UNIVERSIDAD DE LOS ANDES, DEPARTAMENTO DE FÍSICA
FÍSICA ATÓMICA

Oscilador armónico

Juan Barbosa, 201325901
Febrero 23, 2017

$$\boxed{-\frac{\hbar^2}{2m}\nabla^2\Psi(x)+V(x)\Psi(x)=E\Psi(x)}$$

Para un oscilador armónico la energía potencial se escribe:

$$V(x)=\frac{1}{2}kx^2=\frac{1}{2}m\omega^2x^2 \tag{1}$$

La ecuación de Schrödinger se escribe de la forma:

$$\frac{d^2}{dx^2}\Psi(x)=\ddot{\Psi}(x)=\frac{2m}{\hbar^2}\left(\frac{1}{2}m\omega^2x^2-E\right)\Psi(x) \tag{2}$$

La solución a la ecuación anterior debe cumplir que para $x=\pm\infty$, $\Psi(x)=0$. Las soluciones analíticas usan los polinomios de Hermite.

$$\Psi_n(x)=\frac{1}{\sqrt{2^n n!}}\left(\frac{m\omega}{\pi\hbar}\right)^{1/4}e^{-\frac{m\omega x^2}{2\hbar}}H_n\left(\sqrt{\frac{m\omega}{\hbar}}x\right)\qquad\text{donde}\qquad H_n(x)=(-1)^n e^{x^2}\frac{d^n}{dx^n}e^{-x^2} \tag{3}$$

Usando el método de Euler para resolver numéricamente se obtiene:

$$\begin{aligned}\dot{\Psi}_n&=\dot{\Psi}_{n-1}+\ddot{\Psi}_{n-1}\Delta x\\ \Psi_n&=\Psi_{n-1}+\dot{\Psi}_n\Delta x\end{aligned} \tag{4}$$

El sistema de ecuaciones diferenciales es resuelto en C usando $N=100000$ puntos, $dx=0{,}0001$, para $\omega=0{,}4, 0{,}6, 0{,}8, 1{,}0$. Las unidades usadas son arbitrarias tales que $\hbar^2=m=1$. Posteriormente se usa un algoritmo en Python para graficar las soluciones.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int N = 100000;
double dx = 0.0001, E, omega;
double *psi;
double psi_prime;

double *solver();
int main(int argc, char **argv)
{
    double acceptance = 0.5, current = 100;
    psi = malloc(N*sizeof(double));
    int i = 0, j = 0;

    FILE *functions = fopen("functions.dat","w");
    FILE *energies = fopen("energies.dat", "w");
    FILE *omegas = fopen("omegas.dat", "w");
    omega = 0.4;
    for(omega; omega<=1.0; omega+=0.2)
    {
        for(j=0; j<7; j++)
        {
            current = 100;
            E = omega*(1.4+j) - 0.1;
            while (current > acceptance)
            {
                E += 0.001/omega;
                if (j%2 == 0)
                {
                    psi[0] = 0;
                    psi_prime = 1;
                }
                else
                {
                    psi[0] = 1;
                    psi_prime = 0;
                }
                psi = solver();
                current = fabs(psi[N*5/10-1]);
            }
            printf("%f %f\n", omega, E);
            for(i = 0; i<N; i += 10)
                {
                    fprintf(functions, "%f\n", psi[i]);
                }
                fprintf(energies, "%f\n", E);
        }
        fprintf(omegas, "%f\n", omega);
    }
        return 0;
}

double *solver()
{
    int i = 0;
    double x = 0, U = 0;
    for(i = 1; i < N; i++)
    {
        U = 0.5*pow(omega*x, 2);
        x = x + dx;
        psi_prime = psi_prime + 2*(U-E)*psi[i-1]*dx;
        psi[i] = psi[i-1] + psi_prime*dx;
    }
    return psi;
}
```

```
import numpy as np
import matplotlib.pyplot as plt

data = np.genfromtxt("functions.dat")
energies = np.genfromtxt("energies.dat")
omegas = np.genfromtxt("omegas.dat")

N = len(omegas)
n = 10000
dx = 0.001
x = np.linspace(0, (n-1)*dx, n)
data_split = np.split(data, N)
energies_split = np.split(energies, N)
answers = int(data_split[0].shape[0]/n)

fig, axes = plt.subplots(2, 2, sharex=True, sharey=True)
axes = axes.reshape(4)
for j in range(N):
    data = np.split(data_split[j], answers)
    for i in range(answers):
        p = axes[j].plot(x, data[i], label="E = %.3f"%energies_split[j][i], lw = 1)[0]
        if i%2 == 0:
            axes[j].plot(-x, -data[i], "--", lw = 1, c = p.get_color())
        else:
            axes[j].plot(-x, data[i], "--", lw = 1, c = p.get_color())
    axes[j].set_ylim(-2, 2)
    axes[j].set_xlim(-5, 5)
    axes[j].legend(loc = 1, fontsize=5)
    if j%2 == 0:
        axes[j].set_ylabel("$\Psi(x)$")
    if j > 1:
        axes[j].set_xlabel("$x$")
fig.tight_layout()
plt.savefig("plot.pdf")


fig = plt.figure()

for i in range(N):
    energies = energies_split[i]
    n = np.arange(1, answers + 1)
    A, B = np.polyfit(n, energies, 1)
    x = np.linspace(1, 7, 10)
    y = A*x + B
    p = plt.plot(n, energies, "o", label = "$\omega = %.1f$"%omegas[i])[0]
    plt.plot(x, y, "--", c=p.get_color())
    plt.text(i+1.2, i+1.5, "$E = %.2fn+%.2f$"%(A, B), color=p.get_color())

plt.xlabel("$n$")
plt.ylabel("$E$")
plt.legend()
plt.savefig("energies.pdf")
```

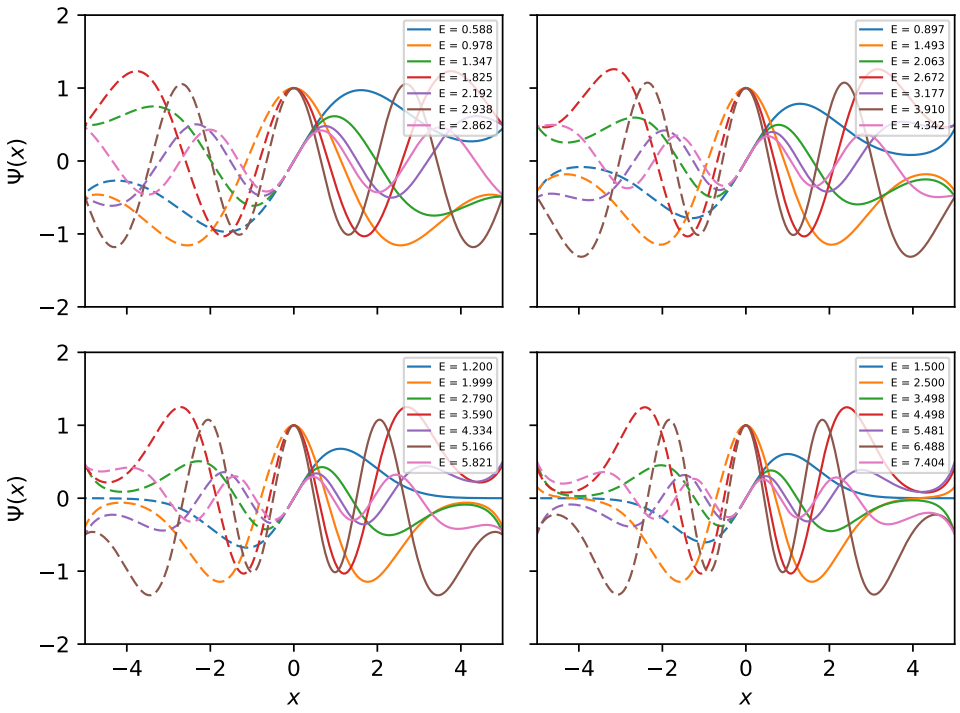

Figura 1: Funciones de onda con mejores comportamientos asistóticos.

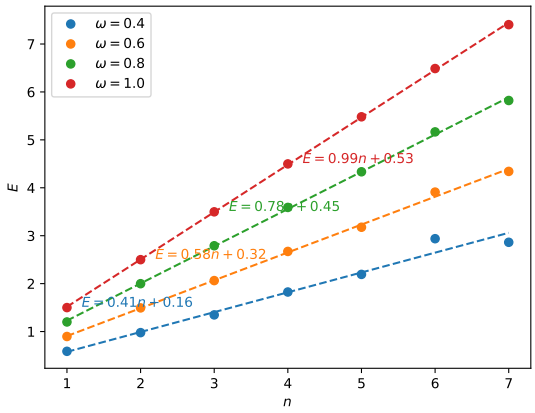

Figura 2: Energía en función de n, para distintos valores de ω.

En la **??** se muestra las funciones de onda para las cuales se considera convergencia en el algoritmo en C, las funciones son simuladas de $x = 0$ hasta $x = 10$, la línea punteada corresponde por simetría con $x < 0$. En la **??** se muestran los valores que adquiere la energía para distintos valores de n y ω. Al observar las regresiones lineales se puede establecer la siguiente relación.

$$E_{n\omega_i} \propto \omega_i n \qquad \Longrightarrow \qquad E_{n\omega} \propto \omega n \tag{5}$$