

## ***Algoritmos en pseudocódigo para mapear registros de datos BBDD de las estructuras de datos de algoritmos de láseres cuánticos:***

To automate and translate the provided scripts into a more structured form, such as pseudocode, we'll focus on key operations and equations related to the quantization of multimode light. This involves setting up the Hamiltonian, field operators, and related calculations. I'll break it down into a step-by-step pseudocode algorithm, including comments to explain each step.

### **Pseudocode for Multimode Quantum Optics**

#### **1. Define Constants and Variables**

```
// Define constants
hbar = Planck's constant divided by 2π
ωm = Array of mode frequencies
am = Array of annihilation operators for each mode
a†m = Array of creation operators for each mode

// Number of modes
num_modes = length(ωm)

// Initialize arrays for electric and magnetic fields
Em = Array of electric field operators for each mode
Bm = Array of magnetic field operators for each mode
```

#### **2. Compute the Total Hamiltonian**

```
// Initialize total Hamiltonian
H_total = 0

// Loop over each mode to accumulate Hamiltonian contributions
for m from 1 to num_modes do
    // Hamiltonian for the m-th mode
    H_m = hbar * ωm[m] * (a†m[m] * am[m] + 0.5)
    H_total = H_total + H_m
end for
```

### 3. Compute Field Operators

```
// Initialize field operators

for m from 1 to num_modes do
    // Electric field operator

    
$$E_m[m](r) = (E_m[m](r) * a + E_m[m](r) * a^\dagger) / 2$$


    // Magnetic field operator

    
$$B_m[m](r) = (\nabla \times E_m[m](r)) / (i * \omega_m[m])$$

end for
```

### 4. Normalize the Electric Field

```
// Define integration function for normalization

function integrate(field, volume):
    // Integration over volume
    result = integrate(field^2, volume)
    return result
end function

// Normalize electric field operators

for m from 1 to num_modes do
    normalization_constant = integrate( $E_m[m](r)$ , volume)
    if normalization_constant !=  $\hbar$  *  $\omega_m[m]$  then
        // Adjust  $E_m[m](r)$  if needed
    end if
end for
```

## 5. Calculate Vacuum Energy

```
// Initialize vacuum energy
E_vacuum = 0

// Loop over each mode to accumulate vacuum energy
for k from 1 to num_modes do
    E_vacuum = E_vacuum + (hbar * ωk[k] / 2)
end for
```

1. **Constants and Variables:** Set up constants such as Planck's constant and arrays for mode frequencies, annihilation, and creation operators.
2. **Hamiltonian Calculation:** Loop through each mode and calculate the Hamiltonian for that mode using the provided formula, summing these to get the total Hamiltonian.
3. **Field Operators:** Define the electric and magnetic field operators for each mode based on given equations. The electric field is normalized, and the magnetic field is derived from the electric field.
4. **Normalization:** Check the normalization condition for the electric field operators and adjust if necessary.
5. **Vacuum Energy:** Calculate the vacuum energy by summing up contributions from all modes.

### Implementation Notes

- In an actual programming language, you'd need to define specific methods for integration and matrix operations.
- For operators ( $a_m$ ,  $a_m^\dagger$ ), a symbolic computation library might be necessary to handle quantum mechanical operations.
- The integrate function is conceptual here; real implementation would depend on the numerical integration methods used.

Feel free to adapt this pseudocode according to the specific programming environment and libraries you are using.

## 1. Allowed Wave Vectors

For a cuboid with side lengths  $L_x$ ,  $L_y$ , and  $L_z$ , the allowed wave vectors are quantized due to boundary conditions. The wave vectors  $k_x$ ,  $k_y$ , and  $k_z$  are given by:

$$k_x = \frac{2\pi l_x}{L_x}$$

$$k_y = \frac{2\pi l_y}{L_y}$$

$$k_z = \frac{2\pi l_z}{L_z}$$

where  $l_x$ ,  $l_y$ , and  $l_z$  are integers (quantum numbers).

where  $l_x$ ,  $l_y$ , and  $l_z$  are integers (quantum numbers).

## 2. Wave Vector Magnitude

The magnitude of the wave vector  $\mathbf{k}$  is:

$$k = \sqrt{k_x^2 + k_y^2 + k_z^2}$$

Substitute the expressions for  $k_x$ ,  $k_y$ , and  $k_z$ :

$$k = \sqrt{\left(\frac{2\pi l_x}{L_x}\right)^2 + \left(\frac{2\pi l_y}{L_y}\right)^2 + \left(\frac{2\pi l_z}{L_z}\right)^2}$$

## 3. Volume of k-Space

### 3. Volume of k-Space

The volume element in  $k$ -space is given by:

$$d^3k = dk_x dk_y dk_z$$

To find the density of states, we need to count the number of states within a spherical shell in  $k$ -space.

## Pseudocódigo para Calcular la Densidad de Estados

### 1. Definir Variables

Primero, define las constantes y variables necesarias, como las longitudes del cuboide y las variables para la densidad de estados:

```
// Definir constantes
pi = 3.141592653589793

// Longitudes del cuboide
Lx = longitud del cuboide en x
Ly = longitud del cuboide en y
Lz = longitud del cuboide en z

// Número de modos
num_modos = número total de modos (si conocido) o rango de
valores para l_x, l_y, l_z
```

### 2. Calcular las Componentes del Vector de Onda

```
// Inicializar arrays para las componentes del vector de onda
kx = array de componentes en x
ky = array de componentes en y
kz = array de componentes en z

// Calcular las componentes del vector de onda para cada modo
for l_x from 1 to num_modos do
  for l_y from 1 to num_modos do
    for l_z from 1 to num_modos do
      // Calcular kx, ky, kz
      kx[l_x, l_y, l_z] = (2 * pi * l_x) / Lx
      ky[l_x, l_y, l_z] = (2 * pi * l_y) / Ly
      kz[l_x, l_y, l_z] = (2 * pi * l_z) / Lz
    end for
  end for
end for
```

```
        end for
    end for
```

### 3. Calcular el Magnitud del Vector de Onda

Calcula la magnitud del vector de onda para cada modo:

```
// Inicializar array para la magnitud del vector de onda
k = array de magnitudes de vector de onda

// Calcular la magnitud del vector de onda para cada modo
for l_x from 1 to num_modes do
    for l_y from 1 to num_modes do
        for l_z from 1 to num_modes do
            k[l_x, l_y, l_z] = sqrt(kx[l_x, l_y, l_z]^2 +
ky[l_x, l_y, l_z]^2 + kz[l_x, l_y, l_z]^2)
        end for
    end for
end for
```

### 4. Calcular la Densidad de Estados

Utiliza la fórmula para la densidad de estados:

```
// Inicializar array para la densidad de estados
g = array de densidad de estados

// Calcular la densidad de estados para cada valor de k
for l_x from 1 to num_modes do
    for l_y from 1 to num_modes do
        for l_z from 1 to num_modes do
            // Calcular la densidad de estados
            g[l_x, l_y, l_z] = (k[l_x, l_y, l_z]^2) / (pi^2)
        end for
    end for
end for
```

```
end for
```

## 5. Sumar la Densidad de Estados (Opcional)

Si quieres obtener la densidad de estados total en un rango específico de magnitudes de  $k$ , puedes sumar o promediar:

```
// Inicializar suma de densidad de estados
total_density = 0

// Sumar la densidad de estados en un rango específico
for l_x from 1 to num_modes do
    for l_y from 1 to num_modes do
        for l_z from 1 to num_modes do
            total_density = total_density + g[l_x, l_y, l_z]
        end for
    end for
end for
```

## Resumen

Este pseudocódigo define y calcula automáticamente la densidad de estados para un cuboide en el espacio de onda. Los pasos incluyen la definición de variables, el cálculo de las componentes y magnitudes del vector de onda, y finalmente la densidad de estados. Puedes adaptar este pseudocódigo a un lenguaje de programación específico para implementar el cálculo en un entorno computacional.

Si tienes un rango específico para los números cuánticos  $l_x$ ,  $l_y$ , y  $l_z$ , ajusta el `num_modes` en los bucles para reflejar esos rangos. Además, si conoces la distribución de modos más detalladamente, puedes ajustar el cálculo en consecuencia.

## 1. Ecuación de Schrödinger Dependiente del Tiempo

La ecuación de Schrödinger dependiente del tiempo es:

$$i\hbar \frac{\partial \Psi(r, t)}{\partial t} = \hat{H} \Psi(r, t)$$

donde  $\hat{H}$  es el operador Hamiltoniano.

Pseudocódigo para la resolución numérica:

```
// Definir constantes
hbar = Planck's constant divided by 2π
dt = paso de tiempo
t_max = tiempo máximo de simulación
```

```

// Definir variables espaciales
Lx = longitud en x
Ly = longitud en y
Lz = longitud en z
N_x = número de puntos en x
N_y = número de puntos en y
N_z = número de puntos en z

// Crear una malla espacial
x = array de tamaño N_x
y = array de tamaño N_y
z = array de tamaño N_z

// Inicializar la función de onda Ψ en el tiempo t=0
Psi = array de tamaño (N_x, N_y, N_z)

// Definir el Hamiltoniano (ejemplo simple de un potencial V)
V = array de tamaño (N_x, N_y, N_z)
H = array de tamaño (N_x, N_y, N_z)

// Tiempo de evolución
t = 0

// Iterar en el tiempo
while t <= t_max do
    // Actualizar la función de onda usando el método de Euler
    explícito
    Psi_new = Psi + ( -i * dt / hbar ) * ( H * Psi )

    // Actualizar Ψ
    Psi = Psi_new

    // Incrementar el tiempo
    t = t + dt
end while


```

## 2. Ecuación de Schrödinger Independiente del Tiempo

La ecuación de Schrödinger independiente del tiempo es:

$$\hat{H}\psi(r) = E\psi(r)$$

donde  $E$  es la energía del sistema.

Pseudocódigo para Resolución  América

Para resolver esta ecuación, se puede utilizar métodos numéricos como el método de diferencias finitas o métodos de matrices:

```

// Definir constantes
hbar = Planck's constant divided by 2π

```



```

// Definir variables espaciales
Lx = longitud en x
Ly = longitud en y
Lz = longitud en z
N_x = número de puntos en x
N_y = número de puntos en y
N_z = número de puntos en z

// Crear una malla espacial
x = array de tamaño N_x
y = array de tamaño N_y
z = array de tamaño N_z

// Inicializar la función de onda  $\Psi$ 
Psi = array de tamaño (N_x, N_y, N_z)

// Definir el potencial V
V = array de tamaño (N_x, N_y, N_z)

// Definir el Hamiltoniano H (por ejemplo, usando diferencias finitas)
H = array de tamaño (N_x, N_y, N_z, N_x, N_y, N_z)

// Construir la matriz del Hamiltoniano H usando diferencias finitas
for i_x from 1 to N_x do
    for i_y from 1 to N_y do
        for i_z from 1 to N_z do
            // Calcular la parte cinética y potencial del Hamiltoniano
            // Implementar diferencias finitas
            H[i_x, i_y, i_z, i_x, i_y, i_z] = ... // Valor correspondiente
        end for
    end for
end for

// Resolver el problema de eigenvalores
eigenvalues, eigenvectors = solve_eigenvalue_problem(H)

// eigenvalues contiene las energías
// eigenvectors contiene las funciones de onda correspondientes

```

## Automatización y Resolución

Automatización en este contexto implica:

- **Definir las variables:** Espaciales, temporales y del sistema.
- **Construir matrices:** Para representar el Hamiltoniano o el operador.
- **Aplicar métodos numéricos:** Para resolver ecuaciones diferenciales o problemas de eigenvalores.
- **Actualizar y analizar:** La evolución temporal o los estados estacionarios.

**Consideraciones Adicionales:**

1. **Métodos Numéricos:** El pseudocódigo anterior asume el uso de métodos numéricos. En la práctica, se pueden usar bibliotecas de álgebra lineal (por ejemplo, LAPACK) y métodos específicos para resolver ecuaciones diferenciales (como el método de Runge-Kutta).
2. **Condiciones de Frontera:** Asegúrate de implementar las condiciones de frontera adecuadas para tu problema específico (por ejemplo, condiciones de frontera de Dirichlet o Neumann).
3. **Manejo de Errores:** Implementar control de errores y validaciones para asegurar que los cálculos sean precisos y estables.

Este enfoque te permitirá automatizar la resolución de problemas cuánticos y realizar simulaciones numéricas más eficientes en el campo de la mecánica cuántica.