

Física Computacional

Alberto Pérez Muñozuri (despacho107)

Contacto:

alberto.perez.munuzuri@usc.es

fisicacomputacional2020@gmail.com

Referencias del Capítulo:

- Numerical Recipes. W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling. Cambridge University Press (1988).
- Computational Techniques for Fluid Dynamics. C.A.J. Fletcher. Springer-Verlag (1991).

NORMAS:

- Básicamente igual que con Moncho y Diego
- Control de asistencia por firma en el listado.
- Programación en Python (o Matlab)
- Evaluación continua: boletines entregados, controles (3 o 4, se avisara unos días antes de cada), defensa de los boletines entregados, asistencia
- Examen final opcional, para los que no hagan completa la evaluación continua, en la fecha prevista por el rectorado.
- Todos los boletines se enviarán a:

fisicacomputacional2020@gmail.com

Ecuaciones Diferenciales Ordinarias (ODE)

Cualquier ecuación diferencial ordinaria se puede reducir a un conjunto de ecuaciones diferenciales ordinarias de primer orden:

$$\text{Ej.: } \frac{d^2x}{dt^2} + q(t)\frac{dx}{dt} = r(t) \Rightarrow \begin{aligned} \frac{dx}{dt} &= y(t) \\ \frac{dy}{dt} &= r(t) - q(t)y \end{aligned}$$

Se procede de modo equivalente para ecuaciones de orden superior. Por tanto, consideramos el caso de ecuaciones diferenciales de primer orden.

$$\frac{dx_i}{dt} = f_i(t, x_1, \dots, x_N) \quad i = 1, \dots, N$$

Para tener el problema correctamente planteado y tener una solución única al mismo necesitamos conocer las condiciones de frontera (boundary conditions) , o condición inicial (initial conditions):

$$x_i(t = t_0) = A_i \quad i = 1, \dots, N$$

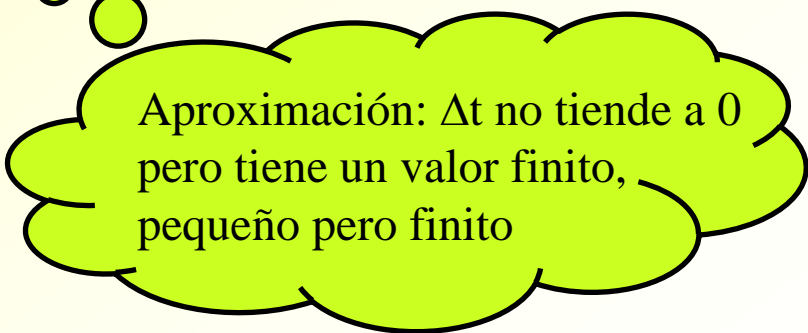
Ec. Diferenciales Ordinarias: Método de Euler

Consideremos el siguiente problema: $\frac{dx}{dt} = \text{sen}(x)$

El ordenador no sabe como hacer derivadas, solo sabe hacer sumas. Aplicamos conceptos vistos en la parte anterior para manipular esta ecuación.

Aplicamos la definición de derivada:

$$\frac{dx}{dt} = \lim_{\Delta t \rightarrow 0} \frac{x(t + \Delta t) - x(t)}{\Delta t} \approx \frac{x(t + \Delta t) - x(t)}{\Delta t}$$



Aproximación: Δt no tiende a 0 pero tiene un valor finito, pequeño pero finito

$$\Rightarrow x(t + \Delta t) = x(t) + \Delta t \text{ sen}(x(t))$$

Podemos conocer el siguiente valor de la variable x a partir de lo que valía un instante de tiempo Δt antes.

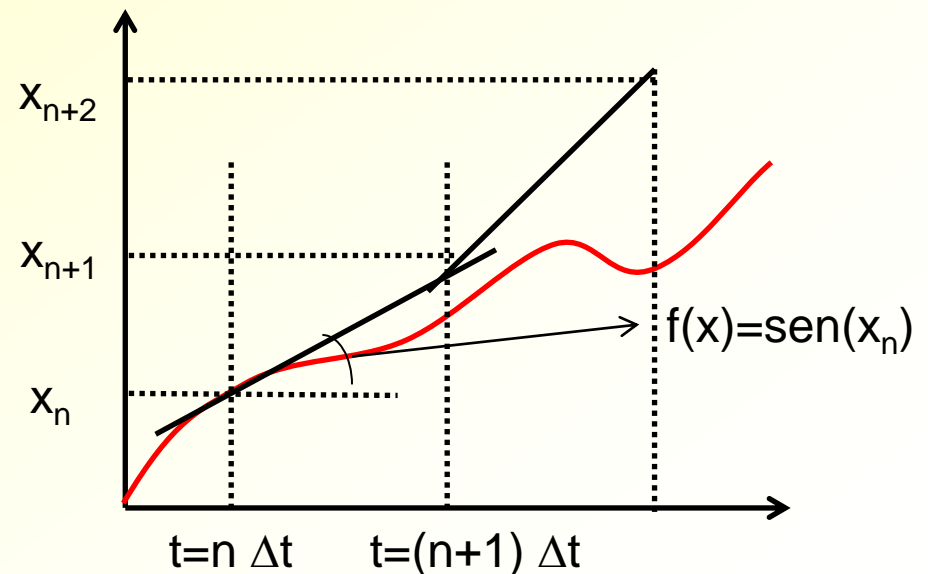
Introducimos la siguiente notación:

$$x(t = n \Delta t) = x_n$$

De modo que la ecuación diferencial anterior queda de la forma

que ya es una ecuación algebraica que el ordenador sabe resolver.

Interpretación geométrica:



Ec. Diferenciales Ordinarias: Método de Euler

Error del esquema de integración para la ecuación general: $\frac{dx}{dt} = f(x, t)$

$$\begin{aligned}x_{n+1} &= x_n + \Delta t f(x_n, t_n) \\ &= x(t + \Delta t)\end{aligned}$$

$$\approx x(t) + \Delta t \left. \frac{dx}{dt} \right|_{\Delta t=0} + O(\Delta t^2)$$

Desarrollo en
Serie de Taylor en
torno a $\Delta t=0$

$$= f(x(t), t)$$

Método de orden 1 \rightarrow
aproximación a primer orden \rightarrow
el error va con Δt^2

```
import numpy as np
import matplotlib.pyplot as plt
```

```
deltat=0.1
```

```
t=0
```

```
x=1
```

```
for i in range(100):
```

```
    t=t+deltat
```

```
    x=x+deltat*np.sin(x)
```

```
    plt.plot(t,x,'*')
```

```
    plt.xlabel('Tiempo')
```

```
    plt.ylabel('x')
```

% Probar diferentes valores de Δt .

% Probar diferentes condiciones iniciales, cada condición inicial da una solución

% diferente.

% Resolución numérica del problema $dx/dt = \sin(x)$ por el Método de Euler
close all; clear all;

deltat=0.01; % Valores de los parámetros
dimt=1000; % Numero total de iteraciones
x(1)=10; % Condición inicial

% Cálculo de las iteraciones sucesivas
for n=1:dimt-1
 x(n+1)=x(n)+deltat*sin(x(n));
end

% Dibuja la dependencia temporal
t=[1:dimt]*deltat;
figure(1);hold off;plot(x,'-','LineWidth',2.000);grid on;

% Probar diferentes valores de Δt .

% Probar diferentes condiciones iniciales, cada condición inicial da una solución
% diferente.



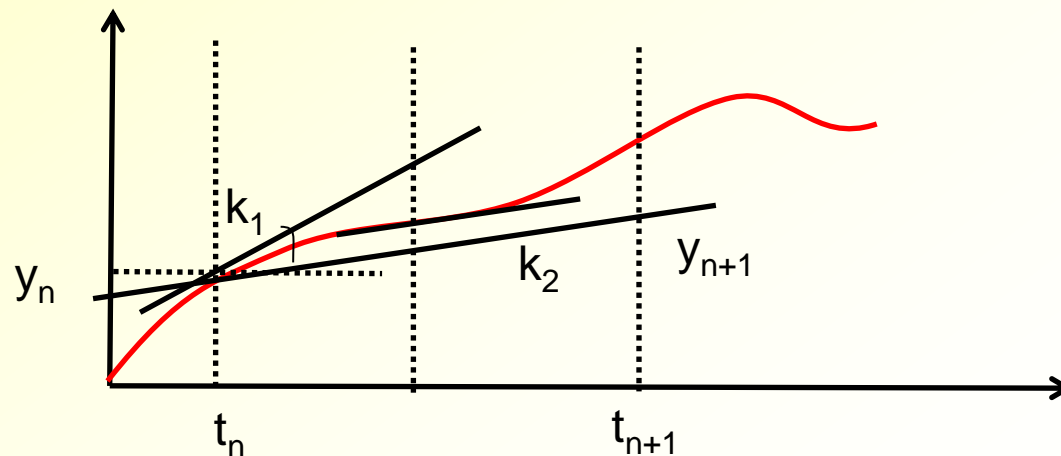
Ec. Diferenciales Ordinarias: Runge-Kutta 2º orden

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

$$k_1 = \Delta t f(t_n, y_n) \quad \text{Como en el método de Euler}$$

$$k_2 = \Delta t f\left(t_n + \frac{1}{2}\Delta t, y_n + \frac{1}{2}k_1\right) \quad k_1 \text{ se emplea para estimar la pendiente de la función en el punto medio del intervalo}$$

$$y_{n+1} = y_n + k_2 + O(\Delta t^3) \quad \text{empleamos } k_2 \text{ para calcular el valor en } \Delta t$$



```
import numpy as np
import matplotlib.pyplot as plt
```

```
deltat=0.1
```

```
t=0
```

```
x=1
```

```
for i in range(100):
```

```
    t=t+deltat
```

```
    k1=deltat*np.sin(x)
```

```
    k2=deltat*np.sin(x+k1/2.)
```

```
    x=x+k2
```

```
    plt.plot(t,x, '*')
```

```
    plt.xlabel('t')
```

```
    plt.ylabel('x')
```

```
plt.xlim(0,10)
```

```
plt.ylim(0,4)
```

```
% Probar diferentes valores de  $\Delta t$ .
```

```
% Probar diferentes condiciones iniciales, cada condición inicial da una solución
```

```
% diferente.
```

```
% Resolución numérica del problema  $dx/dt = \sin(x)$   
% por el Método de Runge-Kutta de 2º orden  
close all; clear all;
```

```
deltat=0.01; % Valores de los parámetros  
dimt=1000; % Numero total de iteraciones  
x(1)=10; % Condición inicial
```

```
% Cálculo de las iteraciones sucesivas
```

```
for n=1:dimt-1  
    k1=deltat*sin(x(n));  
    k2=deltat*sin(x(n)+k1/2);  
    x(n+1)=x(n)+k2;  
end
```

```
% Dibuja la dependencia temporal
```

```
t=[1:dimt]*deltat;  
figure(1);hold off;plot(x,'-', 'LineWidth',2.000);grid on;
```

```
% Probar diferentes valores de  $\Delta t$ .
```

```
% Probar diferentes condiciones iniciales, cada condición inicial da una solución  
% diferente.
```



Ec. Diferenciales Ordinarias: Runge-Kutta 4º orden

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

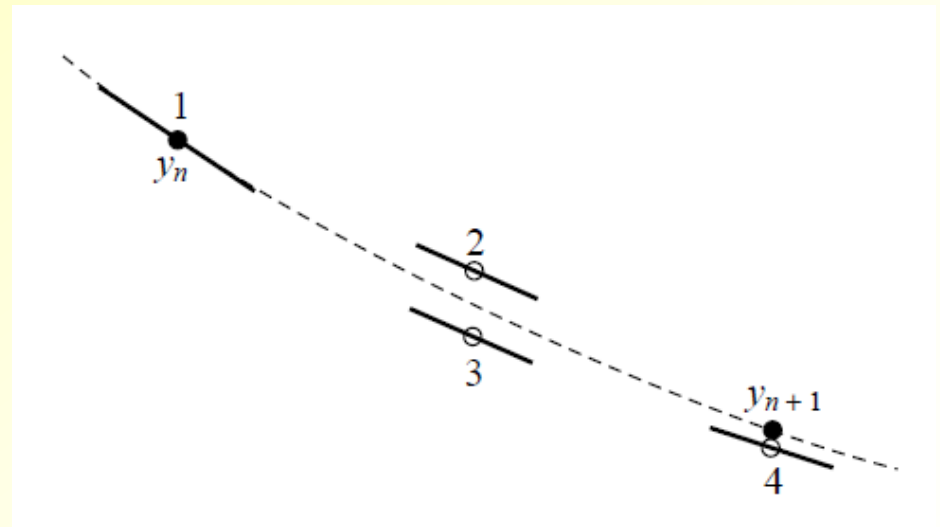
Es una combinación inteligente de pendientes que minimizan el error

$$k_1 = \Delta t f(t_n, y_n)$$

$$k_2 = \Delta t f\left(t_n + \frac{1}{2}\Delta t, y_n + \frac{1}{2}k_1\right)$$

$$k_3 = \Delta t f\left(t_n + \frac{1}{2}\Delta t, y_n + \frac{1}{2}k_2\right)$$

$$k_4 = \Delta t f(t_n + \Delta t, y_n + k_3)$$



$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(\Delta t^5)$$

```

1 from math import *
2 from matplotlib.pyplot import *
3
4 deltat=0.1
5 t=0
6 x=1
7 for i in range(100):
8     t=t+deltat
9     k1=deltat*sin(x)
10    k2=deltat*sin(x+k1/2.)
11    k3=deltat*sin(x+k2/2.)
12    k4=deltat*sin(x+k3)
13    x=x+k1/6.+k2/3.+k3/3.+k4/6.
14    plot(t,x, '*')
15    xlabel('t')
16    ylabel('x')
17
18 xlim(0,10)
19 ylim(0,4)

```

% Probar diferentes valores de Δt .

% Probar diferentes cond. inic., cada cond. inic. da una solución diferente.

```
% Resolución numérica del problema  $dx/dt = \sin(x)$   
% por el Método de Runge-Kutta de 4º orden  
close all; clear all;
```

```
deltat=0.01; % Valores de los parámetros  
dimt=1000; % Numero total de iteraciones  
x(1)=10; % Condición inicial
```

```
for n=1:dimt-1 % Cálculo de las iteraciones sucesivas  
    k1=deltat*sin(x(n));  
    k2=deltat*sin(x(n)+k1/2);  
    k3=deltat*sin(x(n)+k2/2);  
    k4=deltat*sin(x(n)+k3);  
    x(n+1)=x(n)+k1/6+k2/3+k3/3+k4/6;  
end
```

```
t=[1:dimt]*deltat; % Dibuja la dependencia temporal  
figure(1);hold off;plot(x,'-', 'LineWidth',2.000);grid on;
```

```
% Probar diferentes valores de Dt.
```

```
% Probar diferentes cond. inic., cada cond. inic. da una solución diferente.
```



```
% Resolución numérica del oscilador armónico amortiguado por el Método de Euler  
cd 'D:\Users\Alberto\clases\PostgradoDiana\Curso'
```

```
clear all; close all;
```

```
deltat=0.01; b=1; omega=1; % Valores de los parámetros
```

```
dimt=1000; % Numero total de iteraciones
```

```
x(1)=10; y(1)=-1; % Condición inicial
```

```
% Cálculo de las iteraciones sucesivas
```

```
for i=2:dimt
```

```
    x(i)=x(i-1)+deltat*y(i-1);
```

```
    y(i)=y(i-1)+deltat*(-b*y(i-1)-omega*omega*x(i-1));
```

```
end
```

```
% Dibuja los resultados en el espacio de fases
```

```
figure(1);hold off;plot(x,y,'-','LineWidth',2.000);grid on;
```

```
% Dibuja la dependencia temporal de x e y
```

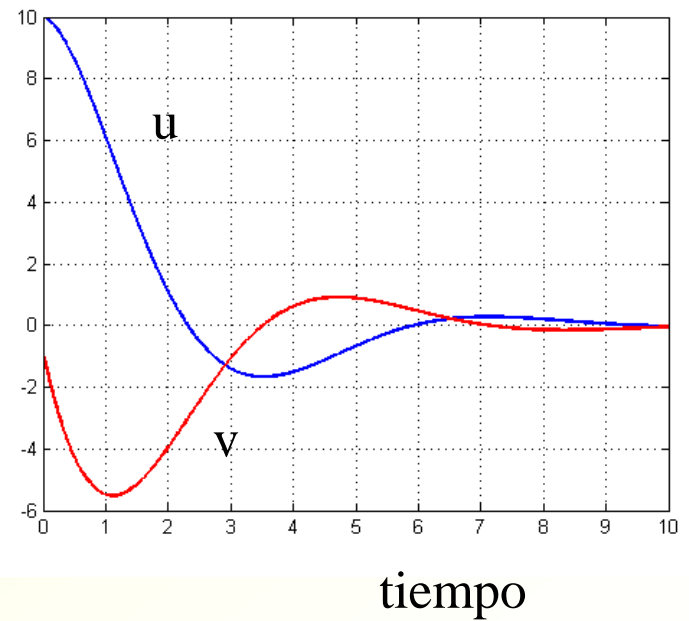
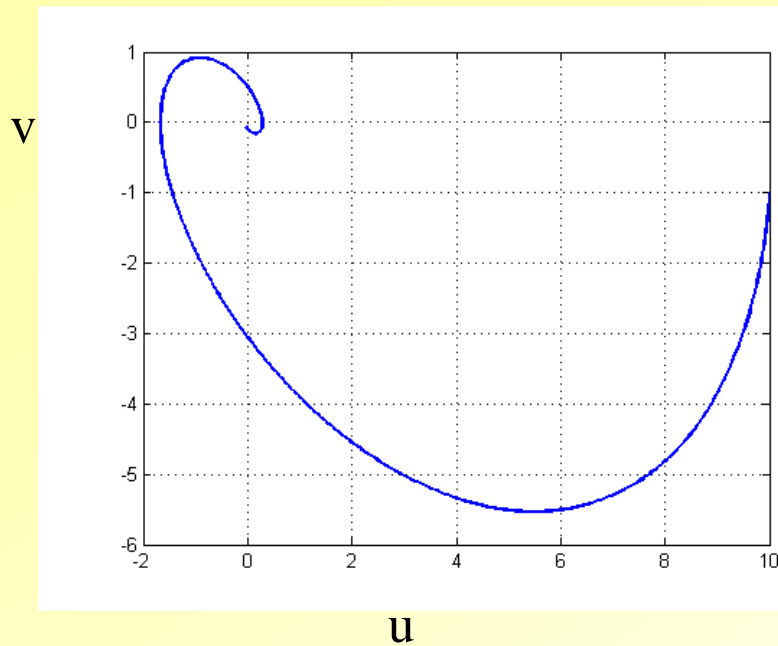
```
t=[1:dimt]*deltat;
```

```
figure(2);hold off;
```

```
plot(t,x,'-b','LineWidth',2.000);hold on;plot(t,y,'-r','LineWidth',2.000);grid on;
```



Ecuaciones Diferenciales Ordinarias



Ecuaciones Diferenciales Ordinarias

Métodos:

- Método de Euler
- Método de Runge-Kutta de 2º orden
- Método de Runge-Kutta de 4º orden
- Método Predictor-Corrector

Ejemplos:

- Resolución de la ecuación del oscilador armónico amortiguado

$$\ddot{x} + b \dot{x} + \omega_0^2 x = 0$$

- Resolución de la ecuación de Lorenz
(emplead los valores de los parámetros
 $\sigma = 3, r = 26.5, b = 1$)

$$\frac{dX}{dt} = \sigma (Y - X)$$

$$\frac{dY}{dt} = rX - Y - XZ$$

$$\frac{dZ}{dt} = XY - bZ$$

Ecuaciones Diferenciales Ordinarias

Solución oscilador armónico amortiguado:

$$\ddot{x} + b \dot{x} + \omega_o^2 x = 0$$

$$\dot{x} = y$$

$$\dot{y} = -b y - \omega_o^2 x \quad \Rightarrow \quad \begin{aligned} x(t + \Delta t) &= x(t) + \Delta t y(t) \\ y(t + \Delta t) &= y(t) + \Delta t (-b y(t) - \omega_o^2 x(t)) \end{aligned}$$

estas ecuaciones algebraicas ya se pueden resolver por ordenador (son sumas y restas que el ordenador entiende)

```
% Resolución numérica del oscilador armónico amortiguado por el Método de Euler  
cd 'D:\Users\Alberto\clases\PostgradoDiana\Curso'
```

```
clear all; close all;
```

```
deltat=0.01; b=1; omega=1; % Valores de los parámetros  
dimt=1000; % Numero total de iteraciones  
x(1)=10; y(1)=-1; % Condición inicial
```

```
% Cálculo de las iteraciones sucesivas
```

```
for i=2:dimt  
    x(i)=x(i-1)+deltat*y(i-1);  
    y(i)=y(i-1)+deltat*(-b*y(i-1)-omega*omega*x(i-1));  
end
```

```
% Dibuja los resultados en el espacio de fases
```

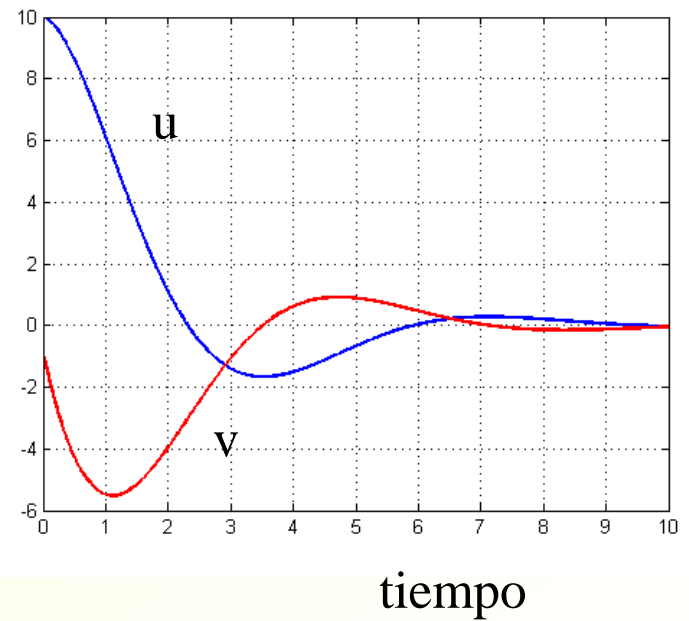
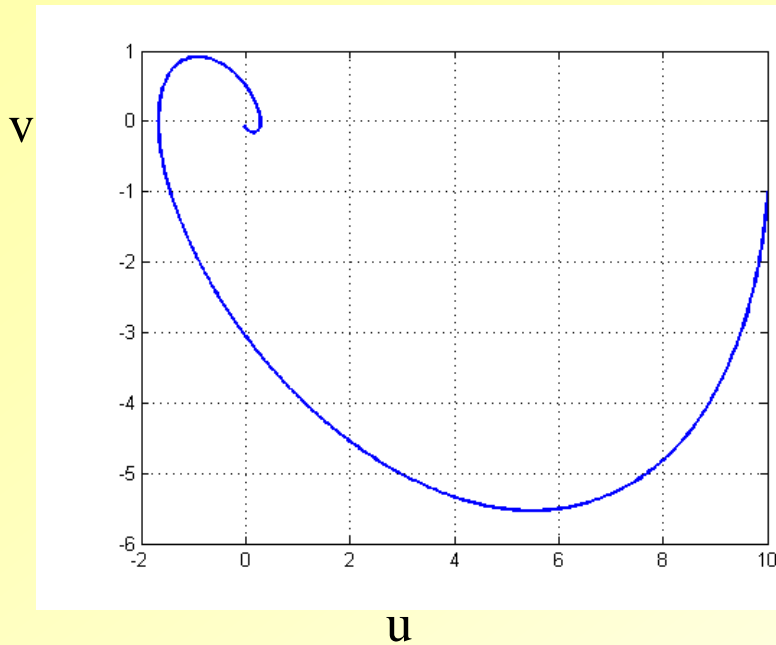
```
figure(1);hold off;plot(x,y,'-','LineWidth',2.000);grid on;
```

```
% Dibuja la dependencia temporal de x e y
```

```
t=[1:dimt]*deltat;  
figure(2);hold off;  
plot(t,x,'-b','LineWidth',2.000);hold on;plot(t,y,'-r','LineWidth',2.000);grid on;
```



M1: Métodos Numéricos en 1D



% Resolución numérico del oscilador armónico amortiguado
% por el Método de Runge-Kutta de 4º orden

cd 'D:\Users\Alberto\clases\PostgradoDiana\Curso'
clear all; close all;

deltat=0.1; b=1; omega=1; % Valores de los parámetros
dimt=100; % Numero total de iteraciones

x(1)=10; y(1)=-1; % Condición inicial
% Cálculo de las iteraciones sucesivas

for i=2:dimt

 kx1=deltat*y(i-1);

 ky1=deltat*(-b*y(i-1)-omega*omega*x(i-1));

 kx2=deltat*(y(i-1)+ky1/2);

 ky2=deltat*(-b*(y(i-1)+ky1/2)-omega*omega*(x(i-1)+kx1/2));

 kx3=deltat*(y(i-1)+ky2/2);

 ky3=deltat*(-b*(y(i-1)+ky2/2)-omega*omega*(x(i-1)+kx2/2));

 kx4=deltat*(y(i-1)+kx3);

 ky4=deltat*(-b*(y(i-1)+ky3)-omega*omega*(x(i-1)+kx3));

 x(i)=x(i-1)+kx1/6+kx2/3+kx3/3+kx4/6;

 y(i)=y(i-1)+ky1/6+ky2/3+ky3/3+ky4/6;

end



% Dibuja los resultados en el espacio de fases

```
figure(1);hold off;
```

```
plot(x,y,'-','LineWidth',2.000);grid on;
```

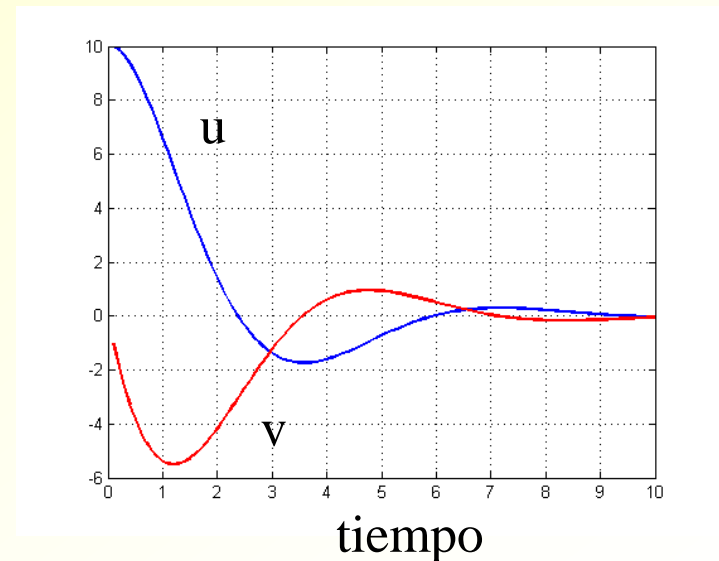
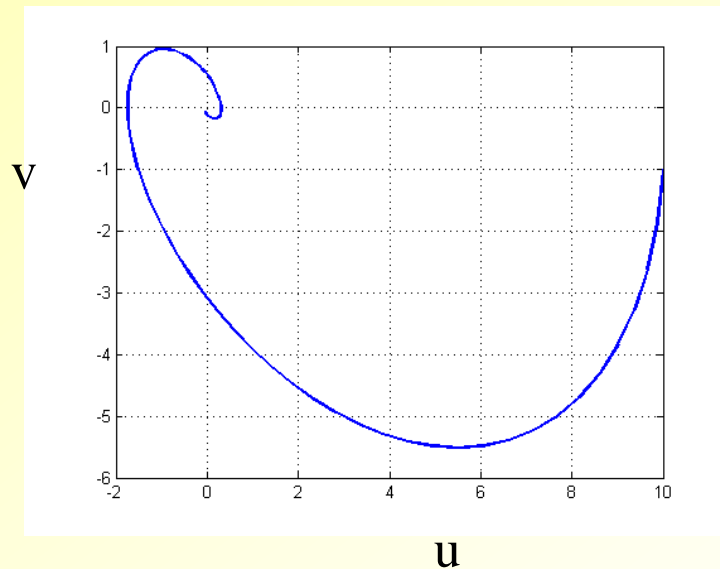
% Dibuja la dependencia temporal de x e y

```
t=[1:dimt]*deltat;
```

```
figure(2);hold off;
```

```
plot(t,x,'-b','LineWidth',2.000);hold on;
```

```
plot(t,y,'-r','LineWidth',2.000);grid on;
```



Modelo de Lorenz

La teoría del Caos empezó en 1963 con el trabajo de Lorenz publicado en una oscura revista de ciencias de la atmósfera. Este trabajo fue desconocido por los físicos y matemáticos hasta que poco a poco se lo llegó a ver tan importante como el nacimiento de una nueva ciencia.

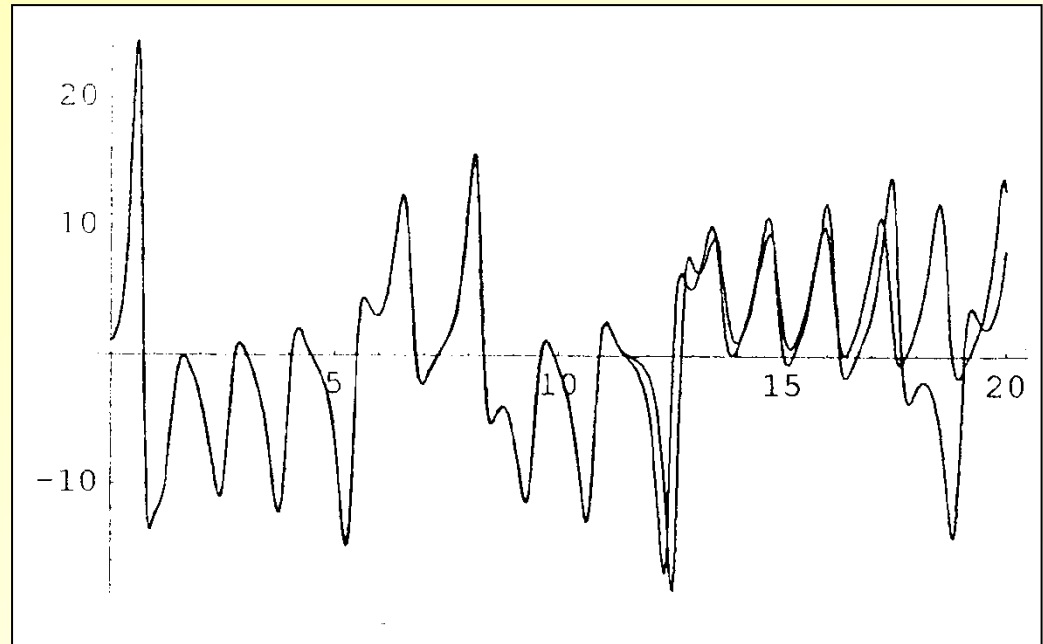
En el trabajo de Lorenz se intentaba dar una explicación y predicción global del clima atmosférico. Las ecuaciones que modelaban la atmósfera y que hoy se saben que son insuficientes, son aproximaciones a las ecuaciones de Navier-Stokes. Se llegaba finalmente a estas tres ecuaciones diferenciales :

$$\begin{aligned}\frac{dX}{dt} &= \sigma (Y - X) \\ \frac{dY}{dt} &= rX - Y - XZ \\ \frac{dZ}{dt} &= XY - bZ\end{aligned}$$

con σ , r , b parámetros definidos positivos.

T1: Dinámica Temporal: Atractor de Lorenz

La sensibilidad a las condiciones iniciales las mostramos en esta figura donde hemos tomado los valores $\sigma = 3$, $r = 26.5$, $b = 1$; $x_1, y_1, z_1 = (0, 1, 0)$; $x_2, y_2, z_2 = (0, 1.1, 0)$



De todos modos quedaría un punto de duda. En todo esto, hemos analizado el sistema de Lorenz numéricamente con computadora. Cabría desconfianza ante este método. Sin embargo Tucker, en 1999, probó que el sistema de Lorenz es un verdadero atractor caótico basándose en el concepto de hiperbolicidad singular y en la técnica de formas normales .