Física Computacional

Resolución de ecuaciones en derivadas parciales: la ecuación de Schrödinger

1. Fundamento Teórico

- En Mecánica cuántica, el estado físico de un sistema unidimensional de una partícula viene descrito completamente por una función de onda compleja $\Phi(x,t)$ que obedece la denominada ecuación de Schrödinger

$$i\hbar\frac{\partial\Phi(x,t)}{\partial t} = \left[-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V(x)\right]\Phi(x,t) = H\Phi \tag{1}$$

  donde $V(x)$ es el potencial al que está sometida la partícula, que supondremos que tiene una masa m independiente del tiempo, $\hbar$ es la constante de Planck reducida y H es el operador Hamiltoniano, que es hermítico.

- La densidad de probabilidad de encontrar a la partícula en un volumen dV alrededor del punto x y en el instante t es

$$dP = |\Phi(x,t)|^2 dV. \tag{2}$$

- Normalizaci'on: Para un sistema compuesto únicamente por una partícula, como es el nuestro, la probabilidad total de encontrar la partícula en algún punto del espacio, en cualquier instante t, es igual la unidad,

$$\int_V |\Phi(x,t)|^2 dV = 1. \tag{3}$$

- Nótese que la integral de probabilidad es constante durante la evolución temporal, lo que habrá que tenerse en cuenta al plantear la resolución numérica del problema.

- Podemos observar que la ecuación de Schrödinger es lineal y que se trata de una ecuación diferencial de primer orden en el tiempo.

- Para evitar el uso continuado de m y $\hbar$, haremos el cambio de variable $t \to t\hbar$ en el tiempo y $x \to x\hbar/\sqrt{2m}$ en el espacio, lo que conduce a una ecuación equivalente a (1) pero en la que $\hbar = 1$ y $m = 1/2$:

$$-H\Phi \equiv \left[\frac{\partial^2}{\partial x^2} - V(x)\right]\Phi(x,t) = -i\frac{\partial\Phi}{\partial t}. \tag{4}$$

- Esta ecuación puede resolverse términos de las autofunciones del hamiltoniano H, que, recordemos, es independiente del tiempo.

- Llamando u_n a los estados acotados y u_k a los estados del continuo, entonces la solución dependiente del tiempo viene dada de forma rigurosa por

$$\Phi(x,t) = \sum_n a_n e^{-iE_n t} u_n(x) + \int dk a(k) e^{-iE(k)t} u_k(x), \tag{5}$$

  con

$$a_n = \langle u_n, \phi(x,0)\rangle, \qquad a(k) = \langle u_k, \phi(x,0)\rangle, \tag{6}$$

  donde $\Phi(x,0)$ es el estado inicial del sistema (y por lo tanto a_n y $a(k)$ son los coeficientes Φ en las bases $\{u_n\}$ y $\{u_k\}$, respectivamente).

- De seguir este procedimiento, toparíamos con los inconvenientes de que:

  - En la mayoría de los casos, las autofunciones y los autovalores del hamiltoniano no se pueden calcular analíticamente y han de determinarse numéricamente.
  - Por motivos numéricos, el estado inicial debe restringirse necesariamente a una combinación lineal *finita* de autofunciones y, en cualquier caso, el número de éstas debe ser el menor posible para reducir el trabajo computacional. Truncar la base tiene como inconveniente añadido que la evolución deja de ser unitaria.

- Por todo ello, en lugar de seguir el procedimiento anterior integrararemos directamente la ecuación (4). Pero hay que ser muy cuidadosos con el método numérico porque, en general, la discretización provoca que no se conserve la normalización de la función de onda (ecuación 3).

- La solución formal de la ecuación de Schrödinger es

$$\Phi(x,t) = e^{-i(t-t_0)H}\Phi(x,t_0) \tag{7}$$

donde el operador e^{-itH} es unitario (su inverso coindice con su adjunto) por ser H hermítico. Esta última propiedad será de utilidad a la hora de hallar la discretización más adecuada para resolver el problema numéricamente como veremos a continuación.

2. Método numérico

- Discretizaremos el espacio y el tiempo tomando $x_j = jh$ y $t_n = ns$, con $j = 0, 1, ..., N$ y $n = 0, 1, 2, ...$, donde h es el espaciado en la discretización espacial y s es el espaciado temporal.

- La función de onda viene dada en cada punto del retículo espacio-temporal por:

$$\Phi(x_j, t_n) \to \Phi(jh, ns) = \Phi_{j,n} \qquad \text{con} \quad j = 0, 1, ..., N \quad \text{y} \quad n = 0, 1, 2, ... \tag{8}$$

- Supondremos que las condiciones de contorno para la función de onda en $j = 0$ y $j = N$ son las correspondientes a la existencia de un potencial infinito en esos puntos, esto es, la densidad de probabilidad de encontrar la partícula en dichos puntos es cero. Así, $\Phi_{0,n} = \Phi_{N,n} = 0$, en cualquier instante.

- Puede definirse de forma inmediata un primer algoritmo a partir de la expresión

$$\Phi_{j,n+1} = e^{-isH}\Phi_{j,n}. \tag{9}$$

Si s es muy pequeño, el operador de evolución $\exp(-isH)$ puede aproximarse por su desarrollo de Taylor a primer orden en t. Si además observamos que el operador Hamiltoniano no es más que la aplicación de una derivada segunda, su discretización espacial es obvia y en total obtenemos el siguiente algoritmo

$$\Phi_{j,n+1} = \left(1 - isH_D + O(sH_D)^2\right)\Phi_{j,n}, \tag{10}$$

donde

$$\Phi_j'' \equiv \frac{\partial^2 \Phi}{\partial x^2} = \frac{1}{h^2}(\Phi_{j+1} - 2\Phi_j + \Phi_{j-1}) + O(h^2) \tag{11}$$

y H_D viene dado por

$$H_D f_j = -\frac{1}{h^2}(f_{j+1} - 2f_j + f_{j-1}) + V_j f_j \tag{12}$$

con $V_j = V(jh)$.

- Este algoritmo tan simple y natural adolece de un grave problema: el operador $(1 - isH)$ **no es unitario**. Esto implica que durante la iteración del algoritmo para encontrar la evolución de la función de onda, la normalización, $\Sigma_j \, h|\Phi_{j,n}|^2$, irá variando con el tiempo, lo que es incompatible con la restricción de normalización a la unidad expresada en (3) y viola la interpretación de Born.

- Así pues, el objetivo será encontrar un operador evolución similar al anterior pero que sea exactamente unitario. Esto se consigue haciendo uso de la aproximación de Cayley para el operador evolución temporal

$$e^{-isH} \approx \frac{1 - isH_D/2}{1 + isH_D/2}, \tag{13}$$

que conduce al algoritmo

$$\Phi_{j,n+1} = \frac{1 - isH_D/2}{1 + isH_D/2}\Phi_{j,n}. \tag{14}$$

- Nótese que además de ser unitario, este operador es exacto hasta orden $(sH_D)^2$.

- Para completar el algoritmo, sólo resta diseñar la estrategia para utilizar eficazmente el operador $1/(1 + isH_D)$. Para ello reescribimos la anterior ecuación como

$$\Phi_{j,n+1} = \left[\frac{2}{1 + isH_D/2} - 1\right]\Phi_{j,n} = \chi_{j,n} - \Phi_{j,n}, \tag{15}$$

donde

$$\chi_{j,n} \equiv \frac{2}{1 + isH_D/2} \Phi_{j,n}, \tag{16}$$

o lo que es lo mismo, dada $\Phi_{j,n}$, $j = 0, .., N$, $\chi_{j,n}$ es la solución de la ecuación

$$\left[1 + isH_D/2\right] \chi_{j,n} = 2\Phi_{j,n}, \tag{17}$$

que en forma explícita puede escribirse como

$$\chi_{j+1,n} + \left[-2 + \frac{2i}{\tilde{s}} - \tilde{V}_j\right] \chi_{j,n} + \chi_{j-1,n} = \frac{4i}{\tilde{s}} \Phi_{j,n} \tag{18}$$

donde $\tilde{s} = s/h^2$ y $\tilde{V}_j = h^2 V_j$.

- De esta forma, el algoritmo de evolución es claro: dada una función de onda en el instante n para toda posición j se resuelve el conjunto de ecuaciones (18) y se obtiene $\chi_{j,n}$ $(j = 0, .., N)$. Con esta solución se recurre a la ecuación (15) para obtener las nuevas $\Phi_{j,n+1}$ $(j = 0, ..., N)$, iterándose el proceso.

- Sólo falta por conocer cómo resolver ecuaciones del tipo (18), es decir, cómo invertir matrices tridiagonales.

- En este caso, hemos de resolver un conjunto de ecuaciones del tipo

$$A_j^- \chi_{j-1,n} + A_j^0 \chi_{j,n} + A_j^+ \chi_{j+1,n} = b_{j,n} \quad j = 1, \ldots, N-1 \tag{19}$$

donde $A_j^- = 1$, $A_j^0 = -2 + 2i/\tilde{s} - \tilde{V}_j$, $A_j^+ = 1$ y $b_{j,n} = 4i\Phi_{j,n}/\tilde{s}$.

- Las condiciones de contorno son $\chi_0 = \chi_N = 0$ (notemos que estas condiciones de contorno implican que en (15) Φ_0 y Φ_N son nulas en cualquier instante).

- Para resolver la recurrencia (19) suponemos que su solución es del tipo

$$\chi_{j+1,n} = \alpha_j \chi_{j,n} + \beta_{j,n} \quad j = 0, \ldots, N-1 \tag{20}$$

donde, para garantizar que se cumple $\chi_{N,n} = 0$, tomaremos $\alpha_{N-1} = \beta_{N-1,n} = 0$.

- Sustituyendo esta expresión en (19) obtenemos

$$\chi_{j,n} = -\frac{A_j^-}{A_j^0 + A_j^+ \alpha_j} \chi_{j-1} + \frac{b_{j,n} - A_j^+ \beta_{j,n}}{A_j^0 + A_j^+ \alpha_j}. \tag{21}$$

- Si identificamos las ecuaciones (20) y (21) podemos definir las recurrencias para los coeficientes α y β así

$$\alpha_{j-1} = -A_j^- \gamma_j, \qquad \beta_{j-1,n} = \gamma_j \left(b_{j,n} - A_j^+ \beta_{j,n} \right). \tag{22}$$

donde $\gamma_j^{-1} = A_j^0 + A_j^+ \alpha_j$.

- Estas ecuaciones nos dan la forma de obtener todas las α y β partiendo de $j = N-1$ y obteniendo en orden decreciente α_j y $\beta_{j,n}$ con $j = N-2, \ldots, 1, 0$.

- Obsérvese que α no depende del tiempo y sólo es necesario calcularlas una vez.

- Una vez obtenidas las α y β, se usa la recurrencia (20) para hallar las χ_j en orden de j crecientes.

- Conocida $\chi_{j,n}$, y con ella $\Phi_{j,n+1}$, queda discutir qué función de onda inicial se usa y con qué potencial. La función de onda inicial que vamos a usar es una onda plana con una amplitud gaussiana, esto es,

$$\Phi(x, 0) = e^{ik_0 x} e^{-(x-x_0)^2/2\sigma^2}. \tag{23}$$

Notemos que con esta elección, la densidad de probabilidad de encontrar inicialmente la partícula en un punto x es una gaussiana centrada en x_0 y de anchura σ.

- El número de oscilaciones completas que la función de onda tiene sobre la red depende de k_0. En particular, $k_0 N h = 2\pi n_{ciclos}$. En lugar de dar como parámetro inicial k_0, daremos n_{ciclos}.

- Obviamente $n_{ciclos} = 0, 1, ..., N$. Pero físicamente no tendría mucho sentido que se produjese una oscilación completa de la función de onda entre dos puntos del retículo, pues querría decir que la discretización no es suficientemente fina. Así pues, restringiremos el parámetro n_{ciclos} a los valores $1, ..., N/4$. De esta forma, un ciclo tendrá 4 puntos como mínimo.

- La posición media inicial y la anchura de la gaussiana serán por ejemplo $x_0 = Nh/4$ y $\sigma = Nh/16$, aunque es recomendable escribir la función de onda inicial de manera genérica y jugar con x_0 y σ.

- Por último, el potencial que usaremos tendrá una anchura $N/5$, estará entrado en $N/2$ y su altura será proporcional a la energía de la función de onda incidente: λk_0^2 (puede usarse, por ejemplo, $\lambda = 0{,}3$).

- En resumen, utilizando las constantes reescaladas, la función de onda en el retículo se tomará

$$\Phi_{j,0} = e^{i\tilde{k}_0 j} e^{-8(4j-N)^2/N^2}, \tag{24}$$

  donde $\tilde{k}_0 = k_0 h = 2\pi n_{ciclos}/N$ donde $n_{ciclos} = 1, ..., N/4$.

- El potencial es

$$\tilde{V}_j = V_j h^2 = \begin{cases} 0 & \text{si } j \notin [2N/5, 3N/5], \\ \lambda \tilde{k}_0^2 & \text{si } j \in [2N/5, 3N/5]. \end{cases} \tag{25}$$

- Sólo queda por fijar el parámetro $\tilde{s} = s/h^2$. Puesto que la energía es proporcional a k_0^2 y el operador dinámico discreto tiende a ser exacto en potencias de Hs, lo óptimo es elegir $||H||s < 1$, esto es, $k_0^2 s < 1$. Así deducimos que $\tilde{s} < 1/\tilde{k}_0^2$. En particular tomaremos $\tilde{s} = 1/4\tilde{k}_0^2$.

- Resumiendo, los parámetro que se han de fijar inicialmente son N, n_{ciclos} y λ, pues todos los demás se determinan a partir de ellos.

- Por lo tanto, el algoritmo para resolver la ecuación de Schrödinger unidimensional puede esquematizarse del siguiente modo:

  1. Dar los parámetros iniciales: N, n_{ciclos} y λ. Generar $\tilde{s}$, $\tilde{k}_0$, $\tilde{V}_j$, $\Phi_{j,0}$ (incluyendo las condiciones de contorno $\Phi_{0,0} = \Phi_{N,0} = 0$) y α.
  2. Calcular β utilizando la recurrencia (22).
  3. Calcular χ a partir de (20).
  4. Calcular $\Phi_{j,n+1}$ de (15).
  5. $n = n + 1$, ir a al paso 2.

3. Problemas

- *Obligatorio:* Resolver la ecuación de Schrödinger unidimensional para un potencial cuadrado. Comprobar que se conserva la norma.

- *Voluntario 1: Estudiar el coeficiente de transmisión*

  El coeficiente de transmisión es la probabilidad de encontrar a la partícula al otro lado del obstáculo para tiempos largos. En sistemas clásicos este coeficiente es 1 si la energía de la partícula es mayor que la energía del escalón y 0 si es menor. En sistemas cuánticos esto cambia por la acción del efecto túnel. Para determinar si una partícula es reflejada o transmitida debemos colocar detectores al final del sistema. Como ya hemos visto, la probabilidad de encontrar a la partícula entre los puntos x_1 y x_2 viene dada por

$$P(x_1, x_2) = \int_{x_1}^{x_2} |\Phi(x)|^2 \, dx \tag{26}$$

Suponemos que tenemos detectores finitos, actuando a derecha e izquierda de la barrera. Si estos detectores tienen un ancho de $N/5$ la probabilidad a tiempo n de detectar la partícula a la derecha vendrá dado por $P_D(n) = \Sigma_{j=4N/5}^{N} |\Phi_{j,n}|^2$, y la probabilidad de detectarla a la izquierda sería $P_I(n) = \Sigma_{j=0}^{N/5} |\Phi_{j,n}|^2$. Después de realizar el experimento m veces el coeficiente de transmisión se calcula como $K = m_T/m$, donde m_T es el número de veces que hemos detectado la partícula a la derecha del potencial.

¿Cómo podemos realizar este proceso de medición con precisión? Un problema en un sistema dinámico de este tipo es saber determinar cuánto tiempo debemos dejar evolucionar el sistema antes de medir. Para tiempos demasiado cortos, la partícula puede no tener tiempo de moverse de su localización inicial. Para tiempos largos, la partícula podría encontrarse en cualquier lugar.

Para resolver este problema se propone el siguiente método. Para una partícula inicialmente localizada a la izquierda de una barrera de potencial y dirigida hacia ella, dejaremos evolucionar el sistema un tiempo t y estudiaremos el comportamiento de la función $P_D(t)$. Para determinar un tiempo de evolución apropiado, buscaremos el valor $t = n_D$ correspondiente al primer máximo local de $P_D(t)$.

Para calcular el coeficiente de transmisión El primer algoritmo que proponemos quedará:

1. Generamos la función de onda inicial.
2. Busca el valor $t = n_D$ correspondiente al primer máximo local de $P_D(t)$.
3. Evolucionamos n_D pasos.
4. Calculamos $P_D(n_D)$.
5. Para simular el proceso de medida siguiendo la regla de Born, generamos un número aleatorio $x \in [0, 1]$. Si $x < P_D(n_D)$, ha habido detección, aumentamos el contador m_T y volvemos al paso 1.

6. Calculamos la función de onda tras la medida en el detector derecho. Hacer $\Phi_j = 0$ para $j \in [4N/5, N]$, calcular $k = \Sigma_{j=0}^{N} |\Phi_j|^2$ y hacer $\Phi_j = \frac{1}{\sqrt{k}}\Phi_j$ para todo j.
7. Calcular $P_I(n_D)$.
8. Generar un número aleatorio x. Si $x < P_I$ e ir a 1.
9. Hacer $\Phi_j = 0$ para $j \in [0, N/5]$, calcular $k = \Sigma_{j=0}^{N} |\Phi_j|^2$ y hacer $\Phi_j = \frac{1}{\sqrt{k}}\Phi_j$ para todo j.
10. Ir a 2.

Para obtener una buena estadística habrá que simular el sistema al menos 10^3 veces.

El segundo algoritmo que proponemos es más sencillo. Podemos primero elegir x_0, k_0 para que la partícula este bien localizada incialmente a la izquierda del pozo con movimiento hacia la derecha. Entonces, calculamos $P_D(n_D)$ para múltiples valores de n_D y maximizamos el valor de dicha probabilidad. Si el paquete de onda inicial se elige de la forma sugerida, el valor máximo de esta cantidad debería ser el coeficiente de transmisión.

Hay que realizar (10 puntos):

- Busca una explicación física sencilla de por qué tiene sentido buscar el punto máximo $t = n_D$. Pista: puedes pensar en el límite clásico del sistema cuántico.
- Estudia la dependencia en N de K realizando simulaciones para $N = 500$, 1000, 2000.
- Estudiar la dependencia en $V(x)$ de K, usando valores $\lambda = 0, 1$; 0,3; 0,5; 1; 5; 10.
- Comparar con resultados teóricos.
- Calcula los valores esperados de distintos observables (posición, momento, energía cinética, energía total, ...) con y sin mediciones y discutir los resultados.

  Nota: El valor esperado de un observable se puede calcular como

$$\langle O \rangle = \int \phi^*(x) \hat{O} \phi(x) dx,$$

donde $\hat{O}$ es el operador correspondiente al observable $\left(\hat{x} = x, \hat{p} = -i\hbar \frac{\partial}{\partial x}, \dots\right)$.

Números complejos en Fortran.

- *Declaración:* implicit double complex (h)
- *Asignación de valores constantes:* $h = (1{,}0d0, 0{,}3d0)$ (=1+i0.3).
- *Asignación de variables:* $h = complex(a, b)$ (=a+ib).

4. Apéndice: resolución de sistemas de ecuaciones con matrices tridiagonales

4.1. Teoría

- En este tema hemos visto como resolver la ecuación de Schrödinger de la mecánica cuántica. El algoritmo que usamos requiere la resolución de un sistema de ecuaciones lineales con "matrices tridiagonales": matrices que tienen ceros casi en todas sus entradas menos en tres bandas centrales. Aunque dichos sistemas pueden resolverse con algoritmos genéricos (véase, p. ej., eliminación Gaussiana), es mucho más eficiente explotar la estructura adicional de la matriz. En concreto, el algoritmo génerico de eliminación Gaussiana tiene coste computacional $O(N^3)$, siendo N el número de variables del sistema de ecuaciones. En nuestro ejemplo, N corresponde al número de puntos espaciales que usamos en nuestra discretización. Como queremos que N sea lo más alto posible y el coste escala como N al cubo, el algoritmo genérico de resolución de sistemas de ecuaciones no es tan eficiente como otros métodos que aprovechan que la matriz es tridiagonal.

- En concreto, visto en clase aprovecha la estructura de la matriz tridiagonal y tiene complejidad computacional $O(N)$. Esto supone una mejora cúbica. Esta mejora es posible, en parte, porque la matriz es casi diagonal. El método que estudiamos en clase para resolver el sistema utiliza dos ideas para conseguir esta mejora:

  1. Se realiza un cambio de coordenadas para poner la matriz en forma triangular.
  2. Se resuelve el sistema con matriz triangular realizando un barrido en escalera

- La teoría debajo este método puede consultarse en Wikipedia:

  - https://es.wikipedia.org/wiki/Algoritmo_para_matrices_tridiagonales
  - https://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm

4.2. Código

El artículo de la Wikipedia anterior (versión española) contiene ejemplos de código para resolver sistemas de ecuaciones tridiagonales en C, Python y Fortran. Adjuntamos aquí los ejemplos de código en C que podrían ser útiles en la resolución de los ejercicios.

- La siguiente función en C resolverá el sistema (aunque sobreescribirá uno de los vectores de entrada conteniendo una de las diagonales de la matriz en el proceso). Se ha de notar que aquí los subíndices están basados en el cero, es decir $n = 0, 1, \ldots, N-1$ donde N es el número de ecuaciones:

```
void solve_tridiagonal_in_place_destructive(float x[], const size_t N,
  ↪ const float a[], const float b[], float c[]) {
    int n;

    /**
```

```
 *  solves Ax = v where A is a tridiagonal matrix consisting of
   ↪ vectors a, b, c
 *  note that contents of input vector c will be modified, making
   ↪ this a one-time-use function
 *  x[] - initially contains the input vector v, and returns the
   ↪ solution x. indexed from [0, ..., N - 1]
 *  N   - number of equations
 *  a[] - subdiagonal (means it is the diagonal below the main
   ↪ diagonal) -- indexed from [1, ..., N - 1]
 *  b[] - the main diagonal, indexed from [0, ..., N - 1]
 *  c[] - superdiagonal (means it is the diagonal above the main
   ↪ diagonal) -- indexed from [0, ..., N - 2]
 */

c[0] = c[0] / b[0];
x[0] = x[0] / b[0];

/* loop from 1 to N - 1 inclusive */
for (n = 1; n < N; n++) {
    float m = 1.0f / (b[n] - a[n] * c[n - 1]);
    if(n < (N-1)) c[n] = c[n] * m;
    x[n] = (x[n] - a[n] * x[n - 1]) * m;
}

/* loop from N - 2 to 0 inclusive */
```

```
    for (n = N - 2; n >= 0; --n) {
        x[n] = x[n] - c[n] * x[n + 1];
    }
}
```

- La siguiente variante preserva el sistema de ecuaciones para reutilizarlo en otras funciones. Se hacen llamadas a la biblioteca para reservar más especiao. Otras variantes usan un puntero a memoria disponible.

```
void solve_tridiagonal_in_place_reusable(float x[], const size_t N,
  ↪ const float a[], const float b[], const float c[]) {
    size_t n;

    /* allocate scratch space */
    float * const cprime = malloc(sizeof(float) * N);

    cprime[0] = c[0] / b[0];
    x[0] = x[0] / b[0];

    /* loop from 1 to N - 1 inclusive */
    for (n = 1; n < N; n++) {
        float m = 1.0f / (b[n] - a[n] * cprime[n - 1]);
        if(n < (N-1)) cprime[n] = c[n] * m;
        x[n] = (x[n] - a[n] * x[n - 1]) * m;
    }

    /* loop from N - 2 to 0 inclusive */
```

```
    for (n = N - 2; n > 0; --n)
        x[n] = x[n] - cprime[n] * x[n + 1];

    /* free scratch space */
    free(cprime);
}
```