Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

IE-0117 Programación Bajo Plataformas Abiertas

MSc. Andrés Mora Zúñiga - II Ciclo 2020

Examen Final

Resolver Osciladores Caóticos con Runge-Kutta de Orden 4

Instrucciones Generales:
El examen se debe realiar de manera individual.
Entregue un archivo comprimido que incluya un directorio llamado `src` con los archivos `.h`, y `.c` que lleven a la solución.
El examen debe entregarse a más tardar el 6 de Diciembre a las 08:00.

1. Introducción

El siguiente examen tiene como objetivo comprobar los conocimientos adquiridos durante el curso. Específicamente se evaluará el buen manejo del lenguaje de programación C y el uso de la línea de comandos de Linux. Los temas a evaluar son: uso de archivos de encabezado y archivos de implementación (.h y .c respectivamente), funciones, buen manejo de punteros y memoria dinámica, correcta implementación en entrada/salida de archivos y por supuesto la funcionalidad del código.

Para ello se presenta como ejercicio académico la simulación de sistemas de ecuaciones diferenciales que modelan sistema caóticos, implementando el método Runge-Kutta de orden 4 (RK4) en el lenguaje de programación C. Debido a que ecuaciones diferenciales y métodos numéricos no son requisito, ni contenido del curso, se le brinda al estudiante una pequeña nota teórica acerca de los osciladores caóticos y el método numérico a implementar.

2. Nota teórica

2.1. Osciladores caóticos

La teoría del caos ha sido un fuerte objeto de estudio en varios ámbitos de la ciencia en las últimas cinco décadas. Por ejemplo en la mecánica, el doble péndulo invertido presenta un comportamiento caótico [1] si se deja caer desde diferentes posiciones o en la electrónica, el circuito de Chua [2] presenta también un comportamiento caótico y se les conoce como osciladores caóticos. Este tipo de sistemas son dinámicos y pueden ser descritos matemáticamente mediante un conjunto de ecuaciones diferenciales ordinarias. Son altamente sensibles a sus condiciones iniciales y son poco predecibles, por lo que tienen gran utilidad y aplicaciones en cifrado de telecomunicaciones.

2.2. Oscilador caótico de Lorenz: variante Tee/Salleh

El oscilador caótico de Lorenz en su variante Tee/Salleh se encuentra modelado por el sistema de ecuaciones diferenciales ordinarias mostrado en (1).

$$\begin{aligned} x'(t) &= a(by(t) - x(t)) \\ y'(t) &= cx(t) - x(t)z(t) \\ z'(t) &= x(t)y(t) - bz(t) \end{aligned} \tag{1}$$

Donde a, b y c parámetros del sistema (constantes), y las funciones del tiempo x(t), y(t) y z(t) son los estados. Con los valores $a = 10$, $b = 4,85$ y $c = 10$, condiciones iniciales $x(0) = 2,3$, $y(0) = -1,3$, $z(0) = 10$ y tiempos desde $t = 0$ hasta $t = 100$ la evolución de los estados se puede apreciar en la Figura 1.

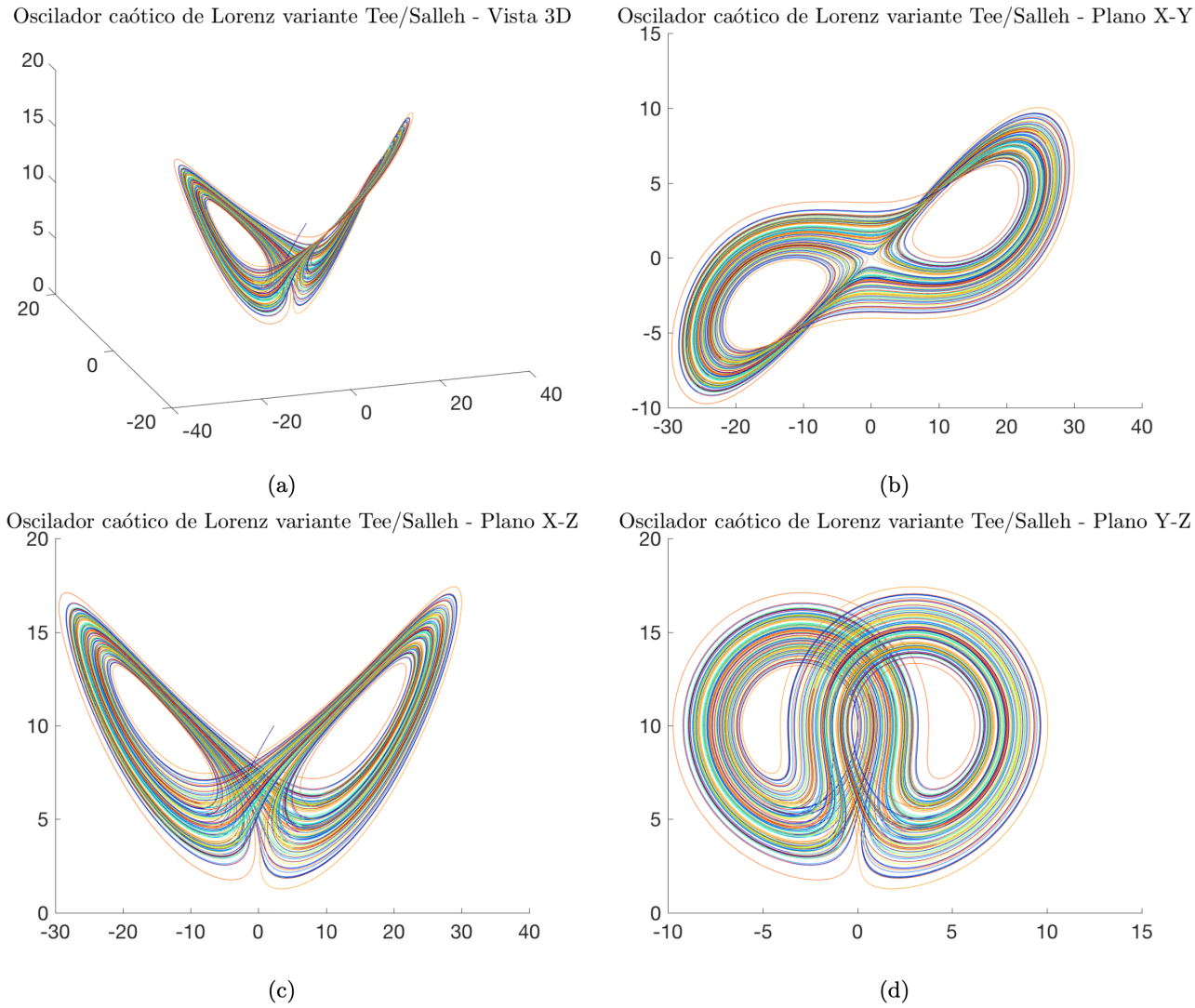

Figura 1: Visualizaciones del oscilador caóotico de Lorenz en su variante Tee/Salleh

2.3. Método numérico Runge-Kutta de Orden 4

Los métodos de la familia Runge-Kutta están hechos para resolver ecuaciones diferenciales ordinarias de la forma mostrada en (2).

$$\frac{dy}{dt} = f(t, y) \tag{2}$$

Existen varios métodos que resuelven este tipo de ecuaciones, como el método de Euler y los de Taylor. Sin embargo, los métodos de la familia RK logran la precisi óon de los métodos de Taylor sin la necesidad de calcular previamente las derivadas de orden superior.

En general, un método RK tienen la forma generalizada de la ecuación (3). En ella se aproxima el nuevo valor a partir de una funcióon incremental ϕ, la cual puede ser interpretada como la pendiente en el intervalo del paso h. La forma general de la función ϕ se muestra en (4).

$$v_{i+1} = v_i + \phi(t_i, v_i, h)h \tag{3}$$

$$\phi = a_1k_1 + a_2k_2 + a_3k_3 + \cdots + a_nk_n \tag{4}$$

El RK4 utiliza (5) como solución a una ecuación diferencial ordinaria. Dicha ecuación fue extraída de [3]. (Las ecuaciones fueron simplificadas eliminando la variable de tiempo para facilidad de los estudiantes, debido a que aunque sus estados dependen del tiempo, las ecuaciones del sistema mostrado en (1) no dependen directamente del tiempo, sino de la evolución de sus estados).

$$v_{t+1} = v_t + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h \tag{5}$$

Donde v_t es el valor actual del estado, v_{t+1} es el siguiente valor del estado y los valores para cada una de las k vienen dadas por:

$$\begin{aligned} k_1 &= f\left(v_t\right) \\ k_2 &= f\left(v_t + \frac{1}{2}k_1h\right) \\ k_3 &= f\left(v_t + \frac{1}{2}k_2h\right) \\ k_4 &= f\left(v_t + k_3h\right) \end{aligned}$$

En un sistema de ecuaciones diferenciales, no se tiene una única función, sino que se cuenta con dos o más que se deben resolver en conjunto debido a que la evolución de una de sus variables depende de las demás. El RK4 se puede extender fácilmente a sistemas como el mostrado en (1). Simplemente se debe de resolver cada ecuación como si fuera independiente, por lo que se debe calcular un k_{1x}, k_{1y} y k_{1z} y así sucesivamente.

RK4 logra simular el sistema de ecuaciones haciendo pequeños pasos de tamaño h hasta llegar a un tiempo máximo de simulación t_{max}. Por lo que si se tiene que llegar hasta $t_{max} = 10s$ y el paso es $h = 0{,}01$ se simularán los tiempos todos entre 0 y 10 en pequeños aumentos de 0.01. Por ejemplo:

- 0.01
- 0.02
- 0.03
- ...
- 9.98
- 9.99
- 10.00

El resultado de la simulación de RK4 para el oscilador antes mencionado es un arreglo de de estados que si se grafican como puntos unidos por líneas dan el resultado que se observa en la Figura 1.

3. Desarrollo

El estudiante deberá de implementar en el lenguaje C una programa capaz de realizar la simulación del oscilador caótico de Lorenz: variante Tee/Salleh utilizando el método numérico Runge-Kutta de Orden 4 y guardar el resultado de la simulación en el tiempo en un archivo de texto plano.

3.1. Código fuente

La implementación que debe realizar el estudiante consta de los siguiente archivos de código fuente:

1. `examen.h`: archivo de encabezado con la declaración de tipos de datos personalizados y las firmas de todas la funciones que se deben de implementar en el examen.
   (15 puntos)

2. `examen.c`: código fuente con la implementación de las funciones descritas en la siguiente sección.
   (75 puntos)

3.2. Requerimientos

En esta sección se enumeran los distintos tipos de datos por definir y las funciones MÍNIMAS a implementar para resolver el examen. El estudiante debe de ordenar su código según lo estipulado en la sección anterior y debe mantener el orden para que su código sea legible y fácil de entender. **Nota:** puede crear más funciones si lo considera necesario para pasos intermedios del programa.

- Tipo de dato `estado`: este tipo se va a utilizar para representar un estado del oscilador caótico. Almacena 3 double: x, y, z.
- Tipo de dato `RK4ks`: este tipo se va a utilizar para representar las 4 k que necesita el método RK4 para calcular el valor del siguiente estado. Almacena 4 double: k1, k2, k3, k4.
- Función `osciladorPrima`: está función recibe y retorna una variable de tipo estado. Su función es evaluar las derivadas del sistema mostrado en (1). Para el sistema, utilice los siguientes valores para sus constantes:

$$a = 10{,}0$$
$$b = 4{,}85$$
$$c = 10{,}0$$

- Función `obtenerKs`: esta función recibe un puntero a un bloque que almacena variables del tipo RK4ks, una variable con el estado actual, y un double con el paso (h). La función calcula las k para cada uno de los estados del sistema y las almacena en el puntero que recibe como parámetro: posición 0: ks para x, posición 1: ks para y, posición 2: ks para z.
- Función `siguienteEstado`: esta función recibe como parámetro el estado actual, el puntero al bloque de ks, el paso. Retorna el siguiente estado de la función utilizando la ecuación (5).
- Función `RK4Oscilador`: esta función recibe el estado inicial, el paso, el tiempo máximo de simulación y un puntero a un entero para poder regresar la cantidad de estados que simuló. Además su valor de retorno es un puntero que apunta a un bloque de memoria que contiene todos los estados intermedios que simuló hasta llegar al tiempo máximo t_{max}. Tiene como trabajo aplicar el método RK4 desde $t = 0$ hasta $t = t_{max}$. Reserva memoria para todos los estados intermedios que tiene que calcular, los almacena, y regresa el puntero. Además reserva memoria para el bloque que va a almacenar 3 variables RK4ks que se necesitan y libera este segundo bloque de memoria cuando ya no sea necesario. Utiliza las funciones `obtenerKs` y `siguienteEstado` para sus respectivas funcionalidades.

- Función `exportarEstados`: esta función recibe, la ruta donde se desea exportar el archivo, el puntero de los estados y la cantidad de estados calculados. Con estos parámetros procede a escribir en un archivo todos los estados calculados (en orden y con 4 decimales) con el siguiente formato:

  ```
  x_1, y_1, z_1
  x_2, y_2, z_2
  ...
  x_tmax, y_tmax, z_tmax
  ```

  Donde:

  - x_t es el estado x en el tiempo t
  - y_t es el estado y en el tiempo t
  - z_t es el estado z en el tiempo t

4. Programa Principal

El programa principal (`main.c`) es proporcionado por el profesor. No puede ser modificado. Asegúrese de que las funciones reciban los parámetros solicitados para que no haya ninguna incompatibilidad en los parámetros.

Nota: puede usar programas principales temporales mientras desarrolla, pero la entrega final debe utilizar el archivo `main.c` proporcionado.

5. Verificación de resultados desde la línea de comandos de Linux

(10 puntos)

Instale (si no tiene instalado ya) `python3` en su distribución de Linux. Proceda a instalar `matplotlib` para `python3` utilizando pip (pip3).

Una vez instalados, ejecute el archivo `plot.py` proporcionado con la siguiente línea:

```
python3 plot.py salida.txt
```

Nota: la línea podría ser sólamente: *python plot.py salida.txt* en caso de que `python3` sea el único Python instalado en su sistema.

Asegúrese de que el archivo ploy.py y salida.txt esten en el mismo directorio desde donde se está ejecutando el comando anterior. El resultado mostrado en pantalla debería verse muy similar al de la Figura 1.

Adjunte una imagen de la gráfica resultante a su solución del examen.

6. Referencias

[1] T. Shinbrot, C. Grebogi, J. Wisdom, and J. A. Yorke, "Chaos in a double pendulum," American Association of Physics Teachers, vol. 60, no. 6, p. 491–499, 1992.

[2] J. Lü and G. Chen, "Generatin multiscroll chaotic attractors: Theories, methods and applications," International Journal of Bifurcation and Chaos, vol. 16, no. 4, p. 775–858, 2006.

[3] S. Chapra, Numerical Methods for Engineers, 7th ed. 2 Penn Plaza, New York, NY 10121: McGraw-Hill Higher Education, 2014.