EVOLUCIÓN TEMPORAL DE LA FDP DE UNA PARTÍCULA PARA DOS POTENCIALES UNIDIMENSIONALES

Juan Felipe Zapata
Kevin Zapata

Física computacional II
Instituto de Física
Universidad de Antioquia

Contenidos

La ecuación de Schrodinger independiente del tiempo

En mecánica cuántica todos los problemas se pueden reducir a dos tipos [1]:

- **Problemas de estructura**
  Se resuelven recurriendo a la ecuación de Schrodinger independiente del tiempo, se hallan autovalores y autofunciones físicamente aceptables para un sistema.

$$\hat{H} |\Psi\rangle = E |\Psi\rangle \tag{1}$$

La ecuación de Schrodinger dependiente del tiempo

- **Problemas dinámicos**

  Se busca conocer la evolución temporal del estado del sistema físico, teniendo la información de su estado inicial; para esto se resuelve la ecuación de Schrodinger dependiente del tiempo:

$$\left[-\frac{1}{2}\frac{\partial^2}{\partial x^2} + V(x)\right]\Psi(x,t) = \hat{H}\Psi(x,t) = i\frac{\partial}{\partial t}\Psi(x,t) \quad (2)$$

Nuestro proyecto consiste en solucionar esta ecuacion. La cual, se puede solucionar con matrices (metodos implicitos) o a traves de metodos recursivos (metodos explicitos). Nosotros usamos un metodo explicito que se desarrolla en [2], [3] y [4]. El reto de este problema consiste en asegurarse que la densidad de probabilidad se conserve mas o menos constanste para todos los tiempos.

Operador evolución

Nuestro objetivo es por lo tanto solucionar la ecuación de Schrodinger dependiente del tiempo para un conjunto discreto de tiempos, para esto se utiliza el operador evolución temporal:

$$\hat{U}(\Delta t) = e^{-i\Delta t\hat{H}} \tag{3}$$

Este operador es solución de la ecuación de Schrodinger, Tal que:

$$\hat{U}(n\Delta t)\Psi(0) = \psi(x, t = n\Delta t) \tag{4}$$

Pasandonos a una notación equivalente tenemos:

$$\hat{U}^n(\Delta t)\psi^0 = \psi^n \tag{5}$$

Serie de Taylor

Juntando (3) con (5) obtenemos:

$$\psi^{n+1} = e^{-i\Delta t\hat{H}}\psi^n \quad ; \quad \psi^{n-1} = e^{i\Delta t\hat{H}}\psi^n \tag{6}$$

Si desarrollamos el operador de evolucion temporal y en series de potencias y aproximamos a primer orden:

$$\begin{aligned} e^{-i\Delta t\hat{H}} &\simeq \hat{1} - i\Delta t\hat{H} = \hat{1} - i\Delta t\left(-\frac{1}{2}\frac{\partial^2}{\partial x^2} + V(x)\right) \\ &= \hat{1} + \frac{i\Delta t}{2}\frac{\partial^2}{\partial x^2} - i\Delta t V(x) \end{aligned} \tag{7}$$

Diferencias finitas

Reemplazando (7) en (6):

$$\psi^{n+1} = [\hat{1} + \frac{i\Delta t}{2}\frac{\partial^2}{\partial x^2} - i\Delta t V(x)]\psi^n \tag{8}$$

Usando la aproximacion de diferencias finitas:

$$\left.\frac{\partial^2\psi}{\partial x^2}\right|_{x=x_j,t=t_n} = \frac{\psi^n_{j+1} + \psi^n_{j-1} - 2\psi^n_j}{\Delta x^2} \tag{9}$$

Donde se tuvo en cuenta que:

$$\psi^n_j = \psi(x = x_j, t = t_n) \tag{10}$$

Ecuacion final

Usando (9) y:

$$V(x = x_j) = V_j \tag{11}$$

Llegamos a la ecuacion:

$$e^{-i\Delta t\hat{H}}\psi_j^n = \psi_j^n + \frac{i}{2}\Delta t(\frac{\psi_{j+1}^n + \psi_{j-1}^n - 2\psi_j^n}{\Delta x^2}) - i\Delta t V_j \psi_j^n \tag{12}$$

Reorganizando terminos se obtine:

$$\psi_j^{n+1} = (1 - i\frac{\Delta t}{\Delta x^2} - i\Delta t V_j)\psi_j^n + i\frac{\Delta t}{2\Delta x^2}(\psi_{j+1}^n + \psi_{j-1}^n) \tag{13}$$

Partes reales e imaginarias

Si se define:

$$\alpha = \frac{\Delta t}{2\Delta x^2} \tag{14}$$

Y e tiene en cuenta que:

$$\psi_j^n = R_j^n + iI_j^n; \psi_j^{n+1} = R_j^{n+1} + iI_j^{n+1}; \tag{15}$$

Se obtienen las siguientes ecuaciones para la parte real e imaginaria

$$R_j^{n+1} = R_j^n - 2\left\{\alpha\left[I_{i+1}^n + I_{i-1}^n\right] - 2\left[\alpha + V_i\Delta t\right] I_i^n\right\} \tag{16}$$

$$I_j^{n+1} = I_j^n + 2\left\{\alpha\left[R_{i+1}^n + R_{i-1}^n\right] - 2\left[\alpha + V_i\Delta t\right] R_i^n\right\} \tag{17}$$

Paquete Gaussiano

La probabilidad es:

$$\rho_j^n = I^2(t) + R\left(t + \frac{\Delta t}{2}\right) R\left(t - \frac{\Delta t}{2}\right) = R_j^{n+1} R_j^{n-1} + \left(I_j^n\right)^2 \tag{18}$$

Inicialmente supondremos una función de onda que se describe como un paquete Gaussiano (electrón) con una rapidez inicial asociada a k_0 y centrado (localizado) en x_0:

$$\psi(x, t = 0) = exp\left[-\frac{1}{2}\left(\frac{x - x_0}{\sigma_0}\right)^2\right] e^{ik_0 x} \tag{19}$$

Definición de la clase


```
#ifndef PARTICULA_H
#define PARTICULA_H

class Particula{

        public:
                Particula(double a, double b, double x, double k0, double dx, double dt, double sigma);
                ~Particula();
                void psi(double(*V)(double));
                double geta();
                double getb();
                double getdx();
        private:
                double a,b, x, k0, dx, dt, sigma;
};


#endif

```

Interfaz de la clase


```cpp
#include <cmath>
#include "potencial.h"
#include <iostream>
#include <fstream>

using namespace std;

Particula::Particula(double a, double b, double x, double k0, double dx, double dt, double sigma){
        this->a = a;
        this->b = b;
        this->x = x;
        this->k0 = k0;
        this->dx = dx;
        this->dt = dt;
        this->sigma = sigma;
}


Particula::~Particula(){
}

double Particula::geta(){
        return a;
}


double Particula::getb(){
        return b;
}


double Particula::getdx(){
        return dx;
}
```

Interfaz de la clase


```
36 void Particula::psi(double(*V)(double)){
37
38         ofstream archivo;
39         ofstream archivo1;
40         ofstream archivo2;
41         ofstream archivo3;
42         ofstream archivo4;
43
44         archivo.open("programa.txt",ios::out);
45         archivo1.open("programa1.txt",ios::out);
46         archivo2.open("programa2.txt",ios::out);
47         archivo3.open("programa3.txt",ios::out);
48         archivo4.open("programa4.txt",ios::out);
49
50         archivo3<<geta()<<endl;
51         archivo3<<getb()<<endl;
52         archivo3<<getdx()<<endl;
53
54         int max = (b-a)/dx;
55         int i, n;
56         int cont=0;
57
58         double psr[max][2]; //parte real
59         double psi[max][2]; //parte imaginaria
60         double p2[max]; //probabilidad
61         double v[max]; //potencial
62
63
64         for(i=0; i<max; i++){
65                 psr[i][0]=exp(-0.5*pow((x/0.5),2))*cos(k0*x);
66                 psi[i][0]=exp(-0.5*pow((x/0.5),2))*sin(k0*x);
67                 v[i]=V(x);
68                 archivo4<<v[i]<<endl;
69                 x+=dx;
70         }
71
72         p2[0]=0;
73         p2[max-1]=0;
74
75
76         for (n=0 ; n < 10000 ; n++){
77
78                 for(i=1;i<max-1;i++){
79                         psr[i][1] = psr[i][0]-dt*(psi[i+1][0]+psi[i-1][0]-2.0*psi[i][0])/(dx*dx)+dt*v[i]*psi[i][0];
80                         p2[i]=psr[i][0]*psr[i][1]+psi[i][0]*psi[i][0]; //para tiempos semi-enteros
81                 }
82
83
84                 for(i=1;i<max-1;i++) {
85                         psi[i][1]=psi[i][0]+dt*(psr[i+1][1]+psr[i-1][1]-2.0*psr[i][1])/(dx*dx)-dt*v[i]*psr[i][1];
86                 }
87
```

Interfaz de la clase


```
88
89                  if(n==0 || n%10==0){
90                          for(i=0; i<max; i++){
91                                  archivo<<p2[i]<<endl;
92                                  archivo1<<psr[i][1]<<endl;
93                                  archivo2<<psi[i][1]<<endl;
94                          }
95                          cont++;
96                  }
97
98                  for(i=0;i<max;i++){
99                          psi[i][0]=psi[i][1];
100                         psr[i][0]=psr[i][1];
101                 }
102         }
103
104         archivo3<<cont<<endl;
105         archivo3.close();
106         archivo.close();
107         archivo1.close();
108         archivo2.close();
109         archivo4.close();
110
111
112 }
```

Implementación de la clase


```cpp
#include "potencial_inter.cpp"
#include <cmath>
#include <fstream>

using namespace std;

double V(double x);

int main(){
        const double pi = 3.14159265358979323846;
        double a=0;
        double b=15;
        double x=-7.5;
        double k0=3.0*pi;  //armónico
        //double k0=17.0*pi; //pozo
        double dx = 0.02;
        double sigma = 0.5;
        double dt=dx*dx/4; //armónico
        //double dt=dx*dx/2; //pozo

        Particula arm( a,  b,  x,  k0,  dx, dt,  sigma);
        arm.psi(&V);
        arm.~Particula();
        return 0;
}

double V(double x){
        return 5*pow(x,2); //potencial de oscilador armónico
}


/*double V(double x){
        return 0; //potencial de pozo infinito.
}*/
```

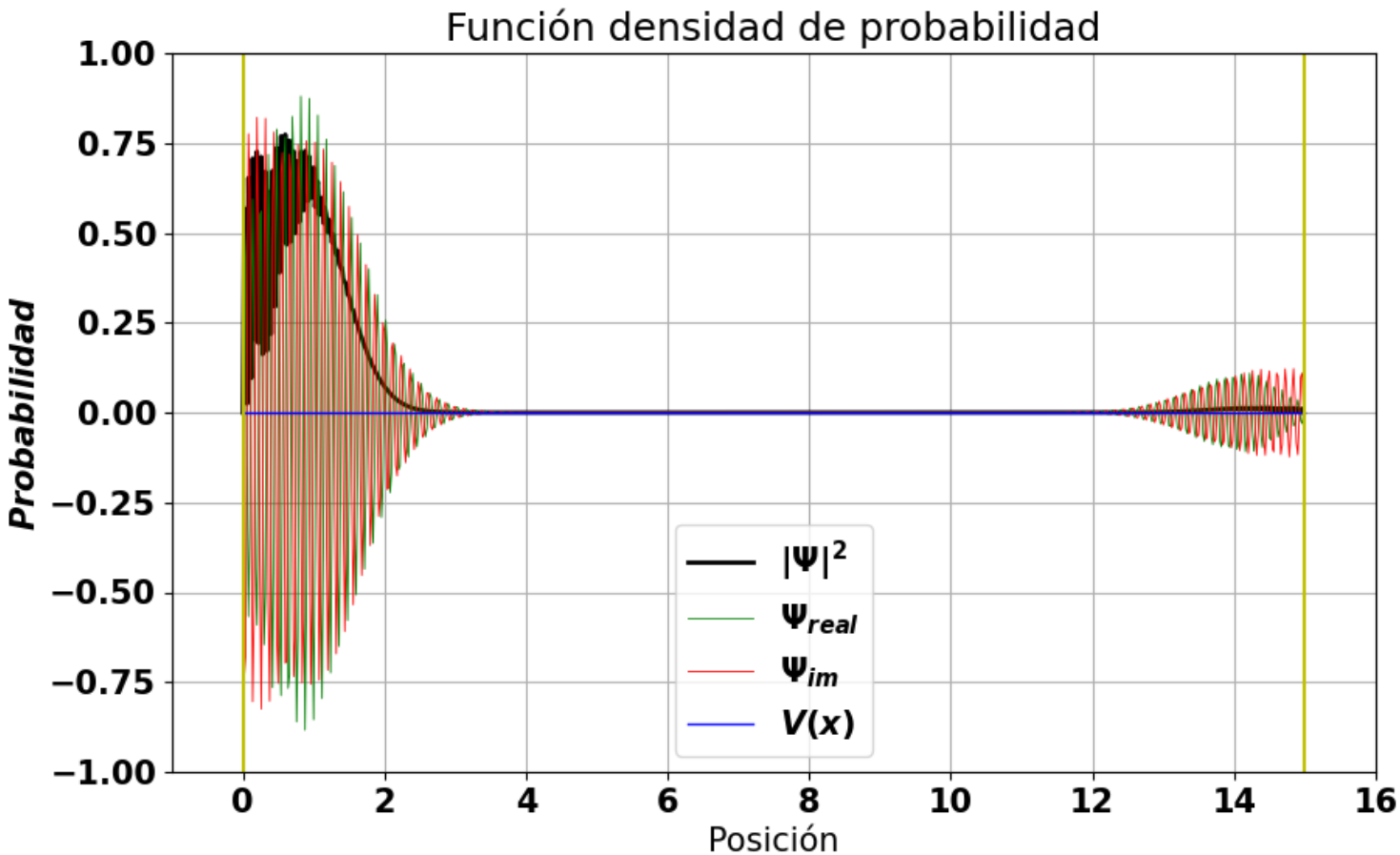
Animación


```
%matplotlib notebook
import numpy as np
from numpy import *
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.animation as animation
from IPython import display

from numpy import *
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import matplotlib
plt.rcParams['figure.figsize'] = 10, 6 #ancho, alto
font = {'weight' : 'bold',
        'size'   : 15}

matplotlib.rc('font', **font)
################################3
datos = np.loadtxt("programa.txt")
datos1 = np.loadtxt("programa1.txt")
datos2 = np.loadtxt("programa2.txt")
datos3 = np.loadtxt("programa3.txt")
y = np.loadtxt("programa4.txt")
################################3
a=datos3[0]
b=datos3[1]
dx=datos3[2]
N=datos3[3]
##############################
Datos=np.split(datos, N)
Datos1=np.split(datos1, N)
Datos2=np.split(datos2, N)
######################
x=np.arange(a,b,dx)
```

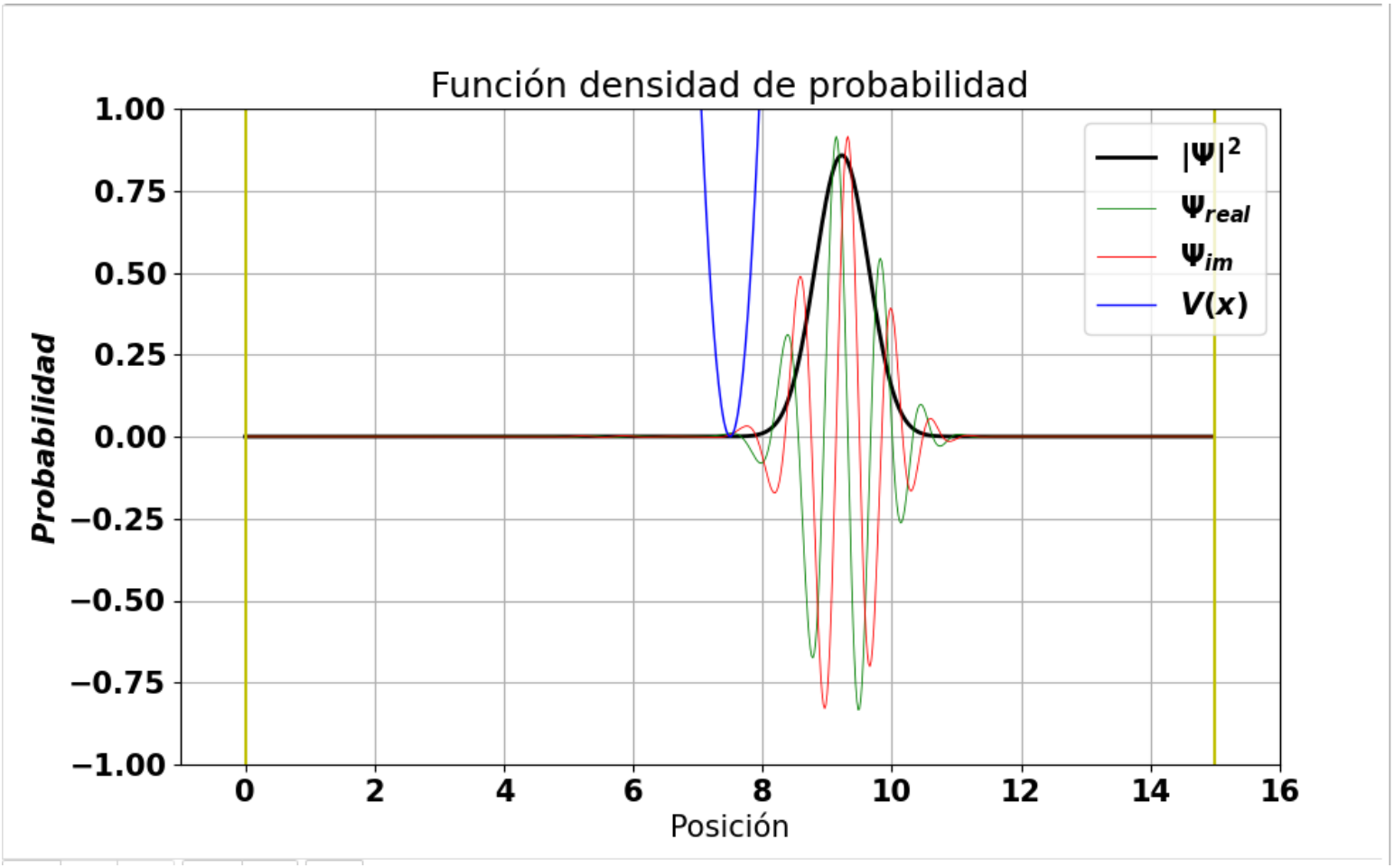
Animación


```
##############################
%matplotlib notebook
#Gráfica
fig1=plt.figure()
ax=fig1.gca()

#Función de actualización

def actualizar1(i):
    ax.clear()
    ax.plot(x,Datos[i], c='k',lw=2, label='$|\Psi|^2$')
    ax.plot(x,Datos1[i],c='g', lw=0.6, label='$\Psi_{real}$')
    ax.plot(x,Datos2[i],c='r', lw=0.6, label='$\Psi_{im}$')
    ax.plot(x,y,c='b', lw=1, label='$V(x)$')
    plt.axvline(x =a, color = 'y')
    plt.axvline(x = b, color = 'y')
    plt.xlim(a-1,b+1) #para que no varíen los ejes
    plt.ylim(-1,1)
    plt.xlabel('Posición')
    plt.ylabel('$Probabilidad$')
    plt.title('Función densidad de probabilidad')
    plt.grid()
    plt.legend()
#Animación

ani1=animation.FuncAnimation(fig1,actualizar1,range(len(Datos)), interval=1, repeat=False)
#en donde está range se ponen los datos para meter en actualizar
#interval está en milisegundos, cada k milisegundos crea la gráfica


plt.show()
```

Pozo infinito de potencial



Potencial armónico simple



Referencias

[1] Campos, D., Romero, D. C. (1997). Fundamentos de física atómica y molecular. Univ. Nacional de Colombia.
[2] Landau, R. H., Páez, M. J., Bordeianu, C. C. (2007). Computational physics. Computer Languages, 7, 2-3.
[3] Askar, A. Cakmak, A. S. (1978). Explicit integration method for the time-dependent Schrodinger equation for collision problems. The Journal of Chemical Physics, 68(6), 2794. https://doi.org/10.1063/1.436072
[4] Visscher, P. B. (1991). A fast explicit algorithm for the time-dependent Schrodinger equation. Computers in Physics, 5(6), 596. https://doi.org/10.1063/1.168415