Osciladores de Van der Pol Acoplados

Documentación de Simulación de Sistemas No Lineales

29 de octubre de 2025

1. Introducción: El Oscilador de Van der Pol

El oscilador de Van der Pol es un sistema dinámico no lineal con un ciclo límite estable. La ecuación diferencial de segundo orden es:

$$\ddot{x} - \mu(1 - x^2)\dot{x} + \omega_0^2 x = 0$$

donde μ controla la no-linealidad y la amortiguación, y ω_0 es la frecuencia natural.

En la simulación, se resuelve un sistema de dos osciladores acoplados linealmente mediante una fuerza de tipo resorte. El sistema implementado se describe mediante ecuaciones de primer orden:

$$\begin{aligned} \frac{dx_i}{dt} &= v_i \\ \frac{dv_i}{dt} &= \mu_i(1 - x_i^2)v_i - \omega_{0i}^2 x_i + F_{\text{acoplamiento},i} \end{aligned}$$

donde la fuerza de acoplamiento es $F_{\text{acoplamiento},1} = k(x_2 - x_1)$ y $F_{\text{acoplamiento},2} = k(x_1 - x_2)$.

2. Método Numérico: Integración RK4

La integración temporal de las ecuaciones diferenciales del sistema acoplado se realiza utilizando el método de Runge-Kutta de Cuarto Orden (RK4). Este método es explícito, de orden $O(\Delta t^4)$, lo que proporciona una alta precisión y estabilidad para la dinámica de este tipo de sistemas no lineales.

El paso de integración para la variable de posición x es:

$$x(t + \Delta t) = x(t) + \frac{\Delta t}{6}(k_{1x} + 2k_{2x} + 2k_{3x} + k_{4x})$$

Un método similar se aplica para la velocidad v. El código implementa este algoritmo resolviendo las derivadas en cuatro puntos intermedios (k_1 a k_4).

3. Análisis de Resultados

Definición de Régimen (Parámetros)

Para la simulación se utilizó un régimen de osciladores idénticos con acoplamiento moderado.

- **Parámetros Individuales:**
$$\mu_1 = \mu_2 = 1{,}0, \quad \omega_{01} = \omega_{02} = 1{,}0$$
- **Constante de Acoplamiento:**
$$k = 0{,}5$$
- **Condiciones Iniciales:**
$$(x_1(0), v_1(0)) = (1{,}0, 0{,}0), \quad (x_2(0), v_2(0)) = (0{,}5, 0{,}0)$$
- **Parámetros de Simulación:**
$$T_f = 200{,}0 \text{ s}, \quad \Delta t = 0{,}01 \text{ s}$$

Validación de Integración (Paso de Tiempo)

El método RK4, con un paso de tiempo de $\Delta t = 0{,}01$ y un tiempo final $T_f = 200{,}0$ s, es adecuado para la dinámica del ciclo límite de Van der Pol, cuya frecuencia natural es $\omega_0 = 1{,}0$.

- La **precisión** del método RK4 asegura que la trayectoria del ciclo límite sea conservada sin desviaciones a largo plazo.

Caracterización de Sincronización y Caos

El objetivo de simular osciladores acoplados es estudiar el fenómeno de **sincronización**. Dado que los parámetros (μ, ω_0) son idénticos y el acoplamiento ($k = 0{,}5$) es moderado, se espera observar **sincronización de fase y amplitud** después de un transitorio inicial.

- **Sincronización (Gráfica x vs. t):** El archivo `x_vs_t.png` mostrará la evolución de $x_1(t)$ y $x_2(t)$. La sincronización se confirma si ambas trayectorias convergen a la misma amplitud y fase, es decir, $|x_1(t) - x_2(t)| \to 0$ para $t \to \infty$.
- **Sincronización (Curva de Lissajous):** El archivo `lissajous.png` (gráfico de x_1 vs. x_2) mostrará el estado estacionario.
  - **Sincronización completa:** El gráfico convergirá a la línea recta $x_1 = x_2$.
  - **Sincronización de fase (cuasi-periódica):** El gráfico mostraría una curva cerrada pero compleja o una banda estrecha, lo cual no se espera en este régimen idéntico.

---

4. Estructura de Carpetas del Proyecto

El proyecto está organizado de manera modular para separar la lógica de simulación, la configuración y el post-procesado (visualización).

- **Carpeta Raíz:** Contiene el archivo de compilación principal (e.g., `Makefile`) y los directorios principales.
- `src/` (Source): Contiene todo el código fuente en C++ (clases y programa principal).
- `scripts/` (Scripts): Contiene archivos para el post-procesado, como scripts de Python o Gnuplot.
- `results/` (Resultados): Directorio donde se almacenan los datos de salida (`trayectorias.dat`).

```
/billar
|-- build/
|-- include/
|   |-- VanDerPol.h
|   |-- Sistema.h
|-- scripts/
|   |-- gnuplot/
|   |-- |-- plot.gp
|   |-- python
|   |-- |-- plot.py
|-- results/
|   |-- datos.txt
|   |-- lissajous_py.png
|   |-- x_vs_t.png
|-- src/
|   |-- Sistema.cpp
|   |-- VanDerPol.cpp
|   |-- main.cpp
|-- CMakeLists.txt
|-- Doxyfile
```

5. Arquitectura Orientada a Objetos: Osciladores de Van der Pol Acoplados

La simulación se implementa mediante un diseño de dos clases principales: `VanDerPol`, que modela el oscilador individual, y `Sistema`, que gestiona el acoplamiento y el proceso de integración.

5.1. Resumen de Clases y Atributos

Clase	Atributos Clave (Públicos/Privados)	Propósito
`VanDerPol`	`x`, `v` (Estado) `mu`, `w0` (Parámetros)	Modelar un único oscilador de Van der Pol. Realizar el paso de integración (RK4).
`Sistema`	`o1`, `o2` (`VanDerPol`) `k` (Constante de acoplamiento)	Gestionar los dos osciladores acoplados. Calcular la fuerza de acoplamiento y ejecutar la simulación.

Cuadro 1: Clases principales del proyecto.

5.2. Clases utilizadas

El diagrama a continuación ilustra las relaciones de composición y dependencia.

- **Composición (`Sistema` a `VanDerPol`):** La clase `Sistema` contiene explícitamente a los dos objetos `VanDerPol` (`o1` y `o2`) como miembros de datos.
- **Dependencia (`main` a `Sistema`):** La función principal configura y ejecuta la simulación a través del método `simulate` de la clase `Sistema`.

Listing 1: Diagrama de Clases (UML Simplificado)

```
classDiagram
    direction LR

    class VanDerPol {
        +x, v : double (Estado)
        +mu, w0 : double (Par metros)
        +set_state(x0, v0)
        +derivatives(t, F_couple, dxdt, dvdt)
        +rk4_step(t, dt, F_couple)
    }

    class Sistema {
        +o1 : VanDerPol
        +o2 : VanDerPol
        +k : double (Constante de acoplamiento)
        +Sistema(mu1, w01, mu2, w02, k_)
        +simulate(t0, tf, dt, outfile)
    }

    Sistema "1" *-- "2" VanDerPol : Contiene (Composici n)
    main ..> Sistema : Usa para simular
```

Código

A continuación, se presentan los listados del código fuente de las clases principales del proyecto.

Archivo `main (1).cpp`

Contiene los parámetros de simulación y las condiciones iniciales.

```
#include <iostream>
#include <string>
#include <filesystem>
#include "Sistema.h"


int main(int argc, char **argv) {
// par metros por defecto (ejemplo)
double mu1 = 1.0, w01 = 1.0;
double mu2 = 1.0, w02 = 1.0;
```

```cpp
double k = 0.5;
double t0 = 0.0, tf = 200.0, dt = 0.01;


// condici n iniciales
double x10 = 1.0, v10 = 0.0;
double x20 = 0.5, v20 = 0.0;


std::string outdir = "../results";
std::filesystem::create_directories(outdir);
std::string outfile = outdir + "/datos.txt";


Sistema S(mu1, w01, mu2, w02, k);
S.o1.set_state(x10, v10);
S.o2.set_state(x20, v20);


std::cout << "Simulando... t=[" << t0 << "," << tf << "] dt=" << dt << " k=" << k << "\n";
S.simulate(t0, tf, dt, outfile);
std::cout << "Datos guardados en: " << outfile << std::endl;
return 0;
}
```

Archivo `VanDerPol.cpp`

Implementa la clase del oscilador individual y el método de integración RK4.

```cpp
#include "VanDerPol.h"
#include <cmath>


VanDerPol::VanDerPol(double mu_, double w0_) : x(0.0), v(0.0), mu(mu_), w0(w0_) {}


void VanDerPol::set_state(double x0, double v0) {
x = x0; v = v0;
}


void VanDerPol::derivatives(double /*t*/, double F_couple, double &dxdt, double &dvdt) const {
dxdt = v;
dvdt = mu * (1.0 - x * x) * v - (w0 * w0) * x + F_couple;
}


void VanDerPol::rk4_step(double t, double dt, double F_couple) {
double k1x, k1v;
double k2x, k2v;
double k3x, k3v;
double k4x, k4v;


derivatives(t, F_couple, k1x, k1v);
// mid 1
double x_temp = x + 0.5 * dt * k1x;
double v_temp = v + 0.5 * dt * k1v;
// temporary oscillator for mid evaluation (we only need derivatives formula)
double dxdt_mid, dvdt_mid;
// For the mid evaluations we assume F_couple stays constant over the small dt (explicit RK4)
// but the state used is the mid-state
// compute k2
dxdt_mid = v_temp;
dvdt_mid = mu * (1.0 - x_temp * x_temp) * v_temp - (w0 * w0) * x_temp + F_couple;
```

```
k2x = dxdt_mid; k2v = dvdt_mid;


// k3
x_temp = x + 0.5 * dt * k2x;
v_temp = v + 0.5 * dt * k2v;
dxdt_mid = v_temp;
dvdt_mid = mu * (1.0 - x_temp * x_temp) * v_temp - (w0 * w0) * x_temp + F_couple;
k3x = dxdt_mid; k3v = dvdt_mid;


// k4
x_temp = x + dt * k3x;
v_temp = v + dt * k3v;
dxdt_mid = v_temp;
dvdt_mid = mu * (1.0 - x_temp * x_temp) * v_temp - (w0 * w0) * x_temp + F_couple;
k4x = dxdt_mid; k4v = dvdt_mid;


x += dt * (k1x + 2.0 * k2x + 2.0 * k3x + k4x) / 6.0;
v += dt * (k1v + 2.0 * k2v + 2.0 * k3v + k4v) / 6.0;
}
```

Archivo `Sistema (1).cpp`

Gestiona el acoplamiento y el bucle de simulación.

```
#include "Sistema.h"
#include <fstream>
#include <iostream>


Sistema::Sistema(double mu1, double w01, double mu2, double w02, double k_) : o1(mu1, w01),
    o2(mu2, w02), k(k_) {}


void Sistema::simulate(double t0, double tf, double dt, const std::string &outfile) const {
// copia local de osciladores para iterar
VanDerPol a = o1;
VanDerPol b = o2;


std::ofstream ofs(outfile);
if (!ofs) {
std::cerr << "No se pudo abrir archivo de salida: " << outfile << std::endl;
return;
}


ofs << "# t x1 v1 x2 v2" << std::endl;


double t = t0;
while (t <= tf + 1e-12) {
// escribir
ofs << t << " " << a.x << " " << a.v << " " << b.x << " " << b.v << "\n";


// calcular fuerzas de acoplamiento (lineal: k*(xj - xi))
double F1 = k * (b.x - a.x);
double F2 = k * (a.x - b.x);


// RK4 paso para ambos (use las mismas fuerzas en cada subpaso)
a.rk4_step(t, dt, F1);
```

```
38 b.rk4_step(t, dt, F2);
39
40
41 t += dt;
42 }
43
44
45 ofs.close();
46 }
```

6. Discusión y Conclusiones

Esta sección resumirá los hallazgos tras el post-procesado de los datos en `datos.txt`, analizando las figuras `x_vs_t.png` y `lissajous.png`.

Resultados de la Sincronización

Se debe concluir si el acoplamiento $k = 0{,}5$ fue suficiente para lograr la sincronización completa entre los dos osciladores idénticos.

- La gráfica `x_vs_t.png` (posición vs. tiempo) debería mostrar que las trayectorias $x_1(t)$ y $x_2(t)$ se superponen a partir de un tiempo T_{sinc} (tiempo de sincronización).
- La **curva de Lissajous** (`lissajous.png`) que representa x_1 vs. x_2 en estado estacionario, debería ser una **línea diagonal perfecta** ($x_1 = x_2$), confirmando la sincronización de amplitud y fase.

Conclusiones Generales

El proyecto demuestra una implementación exitosa del método RK4 para integrar sistemas de ecuaciones diferenciales no lineales. Se confirma que los osciladores de Van der Pol idénticos, con acoplamiento lineal, son susceptibles a la sincronización completa. El sistema permanece en un régimen no caótico, como lo confirman las oscilaciones periódicas (ciclos límite) que convergen a un atractor simple.