# Práctica 9

# Ecuaciones diferenciales 1: Métodos tradicionales y Métodos de un paso

En esta práctica vamos a estudiar métodos numérico para obtener la solución de ecuaciones diferenciales del tipo

$$y'(x) = f(x,y)$$

y

$$y''(x) = f(x,y,y')$$

en un intervalo $[x_0, x_N]$. El intervalo lo dividimos en un conjunto de puntos igualmente espaciados una distancia $h$, que es el paso de integración de la ecuación diferencial.

Estudiaremos casos simples de ecuaciones lineales de la Dinámica Newtoniana.

## 9.1. Método de Euler

El método de Euler consiste en aproximar la derivada por la fórmula de dos puntos

$$y'(x) \sim \frac{y(x+h) - y(x)}{h}$$

con lo que obtenemos la relación de recurrencia

$$y(x+h) = y(x) + hf(x,y)$$

Converge en orden $O(h)$, por lo que hay que tomar $h$ muy pequeño para obtener resultados razonables.

## 9.2. Método del punto medio

El método del punto medio consiste en aproximar la derivada por la fórmula de dos puntos

$$y'(x) \sim \frac{y(x+h) - y(x-h)}{2h}$$

con lo que obtenemos la relación de recurrencia

$$y(x+h) = y(x-h) + 2hf(x,y(x))$$

Converge en orden $O(h^2)$, lo que es en general suficiente para muchas aplicaciones, aunque puede presentar inestabilidades. Esta relación no está definida en el primer punto del intervalo, $x_0+h$. ya que implica el valor de $y$ en $x_0-h$. Para este primer punto se utiliza el método de Euler. Si denotamos $x_n = x_0+nh$ e $y_n = y(x_n)$ podemos escribir

$$\begin{aligned} y_1 &= y_0 + hf(x_0,y_0) \\ y_{n+1} &= y_{n-1} + 2hf(x_n,y_n) \end{aligned}$$

## 9.3. Método de Runge-Kutta de 4 orden

Entre los métodos más utilizados están los de Runge-Kutta de orden 4 y superior. El método de Runge-Kutta de cuarto orden converge en orden $h^4$. Consiste en la siguiente regla:

$$\begin{aligned} y_{n+1} &= y_n + \frac{h}{6}(k_1+2k_2+2k_3+k_4) \\ k_1 &= f(x_n,y_n) \\ k_2 &= f(x_n+\frac{1}{2}h, y_n+\frac{1}{2}hk_1) \\ k_3 &= f(x_n+\frac{1}{2}h, y_n+\frac{1}{2}hk_2) \\ k_4 &= f(x_n+h, y_n+hk_3) \end{aligned} \tag{9.1}$$

Se reduce a la regla de Simpson cuando $f$ no depende de $y$.

## 9.4. Ecuaciones de segundo orden

En el caso de una ecuación $y'' = f(x,y,y')$, reemplazamos la ecuación de segundo orden por dos ecuaciones de primer orden con el cambio de variables

$$\begin{aligned} y_1(x) &= y(x) \\ y_2(x) &= y'(x) \end{aligned}$$

con lo que queda el sistema de ecuaciones diferenciales de primer orden

$$\begin{aligned} y_1'(x) &= y_2(x) \\ y_2'(x) &= f(x,y_1(x),y_2(x)) \end{aligned}$$

que se pueden resolver simultáneamente por cualquier método válido para ecuaciones diferenciales de primer orden, entre ellos los tres que acabamos de ver.

## 9.5. Aplicación de los métodos a casos de interés físico

Vamos a utilizar los métodos de integración numérica en ecuaciones diferenciales sencillas que aparecen en dinámica. En una primera fase utilizaremos el método más simple, es decir el método de Euler. Después podéis utilizar cualquier método más sofisticado. En general el método del punto medio da resultados muy aceptables.

Los dos ejercicios que vamos a estudiar son:

1. Resolver la ecuación diferencial de una pelota que cae de una altura $x_0$, partiendo del reposo. Hay que imponer las condiciones de rebote de que cuando la altura $x$ sea menor que el paso de integración, se integra en esta fracción de paso de integración y la velocidad cambia de signo, es decir si el paso de integración da *xnew* y *vnew* en función de *xold* y *vold*, cuando hay rebote, *xnew=xold* y *vnew=-vold*. Por lo tanto el problema, aunque sencillo, esconde una singularidad que hace que la velocidad cambie de signo. La ecuación diferencial a resolver es
$$m\frac{d^2x}{dt^2} = F$$
donde $F = -mg$. Esta ecuación se resuelve como un conjunto de dos ecuaciones diferenciales:
$$\begin{aligned} \frac{dx}{dt} &= v \\ m\frac{dv}{dt} &= F \end{aligned}$$
El programa `rebote_el_eu.cpp` contiene la programación para el caso elástico, por el método de Euler.

```
,
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;
int main()
{
    const char *tab = "␣␣␣␣␣␣";
    double x0 = 1.;
    double m = 1.;
    double g = 9.81;
    double f = -m * g;
    double k = 0.;
    double x = x0;
    double v0 = 0.;
    double t = 0;
    double h = 0.01;
    double xold = x0;
```

```
    double vold = v0;
    //Numero de pasos de integracion
    int n = 1000;
    double xnew, vnew;
    ofstream fout("rebote_el_eu.res");
    //Integracion
    for (int i = 0; i < n; i++) {
        xnew = xold + vold * h;
        vnew = vold + (f + k * vold) / m * h;
//condicion de rebote;
//delth fraccion de h para que la pelota llegue al suelo
//La velocidad se invierte en el choque
        if (xnew <= 0) {
          xnew=0;
         double  deltah=-xold/vold;
          vnew = -(vold +  (f + k * vold) / m * deltah);
            t = t + deltah;     /*se desprecia el efecto de la aceleracion e
        } else
            t = t + h;
        xold = xnew;
        vold = vnew;
        fout << setprecision(10) << setw(15) << t << tab <<
            setw(15) << xnew << tab << setw(15) << vnew <<
            endl;
    }
    fout.close();
}
```

El programa `rebote_el_pm.cpp` contiene la program ación para el método del punto medio. 3as condiciones iniciales son *x=x0* y *v=0*. Integraréis un número importante de pasos de integración, como *10000* ó *100000*, de forma que haya varios rebotes. El primer aspecto a estudiar es la estabilidad de la integración numérica. así, no os debe extrañar de que si el paso de integración es demasiado grande la pelota rebote cada vez más alto o más bajo debido a una ganancia o pérdida de energía producida por los errores de redondeo al propagarse. Estos problemas deben de disminuir y eventualmente desaparecer al disminuir el paso de integración. Como ejemplo de los resultados que se obtienen, la Fig. 1 da los resultados de integrar con el método de Euler y *h=0.01* y la Fig.2 el resultado de integrar con el método del punto medio. En una segunda fase, cuando el rebote elástico funcione correctamente podéis incluir los siguientes aspectos:

- Una fuerza de rozamiento *kv*, donde *k* es una constante. La podéis elegir como la constante propia para una esfera o simplemente una cantidad numéricamente pequeña si no queréis repasar vuestros conocimientos de aerodinámica en un libro de física general.
- Un coeficiente de inelasticidad kine de forma que en el rebote *vnew=-kine*vold*.

Figura 9.1: Integración por el método de Euler

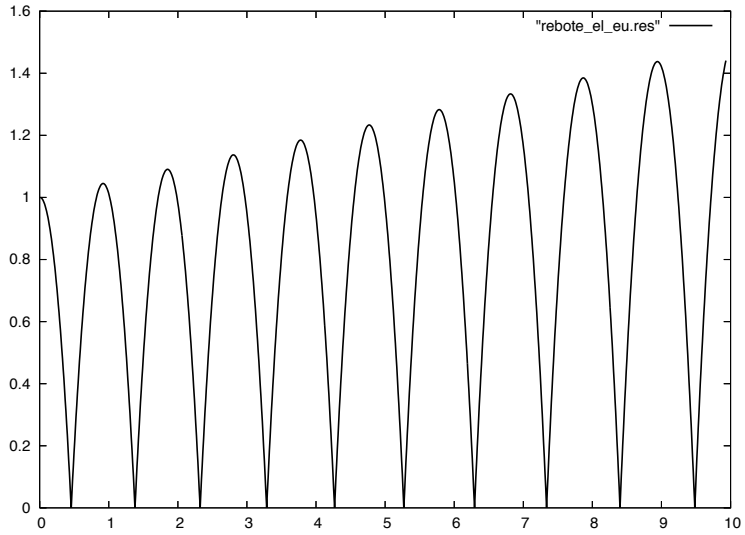
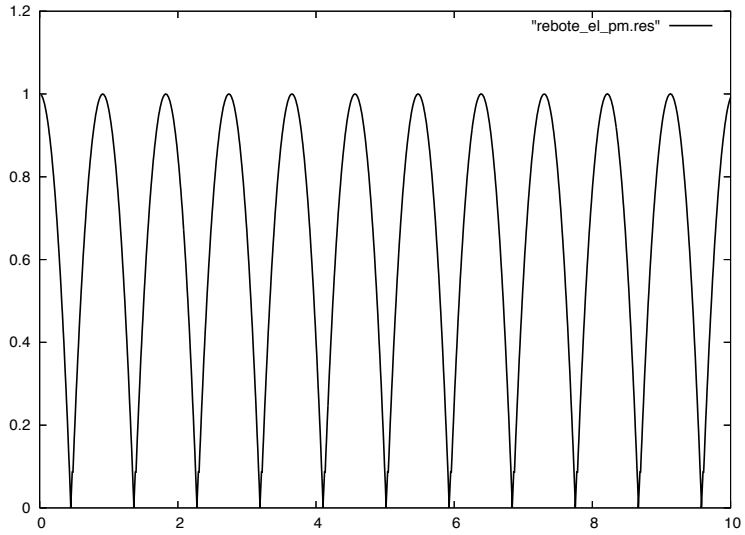



Figura 9.2: Integración por el método del punto medio



- Finalmente podéis incluir un método de integración más eficiente y comparar.
  Como recordatorio os doy lasecuaciones en diferencias en el caso de Euler y del punto medio:
- Euler

$$t_{n+1} = t_n + h$$
$$x_{n+1} = x_n + v_n h$$
$$v_{n+1} = v_n - gh$$

- Euler-Cromer

$$t_{n+1} = t_n + h$$
$$v_{n+1} = v_n - gh$$
$$x_{n+1} = x_n + v_{n+1} h$$

- Punto medio

$$x_{n+1} = x_{n-1} + 2v_n h$$
$$v_{n+1} = v_{n-1} - 2gh$$
$$t_{n+1} = t_n + h$$

con

$$x_1 = x_0 + v_0 h = x_0$$
$$v_1 = v_0 - gh = -gh$$
$$t_1 = h$$

2. Consideremos el movimiento de la Tierra alrededor del Sol. Integramos las ecuaciones de Newton de forma cartesiana. Verificad la estabilidad de la integración integrando varias órbitas consecutivas. Las ecuaciones del movimiento que hay que resolver son

$$m\frac{d^2x}{dt^2} = F_x = \frac{-GMm}{r^3}x$$
$$m\frac{d^2y}{dt^2} = F_y = \frac{-GMm}{r^3}y$$

Estas ecuaciones las escribiréis como cuatro ecuaciones diferenciales de primer orden, que resolveréis em primer lugar por el método de Euler y en segundo lugar por el método del punto medio. Tomad un paso de integración de *h=0.01* e integrad *10000* pasos de integración lo que corresponde a 100 años en intervalos de tres días. Comparar el comportamiento de ambos métodos de integración. Comparad después con *h=0.001*. Para evitar números excesivamente grandes es conveniente utilizar unidades de longitud astronómicas: $1AU = 1.496 \times 10^{11} m$, que corresponde a la distancia promedio de la Tierra al Sol. En estas unidadesel radio de la órbita vale la unidad en el caso de órbitas circulares.

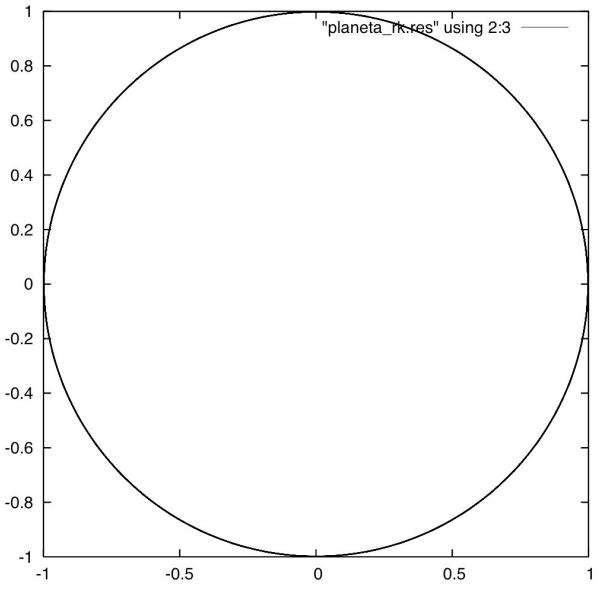



Figura 9.3: Integración de la órbita durante un siglo por el método de Runge-Kutta 4 orden

En estas unidades el producto de la masa de la Tierra por la constante gravitacional vale $GM = 4\pi^2 AU^3/año^2$. Tomad como condiciones iniciales las correspondientes a órbitas circulares $v_0 = \sqrt{v_{x0}^2 + v_{y0}^2} = (GM/R_0)^{\frac{1}{2}}$ . En una segunda etapa considerad órbitas cuya velocidad inicial se desvía por encima y por debajo de la condición de órbitas circulares. Representad $x$ en función de $y$. Discutid el resultado. Representad también $v_x$ en función de $v_y$. Para visualizar la estabilidad del método de integración un buen indicador es el radio de la órbita en función del tiempo. En el caso de órbitas circulares debe de permanecer prácticamente constante integrando durante algunos siglos.

Para visualizar los resultados es conveniente que vuestro programa efectúe una salida en un fichero que se pueda visualizar fácilmente con *Gnuplot*. Para el primer ejercicio por ejemplo, podéis imprimir en un fichero *t,x,v* en forma de tres columnas. Si el fichero se llama *rebote.res* por ejemplo, el comando plot *"rebote.res"* de *Gnuplot* dibujara $x$ en función de $t$. Con el comando *using* de *Gnuplot* podéis dibujar cualquier columna en función de cualquier otra. En el segundo

ejercicio podéis sacar $t,x,y,r$( radio de la órbita),$v_x,v_y$.

Los programas `planeta_eu.cpp`, `planeta_pm.cpp`, `planeta_rk.cpp` resuelven el problema por los métodos de Euler, punto medio y Runge-Kutta, respectivamente. A continuación damos el programa `planeta_rk.cpp :`

```
,
#include <cmath>
#include <iomanip>
#include <fstream>
#include <iostream>
using namespace std;
void rk4(double[], double[], int, double, double, double[],
         void (*derivs) (double, double[], double[]));
void derivs(double, double[], double[]);
int main()
{
  int neq = 4; //numero de ecuaciones de primer orden
    char *tab = "␣␣␣␣␣␣";
    double GM = 4 * pow(M_PI, 2);
    double m = 5.99e24;
    double M = 1.99e30;
    double t = 0;
    double h = 0.02;
    double v0 = sqrt(GM);
    double T = sqrt(4 * pow(M_PI, 2) / GM);
    cout << "Periodo␣teórico=␣␣" << T << endl;
    double tprev = 0.;
    double xold = 1.;
    double vxold = 0;
    double yold = 0;
    double vyold = v0;
    double y[4] = { xold, vxold, yold, vyold };
    double yn[4];
    double k1[4];
    double r = sqrt(pow(xold, 2) + pow(yold, 2));
    int n = 10000;
    double xnew, vxnew, ynew, vynew;
    ofstream fout("planeta_rk.res");
    for (int i = 0; i < n; i++) {
        derivs(t, y, k1);
        rk4(y, k1, neq, t, h, yn, (*derivs));
        xnew = yn[0];
        ynew = yn[2];
        vxnew = yn[1];
        vynew = yn[3];
//condicion de periodo;
```

```
        if (ynew * yold < 0 && xnew > 0) {
            T = t - tprev;
            cout << "tprev=␣" << tprev << tab << "t=␣␣" << t
                << tab << "Periodo=␣␣" << T << endl;
            tprev = t;
        }
        y[0] = yn[0];
        y[1] = yn[1];
        y[2] = yn[2];
        y[3] = yn[3];
        t = t + h;
        r = sqrt(pow(xnew, 2) + pow(ynew, 2));
        fout << t << tab << xnew << tab << ynew << tab <<
            vxnew << tab << vynew << tab << r << endl;
    }
}
void rk4(double y[], double k1[], int n, double x, double h,
         double yout[], void (*derivs) (double, double[],
                                        double[]))
{
    double xh, hh, h6;
    double *k4 = new double[n];
    double *k23 = new double[n];
    double *yt = new double[n];
    hh = h * 0.5;
    h6 = h / 6.0;
    xh = x + hh;
    for (int i = 0; i < n; i++)
        yt[i] = y[i] + hh * k1[i];
    (*derivs) (xh, yt, k4);
    for (int i = 0; i < n; i++)
        yt[i] = y[i] + hh * k4[i];
    (*derivs) (xh, yt, k23);
    for (int i = 0; i < n; i++) {
        yt[i] = y[i] + h * k23[i];
        k23[i] += k4[i];
    }
    (*derivs) (x + h, yt, k4);
    for (int i = 0; i < n; i++)
        yout[i] =
            y[i] + h6 * (k1[i] + k4[i] + 2.0 * k23[i]);
    delete[]yt;
    delete[]k23;
    delete[]k4;
}
```

```
void derivs(double x, double y[], double dy[])
{
    double GM = 4 * pow(M_PI, 2);
    double r = sqrt(y[0] * y[0] + y[2] * y[2]);
    double r3 = r * r * r;
    dy[0] = y[1];
    dy[1] = -GM / r3 * y[0];
    dy[2] = y[3];
    dy[3] = -GM / r3 * y[2];
}
```

## 9.6. Ejercicios a presentar como memoria.

1. Estudiar para que paso de integración no hay cambio apreciable de de altura en 10 rebotes consecutivos con el método de Euler y con el método de Euler-Cromer.

2. Repetir el problema anterior con los métodos de Verlet y del salto de la rana.

3. Estudiar para que paso de integración el método del punto medio tiene errores de integración visibles (cambio de altura en el rebote) en 10 rebotes consecutivos. Ajustad el número total de pasos cada vez que cambiéis el paso de integración para tener siempre unos 10 rebotes.

4. Introducir con el método del punto medio un coeficiente de inelasticidad, en el que se pierda el 10 % de la energía cinética en el rebote.

5. Estudiar la estabilidad con el paso de integración $h$ en el problema del planeta, para los métodos del punto medio y RK4. Aumentad el paso de integración hasta observar que el radio de órbitas circulares varia con el tiempo (debido a errores numéricos).

6. Modificad la velocidad inicial en `planeta_rk.cpp` para obtener órbitas elípticas. Dibujadlas con `Gnuplot`. Utilizad la opción `set size square` para que las escalas en los ejes x e y sean las mismas y no haya distorsiones debidas a las escalas.

7. Obtener la solución de un oscilador anharmónico forzado con una fuerza sinusoidal por el método del punto medio y RK4:

$$\frac{d^2x}{dt^2} + k^2x + \beta x^3 = A\sin(\omega_0 t)$$

   Elegid las constantes $k$, $\beta$, $A$, $\omega_0$ simples, por ejemplo podéis comenzar por 1, 1, 0.1, 1 (unidades MKS). Tomad como condiciones iniciales $\dot{x} = 0$ m/s y $x_0 = 1$ m.

8. Resolver la ecuación anterior por el método de Runge-Kutta-Fehlberg.

# Práctica 10

# Ecuaciones diferenciales ordinarias 2: Métodos avanzados, Métodos multipaso y predictor corrector

## 10.1. Runge-Kutta adaptativo

Uno de los casos en que el método adaptativo es especialmente útil es el caso de trayectorias fuertemente elípticas en el caso de atracción gravitacional. El programa `orbit.cpp` calcula la órbita de un cometa por diferentes métodos: Euler, Euler-Cromer, RK4 y RK4 adaptativo. Uno de los test de órbitas es la conservación de la energía. Si un método tiene errores numéricos importantes o es inestable no conserva la energía. La energía viene dada por

$$E = \frac{1}{2}mv^2 - \frac{GMm}{r}$$

La solución analítica del problema es

$$r(\theta) = \frac{a(1-e^2)}{1-e\cos\theta}$$

donde $a$ es el radio de órbitas circulares

$$a = -\frac{GMm}{2E}$$

Los valores máximo y mínimo de la distancia radial son $r_+ = a(1+e)$ y $r_- = a(1-e)$ que corresponden a $\theta = 0, \pi$ respectivamente. La velocidad en función de la distancia radial es

$$v = \sqrt{GM\left(\frac{2}{r} - \frac{1}{a}\right)}$$

Para poner condiciones iniciales de órbitas muy elípticas ponemos $x_0 = a(1 \pm e)$, $y_0 = 0$, $v_x = 0$, $v_y = \sqrt{GM\left(\frac{2}{a(1\pm e)} - \frac{1}{a}\right)}$ con$e$ cercano a la unidad, digamos $e = 0{,}99$. Por ejemplo, en el caso del

cometa Halley, $e = 0{,}967$, $r_- = 0{,}587$AU con un período de $T = 76{,}03$ años. Tenemos $GM = 4\pi^2$ AU$^3$/año$^2$ por lo que $a = 17{,}7878787878$ AU para el cometa Halley. La velocidad tangencial en el punto de máxima aproximación es $v_y = 11{,}404835349$ AU/año.

Compilar el programa con

```
g++ -I. orbit.cpp gravrk.cpp rka.cpp rk4.cpp -o orbit
```

Para dibujar las órbitas hacer en Gnuplot

```
plot "orbit.txt" using 2:3 with lines
```

### 10.1.1. Método de quinto orden

La función rkck.cpp implementa el método de Cash. Compilar con
g++ orbitrkck.cpp gravrk.cpp rkck.cpp rk4.cpp -o orbitrk
En orbit.dat hay un fichero de datos. Ejecutar con orbitrk<orbit.dat.

## 10.2. Sistemas de ecuaciones

### 10.2.1. Ecuaciones de Lorenz. Sistemas caóticos.

Las ecuaciones de Lorenz son un modelo muy simplificado de la Meteorología. Tienen como característica que una pequeña variación de las condiciones iniciales producen grandes variaciones de las trayectorias. Es lo que se denomina un sistema caótico. Lorenz le dió elnombre de efecto mariposa, en el sentido de que el vuelo de una mariposa podía afectar el comportamiento del tiempo en las antípodas. Las ecuaciones de Lorenz tienen la forma

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y-x) \\ \frac{dy}{dt} &= rx - y - xz \\ \frac{dz}{dt} &= xy - bz \end{aligned}$$

donde $r$,$b$ y $\sigma$ son parámetros. Normalmente se toma $\sigma = 10$, $b = 8/3$. El parámetro $r$ es proporcional al gradiente de temperatura aplicado.

Compilar `lorenz.cpp`,`lorenzrk.cpp`, `rka.cpp`, `rk4.cpp`. Introducir como estado inicial $x = 1$, $y = 1$, $z = 1$, $r = 28$. Con gnuplot dibujar

```
splot [-30:30] [-30:30] [0:50] "total_plot.txt" using 2:3:4 with
lines
```

Se observará dos atractores alrededor de los cuales se forman las trayectorias. Los atractores

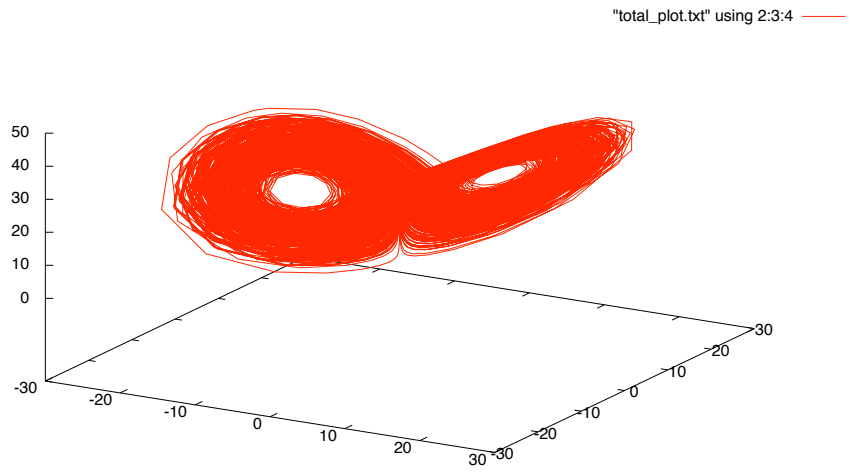



Figura 10.1: Trayectorias obtenidas con el modelo de Lorenz y $x = y = 1$ y $z = 20$ como punto inicial y$r = 28$ como parámetro.

son puntos fijos. Se deducen como los puntos estacionarios

$$\begin{aligned} \frac{dx}{dt} &= 0 = \quad \sigma(y-x) \\ \frac{dy}{dt} &= 0 = \quad rx - y - xz \\ \frac{dz}{dt} &= 0 = \quad xy - bz \end{aligned}$$

La solución es $y = x$, $z = y^2/b$ e $y^3/b + (1-r)y = 0$ que da $y = x = z = 0$, que es la solución trivial, y $y = x = \pm\sqrt{b(r-1)}$, $z = r-1$.

### 10.2.2. Ecuaciones de Lotka-Volterra

Las ecuaciones de Lotka-Volterra describen un sistema predador presa.

$$\begin{aligned} \frac{dx}{dt} &= \quad (a - bx - cy)x \\ \frac{dy}{dt} &= \quad (-d + ex)y \end{aligned}$$

donde $x$ es una especie que sirve como presa e $y$ su predador (ratones y gatos, por ejemplo). El parámetro $a$ es la tasa de crecimmiento de $x$ por reproducción cuando hay una cantidad infinita de alimento, mientras que $b$ tiene en cuenta que el alimento es finito y que incluso sin predadores, $x$ no puede crecer indefinidamente. El parámetro $c$tiene en cuenta la desaparición de $x$a causa de los

predadores. El parámetro $d$ es la tasa de muerte por hambre de $y$ cuando no hay presas, mientras que $e$ tiene en cuenta que si el alimento es suficiente, los predadores $y$ pueden multiplicarse. Hay dos puntos estacionarios, el trivial $x = 0$ e $y = 0$ y el punto de equilibrio, dado por $x = d/e$ e $y = -a/c + bd/ce$. Tomamos $a = d = 10$, $c = e = 0{,}1$ y $b = 10^{-5}$. El modelo predice oscilaciones alrededor del punto de equilibrio, tanto más amplias cuando mas alejadas están las condiciones iniciales del punto de equilibrio.

Compilar `Lotka-Volterra.cpp`, `lotkark.cpp`, `rka.cpp`, `rk4.cpp`. Introducir como estado inicial $x = 110$, $y = 90$. Con gnuplot dibujar

```
splot "total_plot.txt" using 1:2:3 with lines
```

## 10.3. Métodos predictor-corrector

En el programa orbit_pc.cpp están implementados los siguientes pares predictor corrector, con los valores de comienzo estimados por RK4.

Método de Euler modificado:

$$\begin{aligned} y_{n+1}^{(0)} &= y_n + hf_n \\ y_{n+1}^{(i+1)} &= y_n + \frac{h}{2}\left[f_{n+1}^{(i)} + f_n\right] \end{aligned}$$

Método de Adams-Moulton:

$$\begin{aligned} y_{n+1}^{(0)} &= y_n + \frac{h}{12}\left[23f_n - 16f_{n-1} + 5f_{n-2}\right] + O(h^3) \\ y_{n+1}^{(i+1)} &= y_n + \frac{h}{24}\left[9f_{n+1}^{(i)} + 19f_n - 5f_{n-1} + f_{n-2}\right] + O(h^4) \end{aligned}$$

Método de Adams:

$$\begin{aligned} y_{n+1}^{(0)} &= y_n + \frac{h}{24}\left[55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}\right] + O(h^4) \\ y_{n+1}^{(i+1)} &= y_n + \frac{h}{24}\left[9f_{n+1}^{(i)} + 19f_n - 5f_{n-1} + f_{n-2}\right] + O(h^4) \end{aligned}$$

Método de Milne:

$$\begin{aligned} y_{n+1}^{(0)} &= y_{n-3} + \frac{4h}{3}\left(2f_{n-2} - f_{n-1} + 2f_n\right) \\ y_{n+1}^{(i+1)} &= y_{n-1} + \frac{h}{3}\left(f_{n+1}^{(i)} + 4f_n + f_{n-1}\right) \end{aligned}$$

Comparar para diversos valores de $h$ el predictor y las primeras iteraciones del corrector. Para los datos del cometa Halley estudiar la estabilidad en función de $h$.