Équation de Schrödinger à deux dimensions : module Python

1. Introduction

Ce document présente le module python `pyschrodinger2d`, qui permet de résoudre numériquement des problèmes de diffusion d'un paquet d'onde par un potentiel, ou de diffraction par un obstacle (diffraction par des fentes).

La méthode numérique est exposée dans Équation de Schrödinger à deux dimensions.

La représentation graphique de la densité de probabilité est faite sous forme d'image ou sous forme de surfaces.

Si l'on dispose d'une carte graphique moyen ou haut de gamme, on peut utiliser une plateforme OpenCL pour faire les calculs. Avec un processeur multicœurs, l'utilisation d'OpenCL peut être intéressante.

2. Installation

Exécutables d'installation pour python win32 :

- ▷ schrodinger2d-0.1.win32-py2.7.exe
- ▷ schrodinger2d-0.1.win32-py3.4.exe
- ▷ schrodinger2d-0.1.win32-py3.5.exe

Exécutable d'installation pour python 3.5 amd64 :

- ▷ schrodinger2d-0.1.win-amd64-py3.5.exe

Code source :

- ▷ schrodinger2d-0.1.zip

Bibliothèques nécessaires pour la compilation : openGL, glew, freeglut, pthread, openCL (optionnel)

Installation par PIP pour python win32 :

- ▷ Aller dans le dossier `Scripts` de la distribution (python 2.7 win32, 3.4 win32, 3.5 win32, python 3.5 amd64)
- ▷ Exécuter : `pip install schrodinger2d`.
- ▷ Si nécessaire, mettre à jour pip puis recommencer.

3. Scripts de démonstration

- ▷ Fentes
- ▷ Fentes avec montage vidéo
- ▷ Disque de potentiel uniforme
- ▷ Disque de potentiel uniforme avec montage vidéo
- ▷ Barrière de potentiel
- ▷ Potentiel coulombien
- ▷ Tests de vitesse de calcul

4. Interface

4.a. Définition du maillage

La classe principale est `Schrodinger2d`. Elle se trouve dans le module `pyschrodinger2d.main`. Son constructeur effectue les allocations de mémoire pour le maillage, et permet de définir la palette de couleur utilisée pour les représentations graphiques.

Le maillage comporte un nombre de points puissance de 2 dans les deux directions x et y. On définit un maillage réduit et un niveau de subdivision pour définir le maillage utilisé dans les calculs. Le maillage réduit est utilisé pour définir le problème.

```
object = Schrodinger2d.Schrodinger2d(px_min,py_min,levels,Lx,normalize,colormap,g
```

Définition du maillage et du mode de calcul des couleurs

- ▷ `px_min,py_min (integer)` : puissances définissant les dimensions du maillage réduit. Les nombres de points sont 2^{px_min} dans la direction X et 2^{py_min} dans la direction Y.
- ▷ `levels (integer)` : le maillage de calcul a $2^{px_min+levels}$ et $2^{py_min+levels}$ points.
- ▷ `Lx (float)` : largeur du domaine.
- ▷ `normalize (boolean)` : normalisation de la densité de probabilité, c'est-à-dire division par sa valeur maximale à chaque pas de calcul.
- ▷ `colormap (integer)` : palette de couleurs utilisée pour les représentations graphiques de la densité de probabilité. Les deux possibilités sont 1 (niveau de gris) et 2 (palette couleur).
- ▷ `gamma (float)` : coefficient gamma pour la définition de la palette. Un coefficient inférieur à 1 permet de renforcer la visibilité des faibles valeurs.

Une seule instance de cette classe est utilisable, car le programme C sous-jacent ne comporte qu'une instance des données.

```
Schrodinger.close()
```

Libération de l'espace mémoire réservé pour le maillage. À appeler avant d'ouvrir une nouvelle instance de la classe.

4.b. Définition du potentiel

La valeur par défaut du potentiel sur tout le domaine est 0. On peut définir un potentiel constant sur un domaine rectangulaire ou circulaire (définis sur le maillage réduit) avec les deux fonctions suivantes :

```
Schrodinger2d.potential_rectangle(ci0,cj0,width,height,v)
```

Potentiel uniforme sur un domaine rectangulaire

- ▷ `ci0,cj0 (integer)` : indices du centre du rectangle, défini sur le maillage réduit.
- ▷ `width, height (integer)` : demi-largeur et demi-hauteur du rectangle.

▷ `v (float)` : valeur du potentiel dans le rectangle.

```
Schrodinger2d.potential_disk(ci0,cj0,radius,v)
```

Potentiel uniforme sur un disque

▷ `ci0,cj0 (integer)` : indices du centre du disque.

▷ `radius (integer)` : rayon du disque.

▷ `v (float)` : valeur du potentiel dans le rectangle.

Le potentiel peut aussi être défini par une fonction $V(x, y)$. L'origine des coordonnées se trouve en bas à gauche du domaine. La largeur `Lx` du domaine est définie dans le constructeur. La hauteur dépend des proportions du maillage.

```
Schrodinger2d.potential_function(V)
```

Définition du potentiel par une fonction

▷ `V (function)` : fonction de la forme `V(x,y)`.

4.c. Définition du pas de temps et de l'équation de Schrödinger

La fonction suivante définit le pas de temps et calcule les coefficients du schéma numérique pour l'équation de Schrödinger. Elle doit être appelée *après* la définition du potentiel et *avant* la définition des conditions limites.

```
Schrodinger2d.schrodinger(dt)
```

Définition du pas de temps et initialisation des coefficients du schéma numérique.

▷ `dt (float)` : pas de temps.

4.d. Définition de conditions limites

Des conditions limites d'annulation de la fonction d'onde peuvent être définies sur des lignes. Par défaut, la fonction d'onde s'annule sur les bords du domaine.

Les lignes sont définies sur la maille réduite. L'indice d'abscisse i va de 0 à $2^{px_min} - 1$. L'indice d'ordonnée j va de 0 à $2^{py_min} - 1$.

La fonction suivante permet de définir une ligne d'annulation :

```
Schrodinger2d.zero_line(i0,j0,length,direction)
```

Ligne d'annulation de la fonction d'onde, définie sur le maillage réduit.

▷ `i0,j0 (integer)` : indices du point d'origine de la ligne.

▷ `length (integer)` : longueur de la ligne.

▷ `direction (char)` : direction de la ligne : 'u' pour up, 'd' pour down, 'l' pour left, 'r' pour right.

On peut aussi définir une liste de lignes :

```
Schrodinger2d.zero_line_list(line_list)
```

Liste de lignes d'annulation de la fonction d'onde.

▷ `line_list (list)` : liste de listes de la forme `[i9,j0,length,direction]`.

4.e. Ordre des définitions

Pour résumer, voici dans quel ordre les définitions doivent être faites :

▷ Définition du potentiel (si nécessaire).

▷ Appel de la fonction `schrodinger(dt)` pour définir le pas de temps.

▷ Définition de conditions limites (si nécessaire). La condition limite sur le bord du domaine est automatiquement définie.

4.f. Condition initiale

La fonction suivante permet de définir une condition initiale pour un paquet d'onde gaussien se propageant dans la direction x. Voir la page Équation de schrödinger à une dimension pour les conditions à respecter.

```
Schrodinger2d.paquet(x0,y0,k0,sigma0)
```

Définition d'un paquet d'onde initial

▷ `x0,y0 (float)` : centre du paquet. L'abscisse est comprise entre 0 et `Lx` (largeur du domaine). L'ordonnée est comprise entre 0 et Ly (hauteur du domaine).

▷ `k0 (float)` : nombre d'onde.

▷ `sigma0 (float)` : largeur du paquet (écart-type de la position).

4.g. Initialisation OpenCL

Pour le calcul sur une plateforme OpenCL, des fonctions de configuration doivent être appelées après le constructeur.

La première fonction permet d'afficher les plateformes openCL présentes sur le système, et pour chaque plateforme les périphériques associés. Les plateformes et les périphériques sont numérotés à partir de 0.

```
Schrodinger2d.opencl_platforms()
```

Affiche sur la console les plateformes openCL et leurs périphériques associés.

Par défaut, la plateforme 0 et le périphérique 0 sont sélectionnés. La fonction suivante permet de sélectionner une plateforme openCL et un périphérique :

```
Schrodinger2d.set_opencl_platform_device(platform,device)
```

Sélection d'une plateforme et d'un périphérique pour effectuer les itérations.

▷ `platform` : numéro de la plateforme, 0 pour la première

▷ `device` : numéro du périphérique, 0 pour le premier

Si des pilotes OpenCL sont installés pour le processeur central, celui-ci apparaît comme plateforme OpenCL. L'utilisation d'OpenCL n'a vraiment d'intérêt que sur un processeur graphique d'une carte graphique moyen ou haut de gamme (type carte pour jeu ou pour station de travail graphique). Il faut choisir la plateforme correspondante.

La fonction suivante doit être appelée après la sélection de la plateforme :

```
Schrodinger2d.opencl_init()
```

Initialisation du programme OpenCL (compilation des noyaux).

La fonction suivante doit être appelée lorsque toute la configuration du système est faite (après la fonction `init` décrite ci-dessous.

```
Schrodinger2d.opencl_create_memory()
```

Création des tampons de mémoire sur la carte graphique et transfert des données.

Lorsque les calculs sont terminés, on doit libérer la mémoire avec la fonction suivante :

```
Schrodinger2d.opencl_release_memory()
```

Destruction des tampons de mémoire sur la carte graphique.

4.h. Itérations de calcul

Avant d'effectuer les calculs, il faut initialiser les matrices de calcul du schéma numérique avec la fonction suivante :

```
Schrodinger2d.init()
```

Initialisation des matrices de calcul.

La fonction suivante effectue des itérations avec le processeur central (CPU) :

```
Schrodinger2d.iterations(ti,tf,threads=1)
```

Itérations sur CPU.

▷ `ti,tf (float)` : instants initial et final.

▷ `threads (integer)` : 1, 2 ou 4 : nombre de threads.

Seule la différence entre les deux instants importe (on obtient le nombre d'itérations en la divisant par le pas de temps). La possibilité de définir le temps de manière absolue est introduite ici pour une éventuelle future implémentation d'un potentiel dépendant du temps.

Avec un processeur multi-cœurs, l'utilisation de 2 threads apporte un gain d'un facteur 2 environ.

La fonction suivante effectue des itérations sur le processeur graphique (GPU).

```
Schrodinger2d.opencl_iterations(ti,tf)
```

Itérations sur GPU.

▷ `ti,tf (float)` : instants initial et final.

4.i. Récupération de la fonction d'onde

La fonction suivante permet d'obtenir la fonction d'onde sous forme de deux matrices, une pour la partie réelle, l'autre pour la partie imaginaire.

```
(psi_re,psi_im)=Schrodinger2d.get_psi()
```

Obtention de la fonction d'onde

▷ `psi_re (ndarray)` : partie réelle, sous la forme d'une matrice numpy.

▷ `psi_im (ndarray)` : partie imaginaire.

La fonction suivante permet d'obtenir la densité de probabilité, c'est-à-dire le module au carré de la fonction d'onde. Ce module est divisé par sa valeur maximale si l'argument `normalize` introduit dans le constructeur vaut `True`.

```
P=Schrodinger2d.get_proba()
```

Obtention de la densité de probabilité.

▷ `P (ndarray)` : densité de probabilité, sous la forme d'une matrice numpy.

Si `normalize=False`, la fonction d'onde est normalisée au sens des probabilités, dans la mesure ou le paquet d'onde initial est normalisé. La normalisation faite lorsque `normalize=True` est une normalisation au sens du traitement d'image, qui permet de ramener les valeurs dans l'intervalle [0,1].

La fonction suivante permet de récupérer la densité de probabilité sous la forme d'une image RGB. Les couleurs sont calculées avec la palette définie dans le constructeur. Les lignes d'annulation de la fonction d'onde et les zones rectangulaires ou circulaires de potentiel non nuls apparaissent en blanc sur l'image.

```
img=Schrodinger2d.get_proba_colors()
```

Obtention de la densité de probabilité sous la forme d'une image.

▷ `img (ndarray)` : image RGB

Les valeurs des trois couches RGB sont des flottants dans l'intervalle [0,1]. Lorsque `normalize=False`, il y a écrêtage à 1 des valeurs dépassant 1.

4.j. Animation graphique (OpenGL)

Une animation graphique peut être obtenue pendant le calcul avec la fonction suivante :

```
Schrodinger2d.start_rendering(ti,tf,opencl,width,height,surface_height,rendering,
```

Animation sous la forme d'une image ou d'une surface.

- ▷ `ti,tf (float)` : instants initial et final.
- ▷ `opencl (boolean)` : utilisation de la plateforme OpenCL.
- ▷ `width,height (integer)` : taille de la fenêtre.
- ▷ `rendering (integer)` : 1 pour obtenir une image, 2 pour une surface.
- ▷ `surface_height (float)` : hauteur de la surface. La densité de probabilité est multipliée par cette valeur.
- ▷ `angle_x,angle_y (float)` : angles (en degré) définissant l'orientation de la caméra (pour le rendu en surface).

Il peut être nécessaire de générer les images de l'animation pour faire un montage vidéo. La fonction suivante effectue des itérations et renvoit l'image finale (celle-ci n'est pas affichée) :

```
img=Schrodinger2d.offscreen_rendering(ti,tf,opencl,width,height,surface_height,re
```

Itérations et récupération d'une image.

- ▷ `ti,tf (float)` : instants initial et final.
- ▷ `opencl (boolean)` : utilisation de la plateforme OpenCL.
- ▷ `width,height (integer)` : taille de la fenêtre.
- ▷ `rendering (integer)` : 1 pour obtenir une image, 2 pour une surface.
- ▷ `surface_height (float)` : hauteur de la surface. La densité de probabilité est multipliée par cette valeur.
- ▷ `angle_x,angle_y (float)` : angles (en degré) définissant l'orientation de la caméra (pour le rendu en surface).
- ▷ `img (ndarray)` : image RGBA (8 bits par couche), identique à celle qui serait obtenue dans une animation en temps réel.

5. Exemple

```
import numpy
from schrodinger2d.main import Schrodinger2d
from matplotlib.pyplot import *
import imageio
```

On définit un maillage réduit de 128 points par 64. Le niveau de subdivision est 3, ce qui donne un maillage final de 1024 par 512, suffisant pour faire une bonne simulation. La taille du domaine est 2 par 1.

```
px_min = 7
py_min = 6
levels = 3
Lx = 2.0
solver = Schrodinger2d(px_min,py_min,levels,Lx,normalize=False,colormap=2,gamma=0.6)
```

On initialise la plateforme OpenCL (si l'on dispose d'une carte graphique) :

```
solver.opencl_platforms()
solver.set_opencl_platform_device(0,0)
solver.opencl_init()
```

On doit ensuite définir le pas de temps. Un bon choix est deux fois le pas d'espace au carré :

```
dt = 2*solver.dx**2
solver.schrodinger(dt)
```

Pour obtenir la diffraction par deux fentes, on définit trois traits de condition limite nulle :

```
lignes_nulles_fentesA = [[64,0,29,"u"],[64,31,2,"u"],[64,35,29,"u"]]
solver.zero_line_list(lignes_nulles_fentesA)
```

Il est important d'appeler la fonction `schrodinger` avant la fonction `zero_line_list`. Dans le cas contraire, la fonction `schrodinger` écraserait les lignes définies.

On définit un paquet d'onde de déplaçant dans le sens de la longueur, et initialement situé à gauche :

```
x0=0.7
y0=0.5
k0 = 250
sigma0 = 0.1
E=k0*k0
solver.paquet(x0,y0,k0,sigma0)
```

On doit alors initialiser les matrices et les tampons mémoires pour OpenCL :

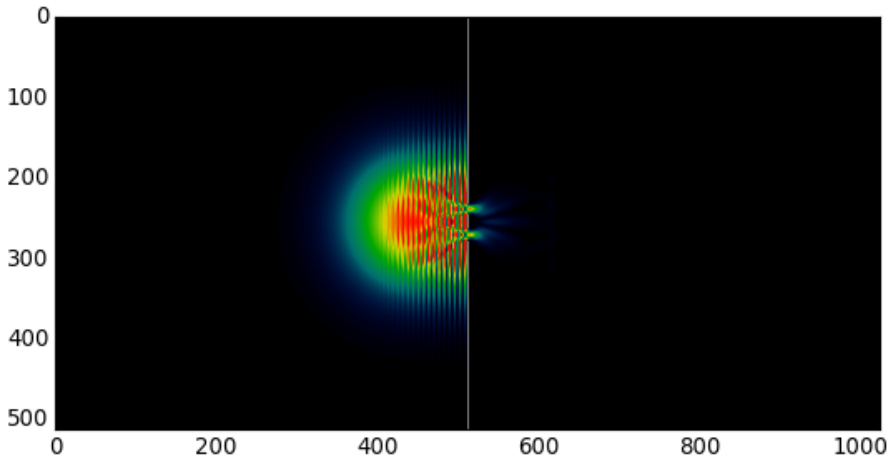
```
solver.init()
solver.opencl_create_memory()
```

Voici le calcul de quelques itérations :

```
ti = 0.0
tf = 50*dt
solver.opencl_iterations(ti,tf)
```

On récupère la densité de probabilité pour l'afficher :

```
img = solver.get_proba_colors()
figure()
imshow(img)
```



Avec cette palette de couleurs, les zones de forte densité sont rouges.

Pour faire un montage vidéo, on peut utiliser le module `imageio` :

```
writer = imageio.get_writer('../../../../figures/numerique/diffusion/pyschrodinger2d/
t = 0
solver.paquet(x0,y0,k0,sigma0)
solver.opencl_release_memory()
solver.opencl_create_memory()
delta_t = dt
while t < 200*dt:
    img = solver.offscreen_rendering(t,t+delta_t,opencl=1,width=solver.width,height=s
    t+=delta_t
    writer.append_data(img)
writer.close()
```

Avec `normalize=False`, il y a une saturation de couleur rouge lorsque la densité probabilité dépasse 1 (ce qui se produit lorsque le paquet est proche des fentes).

Voici comment obtenir une animation avec une surface :

```
writer = imageio.get_writer('../../../../figures/numerique/diffusion/pyschrodinger2d/
t = 0
solver.paquet(x0,y0,k0,sigma0)
solver.opencl_release_memory()
solver.opencl_create_memory()
delta_t = dt
while t < 200*dt:
    img = solver.offscreen_rendering(t,t+delta_t,opencl=1,width=solver.width,height=s
    t+=delta_t
    writer.append_data(img)
writer.close()


solver.opencl_release_memory()
solver.close()
```