer un nouveau programme qui utilise cette méthode et cette représentation de matrice creuse. Attention, il ne peut y avoir aucune ligne du style `A=np.zeros(M+1,M+1)` dans le code car cela crée une matrice pleine et occupe donc de la mémoire pour rien. Il faut, à la place construire les tableaux `row`, `col`, `val` adéquats. N'oubliez pas que suite à l'élimination des points de bords, la bonne gestion des indices en Python sera délicate. Ce nouveau programme acceptera une entrée supplémentaire, `Nval` le nombre de valeurs propres voulu par

l'utilisateur. Il est en effet nécessaire de préciser à l'algorithme combien de valeurs propres on souhaite. On ne sera pas en mesure de calculer toutes les valeurs et fonctions propres mais cela n'est pas très grave : généralement on est intéressé par les plus petites valeurs et fonctions propres.

7. Valider votre programme version creuse en le comparant à la version pleine et pour le même maillage uniforme.
8. Comparer les vitesses d'exécution des deux programmes pour plusieurs maillages $100 \leq M \leq 4000$. Le calcul creux doit renvoyer les 10 premiers vecteurs propres. Relancer ces mesures de temps de calcul, pour le calcul creux, en demandant les 20, 40, 80 premiers modes. Faire un tableau récapitulatif et éventuellement un graphe en échelle logarithmique (temps de calcul vs. M).
9. Calculer les 100 premières énergies propres de l'oscillateur harmonique avec un maillage uniforme de 10001 points ($M = 10000$) entre $x = -10$ et $x = 10$. Montrer que la précision sur ces énergies propres est bien meilleure que ce que vous avez pu trouver avec une résolution plus petites (Attention, ne tentez pas d'utiliser la version pleine avec cette résolution élevée).
10. Utiliser votre programme pour trouver les 10 premiers niveaux d'énergie d'une particule dans un potentiel $V(x)$ un peu plus complexe qui présente un peu plus de variation spatiale (par exemple, un double puit de potentiel, défini par un polynôme d'ordre 4) .

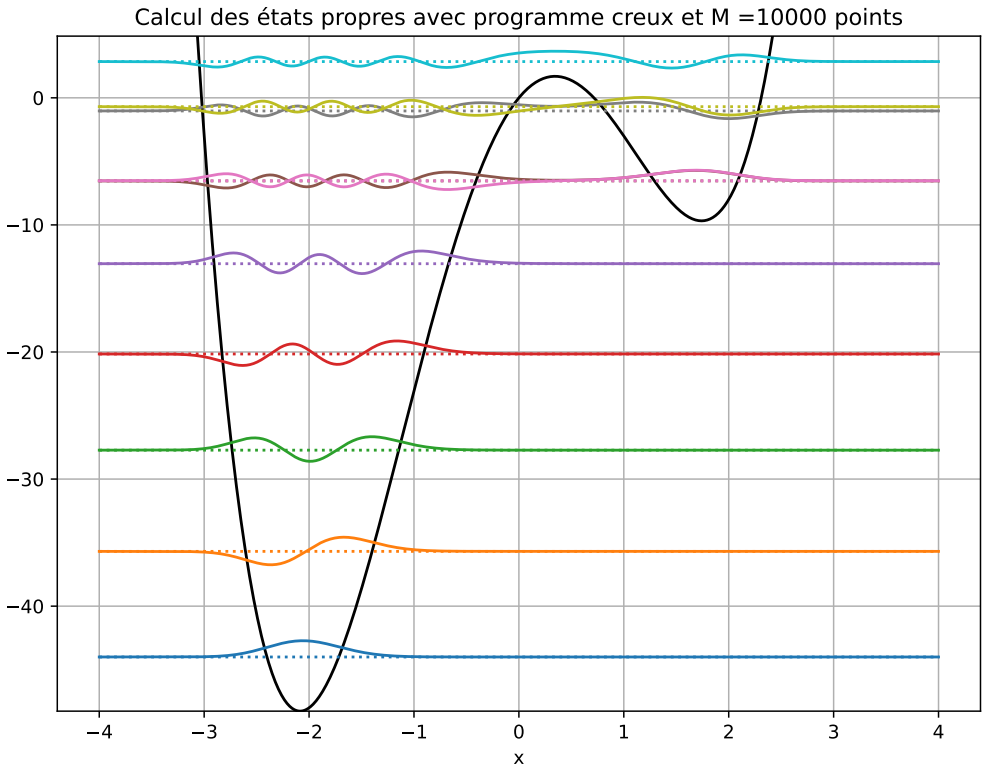


FIGURE 1.2 – Niveaux d'énergie dans un double puit de potentiel, calculé sur un maillage avec 10000 points.

2.2 Projet 2 : états cohérents de l'oscillateur harmonique

Une particule de masse m classique qui se déplace dans un potentiel $V(x)$ satisfait l'équation de mouvement $m\ddot{x} = -dV/dx$. Pour le potentiel $V(x) = m\omega^2 x^2/2$ de l'oscillateur harmonique, on en déduit $\ddot{x} + \omega^2 x = 0$. La solution pour la position $x(t)$ et la quantité de mouvement $p(t) = m\dot{x}(t)$ est

$$x(t) = A\cos(-\omega t + \chi) \quad , \quad p(t) = mA\omega \sin(-\omega t + \chi) \tag{1.18}$$

Ici A et χ sont une amplitude et une phase arbitraire. Les états cohérents de l'oscillateur harmonique quantique sont des états très remarquables qui se rapprochent au maximum de leur cousins classiques. La particule quantique dans cet état est décrite par une fonction d'onde $\psi_\alpha(x,t)$ mais sa position moyenne et sa quantité de mouvement moyenne varient exactement comme pour une particule classique

$$\langle x\rangle(t) = A\cos(-\omega t + \chi) \quad , \quad \langle p\rangle(t) = mA\omega \sin(-\omega t + \chi) \tag{1.19}$$

Ici nous avons $\langle x\rangle(t) = \langle\psi_\alpha|\hat{x}|\psi_\alpha\rangle$ et $\langle p\rangle(t) = \langle\psi_\alpha|\hat{p}|\psi_\alpha\rangle$. Un état cohérent est aussi un état d'incertitude minimale ou $\Delta x \Delta p = \hbar/2$.

Pour définir un état cohérent dans notre formalisme sans dimension, on introduit d'abord un nombre complexe

$$\alpha = \frac{Ae^{i\chi}}{\sqrt{2}} \tag{1.20}$$

où on reconnait A, χ, l'amplitude (adimensionnée) et la phase initiale. La fonction d'onde $\psi_\alpha(x,t)$ associée à un état cohérent de l'oscillateur mécanique quantique est

$$\psi_\alpha(x,t) = \sum_{n=0}^{+\infty} \underbrace{\frac{\alpha^n}{\sqrt{n!}} e^{\frac{-|\alpha|^2}{2}}}_{c_n} \phi_n(x) e^{-iE_n t} \tag{1.21}$$

dans notre base de fonctions propres normalisées et adimensionnées.

1. Calculer les fonctions propres de l'oscillateur harmonique avec votre programme sur un domaine adimensionné $x[-L/2, L/2]$. Choisir L suffisamment grand, mais pas trop grand non plus.
2. Comparer graphiquement les fonction propres numériques aux fonctions propres exactes (adimensionnées). Utiliser les fonctions `hermite` et `factorial` de la librairie `scipy.special`. Faire la comparaison pour quelques valeurs de n entre 0 et 20. Que constatez-vous ?
3. Apporter quelques modifications à la fin de votre programme qui calcule les fonctions propres afin de fixer la phase, ici le signe, des fonctions propres numériques de manière systématique. On conseille l'algorithme suivant. Pour chaque fonction propre numérique, calculer le minimum et le maximum de la partie réelle de cette fonction d'onde sur l'intervalle $x \in [0, L/2]$. Si la valeur absolue du minimum est plus grande que la valeur absolue du maximum, renverser le signe de $\phi_n(x)$.
4. Après ce traitement, vous pouvez utiliser la formule (1.21) pour calculer la fonction d'onde de l'état cohérent. Réaliser une animation qui montre l'évolution temporelle de la partie réelle et du module au carré de ψ_α en fonction du temps. Vous devez voir un paquet d'onde dont le module au carré correspond à une Gaussienne qui se déplace, comme une particule classique. L'amplitude A ne doit pas être ni trop petite, ni trop grande.
5. Construire une représentation discrète l'opérateur impulsion $\hat{p} = -i\dfrac{\partial}{\partial x}$. Le but est de traduire l'opération

   $$\hat{p}\psi(x,t) \longrightarrow \mathtt{P}\psi \tag{1.22}$$

   Ici le ψ de droite correspond au vecteur de valeur nodales et `P` est une matrice carrée. Il suffit d'identifier les bonnes formules d'approximation (différences finies centrées sur les points intérieurs, décentrées sur les points de bord) pour calculer les dérivées en tous les points, puis de récrire ces formules sous forme matricielle. Ceci permet d'identifier le contenu de matrice `P` et ensuite de le programmer. Tester votre matrice `P` sur la fonction $f(x) = e^{ikx}$. L'application $\hat{p}e^{ikx} = ke^{ikx}$. Faire un graphe qui compare donc ke^{ikx} à `Pf` sur des valeurs nodales.
6. Calculer numériquement la position et l'impulsion moyenne $\langle\hat{x}\rangle(t)$ et $\langle\hat{p}\rangle(t)$ et montrer que ces fonctions évoluent comme pour une particule classique comme $\langle x\rangle(t) = A\cos(-t+\chi)$ et $\langle p\rangle(t) = A\sin(-t+\chi)$ dans le format adimensionné.
7. Calculer numériquement $\Delta x(t) = \sqrt{\langle\psi_\alpha|(\hat{x} - \langle x\rangle)^2|\psi_\alpha\rangle}$ et $\Delta p(t) = \sqrt{\langle\psi_\alpha|(\hat{p} - \langle p\rangle)^2|\psi_\alpha\rangle}$. Montrer que $\Delta x(t) \approx \Delta p(t)$ et que $\Delta x(t)\Delta p(t) \approx \ldots$ reste presque constant dans le temps. Vérifier l'inégalité de Heisenberg (adimensionnée) de manière numérique et montrer que l'état cohérent est un état d'incertitude minimal.

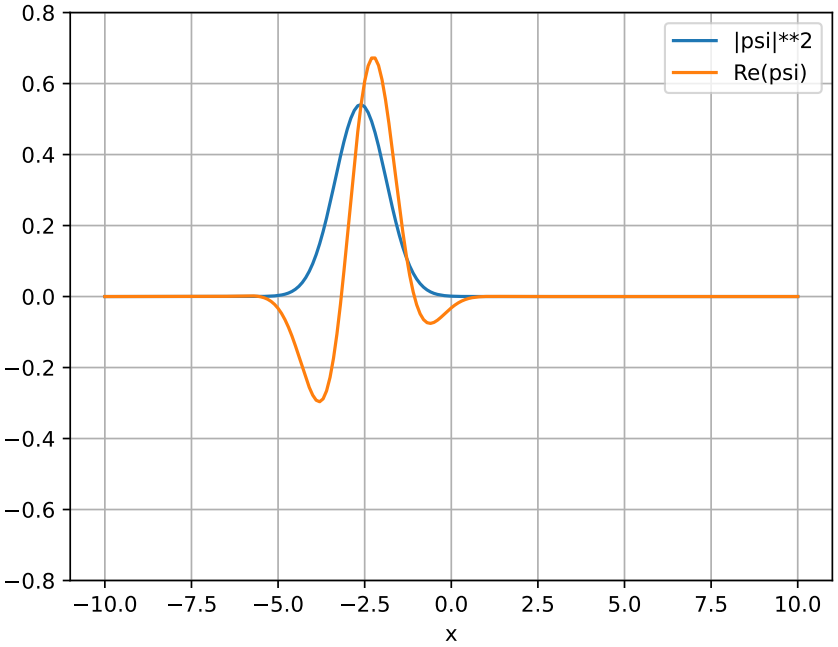


FIGURE 1.3 – Image issue de l'animation qui montre l'évolution temporelle du paquet d'onde cohérent. Le paquet Gaussien se déplace comme un oscillateur harmonique au cours du temps.

2.3 Projet 3 : simulation de l'effet tunnel

L'effet tunnel est l'effet quantique par lequel une particule de masse m avec énergie H_0 initiale arrive à franchir une barrière de potentiel de hauteur V localement plus grande que H_0. Selon les lois de la mécanique classique cela n'est pas possible. Dans l'exercice, nous plaçons notre fine barrière de potentiel au centre d'un puit de potentiel infiniment profond avec une largeur L. On décide d'utiliser cette largeur du puit infini, $L = L_*$ comme échelle spatiale naturelle. Le domaine adimensionné de calcul est alors $x \in [0, 1]$ et on définit le potentiel adimensionné comme

$$V(x) = \begin{cases} 0 & , \quad 0 < x < \dfrac{1-a}{2} \\ V & , \quad \dfrac{1-a}{2} < x < \dfrac{1+a}{2} \\ 0 & , \quad \dfrac{1+a}{2} < x < 1 \\ +\infty & , \quad \text{ailleurs} \end{cases} \tag{1.23}$$

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

$$\psi_0 \approx C e^{ikx} \exp\left(-(x - x_0)^2 / 2d^2\right) \tag{1.24}$$

Ici quelques explications sont nécessaires.

- $C \in \mathbb{R}$ est un coefficient de normalisation qui garantit $\langle \psi_0 | \psi_0 \rangle = 1$. Calculer C analytiquement en supposant que le paquet reste loin des bords (que le domaine d'intégration est $]-\infty, +\infty[$).
- Le paramètre x_0 définit le centre de la Gaussienne et il convient de le choisir entre le bord de gauche et la barrière $x_0 \in [0, (1-a)/2]$ pour placer le paquet devant la barrière.
- Le coefficient d définit la largeur de la Gaussienne et sera petit. Avec $d \ll 1$ on peut localiser le paquet dans une zone de taille d.
- Le nombre $k = 2\pi/\lambda$ est un nombre d'onde. Idéalement, on veut une longueur d'onde courte $\lambda \ll 1$ afin d'avoir un paquet d'onde qui ressemble à un paquet d'onde.

Le paquet d'onde initial et le potentiel sont définis par 5 paramètres k, x_0, d, V, a et une difficulté dans ce projet est qu'il fait les choisir afin de retrouver une situation qui permet de mettre en évidence l'effet Tunnel. Ceci fixe deux contraintes sur les paramètres. La première contrainte est que le paquet d'onde doit se propager vers la droite. La vitesse moyenne initiale doit donc être positive, soit

$$U_0 = \langle \psi_0 | \hat{p} | \psi_0 \rangle > 0 \tag{1.25}$$

Calculer cette vitesse moyenne initiale analytiquement et exprimer-la en fonction des paramètres k, x_0, d. Avec cette vitesse initiale, on peut estimer le temps nécessaire pour que le paquet parcourt le domain de calcul. Donner ce temps

$$T = \ldots \tag{1.26}$$

en fonction des paramètres k, x_0, d. Cette information nous sera utile au moment où on définit la durée de l'animation. Dans un temps T on peut être à peu près certain que le paquet aura atteint la barrière. La deuxième contrainte est que le paquet d'onde doit avoir une énergie initiale H_0 plus petite que V car sinon, on ne peut pas vraiment parler d'effet tunnel. Ainsi on a besoin de définir les paramètres tels que

$$H_0 = \langle \psi_0 | \hat{H} | \psi_0 \rangle > V \tag{1.27}$$

Calculer cette énergie moyenne initiale analytiquement (on suppose que le paquet est dans la zone ou $V = 0$) et exprimer-la en fonction des paramètres k, x_0, d. En pratique, on adaptera la hauteur de la barrière, c'est à dire le paramètre V pour avoir une barrière qui est plus élevée que H_0.

1. Créer une fonction qui renvoie le potentiel adimensionné $V(x)$ sur les points du maillage. Les paramètres V et a sont des variables globales qu'on définira plus tard.
2. Créer une fonction qui calcule la fonction d'onde initiale $\psi_0(x)$ sur le maillage. Les paramètres k, d, x_0 sont des variables globales qu'on définira plus tard.

3. Calculer pour quelques valeurs de V et a les fonctions propres $\phi_n(x)$ et les énergies propres E_n. Il faudra choisir la barrière suffisamment fine et haute. Visualiser les fonctions propres avec leur énergie propre comme en figure 1.1. Nous voulons une situation où il existe de nombreux niveaux d'énergie avec $E_n < V$ car ceci permet de limiter les effets de bord (le fait que notre barrière se trouve dans un puit infini). Utiliser un maillage initial pas trop fin pour tester dans un premier temps. Pour l'animation, nous conseillons un maillage fin avec au minimum $M = 1000$ points.
4. Il faut maintenant choisir k, d, x_0 convenablement. Calculer pour quelques choix de ces paramètres, la fonction ψ_0 puis afficher cette condition initiale ensemble avec la barrière comme en figure 1.4. Il faut aussi calculer H_0 et U_0 et afficher ces valeurs à l'écran. Créer un message pour vous avertir si $E_0 > V$. Si ceci est le cas, retourner au pas précédent, augmenter V la barrière de potentiel, puis recalculer les fonctions propres.
5. Si vous avez réussi à créer un paquet d'onde, suffisamment localisé, suffisamment placé à gauche de la barrière et d'énergie initiale $H_0 < V$, vous pouvez passer à l'étape suivante, la projection. Projeter la condition initiale sur les vecteurs propres avec votre fonction `projection` afin d'obtenir les coefficients d'expansion c_n. Normalement, cette fonction doit générer un graphe qui montre le paquet d'onde ψ_0 et la reconstruction approchée $\psi_{0,approx}$. Vérifier que l'approximation est bonne que $\psi_0(x) \approx \psi_{0,approx}(x)$. Si cela n'est pas le cas, il faut revenir en arrière et augmenter la résolution M ou diminuer k.
6. Une fois la projection validée, vous pouvez faire évaluer votre paquet d'onde dans le temps. Utiliser votre fonction `psi_eval` pour évaluer le paquet d'onde pour un temps total T en $N = 100$ pas de temps. Ce temps final T a été précédemment calculé et dépend du paquet d'onde initial choisi.
7. Générer une animation qui montre l'évolution temporelle du paquet d'onde. En arrière plan, nous voulons voir le potentiel $V(x)$ en noir. Refaire les calculs avec un maillage plus fin afin d'améliorer cette animation si elle vous semble mal résolue. Normalement vous devez voir un paquet d'onde qui avance vers la barrière de potentiel. Selon la hauteur et la largeur de la barrière, V et a, une partie du paquet arrivera à percer la barrière. Ceci est l'effet tunnel.
8. Afin d'étudier l'effet tunnel de manière quantitative, on a veut calculer la probabilité de présence de la particule à gauche ou à droite de la barrière et comment cela varie au cours du temps. Trouver une méthode pour calculer ces probabilités à chaque instant. Faire ensuite une courbe qui montre comment ces probabilités varient au cours du temps. Est-ce qu'on attend un état asymptotique à temps long où les probabilités de présence ne varient plus ?
9. Etudier comment la largeur de la barrière a et son hauteur V influencent l'effet tunnel.

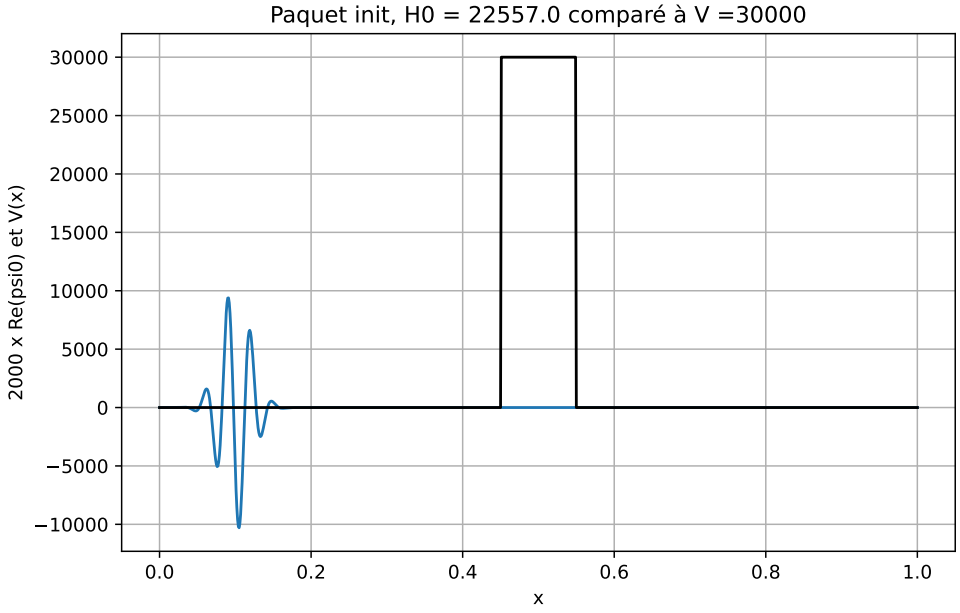


FIGURE 1.4 – Paquet d'onde initial (partie réelle) devant la barrière de potentiel.

2.4 Projet 4 : bandes d'énergie dans les solides

Dans les solides, les noyaux sont rangés dans un ordre cristallin et cela introduit une périodicité dans le potentiel $V(x)$. Cette périodicité mène à l'existence de bandes dans les spectres des énergies et c'est ça qui nous permet de comprendre pourquoi certains matériaux sont des bons conducteurs et d'autres des isolants. Dans ce projet, le but est de mettre en évidence ces bandes d'énergie à l'aide de spectrogrammes. Le potentiel sera L-périodique : $V(x) = V(x+L)$ pour tout x et le domaine n'a plus vraiment de bords. Il faudra utiliser des conditions de périodicité et créer une version modifié du programme `etats_propres`. On utilisera toujours la longueur de périodicité $L_* = L$ comme échelle naturelle et E_* et T_* en découlent. La longueur de périodicité ans le problème adimensionné est donc toujours 1 dans la suite.

1. Faire une version périodique du programme qui calcule les énergies propres au nom `etats_propres_perio`. La forme du potentiel $V(x)$ périodique ainsi que le maillage (réduit) x du calcul doivent pouvoir être donnés par l'utilisateur.
2. Construire un maillage uniforme réduit de M points qui couvre l'intervalle $x \in [0, 1[$. On rappelle que dans le cas périodique, on doit exclure le dernier point de ce maillage (car il est identique au premier).
3. Tester votre programme avec un potentiel $V(x) = 0$ pour une particule libre. Comment doivent être les fonctions et énergies propres (faire le calcul analytique) si on leur impose la même périodicité 1 ?
4. Nous étudions ensuite un potentiel périodique en créneau

$$V(x) = \begin{cases} V & , \quad 0 < (x\%1) < a \\ 0 & , \quad a < (x\%1) < 1 \end{cases} \tag{1.28}$$

   Ici l'opération $(x\%1)$ signifie module après division et c'est exactement de cette manière qu'on le code en Python. Trouver les fonctions propres pour le choix $V = 30$ et $a = 0.8$, $M = 200$. Créer un graphe comme en figure (1.1), limitant l'axe y à l'intervalle de 0 à 70 et montrant les 3 premiers niveaux d'énergie. Adapter bien C (cf. section sur la visualisation) pour bien voir les fonctions propres.
5. En réalité, les fonctions propres ne doivent pas forcément avoir la même périodicité que le potentiel. Si on admet un domaine plus grand alors on verra apparaître d'autres fonctions propres et on témoignera de l'apparition de "bandes d'énergie". Nous voulons mettre en évidence ceci de manière numérique. Pour $p = 1, 2, 4, 8, 16, 32$ répéter la procédure suivante
   (a) Créer un maillage réduit de $p \times M$ points couvrant l'intervalle $x \in [0, p[$. Cea garantit l'utilisation du même pas δx dans le calcul numérique des valeurs propres.
   (b) Trouver les fonctions et énergies propres et créer des figures du genre de (1.1). Nous voulons toujours voir y sur l'intervalle de 0 à 70 et les $3 \times p$ première valeurs propres

   Une fois toutes les figures faites, changer la taille des fenêtres manuellement afin qu'ils aient toutes la même hauteur et que les axes x sont à la même échelle dans les figures (la figure pour $p = 32$, sera idéalement 32 fois aussi large que celle pour $p = 1$ et deux fois aussi large que celle pour $p = 16$). Sauvegarder ces figures. En comparant ces figures entre elles, on peut effectivement constater que les niveaux d'énergie se regroupent autour de "bandes". Un exemple de figure pour $p = 32$ est donnée en figure 1.5, gauche.

L'utilisation d'un très grand domaine de calcul pour capturer des fonctions d'ondes qui n'ont pas la périodicité du réseau n'est pas très pratique. Cela nécessite de plus en plus de points de maillage et rend le calcul numérique lourd. En faite, cela n'est pas non plus nécessaire. Grâce au théorème de Bloch, on sait que la fonction d'onde pour un potentiel périodique de périodicité L est de la forme

$$\phi(x) = e^{iqx}\varphi(x) \tag{1.29}$$

Ici $\varphi(x) = \varphi(x + L)$ est une fonction L-périodique. Si $q = 0$ on retrouve tout simplement les solutions qui ont la même périodicité que le réseau, on parle des solutions harmoniques. Si $q \neq 0$, le préfacteur e^{iqx} donne une modulation à l'onde sur une longueur $2\pi/q$. Cette solution n'est pas forcément périodique et on parle d'une solution sous-harmonique. Sans perte de généralité, nous pouvons limiter le nombre $q \in [-\pi/L, \pi/L]$ (appelé première zone de Brillouin, plus d'info en salle) pour couvrir l'ensemble des possibilités. En réalité, tout cristal ou réseau périodique a une taille fini. Le nombre q ne peut pas alors pas prendre tous les valeurs dans ce continuum, mais cela reste néanmoins une très bonne approximation pour les solides (la périodicité du réseau est la distance inter-atomique, mais un cristal peut être macroscopique et donc très très grand devant cette longueur). Si on propose la solution de Bloch à notre problème au valeurs propres, on trouve

$$E\varphi(x) = -\frac{1}{2}\frac{d^2\varphi(x)}{dx^2} - iq\frac{d\varphi(x)}{dx} - q^2\varphi(x) + V(x)\,\varphi(x) \tag{1.30}$$

pour la fonction φ. On peut maintenant chercher les solutions propres de ce problème au valeurs propres (Hermitien) et cela nous donnera des niveaux d'énergies discrets E_n comme avant, indexable par un entier n. Par contre, chaque valeur $q \in [-\pi/L, \pi/L]$ nous donnera une autre valeur discrète E_n et on a donc des branches d'énergies accessibles qui varient continûment avec q. On note

$$E_n(q) \ : \text{ la variation du n-ième niveau d'énergie avec } q \tag{1.31}$$

En variant $q \in [-\pi/L, \pi/L]$, on obtient les bandes d'énergie $E_0(q), E_1(q), E_2(q), ...$ admissibles.

6. Réaliser un nouveau programme `etats_propres_bloch` permettant d'identifier les états et énergies propres $E_n(q)$ de cette nouvelle équation pour $\varphi(x)$. Ce programme acceptera un nouvel argument q.

Dans notre adimensionnement, on a $L = 1$ comme longueur de périodicité donc $q \in [-\pi, \pi]$. Dans ce continuum de valeurs de q, il y a quelques valeurs très spécifiques qui donnent des fonctions d'onde périodiques sur un domaine de taille pL avec p entier. On a déjà calculé ces ondes propres dans nos calculs précédents, sur des domaines de taille pL et afin de faire le lien avec les solutions de Bloch, on essaie donc d'identifier à quels choix de q cela doit correspondre. Pour identifier ces valeurs de q particulières, il suffit d'exiger la périodicité pL dans la solution de Bloch :

$$\phi(x) = \phi(x + pL) \quad \Leftrightarrow \quad e^{iqx}\varphi(x) = e^{iq(x+pL)}\varphi(x + pL) \quad \Leftrightarrow \quad e^{iqx}\varphi(x) = e^{iqx}\varphi(x)e^{iqpL} \quad \Leftrightarrow \quad 1 = e^{iqpL} \tag{1.32}$$

On déduit de cette relation que la fonction $\varphi(x)$ a exactement une période dans la distance pL si $qpL = 2\pi$ soit $q = 2\pi/pL$. Ainsi

- $q = \pm 2\pi/2L$, fonction d'onde $\phi(x)$ de période $2L$
- $q = \pm 2\pi/3L$, fonction d'onde $\phi(x)$ de période $3L$
- $q = \pm 2\pi/5L$, fonction d'onde $\phi(x)$ de période $5L$
- ...

Cette info nous permet de réaliser quelques tests.

7. Construire un maillage uniforme réduit de M points qui couvre l'intervalle $x \in [0, 1[$. Lancer le nouveau programme qui utilise la solution de Bloch avec $q = 0$ et identifier les 4 premières énergies ($n = 0, 1, 2, 3$) propres $E_n(q = 0)$). Lancer ce programme une deuxième fois pour $q = \pi$ et identifier de nouveau les 4 premières énergies propres $E_n(q = \pi)$). Imprimer toutes ces valeurs à l'écran.
8. Construire un maillage uniforme réduit de $2M$ points qui couvre l'intervalle $x \in [0, 2[$. Lancer votre programme périodique précédent (pas le programme bloch) et imprimer les 8 premières valeurs propres. Si votre code Bloch est correct, vous devez récupérer les mêmes énergies propres aux erreurs d'imprécision près.
9. On veut maintenant mesurer les branches d'énergie admissibles $E_0(q), E_1(q), E_2(q), ...$ pour notre potentiel et les afficher en fonction de q. Ceci donnera les célèbres schémas de bandes, très utilisés en physique du solide. On définit un maillage uniforme $N_q + 1$ valeurs de q qui couvre l'intervalle $[-\pi, \pi]$:

$$[q_0, q_1, \ldots, q_{N_q}] = [-\pi, -\pi + \delta q, -\pi + 2\delta q, \ldots, -\pi + (N_q)\delta q = \pi] \tag{1.33}$$

soit

$$q_k = -\pi + k \underbrace{\frac{2\pi}{N_q}}_{\delta q} \quad \text{avec} \quad k = 0, 1, \ldots, N_q \tag{1.34}$$

Ensuite, pour tous ces valeurs q_k, calculer les 3 énergies les plus basses admissibles selon votre programme Bloch. Mettre ces mesures dans 3 vecteurs

$$\begin{aligned} \mathtt{E_0} &= [E_0(q_0)\ E_0(q_1)\ E_0(q_2)\ \ldots\ E_0(q_{N_q})] \\ \mathtt{E_1} &= [E_1(q_0)\ E_1(q_1)\ E_1(q_2)\ \ldots\ E_1(q_{N_q})] \\ \mathtt{E_2} &= [E_2(q_0)\ E_1(q_1)\ E_2(q_2)\ \ldots\ E_2(q_{N_q})] \end{aligned} \tag{1.35}$$

Utiliser ces vecteurs pour réaliser les schémas en bandes qui montrent $E_0(q), E_1(q), E_2(q)$ en fonction de $q \in [-\pi, \pi]$.

11. Afin de mieux voir les bandes d'énergie, on conseille de faire un "spectrogramme" à bandes. On veut voir des rectangles colorés sur les zones d'énergie admissibles. Il suffit ici calculer le min et le max des $E_0(q), E_1(q), E_2(q)$. Sur les spectrogrammes on veut voir ces valeurs min et max, comme dans la figure 1.5, droite. Si on compare ces diagrammes à ceux produits à la question 5, on constate une ressemblance évidente.

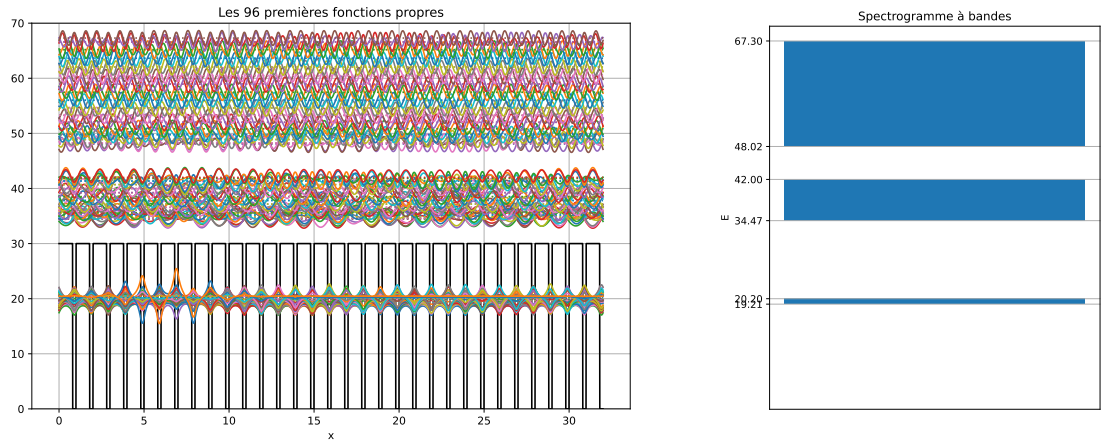
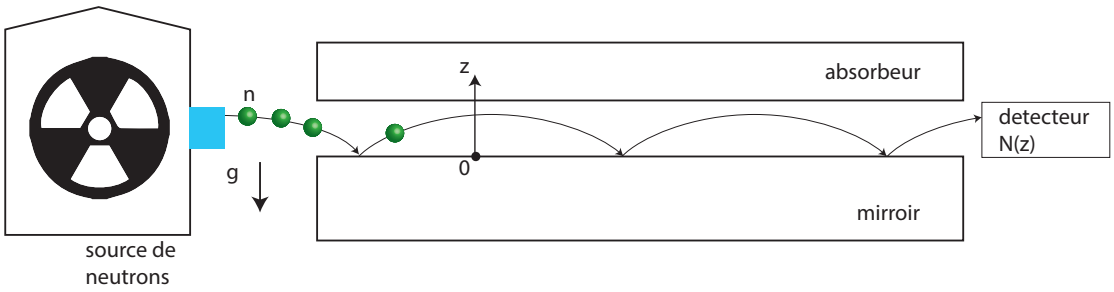


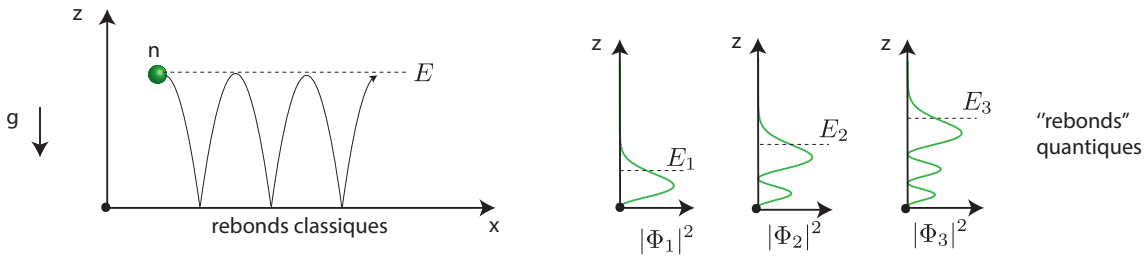
FIGURE 1.5 – Bandes d'énergie avec un potentiel périodique. (gauche) Fonctions propres et énergies discrètes dans un domaine de taille 32. (droite) Spectrogramme à bandes pour un potentiel périodique.

2.5 Projet 5 : rebonds quantiques de neutrons dans le champ de gravité terrestre

En 2002, à l'Institut Laue-Langevin (ILL) de Grenoble, l'équipe de V.V. Nesvizhevsky a réalisé une expérience unique, schématisée ci-dessous.



Des neutrons sortent d'un réacteur nucléaire et entrent une fine fente, qui sépare un miroir à neutron en bas à $z = 0$, d'un absorbeur à neutrons en haut. Dans ce dispostif les neutrons sont ultrafroids et donc lents. Ils ont alors le temps de tomber sous l'effet de la gravité et peuvent donc rebondir sur le miroir avant d'atteindre le détecteur à neutrons, placé à la sortie. Celui-ci compte le nombre de neutrons $N(z)$ en fonction de l'hauteur z et mesure donc la distribution de probabilité $P(z)$, de trouver un neutron à la hauteur z à la sortie.



Si le neutron serait une particule classique de masse m, il ferait des rebonds, comme montré ci-dessus, pouvant atteindre un continuum de hauteurs et donc un continuum d'énergie mécaniques $E = m\dot{z}^2/2 + mgz$ associée aux mouvements verticaux. Mais le neutron n'est pas une particule classique, c'est une particule quantique décrite par

une fonction d'onde. Cette fonction d'onde satisfait l'équation de Schrödinger :

$$i\hbar\frac{\partial\psi}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2\psi}{\partial z^2} + V(z)\psi \tag{1.36}$$

Ici $m = 1,67 \times 10^{-27} kg$ est la masse du neutron. L'énergie potentielle $V(z)$ du système est ici celui du champ gravitationnel :

$$V(z) = \begin{cases} mgz & , \quad z > 0 \\ +\infty & , \quad z < 0 \end{cases} \tag{1.37}$$

avec $g = 9.81 m/s^2$. Le miroir en $z = 0-$, correspond à une zone où l'énergie potentielle du neutron est très grande. En pratique ceci veut dire que le neutron ne pourra jamais atteindre la région $z \leq 0$. Pour la même raison, on peut dire que le neutron ne pourra jamais atteindre la région $z \to +\infty$ car le potentiel y est trop élevé. Ceci nous donne les conditions aux limites

$$CL \ : \ \psi(0,t) = 0 \quad , \quad \psi(+\infty,t) = 0 \tag{1.38}$$

qu'on approchera par

$$CL \ : \ \psi(0,t) = 0 \quad , \quad \psi(D,t) = 0 \tag{1.39}$$

dans le code. Ici D est une distance suffisamment grande.

1. Montrer qu'il existe une longueur naturelle

   $$L* = m^\alpha g^\beta \hbar^\gamma \tag{1.40}$$

   et identifier donc les exposants $\alpha, \beta, \gamma \in \mathbb{Q}$ qui permettent de réécrire l'équation sous la forme adimensionnée déjà mentionnée. Donner la valeur numérique de L_* ainsi que les échelles d'énergie E_* et T_* associées.
2. Pour calculer les états propres, on devrait normalement le faire sur l'intervalle $z \in [0, +\infty[$. Sous quelles conditions peut-on calculer les solutions sur un intervalle fini $z \in [0, D]$? Dans la suite on choisira $D > 10$.
3. Calculer les énergies propres "adimensionnées" des trois premiers états quantiques $\tilde{E}_n$, $n = 0, 1, 2$, utilisant $N + 1 = 201$ points. Exprimer ces énergies en forme dimensionné, utilisant l'échelle d'énergie E_*. Vérifier que vous trouvez bien

   $$E_0 \simeq 1.41\, peV \quad , \quad E_1 \simeq 2.46\, peV \quad , \quad E_2 \simeq 3.32\, peV \tag{1.41}$$

   On rappelle également que $1 peV = 10^{-12} eV = 1.602 \times 10^{-31} J$.
4. Faire trois diagrammes qui montrent z en fonction de $|\phi_n^2|$ (l'axe z est vertical dans les graphes), pour les trois premiers premiers niveaux. Ceci vous montre la densité de probabilité de trouver le neutron à cette hauteur.
5. On définit l'opérateur quantité de mouvement adimensionnée comme $\hat{p} = -i\dfrac{\partial}{\partial z}$. Le but est de traduire l'opération

   $$\hat{p}\psi \longrightarrow \mathtt{P}\psi \tag{1.42}$$

   Ici le ψ de droite correspond au vecteur de valeur nodales et `P` est une matrice carrée. Il suffit d'identifier les bonnes formules d'approximation (différences finies centrées sur les points intérieurs, décentrées sur les points de bord) pour calculer les dérivées en tous les points, puis de récrire ces formules sous forme matricielle. Ceci permet d'identifier le contenu de matrice `P` et ensuite de le programmer. Tester votre matrice `P` sur la fonction $f(z) = e^{ikz}$. L'application $\hat{p}e^{ikz} = ke^{ikz}$. Faire un graphe qui compare donc ke^{ikz} à `Pf` sur des valeurs nodales.
6. Utiliser la matrice `P` pour calculer la valeur moyenne de la vitesse moyenne $\langle\phi_n|\hat{p}|\phi_n\rangle$ de chute du neutron, dans le n-ième état. Pourquoi trouve-t-on ce résultat ?
7. Calculer également la hauteur moyenne $\langle\phi_n|z|\phi_n\rangle$ et diviser cette hauteur moyenne par E_n. Qu'est-ce que vous constatez ?
8. On imagine une masse ponctuelle m classique lancée du sol $z = 0$ à l'instant $t = 0$ et avec une énergie cinétique initiale égale à E. Calculer l'équation horaire $z(t)$ de la masse en trouvant la solution de l'équation de Newton (en fonction de E, m, g). Calculer le temps de retour au sol, le temps d'impact T. Calculer ensuite la hauteur moyenne classique de la particule comme

   $$\langle z\rangle_{classique} = \frac{1}{T}\int_0^T z(t)\,\mathrm{d}t \tag{1.43}$$

   et exprimer cette hauteur moyenne en unités E/mg. Comment est-ce que cette hauteur moyenne se compare-t-ellle à la prédiction quantique de la question précédente ?

9. Calculer la norme de la vitesse au carrée $\langle \phi_n | \hat{p}^2 | \phi_n \rangle$ ainsi que $\langle \phi_n | z^2 | \phi_n \rangle$. Vérifier l'inégalité de Heisenberg (adimensionnée) sur quelques états $n = 0, 1, 2$. Comment varie l'incertitude avec n.

10. On imagine un paquet d'onde initial Gaussien ψ_0 placé à une certaine hauteur au dessus de $z = z_0$ et avec une certaine largeur d, genre

$$\psi_0 \approx C e^{ikz} \exp\left(-(z - z_0)^2 / 2d^2\right) \tag{1.44}$$

Ici quelques explications sont nécessaires.

- $C \in \mathbb{R}$ est un coefficient de normalisation qui garantit $\langle \psi_0 | \psi_0 \rangle = 1$. Calculer C analytiquement en supposant que le paquet reste loin des bords (que le domaine d'intégration est $]-\infty, +\infty[$).

- Le paramètre z_0 définit le centre de la Gaussienne et donc une "hauteur initiale". Il convient donc de le choisir positif pour placer le paquet au dessus du miroir. Ne placer pas la particule trop haute.

- Le coefficient d définit la largeur de la Gaussienne et devra être suffisamment petit pour localiser le paquet d'onde.

- Le nombre $k = 2\pi/\lambda$ est un nombre d'onde. Idéalement, on veut une longueur d'onde suffisamment courte afin d'avoir un paquet d'onde qui ressemble à un paquet d'onde, mais attention à la résolution finie.

Le paquet d'onde initial est donc défini par 3 paramètres k, x_0, d. Il est utile de faire un graphe qui montre le paquet (axe vertical = z) pour vérifier qu'on fait un bon choix. Le paquet doit être suffisamment loin des bords et suffisamment localisé. On conseille également de calculer la vitesse moyenne initiale (analytiquement ou numériquement)

$$U_0 = \langle \psi_0 | \hat{p} | \psi_0 \rangle \tag{1.45}$$

car cela permet d'estimer la hauteur maximale que le paquet atteindra. Cela détermine la taille du domaine, c.a.d. D, sur lequel il faudra résoudre le problème. Proposer une formule pour

$$H_{max} = \dots \tag{1.46}$$

Il faudra toujours réaliser les calculs sur un domaine $D > H_{max}$ car sinon, on va avoir des réflexions non-physiques sur la "paroi" $z = D$. Cette vitesse initiale, nous permet aussi d'estimer le temps de retour au sol du paquet d'onde. Utiliser la formule classique trouvée pour T pour estimer ce temps de retour au sol (on suppose $E = mgH_{max}$, puis on adimensionne)

$$T = \dots \tag{1.47}$$

Cette information est utile pour définir la durée de l'animation.

Une fois une bonne condition initiale et une hauteur D du domaine identifiée, calculer les fonctions propres avec une bonne résolution. Projeter ensuite la condition initiale sur votre base d'ondes propres afin d'obtenir les coefficients d'expansion c_n. Calculer ensuite l'évolution du paquet d'onde à $N = 100$ instants entre $t = 0$ et $t = T$. Utiliser ces données pour faire une animation qui montre l'évolution de la distribution de probabilité au cours du temps. Décrire l'animation à temps court et temps long.

Chapitre 2

Simulations instationnaires

On s'intéresse à la mécanique quantique ondulatoire 1D, d'une particule placée dans un potentiel $V(x)$ stationnaire ou instationnaire $V(x,t)$. Dans ces projets, l'objectif commun sera de :

- réaliser une simulation d'un paquet d'onde dans un potentiel arbitraire
- faire une animation qui montre **l'évolution d'un paquet d'onde initial** dans un potentiel donné

Les codes instationnaires sont peut être un poil plus difficile à faire, mais comparé aux projets du chapitre précédent, il y a bien moins de fonctions à programmer.

1 Première partie du projet : tronc commun

1.1 Programme principal : fonction `schrodinger`

Nous voulons une fonction `schrodinger` capable d'avancer en temps un paquet d'onde qui satisfait l'équation de Schrödinger adimensionnée

$$i\frac{\partial \psi}{\partial t}(x,t) = -\frac{1}{2}\nabla^2\psi(x,t) + V(x,t)\,\psi(x,t) \tag{2.1}$$

sur un domaine fini $x \in [a,b]$ et pour tous les temps $t \in [0,T]$. On disposera d'une condition initiale

$$CI \;:\; \psi(x,0) = \psi_0(x) \tag{2.2}$$

et sur les bords du domaine, on impose des conditions de Dirichlet homogènes

$$CL \;:\; \psi(a,t) = 0 \quad , \quad \psi(b,t) = 0 \tag{2.3}$$

Ceci imite l'effet de barrières de potentiel infiniment hautes ($V(x<a) \to +\infty$ et $V(x>b) \to +\infty$) qui repoussent les particules. Le potentiel $V(x,t)$ ainsi que la condition initiale $\psi_0(x)$ doivent pouvoir être définis par l'utilisateur de la fonction. On veut pouvoir utiliser ce même programme dans des contextes physiques différents. On veut également visualiser la solution par une animation.

On utilisera des **différences finies centrées et un maillage uniforme**. Le schéma temporel est au choix, mais vous devez au minimum tester **un schéma explicite** adapté à tous les potentiels $V(x,t)$. Vous devez aussi programmer **un schéma implicite** pour un potentiel $V(x)$ stationnaire arbitraire. Ne vous limiter pas au schéma d'Euler explicite. L'utilisation de schémas implicites ou semi-implicites est très conseillée car cela permet d'utiliser des pas de temps plus grands et de diminuer le temps de calcul total.

Préparer tout sur une ou plusieurs feuilles de papier, suivant les pas indiqués dans les tutoriels. Démarrer la programmation en partant des programmes "instationnaires" du tutoriel.

Votre fonction doit obligatoirement accepter les arguments suivants, les extrémités `a`,`b` de l'intervalle, le nombre de pas d'espace `M`, le nombre de pas de temps `N` ainsi que le temps final `T`. La fonction doit aussi recevoir le nom de la fonction Python qui définit le potentiel V et celle qui définit la condition initiale $\psi_0(x)$. On imagine ici que c'est l'utilisateur qui définira ces fonctions. La fonction doit sortir le maillage x, les instants discrets t sur lequel on exporte

la solution et une matrice `psi` contenant sur ces colonnes le vecteur d'état aux instants t.

Attention : dans votre code, initialiser tous les vecteurs et matrices qui contiennent des fonctions d'ondes comme des vecteurs et matrices complexes. Sinon Python va uniquement considérer la partie réelle dans les calculs ce qui est évidemment faux.

Validation : puit de potentiel infiniment profond

Une fois le code écrit, on le valide sur un cas simple, le puit de potentiel infiniment profond. Sous forme dimensionnée, le puit de potentiel infiniment profond est défini par

$$V(x) = \begin{cases} 0 & , \quad x \in [0, L] \\ +\infty & , \quad \text{ailleurs} \end{cases} \tag{2.4}$$

La seule grandeur naturelle disponible est L, la largeur du puit et on fixe donc l'échelle naturelle $L_* = L$. En découlent que $E_* = \hbar^2/mL_*^2$ et $T_* = mL_*^2/\hbar$. Le domaine de calcul est donc $x \in [0, 1]$. Nous voulons la solution pour $t \in [0, T]$. Comme condition initiale, on choisit le paquet d'onde parabolique

$$CI \ : \ \psi(x, 0) = \sqrt{30}\, x(1 - x) \tag{2.5}$$

Ce paquet d'onde est bien normalisé comme il le faut, $\langle \psi_0 | \psi_0 \rangle = 1$ et la fonction d'onde s'annule sur les bords $x = 0, 1$. Pour valider le code **explicite** :

1. Créer une fonction `V_puit_infini` qui à la réception d'un maillage x, renvoie un vecteur de la même taille que x, contenant des zéros.
2. Créer une fonction `psi0` qui à la réception d'un maillage x, renvoie la fonction $\sqrt{30}x(1 - x)$ qui définit la condition initiale sur les points du maillage.
3. Lancer la simulation avec $M = 100$, $a = 0$ et $b = 1$. Le nombre de pas de temps dépend fortement de votre schéma. Quelques indications : pour un schéma explicite de bonne qualité, on trouve que $T = 1$ peut être atteint en $N = 10000$ pas de temps, sur un maillage avec $M = 100$. Avec un schéma implicite de bonne qualité, on peut faire le même calcul en $N = 100$ pas de temps.
4. Réaliser une animation du paquet d'onde, comme vous l'avez appris dans le tutoriel 3 sur les problèmes instationnaires. Il suffira de parcourir les colonnes de la matrice `psi`. L'animation ressemble à une corde de guitare tenue par les bords et qui oscille entre les positions haute et basse. Il faut fixer l'étendu de l'axe y. Vous pouvez comparer votre animation à celle produite par vos collègues qui sont sur les projets des états propres.

Ces paramètres fonctionnent pour notre code qui utilise le schéma de Runge-Kutta 4. La validation du code **implicite** passe par exactement les mêmes pas. Dans nos tests, nous avons programmé le schéma de Crank-Nicolson et on observe qu'on peut augmenter le pas de temps drastiquement, sans perdre la stabilité numérique. On trouve que le même calcul instationnaire avec $M = 100$ et sur une durée $T = 1$, se fait en seulement $N = 100$ pas de temps implicites de taille $dt = 0.01$. Le code implicite est donc bien plus rapide et stable.

2 Deuxième partie du projet : choisir 1 sujet parmi 2

Après avoir réalisé ce premier volet du projet, chaque groupe continuera le projet sur l'un des 2 sujets qui sont proposés ici. L'idée est que les différents groupes travaillent sur des sujets différents.

2.1 Projet 6 : un programme performant et implicite, qui utilise des matrices creuses

Les matrices de gauches $\mathbf{O}_g$ et de droites $\mathbf{O}_d$ qui définissent les équations discrets d'une formulation implicite ($\mathbf{O}_g \psi^{n+1} = \mathbf{O}_d \psi^n$) sont très creuses, c'est à dire plein de zéros. Ces zéros n'influencent pas vraiment le calcul mais on les charge néanmoins en mémoire. Puis, on effectue de nombreuses multiplications par zéro ce qui ne fait que ralentir le calcul numérique des vecteurs propres. L'utilisation de matrices creuses (librairie `scipy.sparse`) permet une économie en mémoire et accélère fortement les calculs. C'est aussi le seul moyen pour réaliser des calculs sur des maillages fins avec de très nombreux points. Le but de ce projet est de faire une nouvelle fonction `schrodinger_creux` plus

performant qui utilise des matrices creuses.

Le programme implicite déjà créé dans la première partie du projet sera votre point de départ. Ce projet s'adresse à ceux/celles qui aiment bien programmer en Python.

Pour mener ce projet à terme, procéder par les étapes suivantes.

1. Nous devons apprendre à programmer des matrices creuses. L'idée est de ne plus mettre en mémoire les très nombreux zéros de la matrice. A la place, on mettra en mémoire uniquement les éléments non-nulles (valeurs) et leur coordonnées (indices de colonne et indices de ligne). Donnons un exemple comment procéder en Python. On imagine 3 vecteurs,

$$\texttt{row} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 4 \end{bmatrix} \quad , \quad \texttt{col} = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 4 \end{bmatrix} \quad , \quad \texttt{val} = \begin{bmatrix} -0.1 \\ 3.2 \\ -1.1 \\ 0.9 \end{bmatrix} \tag{2.6}$$

   et on imagine que ceux-ci contiennent les coordonnées (indices de lignes `row`, indices de lignes `col`) et les valeurs (`val`). Pour être encore plus clair, on veut ici une matrice avec

   - à la ligne 0 et la colonne 1, la valeur -0.1,
   - à la ligne 1 et la colonne 2, la valeur 3.2
   - à la ligne 2 et la colonne 0, la valeur -1.1
   - à la ligne 4 et la colonne 4, la valeur 0.9

   Ensuite, on doit encore préciser la taille totale de la matrice. Dans notre exemple, on veut une matrice 5×6. En Python, on crée tout d'abord les vecteurs `row`, `col`, `val`. La longueur de ces vecteurs correspond exactement au nombre d'éléments non-nuls dans la matrice creuse. Toute la difficulté de la programmation des matrices est déplacé dans la programmation de ces 3 vecteurs. Une fois ces vecteurs créés, on utilise la fonction `coo_matrix` de la librairie `scipy.sparse`). L'instruction `A=coo_matrix((val,(row,col)),shape=(5,6))` crée la matrice

$$\texttt{A} = \begin{bmatrix} 0 & -0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3.2 & 0 & 0 & 0 \\ -1.1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.9 & 0 \end{bmatrix} \tag{2.7}$$

   dans un format creux, c'est à dire, sans mettre en mémoire les très nombreux zéros. Il existe plein d'autres formats de matrices creuses, mais ce format dite de coordonnées est un des plus simples selon nous. Créer en tant qu'exercice une matrice diagonale creuse de taille $M+1 \times M+1$ qui contient sous la diagonale les éléments M à 1. Exemple pour $M = 9999$:

$$\texttt{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 9999 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9998 & 0 & 0 & 0 & 0 \\ 0 & 0 & 9997 & 0 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.8}$$

   Il faudra donc trouver un moyen de créer les vecteurs `row`, `col`, `val` qui permettent de localiser et de définir les entrées non-nulles.

2. Estimer le nombre d'octets nécessaires pour mettre en mémoire la matrice de l'exercice précédente en format plein (avec les zéros) et en format creux (sans les zéros mais avec les coordonnées). Un nombre réel = 16 octets en précision double par ailleurs.

3. Un schéma implicite (à deux niveaux) génère une formule de recurrence de la forme

$$\mathbf{O}_g \psi^{n+1} = \mathbf{O}_d \psi^n \tag{2.9}$$

   Ici $\mathbf{O}_g$ et $\mathbf{O}_d$ sont les matrices de gauche et de droite, ψ^n l'état connu à temps t_n et ψ^{n+1} le nouvel état à calculer à temps t_{n+1}. Ecrire/rappeler la structure de ces matrices sur une feuille de papier, puis, tenter de les programmer. En pratique, il faut pour les deux matrices répéter la procédure suivante :

(a) Calculer combien d'éléments non-nuls se trouvent dans ces matrices. Cela définit la longueur des vecteurs `row`, `col` et `val`. Initialiser donc ces trois vecteurs pour qu'ils aient cette longueur. Attention, il faut initialiser le vecteur `val` comme un vecteur complexe.

(b) Trouver une méthode pour programmer le remplissage de ces tableaux. Le plus simple est d'utiliser une boucle qui parcourt tous les points de maillage et de traiter la matrice ligne par ligne. Selon la ligne (le point), il faudra ajouter $3, 1$ aux vecteurs `row`, `col` et `val`.

(c) Créer la matrice creuse avec la méthode `coo_matrix` de la librairie `scipy.sparse`.

4. Pour tester la bonne programmation de ces matrices, le plus simple est de les comparer aux matrices pleines, déjà disponible dans le code précédent. Il suffit pour cela de travailler à basse résolution ($M < 4000$) et d'utiliser la méthode `toarray` applicables à toutes les matrices creuses.

   `Og_plein` = Og.toarray()

5. Pour pouvoir calculer avec les matrices creuses, on a besoin de convertir les matrices creuses dans un format comprimé. Toute matrice `M` creuse définie par l'instruction `coo_matrix` devra être transformée en format "csr" avec l'instruction suivante

   ```
   M = M.tocsr()
   ```

6. Dans la librairie `scipy.sparse.linalg` il existe deux fonctions qui vont être utiles pour résoudre le système linéaire $\mathbf{O}_g \psi^{n+1} = \mathbf{O}_d \psi^n$:

   - `spsolve` : méthode qui permet de résoudre le système linéaire avec une matrice creuse. Il faut tout d'abord importer cette fonction avec

     ```
     from scipy.sparse.linalg import spsolve
     ```

     Ensuite il est très facile de l'utiliser. Pour le schéma donné et dans la boucle d'avancement temporel, on imagine une ligne de mise à jour d'un vecteur `f` comme

     ```
     f=spsolve(Og,Od.dot(f))
     ```

   - `splu` : factorisation LU (lower upper) de la matrice de gauche. Cette factorisation sera très utile si vous avez à résoudre de nombreuses fois à résoudre un système linéaire $\mathbf{O}_g \psi^{n+1} = \mathbf{O}_d \psi^n$ avec la même matrice creuse de gauche $\mathbf{O}_g$. Ceci permet d'accélèrer encore plus le code. Il faut tout d'abord importer cette fonction avec

     ```
     from scipy.sparse.linalg import splu
     ```

     Juste après la création de la matrice, avant de rentrer dans la boucle d'avancement temporel, on réalise la factorisation LU

     ```
     lu=splu(Og)
     ```

     Ceci crée un objet `lu` avec une méthode de résolution de système linéaire associée. Dans la boucle d'avancement temporel, la ligne de mise à jour d'un vecteur `f` sera

     ```
     f=lu.solve(Od.dot(f))
     ```

   On vous conseille de vous renseigner un minimum sur ces fonctions et comment les utiliser en étudiant les documentations en ligne.

7. Créer un nouveau programme qui utilise cette représentation de matrice creuse et ces méthodes de résolution. Commencer avec un code qui utilise `spsolve`. Dans un deuxième temps vous pouvez l'adapter très facilement

pour utiliser la méthode `splu`.

Attention : il ne peut y avoir aucune ligne du style `A=np.zeros(M+1,M+1)` dans le code car cela crée une matrice pleine et occupe donc de la mémoire pour rien. Il faut, à la place construire les tableaux `row`, `col`, `val` adéquats.

8. Valider votre programme version creuse en le comparant à la version pleine et pour le même maillage uniforme de $M = 100$ pas d'espace et $N = 100$ pas de temps, sur un temps $T = 1$. Utiliser le potentiel du puit infiniment profond.
9. Augmenter maintenant la résolution spatiale et temporelle. Dans nos tests le calcul passe parfaitement en 30s de calcul avec $M = 10000$ pas d'espace et $N = 10000$ pas de temps, sur un temps $T = 1$. Si vous utiliser la méthode `splu`, le temps de calcul est encore diminué.
10. Si vous avez réalisé ce code performant, autant en profiter et s'amuser avec. Réaliser une simulation instationnaire de neutrons qui tombent das le champ de gravité terrestre (cf. projet 5 du chapitre précédent). Votre prof vous définira le domaine, le potentiel et la condition initiale à utiliser. Vous pouvez aussi collaborer avec le groupe qui a choisi ce sujet.

2.2 Projet 7 : simulation instationnaire de l'effet tunnel

L'effet tunnel est l'effet quantique par lequel une particule de masse m avec énergie E_0 initiale arrive à franchir une barrière de potentiel de hauteur V localement plus grande que E_0. Selon les lois de la mécanique classique cela n'est pas possible. Dans l'exercice, nous plaçons notre fine barrière de potentiel au centre d'un puit de potentiel infiniment profond avec une largeur L. On décide d'utiliser cette largeur du puit infini, $L_=L*$ comme échelle spatiale naturelle. Le domaine adimensionné de calcul est alors $x \in [0, 1]$ et on définit le potentiel adimensionné comme

$$V(x) = \begin{cases} 0 & , \quad 0 < x < \dfrac{1-a}{2} \\ V & , \quad \dfrac{1-a}{2} < x < \dfrac{1+a}{2} \\ 0 & , \quad \dfrac{1+a}{2} < x < 1 \\ +\infty & , \quad \text{ailleurs} \end{cases} \tag{2.10}$$

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

$$\psi_0 \approx C e^{ikx} \exp\left(-(x - x_0)^2/2d^2\right) \tag{2.11}$$

Ici quelques explications sont nécessaires.

- $C \in \mathbb{R}$ est un coefficient de normalisation qui garantit $\langle\psi_0|\psi_0\rangle = 1$. Calculer C analytiquement en supposant que le paquet reste loin des bords (que le domaine d'intégration est $]-\infty, +\infty[$).
- Le paramètre x_0 définit le centre de la Gaussienne et il convient de le choisir entre le bord de gauche et la barrière $x_0 \in [0, (1-a)/2]$ pour placer le paquet devant la barrière.
- Le coefficient d définit la largeur de la Gaussienne et sera petit. Avec $d \ll 1$ on peut localiser le paquet dans une zone de taille d.
- Le nombre $k = 2\pi/\lambda$ est un nombre d'onde. Idéalement, on veut une longueur d'onde courte $\lambda \ll 1$ afin d'avoir un paquet d'onde qui ressemble à un paquet d'onde.

Le paquet d'onde initial et le potentiel sont définis par 5 paramètres k, x_0, d, V, a et une difficulté dans ce projet est qu'il fait les choisir afin de retrouver une situation qui permet de mettre en évidence l'effet Tunnel. Ceci fixe deux contraintes sur les paramètres. La première contrainte est que le paquet d'onde doit se propager vers la droite. La vitesse moyenne initiale doit donc être positive, soit

$$U_0 = \langle\psi_0|\hat{p}|\psi_0\rangle > 0 \tag{2.12}$$

Calculer cette vitesse moyenne initiale analytiquement et exprimer-la en fonction des paramètres k, x_0, d. Avec cette vitesse initiale, on peut estimer le temps nécessaire pour que le paquet parcourt le domain de calcul. Donner ce temps

$$T = \dots \tag{2.13}$$

en fonction des paramètres k, x_0, d. Cette information nous sera utile au moment où on définit la durée de l'animation. Dans un temps T on peut être à peu près certain que le paquet aura atteint la barrière. La deuxième contrainte est que le paquet d'onde doit avoir une énergie initiale H_0 plus petite que V car sinon, on ne peut pas vraiment parler d'effet tunnel. Ainsi on a besoin de définir les paramètres tels que

$$H_0 = \langle \psi_0 | \hat{H} | \psi_0 \rangle > V \tag{2.14}$$

Calculer cette énergie moyenne initiale et exprimer-la en fonction des paramètres k, x_0, d. En pratique, on adaptera la hauteur de la barrière, c'est à dire le paramètre V pour avoir une barrière qui est plus élevée que H_0.

1. Créer une fonction qui renvoie le potentiel adimensionné $V(x)$ sur les points du maillage. Les paramètres V et a sont des variables globales qu'on définira plus tard.
2. Créer une fonction qui calcule la fonction d'onde initiale $\psi_0(x)$ sur le maillage. Les paramètres k, d, x_0 sont des variables globales qu'on définira plus tard.
3. Il faut maintenant choisir k, d, x_0 convenablement. Calculer pour quelques choix la fonction ψ_0 puis afficher la condition initiale ensemble avec la barrière comme en figure 2.1. Il faut aussi calculer H_0 et U_0 et afficher ces valeurs à l'écran. Créer un message pour vous avertir si $E_0 > V$. Si ceci est le cas, retourner au pas précédent, augmenter V la barrière de potentiel, puis recalculer les fonctions propres.
4. Si vous avez réussi à créer un paquet d'onde, suffisamment localisé, suffisamment placé à gauche de la barrière et d'énergie initiale $H_0 < V$, vous pouvez passer à l'étape suivante, la simulation elle-même. Il faudra prendre un nombre suffisant de points, typiquement $M = 1000$ sera suffisant. Comme dans toutes les simulations instationnaires, l'utilisation du programme implicite est vivement conseillée.
5. Générer une animation qui montre l'évolution temporelle du paquet d'onde. En arrière plan, nous voulons voir le potentiel $V(x)$ en noir. Refaire les calculs avec un maillage plus fin afin d'améliorer cette animation si elle vous semble mal résolue. Normalement vous devez voir un paquet d'onde qui avance vers la barrière de potentiel. Selon la hauteur et la largeur de la barrière, V et a, une partie du paquet arrivera à percer la barrière. Ceci est l'effet tunnel.
6. Afin d'étudier l'effet tunnel de manière quantitative, on a veut calculer la probabilité de présence de la particule à gauche ou à droite de la barrière et comment cela varie au cours du temps. Trouver une méthode pour calculer cette probabilité à chaque instant. Faire ensuite une courbe qui montre comment ces deux probabilités varient au cours du temps. Est-ce qu'on attend un état asymptotique à temps long où les probabilités de présence ne varient plus ?
7. Etudier comment la largeur de la barrière a et son hauteur V influencent l'effet tunnel.

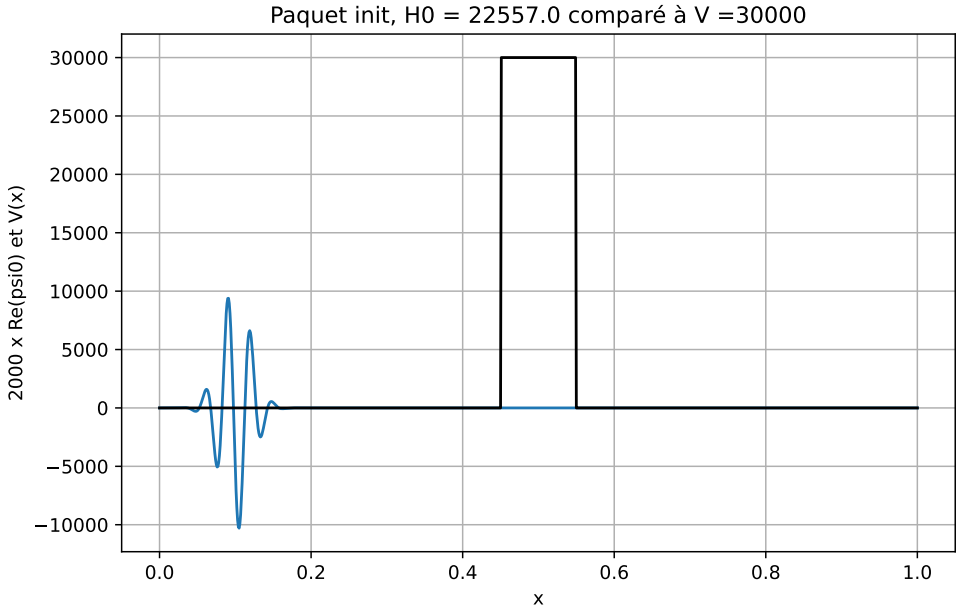


FIGURE 2.1 – Paquet d'onde initial (partie réelle) devant la barrière de potentiel.