Métodos numéricos para la Ciencia e Ingeniería
Tarea 6: Ecuación de reacción-difusión

Felipe Toledo Bittner

November 6, 2015

1 Introducción

Los sistemas de reacción-difusión se utilizan para modelar matemáticamente fenómenos espaciales en que el valor de una variable en cada punto presenta dependencia tanto de reacciones locales como de difusión desde los puntos cercanos. Su forma general puede verse en la ecuación (1). El primer término de la derecha corresponde a la componente difusiva y el segundo a la reactiva, en este caso modelada como el polinomio $P(n)$.

$$\frac{dn}{dt} = \gamma \frac{d^2 n}{dx^2} + P(n) \tag{1}$$

En este trabajo se implementa una clase en python que permite crear un sistema de reacción-difusión unidimensional y resolverlo para un intervalo de tiempo escogido por el usuario. Se incluyen métodos que permiten setear la variable γ, los coeficientes del polinomio $P(n)$ y las condiciones iniciales y de borde.

2 Descripción de la solución

La clase implementada permite resolver cualquier sistema de reacción-difusión que posea la forma de la ecuación (1). En el archivo *main_p1.py* adjunto a este informe hay un ejemplo bien comentado con los pasos a seguir para inicializar el programa, en este caso con la ecuación de Fisher-KPP.

2.1 Algoritmo de integración

Para integrar la ecuación de forma numérica se opta por utilizar dos métodos, Crank-Nicolson para la parte de difusión y una versión modificada del método de Euler para la de reacción. La modificación ha sido una elaboración propia, y es justificada a continuación.
Inicialmente consideremos el caso en que no hay componente de difusión. La ecuación (1) queda escrita en forma numérica como se muestra en (2). Los subíndices (x, t) en este caso representan posiciones y tiempos discretos y h es el paso temporal de la integración.

$$\frac{n_{x,t+1} - n_{x,t}}{h} = P(n_{x,y}) \tag{2}$$

Que se puede integrar usando el método de Euler, como se indica en la ecuación (3). Se observa que el lado derecho consiste simplemente en la actualización de la variable n usando su pendiente en cada instante de tiempo. Apartemos esto un momento para estudiar la parte difusiva, pero no debemos olvidarlo aún.

$$n^e_{x,t+1} = n_{x,t} + hP(n_{x,t}) \tag{3}$$

Suponiendo ahora que sólo hay componente difusiva, la ecuación original puede escribirse como (4). Aquí se utiliza el promedio entre las segundas derivadas calculadas hacia adelante y atrás para aproximar mejor el cambio en n.

$$\frac{n_{x,t+1} - n_{x,t}}{h} = \frac{1}{2}\left(\frac{n_{x+1,t+1} - 2n_{x,t+1} + n_{x-1,t+1}}{h^2} + \frac{n_{x+1,t} - 2n_{x,t} + n_{x-1,t}}{h^2}\right) \tag{4}$$

A partir de aquí se puede plantear un sistema de ecuaciones lineales para encontrar el valor de los $n_{x,t+1}$ desconocidos a partir de los $n_{x,t}$ conocidos para todos los puntos x simultáneamente. Éste es el llamado método de Crank-Nicolson, explicado en múltiples excelentes referencias [1].
El resultado de esta operación calcula todos los $n_{x,t+1}$. A grandes rasgos el resultado de una iteración puede describirse como se ve en la ecuación (5), donde $CN(\Delta x, h, \mathbf{n})$ es el término de actualización, Δx la distancia entre puntos en unidades de longitud físicas y $\mathbf{n}$ el vector que contiene todos los $n_{x,t}$.

$$n^{cn}_{x,t+1} = n_{x,t} + CN(\Delta x, h, \mathbf{n}) \tag{5}$$

En principio, calcular el valor completo de $n_{x,t+1}$ debiese ser tan sencillo como sumar $n_{x,t+1} = n^{cn}_{x,t+1} + n^{e}_{x,t+1}$, pero al hacer esto aparece un problema. Como se ve en la ecuación (6), al hacer esta operación aparece un término $2n_{x,t}$. Este término es espurio y debe corregirse ya que el espíritu de la integración numérica es ir actualizando $n_{x,t}$ usando sus tasas de cambio, dadas en este caso por $CN(...)$ y $P(n)$.

$$n_{x,t+1} = n^{cn}_{x,t+1} + n^{e}_{x,t+1} = \mathbf{2n_{x,t}} + CN(\Delta x, h, \mathbf{n}) + hP(n_{x,t}) \tag{6}$$

Para corregir el inconveniente, se opta por modificar el método de euler, dejándolo de la forma $n^{e^*}_{x,t+1} = hP(n_{x,t})$. Tras esta modificación, la ecuación (6) toma la forma (7), que como se ve en secciones posteriores logra integrar correctamente el sistema de difusión-reacción.

$$n_{x,t+1} = n^{cn}_{x,t+1} + n^{e^*}_{x,t+1} = n_{x,t} + CN(\Delta x, h, \mathbf{n}) + hP(n_{x,t}) \tag{7}$$

3 Resultados

3.1 Caso 1

En este caso se modela un sistema de Fisher-KPP, el cual cumple con la siguiente ecuación:

$$\frac{\partial n}{\partial t} = \gamma \frac{\partial^2 n}{\partial x^2} + \mu n - \mu n^2 \tag{8}$$

Los parámetros usados son:

1. Largo de 1 [m]
2. Discretizado de 500 puntos
3. $\gamma = 0.001$
4. $\mu = 1.5$
5. Resolución: $h = 0.001$, $\Delta x = 0.002$
6. Condiciones de borde: $n(t, x = 0) = 1$ y $n(t, x = 1) = 0$
7. Condiciones iniciales: $n(0, x) = e^{-x^2/0.1}$

[1] Por ejemplo el documento "Solucion Numerica de la Ecuacion de Calor por el Metodo de las Diferencias Finitas", realizado por Miguel Caro C. del Departamento de Matematicas de la Universidad del Atlantico de Colombia, entre otros. Puede encontrarse en línea en el sitio

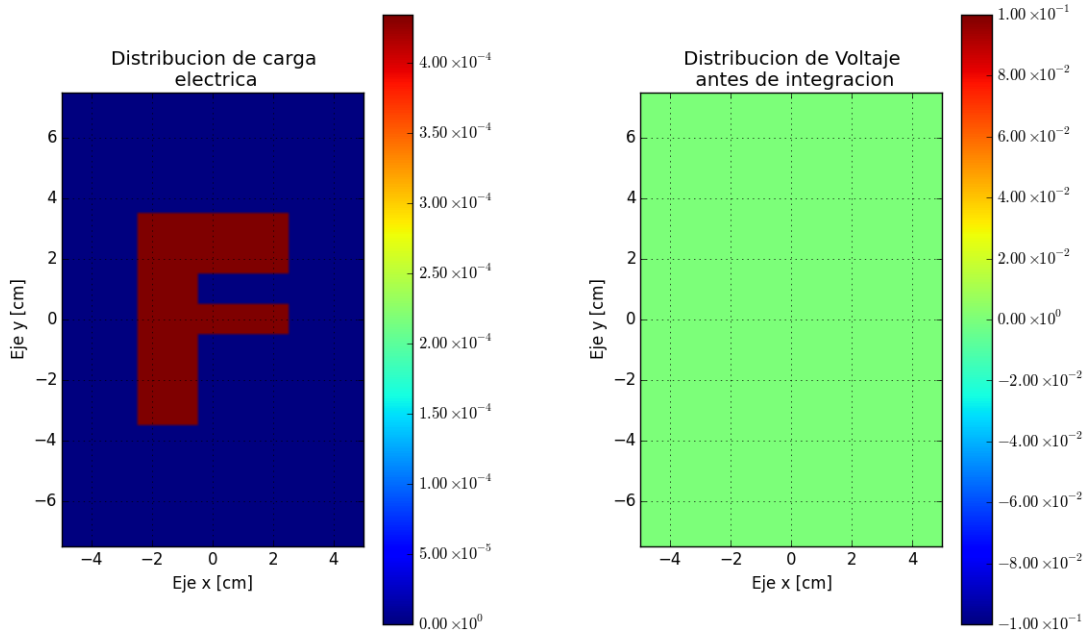

Figura 1: Situacion inicial de las cajas de carga y voltaje. Se observa la letra 'F' dibujada en la caja de carga y las condición $V = 0$ en todo el borde de la caja de potencial. Las unidades de carga son Coulomb y las de potencial, Voltaje.

3.2 Software

La solución del problema se realizó implementando los archivos *main.py* y *box.py*.
El archivo *box.py* es una clase que contiene todos los métodos necesarios para operar y simular el problema solicitado, incluyendo elementos que permiten operar con coordenadas en centímetros, dibujar cargas en la caja, escribir condiciones de borde incluyendo derivativas e integrar la Ecuación de Poisson.
Por otro lado el archivo *main.py* es el archivo ejecutable. Se utiliza para crear objetos caja, definir sus parámetros, iniciar las operaciones de cálculo y presentar los resultados.
La caja se representa usando dos matrices con resolución $h = 0.1$ [cm]. Una matriz se utiliza para almacenar los valores de voltaje y condiciones de borde y la otra almacena la distribución de carga en el espacio. En la Figura (1)
Esta solución posee la ventaja de almacenar la carga de inmediato en memoria. Así, para calcular el potencial en cada punto basta leer su casilla correspondiente de la matriz de carga para obtener su valor local. Con ellos se simplifica la implementación del programa y se adquiere más velocidad al privilegiar un acceso directo a memoria sobre otras alternativas como consulta de índices o árboles de decisión.

3.3 Algoritmo de relajación en 2D

El algoritmo de relajación consiste en calcular el valor de cada punto de una matriz usando los adyacentes durante sucesivas iteraciones, deteniéndose cuando se alcanza convergencia. Por convergencia se entiende el instante en que la diferencia entre dos iteraciones es menor a una determinada tolerancia para todos los puntos. En el caso de la ecuación de poisson, cada iteración del algoritmo de relajación calcula el voltaje de la celda (i, j) usando la ecuación (9). Para resaltar el efecto de las cargas se

escoge $\epsilon = 5 \cdot 10^{-4}$ [F/m].

$$V_{i,j}^{next} = \frac{1}{4}(V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1} + h^2 \frac{\rho_{i,j}}{\epsilon}) \tag{9}$$

En el programa se implementa una modificación a este algoritmo, que se puede observar en la fórmula (10). Con ésta se aprovecha que el cálculo de dos de los términos de la siguiente iteración se realizaron con anterioridad, que junto con el nuevo parámetro w permiten acelerar la convergencia.

$$V_{i,j}^{next} = (1-w)V_{i,j} + \frac{w}{4}(V_{i+1,j} + V_{i-1,j}^{next} + V_{i,j+1} + V_{i,j-1}^{next} + h^2\rho_{i,j}) \tag{10}$$

Se escoge una tolerancia de 10^{-7} para determinar la finalización del algoritmo. Si se realizan más de 800 iteraciones se asume que no hay convergencia bajo los parámetros entregados y se interrumpe la ejecución.

3.4 Condición de borde derivativa

Para implementar la condición de borde derivativa se discretiza la derivada $\frac{dV}{dy} = 1$, quedando:

$$\frac{(y_{i,j_0+1} - y_{i,j_0})}{h} = 1 \rightarrow y_{i,j_0+1} = h + y_{i,j_0} \tag{11}$$

Donde j_0 es el índice de la casilla inmediatamente inferior a la línea $y = -5.5$ [cm], y los i corresponden a los elementos ubicados en $x = [-3 : 3]$ [cm]. Luego de cada iteración del algoritmo de relajación se impone nuevamente la condición de borde, recalculando y_{i,j_0+1}. Como $h = 0.1$, la línea queda justo entre dos casillas, evitándose que queden puntos sobre ella. Esto simplifica la implementación ya que no hay que calcular potencial justo encima de la condición de borde.

4 Resultados

Los resultados del algoritmo de cálculo pueden verse en la Figura (2). Como se indicó en la sección 3.3, se escoge un ϵ tal que el potencial causado por las cargas pueda apreciarse junto a la condición de borde.

5 Conclusiones

El algoritmo de relajación ha mostrado ser una herramienta simple y bastante eficaz para resolver el problema de integración de la ecuación de Poisson. Se debe notar que si se utilizase una distribución de carga con dependencia temporal $\rho(\vec{x}, t)$, el algoritmo de relajación puede ser aplicado para calcular el potencial de la caja en cada instante de la malla de tiempos, permitiendo simular sistemas con comportamiento dinámico en régimen electro cuasiestático.
Además se puede intuir que implementar este algoritmo en sistemas de tres dimensiones debiese ser análogo, lo que abre posibilidades de aplicación de veriones mejoradas de este programa en áreas como la ingeniería, en particular para asistir el diseño de condensadores y elementos similares que operen en bajas frecuencias.
En el área de software, tanto el algoritmo como el uso de la caja de cargas permiten concluir que una buena forma de acelerar los cálculos es evitar el cálculo redundante aprovechando siempre que se pueda la posibilidad de utilizar datos almacenados en la memoria.

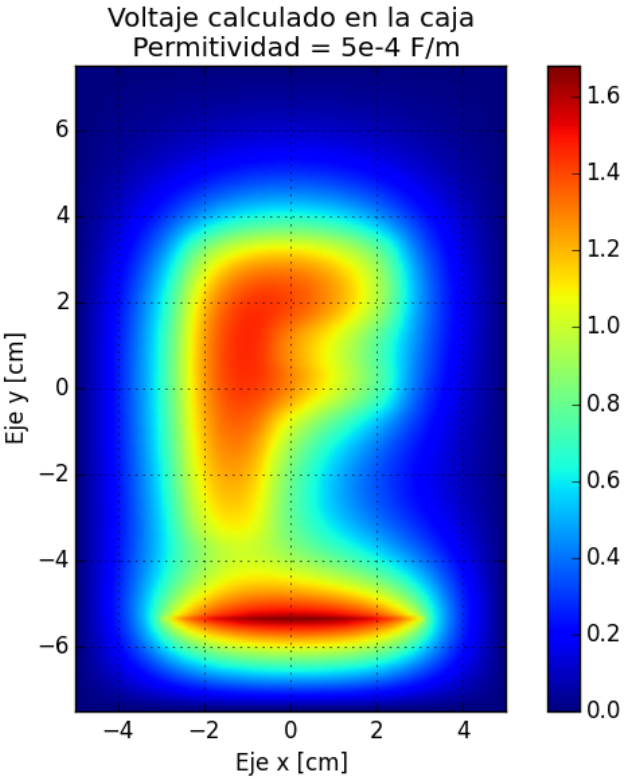

Figura 2: Resultado del algoritmo de integración. Se observa claramente el efecto de la distribución de las cargas y las condiciones de borde en el potencial.