Resolución numérica de la ecuación de Schrödinger dependiente del tiempo

César de la Rosa Sobrino
(Fecha: 5 de noviembre de 2024)

En este trabajo, se aborda la resolución numérica de la ecuación de Schrödinger dependiente del tiempo, aplicando diferentes métodos de discretización para observar la evolución temporal de partículas en diversos potenciales.

I. INTRODUCCIÓN

La ecuación de Schrödinger dependiente del tiempo es fundamental en la mecánica cuántica, ya que describe cómo el estado cuántico de un sistema evoluciona a lo largo del tiempo. Resolver esta ecuación de forma numérica es crucial para simular sistemas complejos donde las soluciones analíticas no son posibles.

II. ECUACIONES EN DERIVADAS PARCIALES

a

III. PROPAGACIÓN DE UNA ONDA SINUSOIDAL

b

IV. ECUACIÓN DE SCHRÖDINGER DEPENDIENTE DEL TIEMPO

c

V. DISCRETIZACIÓN DE LA ECUACIÓN DE SCHRÖDINGER

d

VI. EVOLUCIÓN TEMPORAL DE UNA PARTÍCULA LIBRE

e

VII. EVOLUCIÓN TEMPORAL DE UNA PARTÍCULA VIAJERA

f

VIII. FUNCIONES DEL OSCILADOR ARMÓNICO

g

IX. SISTEMA DE ELECCIÓN PROPIA

h

X. CONCLUSIONES

i

Apéndice A: Código fuente

Listing 1. main.m

```
% main.m
clear; clc;

xmin = -10;
xmax = 10;
Nx = 1000;
tmax = 5;
Nt = 500;
dx = (xmax - xmin) / Nx;
dt = tmax / Nt;

x = linspace(xmin, xmax, Nx);
```

```
t = linspace(0, tmax, Nt);

sigma = 1;
p0 = 1;
V0 = 1;

psi = init_gaussian(x, sigma, p0);

[free_psi] = free_evol(psi, x, dx, dt, Nt);
[travel_psi] = travel_evol(psi, x, dx, dt, Nt, V0);
[oscilator_psi] = oscilator_evol(psi, x, dx, dt, Nt);

plot_system(x, t, free_psi, 'Partícula Libre');
plot_system(x, t, travel_psi, 'Partícula Viajera');
plot_system(x, t, oscilator_psi, 'Oscilador Armónico');
```

Listing 2. init_gaussian.m

```
% init_gaussian.m

function psi = init_gaussian(x, sigma, p0)
        A = 1;
        psi = A * exp(-0.5 * (x / sigma).^2) .* exp(1i * p0 * x);
end
```

Listing 3. free_evol.m

```
% free_evol.m

function [psi_evol] = free_evol(psi, x, dx, dt, Nt)
    for j = 1:Nt
        psi(2:end-1) = psi(2:end-1) + ...
            (1i * dt / (2 * dx^2)) * (psi(3:end) - 2 * psi(2:end-1) + psi(1:end-2));
        psi_evol(:, j) = psi;
    end
end
```

Listing 4. oscilator_evol.m

```
% oscilator_evol.m

function [psi_evol] = evolucion_oscilador(psi, x, dx, dt, Nt)
    V = 0.5 * x.^2;
    for j = 1:Nt
        psi(2:end-1) = psi(2:end-1) + ...
            (1i * dt / (2 * dx^2)) * (psi(3:end) - 2 * psi(2:end-1) + ...
                        psi(1:end-2)) - (1i * dt) * V(2:end-1) .* psi(2:end-1);
        psi_evol(:, j) = psi;
    end
end
```

Listing 5. travel_evol.m

```
% travel_evol.m

function [psi_evol] = evolucion_viajera(psi, x, dx, dt, Nt, V0)
    xb = 0;
    V = V0 * (x >= xb);
    for j = 1:Nt
        psi(2:end-1) = psi(2:end-1) + ...
            (1i * dt / (2 * dx^2)) * (psi(3:end) - 2 * psi(2:end-1) + ...
                        psi(1:end-2)) - (1i * dt) * V(2:end-1) .* psi(2:end-1);
        psi_evol(:, j) = psi;
    end
end
```