Entrega Simulación FQ 1

Jordi Frias, Joana Pi-Suñer

19 de enero de 2017

Estudio numerico de la dinámica del estado fundamental de un potencial harmónico.
Simulación en Python

1. Marco Teórico. El oscilador armónico cuántico

El oscilador armónico es uno de los sistemas más estudiados en todos los campos de la física. El potencial es del tipo $V = \frac{1}{2}kx^2$ y su Hamiltoniano sigue la expresión $H = T+V = \frac{1}{2m}p^2+\frac{1}{2}kx^2$.

Las funciones de onda que son solución a la ecuación de Schrödinger unidimensional son:

$$\Phi_n(x) = A_n e^{-\frac{m\omega x^2}{2\hbar}} H_n\left(\sqrt{\frac{m\omega}{\hbar}}x\right) n = 1, 2, 3, \cdots$$

con $A_n = \frac{1}{\sqrt{2^n n!}}(\frac{m\omega}{\pi h})^{\frac{1}{4}}$, H_n los polinomios de Hermite de grado n $(1, 2x, 2x^2-2, \cdots)$ y $\omega = \sqrt{\frac{k}{m}}$
.

Las energías correspondientes para cada función són : $E_n = \hbar\omega(n + \frac{1}{2})$

2. Planteamiento inicial del problema

Durante nuestra simulación el sistema estudiado no es propio del Hamiltoniano, el programa calculará la evolución temporal de nuestra función de onda. Primeramente, generaremos las funciones base de nuestro sistema, enseguida definiremos la función de onda con la que trabajaremos. Calcularemos también sus coeficientes en la base propia del hamiltoniano y su respectiva evolución temporal.

QP1: Dada una función inicial $\psi(x)$ podemos encontrar $\psi(x,t)$ simplemente resolviendo numéricamente, para cada tiempo t, la ecuación de Schrödinger dependiente del tiempo. Si sólo queremos encontrar la solución para un determinado tiempo (y no evolucionarlo) ésta forma tal vez sea mejor por ser más sencilla y directa. Pero, si queremos evolucionar el tiempo, no es la manera más recomendable, puesto que estaremos introduciendo un error en cada tiempo t. Además, no es demasiado eficiente computacionalmente puesto que tendríamos que resolver numéricamente una ecuación diferencial de segundo grado en cada instante de tiempo. De la manera que presentamos en esta simulación sólo tendremos que computar los coeficientes una vez por cada operación, en el instante $t = 0$.

3. Traslación y Kick al estado fundamental

En esta parte de la simulación estudiaremos la dinámica del estado fundamental del potencial armónico (i.e. $\Phi_0(x)$) después de haberlo desplazado a una cierta distancia x_0 y/o haberle dado un cierto momento lineal inicial p_0.

Tendremos los operadores unitarios traslación y kick (patada) respectivamente:

$$U_{x_0} = e^{-ix_0p/h} \qquad\qquad U_{p_0} = e^{-ip_0x/h}$$

Definimos las energías propias y las funciones propias en nuestro programa. Y expresaremos la función de onda inicial a $t = 0$ (solución independiente del tiempo) en la base de $\Phi_n(x)$.

$$\phi_{x_0}(0, x) = U_{x_0}\Phi_0(x) \qquad\qquad \phi_{p_0}(0, x) = U_{p_0}\Phi_0(x)$$

Pregunta 1: En el caso clásico, darle una patada (kick) o aplicar una traslación a nuestro estado fundamental es exactamente lo mismo que aplicar la operación inversa. Aprovecharemos el hecho de que $e^A \cdot e^B = e^{A+B} \cdot e^{\frac{1}{2}[A,B]}$ y $[x, p] = i\hbar$ para verlo. Empecemos por el caso $U_{x_0} \cdot U_{p_0}$:

$$U_{x_0}U_{p_0} = e^{-ix_0p/\hbar}e^{-ip_0x/\hbar} = e^{-\frac{i}{\hbar}(x_0p+p_0x)}e^{\frac{1}{2}(i/\hbar)^2x_0p_0[p,x]} =$$
$$= e^{-\frac{i}{\hbar}(x_0p+p_0x)}e^{\frac{1}{2}(-1/\hbar^2)x_0p_0(-i\hbar))} = e^{-\frac{i}{\hbar}(x_0p+p_0x)}e^{\frac{1}{2}i\hbar x_0p_0} \equiv e^{-\frac{i}{\hbar}(x_0p+p_0x)}$$

La equivalencia viene del hecho que, al no existir ningún operador en el término $e^{\frac{1}{2}i\hbar x_0p_0}$, al aplicarlo a un estado quedaría como una fase, que podemos eliminar sin temores.

Veamos, de la misma forma, que pasa en el caso $U_{p_0} \cdot U_{x_0}$:

$$U_{p_0}U_{x_0} = e^{-ip_0x/\hbar}e^{-ix_0p/\hbar} = e^{-\frac{i}{\hbar}(p_0x+x_0p)}e^{\frac{1}{2}(i/\hbar)^2x_0p_0[x,p]} =$$
$$= e^{-\frac{i}{\hbar}(p_0x+x_0p)}e^{\frac{1}{2}(-1/\hbar^2)x_0p_0i\hbar)} = e^{-\frac{i}{\hbar}(p_0x+x_0p)}e^{-\frac{1}{2}i\hbar x_0p_0} \equiv e^{-\frac{i}{\hbar}(x_0p+p_0x)}$$

De la misma manera que en el caso anterior, quitamos la fase y, además aprovechamos el hecho de que la suma de operadores conmuta: $A + B = B + A$. Así pues, queda visto que $U_{x_0} = U_{p_0}$.

Pregunta 2: Ya que el operador traslación tiene una derivada dentro del exponencial, nos puede causar complicaciones numéricas, vamos a facilitarlo. Demostraremos que $U_{x_0} f(x) = f(x - x_0)$ mediante una expansión de Taylor alrededor de x:

$$U_{x_0}\psi(x) = e^{-ix_0 p/\hbar}\psi(x) = e^{-x_0 \frac{\partial}{\partial x}}\psi(x) =$$
$$= \sum_{n=0}^{\infty} \frac{(-x_0)^n}{n!} \frac{\partial^n}{\partial x^n}\psi(x) = \sum_{n=0}^{\infty} \frac{\psi^{(n)}(x)}{n!}(-x_0)^n = \psi(x - x_0)$$

La última igualdad crea algo de confusión pero es sencilla de ver pensando en que expandimos alrededor de x, y tomando $x - x_0$ como variable. Es decir que $(-x_0)^n$ viene de: $(x - x_0 - x)^n = (-x_0)^n$.

Tanto para la traslación como para el kick tendremos que los coeficientes a t=0 valen:

$$C_n(0) = \int_{-\infty}^{\infty} dx \phi_n^*(0, x)\Phi_{x_0}(x) \qquad C_n(0) = \int_{-\infty}^{\infty} dx \phi_n^*(0, x)\Phi_{p_0}(x)$$

Haremos evolucionar estos coeficientes según la ecuación de Schrödinger:

$$C_n(t) = C_n(0).e^{-\frac{iE_n t}{\hbar}}$$

Para acabar, nadamás falta sumar estos coeficientes con sus funciones de onda correspondientes:

$$\Phi(t, x) = \sum_{n=0} NC_n(t)\phi_n(x)$$

Pregunta 3: Si tomamos $\hbar = 1$, $\frac{p_0}{m}$ y $m\omega^2 x_0$ dentro del intervalo $(1, 8)$, para valores de tiempo $\Delta t = 0,1$ y $N = 60$ no introduciremos ningún error numérico significante. En nuestro caso tomamos $\hbar = 1$, $m = 1$ y $k = 1$. El numero de elementos de la base afecta dramáticamente al tiempo de ejecución de cálculo, así que hay que optimizarlo lo más posible.

Una forma de comprobar que N es óptimo es mirar que la probabilidad acumulada se acerca mucho a 1. Es decir fijar una tolerancia e ir calculando los coeficientes c_n hasta que la suma de sus valores absolutos al cuadrado sea mayor que la tolerancia:

$$\sum_{n=0}^{N} |c_n|^2 > tol \simeq 1$$

Por ejemplo, podemos tomar $tol = 1 - 10^{-4}$. En nuestro caso, para hacer los cálculos más rápidos hemos tomado $tol = 0{,}99$. Perdemos algo de precisión pero para ver que todo encaja hay más que suficiente.

Nuestra forma de proceder asegura un error al principio de la ejecución (para $t = 0$) que no se propaga con el tiempo, puesto que una vez tenemos los coeficientes calculados no es necesario calcularlos en cada instante de tiempo. Otra forma posible de proceder sería aliviar la complejidad del cálculo para $t = 0$ e ir calculando los coeficientes en cada instante de tiempo, por ejemplo, resolviendo numéricamente la ecuación de Schrödinger dependiente del tiempo. Éso conduciría a una acumulación del error en el tiempo no demasiado recomendable.

QP3: Si la partícula se encuentra en el estado fundamental y le damos un kick (o hacemos una traslación) el potencial actúa como una pista de skateboard y la partícula como una pelota que sube y baja. Ya que no hay pérdida de energía en este oscilador, la partícula no para de subir y bajar deslizándose como una patineta en un 'skatepark' con velocidad constante. El valor de indeterminación de x y p no creemos que vaya a variar pero su valor esperado correspondiente si que varía.

Pregunta 4: Hacemos que el programa haga una representación temporal de las partes reales e imaginarias de $\Phi(t, x)$ y de la probabilidad $\Phi(t, x)^2$. En las Figuras 1, 2, 3 se puede ver la función

de onda inicial, trasladada y su evolución con un kick. Además, hemos subido en una lista de reproducción de Youtube los vídeos mostrando la evolución de los estados. Los vídeos del 1 al 4 son los que corresponden con ésta sección de la simulación. El último, como extra, corresponde a la aplicación de un kick y una traslación simultáneamente.

Vemos que, como esperábamos, la traslación y el kick hacen que el valor esperado se muevan como si la partícula estuviese subiendo y bajando una cuesta mientras que, la desviación estándar varía poquito (no 0 como habíamos previsto, se mueve algo).

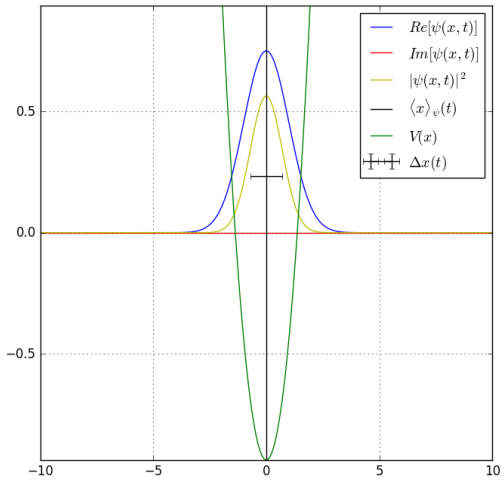

Figura 1: Plot del estado fundamental ψ_0.

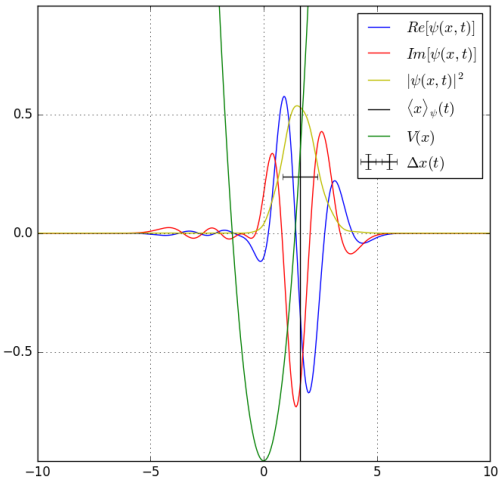

Figura 2: Plot de la traslación con $x_0 = 3$.

Pregunta 5: Una vez vista la evolución temporal de la función de onda y la distribución de la probabilidad de posición, vamos a ver como evolucionan en el tiempo los valores esperados de la posición, el momento lineal y del Hamiltoniano. Las gráficas que representaremos en la simulación son las de la figura 4 y la figura 5.

Representaremos estas gráficas para los dos casos estudiados: traslación y kick, para poder enseguida compararlos. En estas observamos el valor esperado de la posición (y el momento), con su

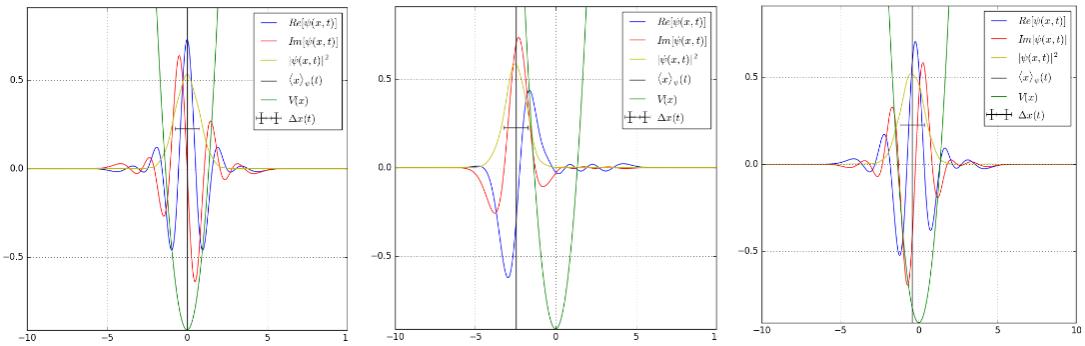

Figura 3: Plot de la evolución del kick con $p_0 = 3$.

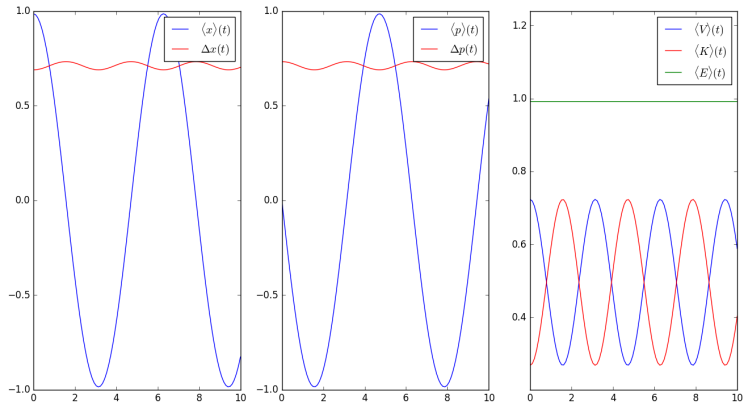

Figura 4: Plot de los valores esperados con una traslación $x_0 = 1$.

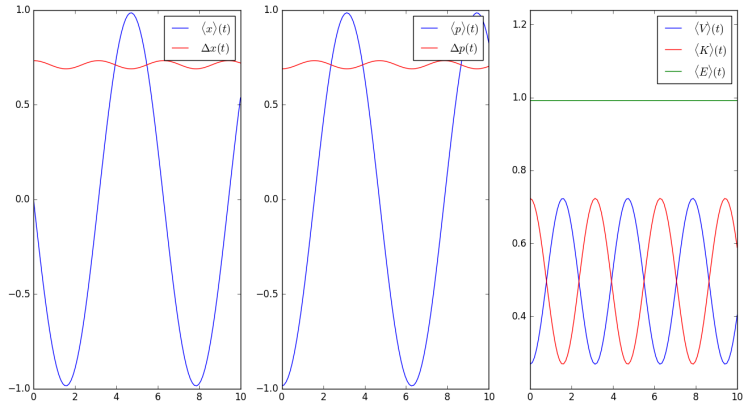

Figura 5: Plot de los valores esperados con un kick $p_0 = 1$.

respectiva incertidumbre. Vemos que la incertidumbre varía ligeramente con el tiempo. El hecho de que exista ésta incertidumbre y que ésta varíe con el tiempo constituye la principal diferencia entre la mecánica clásica y la cuántica (comparando a lo que habíamos predicho en la QP3, pensado en modo clásico). Podemos observar que la gráfica representativa del kick y de la traslación se parecen mucho, tienen la misma amplitud de onda y el mismo periodo, tienen una fase de diferencia. Podemos ver también que el Hamiltoniano es constante para todo t, como podíamos esperar.

Pregunta 6: Una buena representación para ver la evolución temporal de cualquier partícula es el espacio x-p, en el cual las coordenadas de una partícula se ven determinadas por su posición en el eje x, y su momento en el ej y. Representamos también la incertidumbre de la posición y del momento en el mismo gráfico.

En este espacio, después de una traslación o un kick el valor esperado del momento será inversamente proporcional al valor esperado en la posición, por tanto el valor esperado x-p irá dibujando una suerte de circunferencia. Además la incertidumbre variará y los ejes (en nuestro caso adornados con un circulo azul) se irán achatando o agrandando cumpliendo siempre que $\Delta x \Delta p = \frac{\hbar}{2}$, ya que estamos siempre trabajando con Gaussianas. En la Figura 6 podemos ver la evolución del estado en el espacio tradicional conjuntamente con el espacio x-p. Sólo lo hemos puesto para un kick ya que en nuestra lista de reproducción se puede observar en forma de vídeo la evolución para el kick, la traslación y los dos a la vez. Los vídeos del 1 al 4 corresponden con ésta sección.

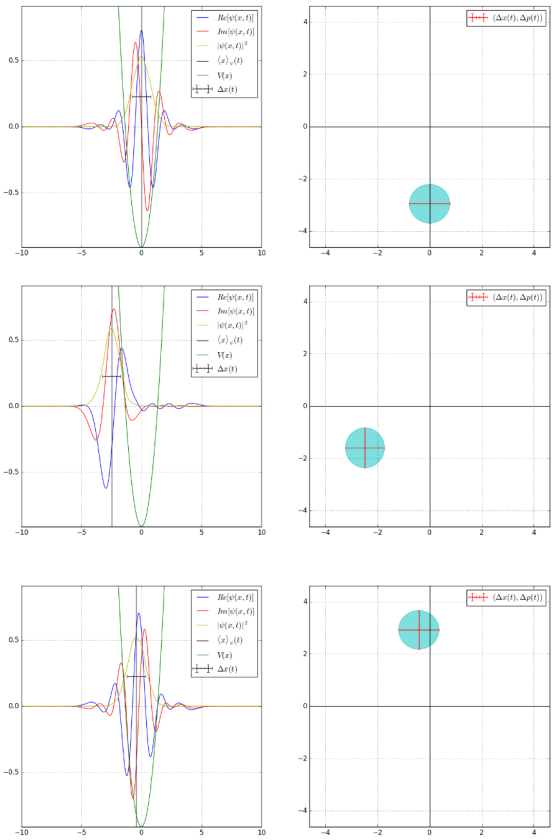

Figura 6: Plot de la evolución del kick en el espacio tradicional conjuntamente con el espacio x-p. $p_0 = 3$.

Pregunta 7: Para acabar, vamos a comprobar el Teorema de Ehrenfest de forma visual. En la Figura 7 se puede apreciar cómo las igualdades efectivamente se cumplen.

Pregunta 8: Teorema de Ehrenfest La mecanica cuántica se diferencia de la clásica por su carácter determinista. Un ejemplo perfecto para ilustrar ésto es el lanzamiento de un electrón hacia una doble ranura, clásicamente conociendo x y p iniciales podemos saber por donde pasará el electrón. Pero en cambio de punto de vista cuántico, el azar juega un papel importante y nos es imposible medir exactamente su posición.

El Teorema de Ehrenfest nos permite describir la dinámica del sistema gracias a las ecuaciones clásicas de los valores esperados.

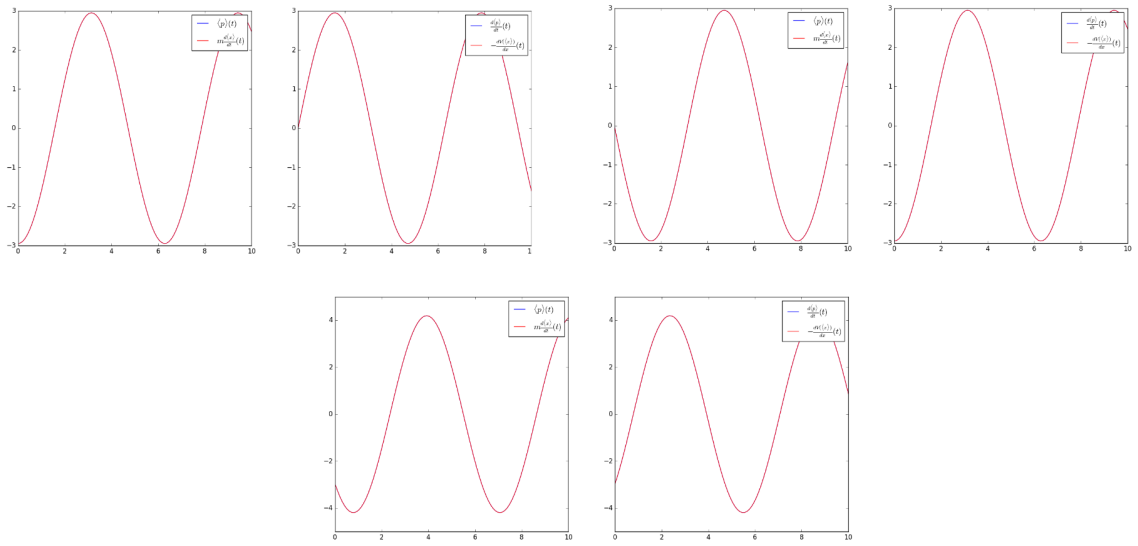

Figura 7: Las dos igualdades del teorema de Ehrenfest. La primera es para una una traslación con $x0 = 3$ y la segunda para un kick con $p_0 = 3$. Vemos que en ambos casos las líneas azules y rojas son coincidentes, lo que indica que la igualdad realmente se cumple

$$\frac{\partial \langle \hat{A} \rangle_\phi}{\partial t} = \frac{1}{i\hbar} \langle [\hat{A}, \hat{H}] \rangle_\phi + \langle \frac{\partial \hat{A}}{\partial t} \rangle_\phi$$

Si conocemos $\langle x_0 \rangle$ y $\langle p_0 \rangle$ iniciales (simplificamos la fuerza $F(x)$), podremos conocer la evolución temporal del valor esperado del momento y de la posición. ($\langle x \rangle(t)$ y $\langle p \rangle(t)$).

Para una partícula de masas m y un momento p, moviéndose en el seno de un potencial real $V(x,t)$, las ecuaciones fundamentales de la mecánica clásica para los valores esperados (leyes de Newton cuánticas):

$$\frac{d\langle \hat{x} \rangle_\phi}{dt} = \frac{\langle \hat{P} \rangle_\phi}{m} \qquad\qquad \frac{d\langle \hat{P} \rangle_\phi}{dt} = -\langle \Delta V \rangle_\phi = \langle F(x) \rangle_\phi$$

Las ecuaciones de Ehrenfest no permiten afirmar las leyes clásicas, pero se pueden aproximar cuando el potencial se desarrollo en serie no tiene inármonicidades. Si desarrollamos en serie de Taylor, las componentes ármonicas tienen que aproximarse a cero.

$$F(x_0 - h) = f(x_0) + f'(x_0)h + \frac{f''(x_0)}{2!}h^2 + \cdots + h^2\epsilon(h)$$

$$\langle F(x,t) \rangle_\phi \simeq F(\langle x \rangle, t) ssi \Delta t << 1 \Rightarrow f'(x_0)h + \frac{f''(x_0)}{2!}h^2 + \cdots + h^2\epsilon(h) \simeq 0$$

Entonces concluimos la evolución temporal:

$$\langle \hat{x}(t) \rangle_\phi = \int_0^t \frac{\langle \hat{P}(t) \rangle_\phi}{m}, dt \qquad\qquad \langle \hat{P}(t) \rangle_\phi = \int_0^t \langle F(x) \rangle_\phi, dt$$

4. Estados de mínima indeterminación

Como ya sabemos, el estado fundamental del oscilador armónico cumple la condición de los estados de mínima indeterminación ($\Delta x \Delta p = \frac{\hbar}{2}$). Si cambiamos ahora el valor Δx manteniendo la forma gaussiana de la función, la nueva función de onda no será propia del Hamiltoniano y evolucionará con el tiempo. En $t = 0$ esta nueva función sigue cumpliendo $\Delta x \Delta p = \frac{\hbar}{2}$.

También sabemos que Δx depende de ξ, por lo tanto si variamos ξ podemos preparar estados gausianos de amplitud arbitraria, que satura Heisenberg en ciertos puntos. Estos estados con

$\Delta x \neq \Delta x_0 = \sqrt{\frac{\hbar}{2m\omega}}$ se llaman estados "squeezed". De la misma manera podemos crear los estados "squeezed.en el momento con $\Delta p \leq \Delta p_0 = \sqrt{\frac{\hbar m\omega}{2}}$.

QP5: Creemos que, al variar Δx, cómo $\Delta x \Delta p = \frac{\hbar}{2}$ se mantendrá invariante, pasará que Δp variará inversamente que Δx. Es decir, que nuestro círculo x-p pasará a ser una elipse mucho más pronunciada. Lo que no sabemos decir a priori es si ésta diferencia entre Δx y Δp se mantendrá constante en el tiempo o irá variando. Posteriormente, en la práctica hemos visto como, efectivamente va variando.

El estado "squeezedçon una traslación y/o un kick es uno de los estados más generales que saturan al teorema de Heisenberg. Éstos estados pueden estar desplazados en posición y momento.

$$\Psi(0,x) = Ce^{\frac{ip_0 x}{\hbar}} e^{-\frac{x^2}{4(\Delta x)^2}}$$

Con $\Delta x = \sqrt{\frac{\hbar}{2m\omega}} e^{-\xi}$, el parámetro $\xi \in \Re$ permita expandir/comprimir nuestra función de onda.

Pregunta 9 Debería pasar que Δp compensara siempre a Δx para que $\Delta x \Delta p = \frac{\hbar}{2}$ en tiempo 0. En nuestro caso ésto se cumple. En tiempo 0 si que coincide con lo que esperábamos, pero nosotros pensábamos que al avanzar el tiempo seguiríamos saturando el principio de incertidumbre de Heisenberg, pero parece que sólo se cumple en los ejes $\langle x \rangle = 0$ y $\langle p \rangle = 0$. Si hacemos que $\xi = 0$, es claro que volvemos a la primera sección y entonces sí se satura Heisenberg en el tiempo. En los dos últimos vídeos se pueden ver los dos ejemplos mencionados.

La constante de normalización para nosotros es $C = 0{,}751125544465$ en el caso $x_0 = 0$, $p_0 = 2$.

QP6: Los estados de mínima indeterminación comparten todos su forma Gaussiana, aunque con diferentes desviaciones estándar Δx. Si además aplicamos traslaciones, éstos también pueden estar centrados en diferentes x_0.

Si $\xi = 0$ recuperamos el estado fundamental, cómo vemos en el vídeo referenciado en la pregunta anterior.

Nosotros pensábamos que $\Delta x \Delta p$ se iba a mantener constante aunque ésto no necesariamente implica que las dos se mantengan constantes. Para que se mantengan constantes deberíamos no trasladar i/o kickear el estado y además deberíamos hacer que $\xi = 0$.

Para visualizar estos estados de manera clara, sólo lo vamos a estudiar teóricamente con un kick, es decir, aplicando el operador U_{p_0}. De esta forma, utilizaremos la función siguiente:

$$\Psi(0,x) = C * e^{\frac{ip_0 x}{\hbar}} * e^{-\frac{x^2}{4(\Delta x)^2}}$$

Simplemente cuando calculamos los coeficientes $C_n(0)$, remplazamos $U_{x_0,p_0}\psi_0(x)$ por la nueva función. Con este simple cambio podremos ver y estudiar como evolucionan los coeficientes dentro del pozo armónico. Hemos escogido valores 1 y -1 para ξ.

$$C_n(0) = \int_{-\infty}^{\infty} dx \phi_n^*(0,x) \Psi_{x_0}(x)$$

Evolucionamos estos coeficientes según la ecuación de Schrödinger:

$$C_n(t) = C_n(0).e^{-\frac{iE_n t}{\hbar}} \rightarrow \Psi(t,x) = \sum_{n=0} NC_n(t)\phi_n(x)$$

Pregunta 10 En la Figura 8 se aprecia cómo evoluciona con el tiempo la distribución de probabilidad. En comparación con el caso previo, en que la forma de la distribución siempre era claramente gaussiana y picuda, emulando una bola en un 'skatepark', en éste caso todo es mucho menos claro. La distribución varía mucho con el tiempo y, aunque el valor esperado sí evoluciona como esperábamos, todo se vuelve mucho más difuso y anti-intuitivo.

Pregunta 11 En la Figura 8 se puede apreciar la evolución x-p del sistema. Además, se puede ver de forma animada en los vídeos del 5 al 12 con diferentes configuraciones. Los valores de Δx y Δp covarían durante todo el movimiento haciéndose su producto $\frac{\hbar}{2}$ sólo en los ejes $\langle x \rangle = 0$ o

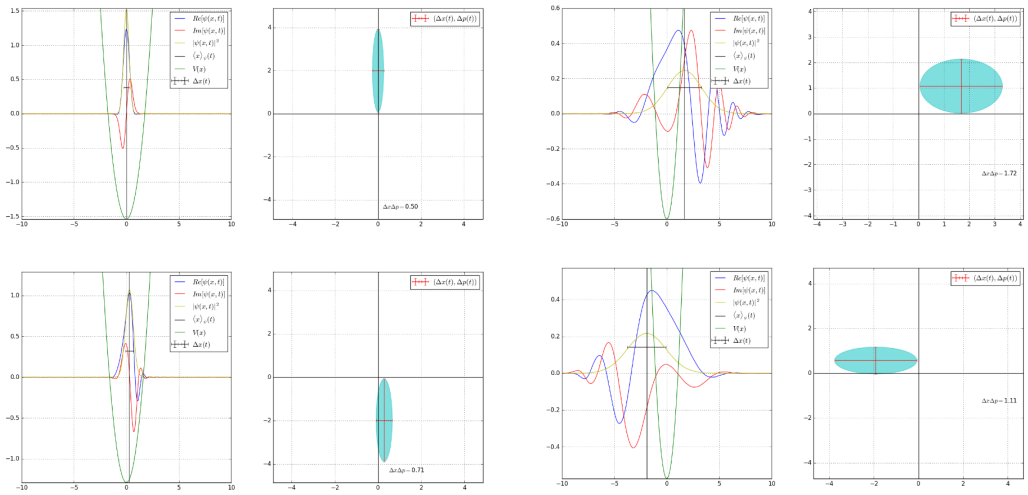

Figura 8: Evolución de la función de onda, de las distribuciones, de los valores esperados y incertidumbres conjuntamente con la evolución del plot x-p.

$\langle p \rangle = 0$, es decir, cuándo la energía cinética o la potencial son una 0 y la otra máxima. En la Figura 9 podemos observar la evolución de los valores esperados y del producto $\Delta x \Delta p$. Cómo ya hemos dicho, el producto es $\frac{\hbar}{2}$ tan sólo en algunos puntos, que se corresponden con los ejes.

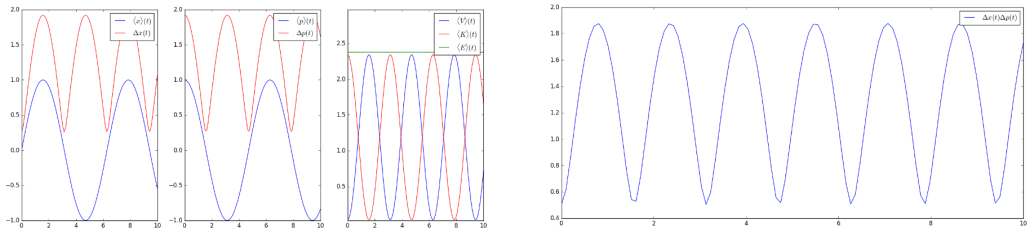

Figura 9: En la izquierda, evolución de los valores esperados e indeterminaciones con el tiempo. En la derecha, evolución de $\Delta x \Delta p$ en el tiempo. Observamos cómo se vuelve $\frac{\hbar}{2} = \frac{1}{2}$ en determinados puntos, correspondientes a los ejes $\langle x \rangle = 0$, $\langle p \rangle = 0$

Nosotros esperábamos mantener que $\Delta x \Delta p = \frac{\hbar}{2}$ pero éso sólo pasa en los ejes. En cuanto a los valores esperados, si esperábamos que hubiera una correlación entre $\langle p \rangle$ y $\langle K \rangle$, así como una covariación de los valores esperados de x y p con el tiempo.

Pregunta 12 En Figura 8 podemos ver cómo evoluciona el estado en el espacio x-p. Está hecho solo para un Kick. En el caso de no aplicar Kick, para no sobrecargar demasiado el pdf de imágenes nos referimos a los vídeos y vemos cómo, tal y como esperamos, no evoluciona su valor esperado con el tiempo pero sí lo hacen sus indeterminaciones x y p.

5. Código

5.1. Estado y Main

Hemos procurado de mantener el código siempre limpio y bien comentado para que sea autoexplicativo. El Main se encuentra en el fichero `estat.py` y es fácil de utilizar. La mayor parte de la funcionalidad se encuentra en el mismo fichero. A destacar las dos clases creadas `Estat` y `FuncioOna`. `Estat` sirve cómo una ligera capa por encima de `FuncioOna` para controlar y estructurar el desarrollo del programa. A su vez, `FuncioOna` acumula prácticamente todo el trabajo de cómputo e implementación de las matemáticas. Finalmente, en el fichero `altres_funcions.py`

tenemos varias funciones útiles para nosotros, como por ejemplo, los plots de Ehrenfest y valores esperados.

Nos hubiera gustado dar una opción de perder algo de precisión pero ejecutar el programa mucho más rápido guardando una única vez por cada par (x, t) los valores de la función de onda y ejecutar todo el cálculo numéricamente sobre éstos valores, pero por cuestión de tiempo y dificultad a la hora de leer el código hemos decidido dejarlo aparte. Para acabar, dejamos aquí escrito todo el código, aunque es recomendable leerlo directamente desde nuestro IDE (PyCharm por ejemplo) e ir experimentando con él.

Muchas gracias por el esfuerzo y la calidad de ésta práctica. Hemos disfrutado implementándola y lo hemos hecho lo mejor que hemos podido.

```
# coding=utf-8
import math
import cmath
import numpy as np
# from scipy import constants
from scipy import integrate
from scipy.misc import derivative
import matplotlib.pylab as plt
import matplotlib.animation as animation
from matplotlib.patches import Ellipse
import sys
from altres_funcions import *

'''Constants globals.'''
HBAR = 1
M = 1
K = 1
'''Per fer-ho anar tot més ràpit, es recomanable baixar la tolerància si no volem massa precissió'''
TOL = 1e-4

'''Energía per cada n'''
def E_n(n, omega):
    return HBAR * omega * (float(n) + 1/2)

'''Part temporal de la solució de l'equació d'Schrodinger depenent del temps.'''
def exp_t(E_n,t):
    return cmath.exp(-1j*E_n*t/HBAR)

'''Solucions estàcionaries de l'equació d'Schrodinger.'''
def phi_n(x, n, a0):
    coeff = np.zeros((n+1,))
    coeff[n] = 1
    fn = 1/( math.pi**(1./4.) * math.sqrt(2**n*math.factorial(n)*a0) )
    xbar = x/a0
    fx = math.exp(-xbar**2/2) * np.polynomial.hermite.hermval(x=xbar, c=coeff)
    return fn * fx

'''Funció secció 4 dels estats de mínima indeterminació.'''
def psi_squeezed(x, p0=0, x0=0, m=M, k=K, xi=0):
    omega = math.sqrt(k/m)
    delta_x = math.sqrt(HBAR/(2.*m*omega))*math.exp(-xi)
    return cmath.exp(1j*p0*x/HBAR) * math.exp(-(x-x0)**2/(4.*delta_x**2))

''' Classe Estat. Aquí emmagatzemo tota la informació rellevant del estat. Gran part de la funcionalitat es subordinarà
    a la classe FuncioOna. Podem definir un estat mitjançant els coeficients de les funcions pròpies 'coeffs' o
     mitjançant una funció qualsevol 'fx' (com fem a la secció 4). La funció fx rep el valor x com a primer argument i
     NO s'ha d'especificar a fx_args. La resta d'arguments de la funció (si n'hi han) vindràn en ordre a fx_args.'''
class Estat:
    def __init__(self, coeffs=None, m=M, k=K, fx = None, fx_args=()):
        assert (coeffs is not None or fx is not None)
        self.m = m
        self.k = k
        self.coeffs = coeffs
        if(coeffs is not None):
            self.coeffs = coeffs / np.linalg.norm(coeffs)
        '''Dintre d'estat guardem la funció d'ona.'''
        self.ona = FuncioOna(coeffs=self.coeffs,m=self.m,k=self.k,fx=fx,fx_args=fx_args)
        if(fx is not None):
            self.ona.update_coeffs()
```

```
            self.ona.fx = None
            self.coeffs = self.ona.coeffs


    '''Fer Kick implica donar-li un impuls p0 al moment emmagatzemat a la funció d'ona. Posteriorment recalculem la
    descomposició de la ona en les funcions pròpies.'''
    def kick(self,p0):
        self.ona.p0 = p0
        self.ona.update_coeffs()
        self.ona.p0 = 0
        self.coeffs = self.ona.coeffs

    '''Traslladar implica aplicar un desplaçament a la x0 emmagatzemada a la funció d'ona. Posteriorment recalculem
    la descomposició de la ona en les funcions pròpies.'''
    def traslacio(self,x0):
        self.ona.x0 = x0
        self.ona.update_coeffs()
        self.ona.x0 = 0
        self.coeffs = self.ona.coeffs

    '''Traslació i kick simultanis.'''
    def traslacio_kick(self,x0,p0):
        self.ona.x0 = x0
        self.ona.p0 = p0
        self.ona.update_coeffs()
        self.ona.x0 = 0
        self.ona.p0 = 0
        self.coeffs = self.ona.coeffs

    '''Valor esperat'''
    def valor_esperat(self, t, operator):
        return self.ona.expected_value(operator,t)

    '''Desviació estàndard'''
    def std(self, t, operator):
        assert(operator.lower() in ['x','p'])
        return math.sqrt(self.ona.expected_value(operator+'2',t) - self.ona.expected_value(operator,t)**2)

''' Classe Funció d'ona. Aquí avaluem la funció d'ona i els seus valors esperats i fem els plots. La classe Estat
s'encarrega de gestionar els passos generals a seguir i aquí es fan els càlculs. De la mateixa manera que hem dit a
Estat, podem partir dels coeficients de les funcions pròpies o d'una funció qualsevol de x 'fx'.'''
class FuncioOna:
    ''' Inicialització de la classe. Guardem les constants i els coeficients cn. '''
    def __init__(self, coeffs=None, m=M, k=K, fx=None, fx_args=()):
        assert(coeffs is not None or fx is not None)
        self.coeffs = coeffs
        self.fx = fx
        self.fx_args = fx_args
        '''Si partim d'una funció fx, la necessitem normalitzada.'''
        if(self.fx is not None):
            abs_fx_2 = lambda x: abs(self.fx(x, *self.fx_args))**2
            self.normaliztion_factor = math.sqrt(1./integrate.quad(func=abs_fx_2,a=-np.inf,b=np.inf)[0])
            print('[Normalització] Constant de normalització C: ' + str(self.normaliztion_factor))
        self.x0 = 0
        self.p0 = 0
        self.m = m
        self.k = k
        self.omega = math.sqrt(k/m)
        self.a0 = math.sqrt(HBAR/(m*self.omega))

    ''' Funcionalitat Important. Avaluem la ona als diferents punts (sense fer ni kick ni translació).'''
    def eval0(self,x,t):
        N = np.size(self.coeffs)

        return np.sum(np.array([self.coeffs[n]*phi_n(x=x,n=n,a0=self.a0) * exp_t(E_n=E_n(n,self.omega),t=t)
                                for n in range(N)]))

    '''Apliquem el Kick.'''
    def eval1(self,x,t):
        return cmath.exp(-1j*self.p0*x/HBAR) * self.eval0(x,t)

    '''Apliquem la Traslació'''
    def eval2(self,x,t):
```

```
    '''De forma pura hauriem d'aproximar l'operador de traslació tal que així:'''
    # '''Defineixo aquesta funció auxiliar perquè només volem calcular la parcial respecte de x.'''
    # def fx(x):
    #     return self.eval1(x,t)
    # '''Aproximació Taylor de grau 1 de U(x0)*f(x,t) = exp(-i*x0*p/hbar)*f(x,t). Això sol serveix per x molt petites,
    #   en un cas real hauriem de aproximar amb més graus.'''
    # return fx(x) - self.x0*sp.misc.derivative(fx,x,dx=1e-18)
    '''Però, com que sabem que U(x0)*f(x,t) = f(x-x0,t):'''
    return self.eval1(x - self.x0, t)

'''Avaluem la funció. En cas que haguem definit la funció amb coeficients, retornem eval2 que és la funció amb el Kick
   i la Traslació (si n'hi han). En cas contrari, si partim d'una funció, com a la secció 4, tornem el valor de la
   funció normalitzada amb els seus arguments.'''
def eval(self,x,t):
    if(self.fx is None):
        return self.eval2(x,t)
    else:
        return self.normaliztion_factor * self.fx(x, *self.fx_args)

'''Actualitzem els coeficients calculant la integral de phi_n'(x)*ona(x,0).'''
def update_coeffs(self):
    '''Definim els integrands'''
    integrand_real = lambda x, n: (np.conj(phi_n(x=x,n=n,a0=self.a0)) * self.eval(x=x,t=0)).real
    integrand_imag = lambda x, n: (np.conj(phi_n(x=x,n=n,a0=self.a0)) * self.eval(x=x,t=0)).imag

    '''Calculem els coeficients cn fins que la suma de les probabilitats excedeix 0.99'''
    accum_prob = 0
    n=0
    coeffs_aux = []
    while (accum_prob<(1.-TOL)):
        cn = integrate.quad(func=integrand_real, a=-np.inf, b=+np.inf, args=(n,))[0] + 1j * \
                                    integrate.quad(func=integrand_imag, a=-np.inf, b=+np.inf, args=(n,))[0]
        coeffs_aux = coeffs_aux + [cn]
        accum_prob = accum_prob + abs(cn)**2
        n = n + 1
    print(str(n) + ' coeficients amb precisió ' + str(accum_prob))
    self.coeffs = np.array(coeffs_aux)

'''Funcio per calcular els valors esperats. Per seleccionar el valor què vols s'ha de escollir un operador x, x2, p, p2.'''
def expected_value(self, operator, t):
    '''Definim les funcions per integrar'''
    func_prob = lambda x, t: abs(self.eval(x,t))**2
    func_x = lambda x,t: func_prob(x,t)*x
    func_x2 = lambda x,t: func_prob(x,t)*x**2
    func_p = lambda x,t: (np.conj(self.eval(x,t)) * HBAR/1j * derivative(func=self.eval,x0=x,dx=1e-2,n=1,
                                                                         args=(t,))).real
    func_p2 = lambda x,t: (np.conj(self.eval(x,t)) * (-HBAR**2) * derivative(func=self.eval,x0=x,dx=1e-2,n=2,
                                                                             args=(t,))).real

    '''Calculem els valors esperats segons l'operador.'''
    if(operator.lower()=='x'):
        return integrate.quad(func=func_x, a=-np.inf, b=+np.inf, args=(t,))[0]
    elif(operator.lower()=='x2'):
        return integrate.quad(func=func_x2, a=-np.inf, b=+np.inf, args=(t,))[0]
    elif (operator.lower() == 'p'):
        return integrate.quad(func=func_p, a=-np.inf, b=+np.inf, args=(t,))[0]
    elif (operator.lower() == 'p2'):
        return integrate.quad(func=func_p2, a=-np.inf, b=+np.inf, args=(t,))[0]
    else:
        sys.exit('El valor de operador ha de ser un string d\'aquests: \'x\', \'x2\', \'p\', \'p2\'')



''' Dibuixem la ona en un rang de X i de T. Aprofitem que només hem de reescriure les línies i no els eixos i tota
 la resta de la figura per fer-ho de forma eficient amb blit.'''
def plot(self, x0=-10, xf=10, t0=0, tf=10, nx=480, nt=100):
    '''Valors de les X i T i valors inicials de la funció d'ona Y.'''
    X = np.linspace(x0, xf, nx)
    T = np.linspace(t0, tf, nt)
    V = 1./2.*self.k*X**2
    prefix = '[Funció Ona] Calculant valors de l\'ona i valors esperats:'
    sufix = 'Acabat'
    print_progress(0, 5, prefix=prefix, suffix=sufix, bar_length=50)
```

```
'''Calculem tots els valors i mostrem el progrés per pantalla'''
Y = [np.array([self.eval(x=x, t=t) for x in X]) for t in T]
print_progress(1, 5, prefix=prefix, suffix=sufix, bar_length=50)

EXP_VAL_X = np.array([self.expected_value(operator='x', t=t) for t in T]) / (math.sqrt(self.m*self.omega))
print_progress(2, 5, prefix=prefix, suffix=sufix, bar_length=50)

STD_X = np.sqrt(np.array([self.expected_value(operator='x2', t=t) for t in T]) - EXP_VAL_X ** 2) / \
        (math.sqrt(self.m*self.omega))
print_progress(3, 5, prefix=prefix, suffix=sufix, bar_length=50)

EXP_VAL_P = np.array([self.expected_value(operator='p', t=t) for t in T]) * (math.sqrt(self.m*self.omega))
print_progress(4, 5, prefix=prefix, suffix=sufix, bar_length=50)

STD_P = np.sqrt(np.array([self.expected_value(operator='p2', t=t) for t in T]) - EXP_VAL_P ** 2) * \
        (math.sqrt(self.m*self.omega))
print_progress(5, 5, prefix=prefix, suffix=sufix, bar_length=50)

'''Definició/Inicialització de tots els plots a les 2 differents subfigures. 1a: Plot XY, 2a: Plot XP.'''
fig, (ax1, ax2) = plt.subplots(nrows=1,ncols=2)
fig.set_size_inches(w=17,h=16./9.*17,forward=True)
ax1.set_xlim(np.min(X), np.max(X))
ylim = 1.25*np.max([abs(np.min(Y[0])), abs(np.max(Y[0]))])
ax1.set_ylim(-ylim, ylim)

'''Inicialització plot XY'''
ax1.grid(True, which='both')
linia_real = ax1.plot(X, Y[0].real, 'b', label=r'$Re[\psi(x,t)]$', animated=True)[0]
linia_imag = ax1.plot(X, Y[0].imag, 'r', label=r'$Im[\psi(x,t)]$', animated=True)[0]
linia_prob = ax1.plot(X, abs(Y[0])**2, 'y', label=r'$|\psi(x,t)|^2$', animated=True)[0]
linia_exp = ax1.plot([EXP_VAL_X[0],EXP_VAL_X[0]], [-ylim,ylim], 'k', label=r'$\langle x \rangle_\psi(t)$',
                     animated=True)[0]
errobj = ax1.errorbar(EXP_VAL_X[0], ylim/4., color='k', xerr=STD_X[0], yerr=0, label=r'$\Delta x(t)$',
                      animated=True)
linia_pot = ax1.plot(X, V-ylim, 'g', label=r'$V(x)$', animated=False)[0]
handles, labels = ax1.get_legend_handles_labels()
ax1.legend(handles, labels)
lines = [linia_real, linia_imag, linia_prob, linia_exp, linia_pot]

'''Inicialització plot XP'''
ax2.grid(True, which='both')
ax2.axhline(y=0, color='k')
ax2.axvline(x=0, color='k')
xlim_xp = max(abs(np.min(EXP_VAL_X-STD_X)), abs(np.max(EXP_VAL_X+STD_X)))*1.25
ylim_xp = max(abs(np.min(EXP_VAL_P-STD_P)), abs(np.max(EXP_VAL_P+STD_P)))*1.25
lim_xp = max(xlim_xp,ylim_xp)
ax2.set_xlim(-lim_xp, lim_xp)
ax2.set_ylim(-lim_xp, lim_xp)
errobj_xp = ax2.errorbar(EXP_VAL_X[0], EXP_VAL_P[0], color='r', xerr=STD_X[0], yerr=STD_P[0],
                     label=r'$(\Delta x(t),\Delta p(t))$',
                     animated=True)
ellipse = Ellipse(xy=(EXP_VAL_X[0], EXP_VAL_P[0]), width=2. * STD_X[0], height=2. * STD_P[0], color='c',
                  alpha=0.5)
heisenberg = ax2.text(xlim_xp*0.6, -ylim_xp*0.9, r'$\Delta x\Delta p=$' + "{0:.2f}".format(STD_X[0]*STD_P[0]),
                      fontsize=12)
ax2.add_patch(ellipse)
handles, labels = ax2.get_legend_handles_labels()
ax2.legend(handles, labels)



'''Funció auxiliar per computar la animació. Avalua per cada temps T la funció d'ona Y als X donats,
el seu modul al quadrat, els valors esperats i std al plot XY i al XP....'''
def animate(n):
    '''Plot XY'''
    lines[0].set_ydata(Y[n].real)
    lines[1].set_ydata(Y[n].imag)
    lines[2].set_ydata(abs(Y[n])**2)
    lines[3].set_xdata([EXP_VAL_X[n],EXP_VAL_X[n]])
    lines_err = adjust_err_bar(errobj=errobj,x=EXP_VAL_X[n],y=ylim/4.,x_error=STD_X[n],y_error=0)
    '''Plot XP'''
    # patches = [ax2.add_patch(Ellipse(xy=(EXP_VAL_X[n],EXP_VAL_P[n]), width=2.*STD_X[n], height=2.*STD_P[n],
```

```
            # color='c', alpha=0.5))]
            ellipse.center = (EXP_VAL_X[n], EXP_VAL_P[n])
            ellipse.width = 2.*STD_X[n]
            ellipse.height = 2.*STD_P[n]
            lines_err_xp = adjust_err_bar(errobj=errobj_xp,x=EXP_VAL_X[n],y=EXP_VAL_P[n],x_error=STD_X[n],
                                          y_error=STD_P[n])
            heisenberg.set_text(r'$\Delta x\Delta p=$' + "{0:.2f}".format(STD_X[n]*STD_P[n]))
            '''Tornem tot el que volem actualitzar. La resta de la figura es mantindrà intacta per estalviar recursos.'''
            return lines + lines_err + lines_err_xp + [ellipse] + [heisenberg]

        '''Animació eficient.'''
        ani = animation.FuncAnimation(fig, animate, range(len(T)),
                                      interval=100, blit=True, repeat=True)
        plt.show()
        return ani, plt


'''Main per fer petites proves'''
if __name__ == '__main__':
    xi = -10
    xf = 10
    ti = 0
    tf = 10
    nx = 480
    nt = 100

    '''Per començar, definim la nostra funció inicial amb els coeficients. Podem escollir altres coeficients també,
    per exemple [1,1] per la combinació 1/sqrt(2)*(psi_0 + psi_1).'''
    coeffs = np.array([1])
    estat = Estat(coeffs=coeffs, m=M, k=K)
    #
    '''Animació de l'evolució de la funció d'ona. Es poden guardar les animacions fàcilment com un video amb
    ani.save('nom_fitxer').'''
    ani0, _ = estat.ona.plot(x0=xi,xf=xf,t0=ti,tf=tf,nx=nx,nt=nt)
    # ani0.save('ona0.mp4',bitrate=6500)

    '''Aplicar un operador es tan senzill com fer estat.operador(valor0).'''
    # estat.traslacio(x0=3)
    # ani_traslacio, _ = estat.ona.plot(x0=xi,xf=xf,t0=ti,tf=tf,nx=nx,nt=nt)
    # ani_traslacio.save('ona_trasl.mp4', bitrate=6500)
    # plot_ehrenfest(estat, t0=ti, tf=tf, nt=nt)
    estat.kick(p0=3)
    ani, _ = estat.ona.plot(x0=xi,xf=xf,t0=ti,tf=tf,nx=nx,nt=nt)
    # ani.save('ona_kick.mp4', bitrate=6500)
    '''Podem, a més del propi plot de l'evolució de l'ona al espai habitual i al espai x-p, imprimir
    la comprovació del teorema d'Ehrenfest, imprimir els valors esperats i les indeterminacions, i imprimir també
    el valor de la incertesa de Heisenberg (delta_x*delta_p).'''
    plot_ehrenfest(estat, t0=ti, tf=tf, nt=nt)
    plot_valoresp(estat, t0=ti, tf=tf, nt=nt)
    plot_heisenberg(estat, t0=ti, tf=tf, nt=nt)

    '''Podem fer una traslació i un kick simultanis també.'''
    # estat.traslacio_kick(x0=3,p0=3)
    # ani, _ = estat.ona.plot(x0=xi, xf=xf, t0=ti, tf=tf, nx=nx, nt=nt)
    # ani.save('ona_kick_traslacio.mp4', bitrate=6500)
    # plot_ehrenfest(estat, t0=ti, tf=tf, nt=nt)

    '''Si no coneixem o no necessitem els coeficients inicials (com en el cas de la secció 4), podem definir l'estat
    a partir d'una funció fx(x). Opcionalment, es poden enviar arguments extra a la funció (la x ha d'anar al inici de
    fx(x,*args) SEMPRE! i NO la enviem aquí, es tracta internament).'''
    # fx_args = (p0,x0,m,k,xi) en aquest ordre!!
    (p0,x0,m,k,xi_scale) = (2, 0, M, K, 1)
    estat = Estat(m=M,k=K,fx=psi_squeezed,fx_args=(p0,x0,m,k,xi_scale))
    # ani, _ = estat.ona.plot(x0=xi,xf=xf,t0=ti,tf=tf,nx=nx,nt=nt)
    # plot_ehrenfest(estat, t0=ti, tf=tf, nt=nt)
    # plot_valoresp(estat, t0=ti, tf=tf, nt=nt)
    plot_heisenberg(estat, t0=ti, tf=tf, nt=nt)
    # ani.save('estat_squeezed_kick_xi0_heisenberg.mp4',bitrate=6500)



    print('Sortida Correcta')
```

5.2. Otras funciones

```
# coding=utf-8
import numpy as np
from scipy.misc import derivative
import matplotlib.pyplot as plt
import sys

'''Funció per actualitzar la línia d'error.'''
def adjust_err_bar(errobj, x, y, x_error, y_error):
    ln, (errx_top, errx_bot, erry_top, erry_bot), (barsx, barsy) = errobj

    if(not isinstance(x, np.ndarray)):
        x_base = np.array([x])
    else:
        x_base = x
    if(not isinstance(y, np.ndarray)):
        y_base = np.array([y])
    else:
        y_base = y

    ln.set_data(x_base, y_base)

    xerr_top = x_base + x_error
    xerr_bot = x_base - x_error
    yerr_top = y_base + y_error
    yerr_bot = y_base - y_error

    errx_top.set_xdata(xerr_top)
    errx_bot.set_xdata(xerr_bot)
    errx_top.set_ydata(y_base)
    errx_bot.set_ydata(y_base)

    erry_top.set_xdata(x_base)
    erry_bot.set_xdata(x_base)
    erry_top.set_ydata(yerr_top)
    erry_bot.set_ydata(yerr_bot)

    new_segments_x = [np.array([[xt, y], [xb,y]]) for xt, xb, y in zip(xerr_top, xerr_bot, y_base)]
    new_segments_y = [np.array([[x, yt], [x,yb]]) for x, yt, yb in zip(x_base, yerr_top, yerr_bot)]
    # print(new_segments_x)
    barsx.set_segments(new_segments_x)
    barsy.set_segments(new_segments_y)
    return [ln, errx_top, errx_bot, erry_top, erry_bot, barsx, barsy]

'''Funció per mostrar progressos de còmput en una sola línia de terminal.'''
def print_progress(iteration, total, prefix = '', suffix = '', decimals = 1, bar_length = 100, fill = 'I'):
    '''
        iteration   : Iteració actual (int)
        total       : Total iteracions (int)
        prefix      : Frase per mostrar abans (str)
        suffix      : Frase per mostrar després (str)
        decimals    : Nombre de decimals (int)
        bar_length   : Longitud de la barra (int)
    '''
    percent = ("{0:." + str(decimals) + "f}").format(100 * (iteration / float(total)))
    filled_length = int(bar_length * iteration // total)
    bar = fill * filled_length + '-' * (bar_length - filled_length)
    sys.stdout.write('\r%s |%s| %s%s %s' % (prefix, bar, percent, '%', suffix)),
    if iteration == total:
        sys.stdout.write('\n')
    sys.stdout.flush()

'''Funció per mostrar les dos igualtats del teorema de Ehrenfest.'''
def plot_ehrenfest(estat, t0=0,tf=10, nt=100):
    m = estat.m
    k = estat.k
    T = np.linspace(t0, tf, nt)

    potential = lambda x: 1./2.*k*x**2

    prefix = '[Ehrenfest] Calculant valors esperats i derivades:'
    sufix = 'Acabat'
```

```
    print_progress(0, 5, prefix=prefix, suffix=sufix, bar_length=50)

    '''Calculem tots els valors'''
    X = np.array([estat.valor_esperat(t=t,operator='x') for t in T])
    print_progress(1, 5, prefix=prefix, suffix=sufix, bar_length=50)

    V = np.array([-derivative(func=potential, x0=x,dx=1e-2,n=1) for x in X])
    print_progress(2, 5, prefix=prefix, suffix=sufix, bar_length=50)

    P = np.array([estat.valor_esperat(t=t,operator='p') for t in T])
    print_progress(3, 5, prefix=prefix, suffix=sufix, bar_length=50)

    DX = m*np.array([derivative(func=estat.valor_esperat, x0=t, dx=1e-2, n=1, args=('x',)) for t in T])
    print_progress(4, 5, prefix=prefix, suffix=sufix, bar_length=50)

    DP = np.array([derivative(func=estat.valor_esperat, x0=t, dx=1e-2, n=1, args=('p',)) for t in T])
    print_progress(5, 5, prefix=prefix, suffix=sufix, bar_length=50)

    '''Definició i impressió dels plots.'''
    fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2)
    fig.set_size_inches(w=17, h=16. / 9. * 17, forward=True)

    ax1.plot(T, P, 'b', label=r'$\langle p \rangle(t)$')
    ax1.plot(T, DX, 'r', label=r'$m \frac{d \langle x \rangle}{dt}(t) $')
    handles1, labels1 = ax1.get_legend_handles_labels()
    ax1.legend(handles1, labels1)

    ax2.plot(T, DP, 'b', label=r'$ \frac{d \langle p \rangle}{dt}(t) $')
    ax2.plot(T, V, 'r', label=r'$ -\frac{d V(\langle x \rangle)}{dx}(t) $')
    handles2, labels2 = ax2.get_legend_handles_labels()
    ax2.legend(handles2, labels2)

    plt.show()

    return plt

'''Funció per hacer los plots de los valores esperados.'''
def plot_valoresp(estat, t0=0, tf=10, nt=100):
    m = estat.m
    k = estat.k
    T = np.linspace(t0, tf, nt)

    prefix = '[Valors esperats] Calculant valors esperats:'
    sufix = 'Acabat'
    print_progress(0, 7, prefix=prefix, suffix=sufix, bar_length=50)

    '''Calculem tots els valors'''
    X = np.array([estat.valor_esperat(t=t, operator='x') for t in T])
    print_progress(1, 7, prefix=prefix, suffix=sufix, bar_length=50)

    P = np.array([estat.valor_esperat(t=t, operator='p') for t in T])
    print_progress(2, 7, prefix=prefix, suffix=sufix, bar_length=50)

    DELTAX = np.array([estat.std(t=t, operator='x') for t in T])
    print_progress(3, 7, prefix=prefix, suffix=sufix, bar_length=50)

    DELTAP = np.array([estat.std(t=t, operator='p') for t in T])
    print_progress(4, 7, prefix=prefix, suffix=sufix, bar_length=50)

    V = k / 2. * np.array([estat.valor_esperat(t=t, operator='x2') for t in T])
    print_progress(5, 7, prefix=prefix, suffix=sufix, bar_length=50)

    K = 1. / (2 * m) * np.array([estat.valor_esperat(t=t, operator='p2') for t in T])
    print_progress(6, 7, prefix=prefix, suffix=sufix, bar_length=50)

    E = K + V
    print_progress(7, 7, prefix=prefix, suffix=sufix, bar_length=50)

    '''Definició i impressió dels plots.'''
    fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3)
    fig.set_size_inches(w=17, h=16. / 9. * 17, forward=True)

    ax1.plot(T, X, 'b', label=r'$\langle x \rangle(t)$')
```

```
    ax1.plot(T, DELTAX, 'r', label=r'$\Delta x (t) $')
    handles1, labels1 = ax1.get_legend_handles_labels()
    ax1.legend(handles1, labels1)

    ax2.plot(T, P, 'b', label=r'$\langle p \rangle(t)$')
    ax2.plot(T, DELTAP, 'r', label=r'$\Delta p (t) $')
    handles2, labels2 = ax2.get_legend_handles_labels()
    ax2.legend(handles2, labels2)

    ax3.plot(T, V, 'b', label=r'$\langle V \rangle(t)$')
    ax3.plot(T, K, 'r', label=r'$\langle K \rangle(t)$')
    ax3.plot(T, E, 'g', label=r'$\langle E \rangle(t)$')
    ax3.set_ylim(np.min(V) - 0.25 * np.min(V), np.max(E) + 0.25 * np.max(E))
    handles3, labels3 = ax3.get_legend_handles_labels()
    ax3.legend(handles3, labels3)

    plt.show()

    return plt

'''Funció per fer els plot de deltax(t)*deltap(t).'''
def plot_heisenberg(estat, t0=0, tf=10, nt=100):
    T = np.linspace(t0, tf, nt)

    prefix = '[Heisenberg] Calculant les incerteses:'
    sufix = 'Acabat'
    print_progress(0, 2, prefix=prefix, suffix=sufix, bar_length=50)

    '''Calculem tots els valors'''
    DELTAX = np.array([estat.std(t=t, operator='x') for t in T])
    print_progress(1, 2, prefix=prefix, suffix=sufix, bar_length=50)

    DELTAP = np.array([estat.std(t=t, operator='p') for t in T])
    print_progress(2, 2, prefix=prefix, suffix=sufix, bar_length=50)

    '''Definició i impressió dels plots.'''
    fig, (ax1) = plt.subplots(nrows=1, ncols=1)
    fig.set_size_inches(w=17, h=16. / 9. * 17, forward=True)

    ax1.plot(T, np.multiply(DELTAX,DELTAP), 'b', label=r'$\Delta x(t) \Delta p(t)$')
    handles1, labels1 = ax1.get_legend_handles_labels()
    ax1.legend(handles1, labels1)

    plt.show()

    return plt
```