Cómo Programar en C++ el Método Runge-Kutta de 4to Orden

Felipe González Cataldo
www.gnm.cl/fgonzalez

November 1, 2017

Contents

Chapter 1

Introducción

Este artículo está dirigido a alumnos y profesores que buscan una guía práctica para implementar el método de Runge-Kutta en el lenguaje `C++`, para resolver distintos tipos de ecuaciones diferenciales. Cada tipo de ecuación diferencial cuenta con un ejemplo, para el cual se aplican las definiciones propias del método, de modo que las funciones queden explícitamente definidas para el caso particular para lograr mayor claridad al momento de programar. Luego se presenta un código autocontenido que implementa el método para el ejemplo en cuestión. Usuarios que únicamente buscan resolver numéricamente una ecuación diferencial podrón sacar provecho de esta guía, ya que basta copiar el código y reemplazar con la ecuación que buscan resolver, sin necesidad de entender cómo funciona el método.

Espero que este artículo sea de utilidad para la comunidad científica, que comunmente se ve en la necesidad de resolver ecuaciones diferenciales numéricamente y no encuentra manuales claros para hacerlo. Este documento pretende ser uno.

Cualquier sugerencia, no duden en mandar un email a `fgonzalez@lpmd.cl`.

Chapter 2

Ecuaciones Diferenciales de Primer Orden

Considere una función diferenciable $y : \mathbb{R} \to \mathbb{R}$ que cumple una ecuación diferencial de primer orden del tipo

$$y'(t) = f(y(t), t), \tag{2.1}$$

donde f es cierta función que depende de la función y y del parámetro $t \in \mathbb{R}$. Conociendo el valor de y en algún punto, digamos $y(t_n) \equiv y_n$, el método de Runge-Kutta permite aproximar, para un $\tau \in \mathbb{R}$ dado, el valor de y en $t_n + \tau$, que definiremos como $y_{n+1} \equiv y(t_n + \tau)$, con un error del orden de τ^5. El método consiste en definir

$$k_1 \equiv f\left(y_n, t_n\right) \tag{2.2a}$$

$$k_2 \equiv f\left(y_n + \frac{\tau}{2}k_1, t_n + \frac{\tau}{2}\right) \tag{2.2b}$$

$$k_3 \equiv f\left(y_n + \frac{\tau}{2}k_2, t_n + \frac{\tau}{2}\right) \tag{2.2c}$$

$$k_4 \equiv f\left(y_n + \tau k_3, t_n + \tau\right), \tag{2.2d}$$

y tomar

$$y_{n+1} = y_n + \frac{\tau}{6}\left[k_1 + 2k_2 + 2k_3 + k_4\right]. \tag{2.3}$$

Podemos realizar recursivamente esta operación para obtener $y(t)$ en cualquier $t > t_n$.

2.1 Ejemplo: ecuación diferencial de primer orden

Consideremos la ecuación diferencial

$$y'(x) = y(x)\tan(x) + x^2,$$

y la condición inicial $y(x_0) = y_0$ para cierto $x_0 \in \mathbb{R}$. En este caso, la función f está dada por

$$f(a, b) = a\tan(b) + b^2,$$

para la cual se tiene

$$f(y(x), x) = y(x)\tan(x) + x^2 = y'(x),$$

como en la ecuación (2.1). Para un valor de τ suficientemente pequeño, tenemos:

$$
\begin{aligned}
k_1 &= f(y_0, x_0) = y_0 \tan(x_0) + x_0^2 \\
k_2 &= f\left(y_0 + \frac{\tau}{2}k_1, x_0 + \frac{\tau}{2}\right) = \left(y_0 + \frac{\tau}{2}k_1\right)\tan\left(x_0 + \frac{\tau}{2}\right) + \left(x_0 + \frac{\tau}{2}\right)^2 \\
k_3 &= f\left(y_0 + \frac{\tau}{2}k_2, x_0 + \frac{\tau}{2}\right) = \left(y_0 + \frac{\tau}{2}k_2\right)\tan\left(x_0 + \frac{\tau}{2}\right) + \left(x_0 + \frac{\tau}{2}\right)^2 \\
k_4 &= f\left(y_0 + \tau k_3, x_0 + \tau\right) = \left(y_0 + \tau k_3\right)\tan\left(x_0 + \tau\right) + \left(x_0 + \tau\right)^2 \\
y_1 &= y_0 + \frac{\tau}{6}\left[k_1 + 2k_2 + 2k_3 + k_4\right].
\end{aligned}
$$

Ahora que tenemos $y_1 \approx y(x_1)$, con $x_1 = x_0 + \tau$, podemos repetir el proceso y obtener $y_2 \approx y(x_2)$, donde $x_2 = x_1 + \tau$. Así, $k_1 = f(y_1, x_1)$, $k_2 = f(y_1 + \frac{\tau}{2}k_1, x_1 + \tau/2)$, etc. Podemos iterar este proceso hasta llegar a algún valor deseado de x, por ejemplo $x = 25$. Para ello basta repetir el proceso n veces, en donde x_n será tal que $x_n \leqslant 25 < x_n + 1$. De este modo tendremos el valor de y para todo $x \in \{x_0, \ldots, x_n\}$. Si τ es lo suficientemente pequeño, podemos considerar que hemos obtenido el valor de $y(x)$ para todo $x \in [x_0, x_n]$.

2.2 Programa para ecuación diferencial de primer orden

Existen muchas formas de programar el método de Runge-Kutta, pero lo esencial es la definicion de k_1, k_2, k_3, k_4 y la función f. A continuación presento un programa hecho en C++ que resuelve la ecuación diferencial de primer orden del ejemplo anterior,

$$y'(x) = y(x)\tan(x) + x^2,$$

para la condición incial $y(x_0) = y_0$, usando el método Runge-Kutta. La forma en que el programa está hecho pretende ser lo más clara posible, separando el método de integración de la ecuación a resolver y del programa principal.

```
/*********************************************************/
/*     INTEGRADOR DE ECUACIONES DIFERENCIALES            */
/*********************************************************/
// AUTOR: FELIPE GONZALEZ CATALDO, Abril 2011.
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
/* Funcion Runge Kutta para ecuaciones de primer orden , como y'(x) = f(y,x) */
void RK4(double & x, double & y, double f(double,double), double tau)
{
 double k1,k2,k3,k4;
 k1 = f(y,x);
 k2 = f(y+tau*k1/2.0, x+tau/2.0);
 k3 = f(y+tau*k2/2.0, x+tau/2.0);
 k4 = f(y+tau*k3, x+tau);
 y = y + (tau/6.0)*(k1+2*k2+2*k3+k4);
}
//----------------- Que ecuacion desea resolver? -----------------//
double Ec_1er_Orden(double y, double x){ return y*tan(x)+x*x;}
//---------------------------------------------------------------//
int main()
{
 cout << "|--------------------------------|" << endl;
 cout << "|  Resuelva ecuaciones del tipo  |" << endl;
 cout << "|       y'(x)=f(y,x)             |" << endl;
 cout << "|--------------------------------|" << endl;
 ofstream file("Solucion-1erOrden.txt");
 double x, y, x0, y0, xmax=25.0, tau=0.1; // Condiciones iniciales: x = xo, y = y(xo)
 cout << "Su ecuacion necesita una condicion inicial: y(xo)=yo.\n";
 cout << "Ingrese el valor de xo: "; cin >> x0;
 cout << "Ingrese el valor de y(xo): "; cin >> y0;
 x = x0; y = y0;
 for( x=x0; x<xmax; x+=tau )
 {
  RK4(x,y,Ec_1er_Orden,tau);
  file << x << " " << y << endl;
 }
 file.close();
 cout << "Su solucion se encuentra en el archivo Solucion-1erOrden.txt" << endl;
 return 0;
}
```

Listing 2.1: Programa `rk4-1ord.cc` en lenguaje C++.

El programa consiste en lo siguiente:

1. Se define la función `RK4` que toma un `x` (que al principio vale x_0) y un `y` (que al principio vale $y_0 = y(x_0)$). Toma también una función `f` de dos parámetros, que representa nuestra función $f(y, x) = y\tan(x) + x^2$, que más adelante se declara como `Ec_1er_Orden`, y un número `tau` que representa el paso de tiempo τ. Usando esto, encuentra cuánto vale y(`x+tau`).

2. Cada vez que esta función es llamada, ésta cambia el valor de `x` por `x+tau`, y el de `y` por y(`x+tau`), es decir, hace avanzar `x` e `y` un paso.

3. En el programa principal simplemente pedimos al usuario las condiciones iniciales (`xo` e y(`xo`) = `yo`), y llamamos a la función `Ec_1er_Orden` con un paso de tiempo `tau`= 0.1 dentro de un ciclo `for`. El ciclo se hace hasta que `x`=`xmax`=25.0, por lo que obtendremos $y(x)$ para los x entre x_0 y 25. En cada *loop*, el programa arroja `x` e `y` al archivo `Solucion-1erOrden.txt`.

Gráfico de la solución

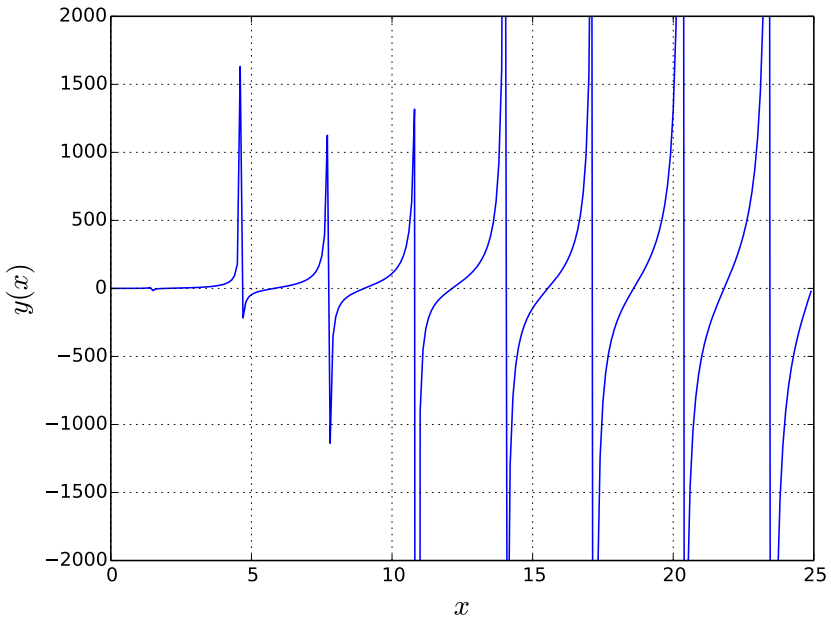

Figure 2.1: Gráfico de $y(x)$, solución a la ecuación $y'(x) = y(x)\tan(x) + x^2$, con condición inicial $y(0) = 0$.

Chapter 3

Ecuaciones Diferenciales de Segundo Orden

Considere una función dos veces diferenciable $y : \mathbb{R} \to \mathbb{R}$ que cumple una ecuación diferencial de segundo orden del tipo

$$y''(t) = F(y(t), y'(t), t), \tag{3.1}$$

donde F es cierta función que depende de la función y, de su derivada y' y del parámetro $t \in \mathbb{R}$. Conociendo el valor de y e y' en algún punto, digamos $y(t_n) \equiv y_n$ e $y'(t_n) \equiv yp_n$, el método de Runge-Kutta permite aproximar, para un $\tau \in \mathbb{R}$ dado, el valor de y en $t_n + \tau$, que definiremos como $y_{n+1} \equiv y(t_n + \tau)$, con un error del orden de τ^5. La idea es convertir esta ecuación diferencial de segundo orden en dos ecuaciones diferenciales de primer orden y aplicar el método anterior. Para ello, definimos

$$x_1(t) \equiv y(t) \tag{3.2a}$$
$$x_2(t) \equiv y'(t) \tag{3.2b}$$

para las cuales se tiene

$$x_1'(t) = x_2(t) \tag{3.3a}$$
$$x_2'(t) = y''(t) = F(x_1(t), x_2(t), t) \tag{3.3b}$$

Ahora nuestro problema no es la ecuación (3.1), sino el conjunto de ecuaciones acopladas en (3.3). Sobre este conjunto, aplicamos el método de Runge-Kutta: al igual que en el capítulo anterior, definimos una función f, una para cada ecuación diferencial:

$$x_1'(t) = f_1(x_1(t), x_2(t), t) \equiv x_2(t), \tag{3.4a}$$
$$x_2'(t) = f_2(x_1(t), x_2(t), t) \equiv F(x_1(t), x_2(t), t). \tag{3.4b}$$

Ahora aplicamos las definiciones en (2.2) a cada una de las ecuaciones diferenciales de primer orden en (3.4). Como cada ecuación diferencial tiene cuatro k-es asociados, necesitaremos cuatro para cada una, ocho en total. Definiendo $k_j^{(i)}$ como el j-ésimo k asociado a f_i ($j = 1, 2, 3, 4$; $i = 1, 2$) tenemos

$$k_1^{(1)} = f_1\left({x_1}^{(n)}, {x_2}^{(n)}, t_n\right) \tag{3.5a}$$
$$k_1^{(2)} = f_2\left({x_1}^{(n)}, {x_2}^{(n)}, t_n\right) \tag{3.5b}$$
$$k_2^{(1)} = f_1\left({x_1}^{(n)} + \frac{\tau}{2}k_1^{(1)}, {x_2}^{(n)} + \frac{\tau}{2}k_1^{(2)}, t_n + \frac{\tau}{2}\right) \tag{3.5c}$$
$$k_2^{(2)} = f_2\left({x_1}^{(n)} + \frac{\tau}{2}k_1^{(1)}, {x_2}^{(n)} + \frac{\tau}{2}k_1^{(2)}, t_n + \frac{\tau}{2}\right) \tag{3.5d}$$
$$k_3^{(1)} = f_1\left({x_1}^{(n)} + \frac{\tau}{2}k_2^{(1)}, {x_2}^{(n)} + \frac{\tau}{2}k_2^{(2)}, t_n + \frac{\tau}{2}\right) \tag{3.5e}$$
$$k_3^{(2)} = f_2\left({x_1}^{(n)} + \frac{\tau}{2}k_2^{(1)}, {x_2}^{(n)} + \frac{\tau}{2}k_2^{(2)}, t_n + \frac{\tau}{2}\right) \tag{3.5f}$$
$$k_4^{(1)} = f_1\left({x_1}^{(n)} + \tau k_3^{(1)}, {x_2}^{(n)} + \tau k_3^{(2)}, t_n + \tau\right) \tag{3.5g}$$
$$k_4^{(2)} = f_2\left({x_1}^{(n)} + \tau k_3^{(1)}, {x_2}^{(n)} + \tau k_3^{(2)}, t_n + \tau\right) \tag{3.5h}$$
$${x_1}^{(n+1)} = {x_1}^{(n)} + \frac{\tau}{6}\left[k_1^{(1)} + 2k_2^{(1)} + 2k_3^{(1)} + k_4^{(1)}\right] \tag{3.5i}$$
$${x_2}^{(n+1)} = {x_2}^{(n)} + \frac{\tau}{6}\left[k_1^{(2)} + 2k_2^{(2)} + 2k_3^{(2)} + k_4^{(2)}\right]. \tag{3.5j}$$

donde ${x_1}^{(n)} \equiv y_n$ y ${x_2}^{(n)} \equiv yp_n$. Es importante notar que el orden en que estas ecuaciones son declaradas: todos los k_1 deben ser declarados antes de los k_2, ya que para obtener $k_2^{(i)}$, es necesario conocer el valor de todos los $k_1^{(i)}$. Lo mismo para k_3 y k_4. **Es un error común** en la implementación del código definir primero los cuatro k asociados a f_1 $(k_1^{(1)}, k_2^{(1)}, k_3^{(1)}, k_4^{(1)})$ y luego los k asociados a f_2. Para resumir estos resultados, definimos

$$\mathbf{x}(t) \equiv (x_1(t), x_2(t)) \tag{3.6a}$$
$$\mathbf{f}(\mathbf{x}, t) \equiv (f_1(\mathbf{x}, t),\ f_2(\mathbf{x}, t)). \tag{3.6b}$$

Con esto,

$$\begin{aligned}
\mathbf{x}'(t) &= (x_1'(t),\ x_2'(t)) \\
&= (x_2(t),\ F(x_1(t), x_2(t), t)) \\
&= (f_1(x_1(t), x_2(t), t),\ f_2(x_1(t), x_2(t), t)) \\
&= (f_1(\mathbf{x}(t), t),\ f_2(\mathbf{x}(t), t)) \\
&= \mathbf{f}\left(\mathbf{x}(t), t\right),
\end{aligned}$$

es decir, UNA ecuación diferencial (vectorial) de primer orden. Estas definiciones permiten que $\mathbf{x}$ guarde las posiciones y velocidades $(y, y') = (x_1, x_2)$, mientras que $\mathbf{f}$ guarda las velocidades y aceleraciones $(y', y'') = (x_1', x_2')$. Ahora, definiendo $\mathbf{k}_1 \equiv (k_1^{(1)}, k_1^{(2)})$, $\mathbf{k}_2 \equiv (k_2^{(1)}, k_2^{(2)})$, $\mathbf{k}_3 \equiv (k_3^{(1)}, k_3^{(2)})$, $\mathbf{k}_4 \equiv (k_4^{(1)}, k_4^{(2)})$, las ecuaciones en (3.5) se reducen a

$$\mathbf{k}_1 = \mathbf{f}\left(\mathbf{x}_n, t_n\right) \tag{3.7a}$$
$$\mathbf{k}_2 = \mathbf{f}\left(\mathbf{x}_n + \frac{\tau}{2}\mathbf{k_1}, t + \frac{\tau}{2}\right) \tag{3.7b}$$
$$\mathbf{k}_3 = \mathbf{f}\left(\mathbf{x}_n + \frac{\tau}{2}\mathbf{k_2}, t + \frac{\tau}{2}\right) \tag{3.7c}$$
$$\mathbf{k}_4 = \mathbf{f}\left(\mathbf{x}_n + \tau\mathbf{k_3}, t + \tau\right) \tag{3.7d}$$
$$\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{\tau}{6}[\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4], \tag{3.7e}$$

Con esto tenemos, básicamente, el mismo set que en (2.2). Lo único que cambia en el método es que ahora tanto los k_j como los x_n son vectores con 2 coordenadas escalares.

3.1 Ejemplo: ecuación diferencial de segundo orden

Consideremos la ecuación diferencial que rige el movimiento angular de un péndulo simple:

$$\theta''(t) + \frac{g}{L}\sin(\theta(t)) = 0,$$

donde g es la aceleración de gravedad y L el largo del péndulo. Consideremos las condiciones inciales $\theta(t_0) \equiv \theta_0$ y $\theta'(t_0) \equiv \omega_0$ para cierto $t_0 \in \mathbb{R}$. En este caso, de acuerdo a (3.4), las funciones f están dadas por

$$f_1(a,b,c) = b, \quad f_2(a,b,c) = -\frac{g}{L}\sin(a),$$

para las cuales se tiene

$$f_1(\theta(t), \theta'(t), t) = \theta'(t), \quad f_2(\theta(t), \theta'(t), t) = -\frac{g}{L}\sin(\theta(t)) = \theta''(t),$$

como en la ecuación (3.1). Para un valor de τ suficientemente pequeño, tenemos:

$$\begin{aligned}
k_1^{(1)} &= f_1\left({x_1}^{(0)}, {x_2}^{(0)}, t_0\right) = {x_2}^{(0)} \\
k_1^{(2)} &= f_2\left({x_1}^{(0)}, {x_2}^{(0)}, t_0\right) = -\frac{g}{L}\sin({x_1}^{(0)}) \\
k_2^{(1)} &= f_1\left({x_1}^{(0)} + \frac{\tau}{2}k_1^{(1)}, {x_2}^{(0)} + \frac{\tau}{2}k_1^{(2)}, t_0 + \frac{\tau}{2}\right) = {x_2}^{(0)} + \frac{\tau}{2}k_1^{(2)} \\
k_2^{(2)} &= f_2\left({x_1}^{(0)} + \frac{\tau}{2}k_1^{(1)}, {x_2}^{(0)} + \frac{\tau}{2}k_1^{(2)}, t_0 + \frac{\tau}{2}\right) = -\frac{g}{L}\sin\left({x_1}^{(0)} + \frac{\tau}{2}k_1^{(1)}\right) \\
k_3^{(1)} &= f_1\left({x_1}^{(0)} + \frac{\tau}{2}k_2^{(1)}, {x_2}^{(0)} + \frac{\tau}{2}k_2^{(2)}, t_0 + \frac{\tau}{2}\right) = {x_2}^{(0)} + \frac{\tau}{2}k_2^{(2)} \\
k_3^{(2)} &= f_2\left({x_1}^{(0)} + \frac{\tau}{2}k_2^{(1)}, {x_2}^{(0)} + \frac{\tau}{2}k_2^{(2)}, t_0 + \frac{\tau}{2}\right) = -\frac{g}{L}\sin\left({x_1}^{(0)} + \frac{\tau}{2}k_2^{(1)}\right) \\
k_4^{(1)} &= f_1\left({x_1}^{(0)} + \tau k_3^{(1)}, {x_2}^{(0)} + \tau k_3^{(2)}, t_0 + \tau\right) = {x_2}^{(0)} + \tau k_3^{(2)} \\
k_4^{(2)} &= f_2\left({x_1}^{(0)} + \tau k_3^{(1)}, {x_2}^{(0)} + \tau k_3^{(2)}, t_0 + \tau\right) = -\frac{g}{L}\sin\left({x_1}^{(0)} + \tau k_3^{(1)}\right) \\
{x_1}^{(1)} &= {x_1}^{(0)} + \frac{\tau}{6}\left[k_1^{(1)} + 2k_2^{(1)} + 2k_3^{(1)} + k_4^{(1)}\right] \\
{x_2}^{(1)} &= {x_2}^{(0)} + \frac{\tau}{6}\left[k_1^{(2)} + 2k_2^{(2)} + 2k_3^{(2)} + k_4^{(2)}\right].
\end{aligned}$$

Ahora que tenemos ${x_1}^{(1)} = \theta_1 \approx \theta(t_1)$, con $t_1 = t_0 + \tau$, podemos repetir el proceso y obtener ${x_1}^{(2)} = \theta_2 \approx \theta(t_2)$, donde $t_2 = t_1 + \tau$. Así, $k_1^{(1)} = f_1({x_1}^{(1)}, {x_2}^{(1)}, t_1)$, $k_2^{(1)} = f_1({x_1}^{(1)} + \frac{\tau}{2}k_1^{(1)}, {x_2}^{(1)} + \frac{\tau}{2}k_1^{(2)}, t_1 + \tau/2)$, etc. Podemos iterar este proceso hasta llegar a algún valor deseado de t.

3.2 Programa para ecuación diferencial de segundo orden

Existen muchas formas de programar el método de Runge-Kutta, pero lo esencial es la definicion de k_1, k_2, k_3, k_4 y la función f. A continuación presento un programa hecho en `C++` que resuelve la ecuación diferencial de segundo orden del ejemplo anterior,

$$\theta''(t) + \frac{g}{L}\sin(\theta(t)) = 0,$$

para $g = 9.8$ m/s^2 y $L = 1$ m, con las condiciones iniciales $\theta(t_0) = \theta_0$ y $\theta'(t_0) = \omega_0$ usando el método Runge-Kutta. La forma en que el programa está hecho pretende ser lo más clara posible, separando el método de integración de la ecuación a resolver y del programa principal.

```
/*****************************************************************/
/*      INTEGRADOR DE ECUACIONES DIFERENCIALES                   */
/*****************************************************************/
// AUTOR: FELIPE GONZALEZ CATALDO, Marzo 2011.
// www.gnm.cl/fgonzalez
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;


/* Funcion Runge Kutta para ecuaciones de segundo orden , como y''(x) = f(y,y',x) */
double f1(double x1, double x2, double t){ return x2;}
void RK4(double & t, double & x1, double & x2, double f2(double,double,double), double tau)
{
 double k1[2],k2[2],k3[2],k4[2];
 k1[0] = f1(x1,x2,t);
 k1[1] = f2(x1,x2,t);
 k2[0] = f1(x1+tau*k1[0]/2.0, x2+tau*k1[1]/2.0, t+tau/2.0);
 k2[1] = f2(x1+tau*k1[0]/2.0, x2+tau*k1[1]/2.0, t+tau/2.0);
 k3[0] = f1(x1+tau*k2[0]/2.0, x2+tau*k2[1]/2.0, t+tau/2.0);
 k3[1] = f2(x1+tau*k2[0]/2.0, x2+tau*k2[1]/2.0, t+tau/2.0);
 k4[0] = f1(x1+tau*k3[0], x2+tau*k3[1], t+tau);
 k4[1] = f2(x1+tau*k3[0], x2+tau*k3[1], t+tau);
 x1 = x1 + (tau/6.0)*(k1[0]+2*k2[0]+2*k3[0]+k4[0]);
 x2 = x2 + (tau/6.0)*(k1[1]+2*k2[1]+2*k3[1]+k4[1]);
 t = t + tau;
}


//----------------- Que ecuacion desea resolver? -------------------//
double Ec_2o_Orden(double y, double yp,double x){ return -9.8*sin(y);}
//------------------------------------------------------------------//


int main()
{
 cout << "|--------------------------------|" << endl;
 cout << "| Resuelva ecuaciones del tipo   |" << endl;
 cout << "|       y''(x)=f(y,y',x)         |" << endl;
 cout << "|--------------------------------|" << endl;
 ofstream file("Solucion-2oOrden.txt");
 double t, y, yp, t0, y0, yp0, tmax=20.0, tau=0.01; // Condiciones iniciales: xo=x, y(xo)=y, y'(xo)=yp
 cout << "Su ecuacion necesita dos condiciones iniciales: y(xo)=yo, y'(xo)=ypo.\n";
```

```
45  cout << "Ingrese el valor de xo:  "; cin >> t0;
46  cout << "Ingrese el valor de yo:  "; cin >> y0;
47  cout << "Ingrese el valor de ypo: "; cin >> yp0;
48  t = t0; y = y0; yp = yp0;
49  for(t=0; t<tmax; t+=tau)
50  {
51   RK4(t,y,yp,Ec_2o_Orden,tau);
52   file << t << " " << y << " " << yp << endl;
53  }
54  file.close();
55  cout << "Ecuacion resuelta. Su solucion se encuentra en el archivo Solucion-2oOrden.txt" << endl;
56  return 0;
57 }
```

Listing 3.1: Programa `rk4-2ord.cc` en lenguaje `C++`.

El programa consiste en lo siguiente:

1. Se define la función `RK4` que toma un `t` (que al principio vale t_0), un `x1` que al principio vale $y_0 = \theta(t_0)$ y un `x2` que al principio vale $yp_0 = \theta'(t_0)$.

2. Toma también una función `f2` de tres parámetros, que representa nuestra función $f_2(x_1, x_2, t) = -\frac{g}{L}\sin(x_1)$, que más adelante se declara como `Ec_2o_Orden`, y un número `tau` que representa el paso de tiempo τ.

3. Usando esto, encuentra cuánto vale θ(`t+tau`) y θ'(`t+tau`).

4. Cada vez que esta función es llamada, se actualiza el valor de `x1` por θ(`t+tau`) y el de `x2` por θ'(`t+tau`), es decir, hace avanzar `x1` y `x2` un paso. El ciclo `for` se encarga de cambiar `t` por `t+tau`.

5. En el programa principal simplemente pedimos al usuario las condiciones iniciales (`xo`, θ(`xo`) = `yo` y θ'(`xo`) = `ypo`), y llamamos a la función `Ec_2o_Orden` con un paso de tiempo `tau`$= 0.01$ dentro de un ciclo `for`. El ciclo se hace hasta que $t = 20$.

6. En cada *loop*, el programa arroja `t`, `y` e `yp` (que representan a t, $\theta(t)$ y $\theta'(t)$, respectivamente) al archivo `Solucion-2oOrden.txt`.

Gráfico de la solución

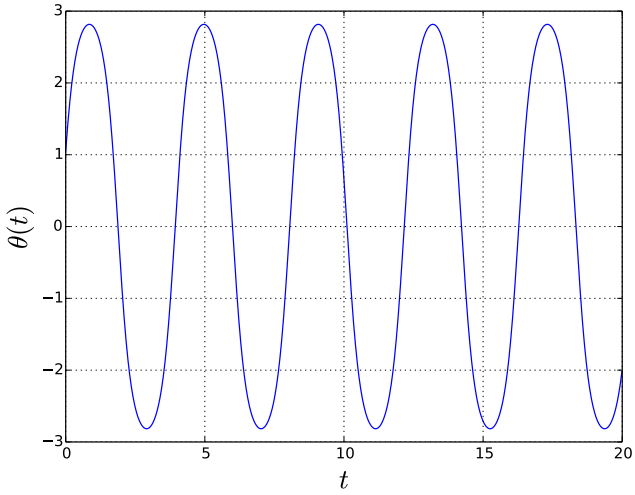

Figure 3.1: Gráfico de $\theta(t)$, solución a la ecuación $\theta''(t) = -\frac{g}{L}\sin(\theta(t)$, con condiciones iniciales $\theta(0) = 1$, $\theta'(0) = 5.4$.

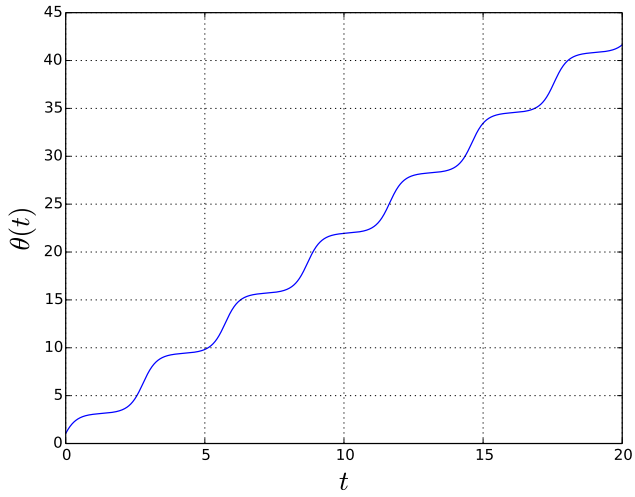

Figure 3.2: Gráfico de $\theta(t)$, solución a la ecuación $\theta''(t) = -\frac{g}{L}\sin(\theta(t)$, con condiciones iniciales $\theta(0) = 1$, $\theta'(0) = 5.5$.

Chapter 4

Ecuaciones Diferenciales Vectoriales de Primer Orden

Considere una función diferenciable $y : \mathbb{R} \to \mathbb{R}^m$,

$$y(t) \equiv \mathbf{y}(t) = (y_1(t), y_2(t), \ldots, y_m(t)),$$

que cumple una ecuación diferencial de primer orden del tipo

$$\mathbf{y}'(t) = \mathbf{f}(\mathbf{y}(t), t) \equiv (f_1(\mathbf{y}(t), t), f_2(\mathbf{y}(t), t), \ldots, f_m(\mathbf{y}(t), t)), \tag{4.1}$$

donde $\mathbf{f} : \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^m$ es un campo vectorial que depende de la función $\mathbf{y}$ y el parámetro $t \in \mathbb{R}$. Conociendo el valor de $\mathbf{y}$ en algún punto $t = t_n$, digamos $\mathbf{y}(t_n) \equiv \mathbf{y}_n = ({y_n}^{(1)}, {y_n}^{(2)}, \ldots, {y_n}^{(m)})$, el método de Runge-Kutta permite aproximar, para un $\tau \in \mathbb{R}$ dado, el valor de $\mathbf{y}$ en $t_n + \tau$, que definiremos como $\mathbf{y}_{n+1} \equiv \mathbf{y}(t_n + \tau)$, con un error del orden de τ^5.

Separemos primero la ecuación diferencial (4.1) por componentes:

$$y_1'(t) = f_1(\mathbf{y}(t), t) = f_1(y_1(t), y_2(t), \ldots, y_m(t), t) \tag{4.2a}$$
$$y_2'(t) = f_2(\mathbf{y}(t), t) = f_2(y_1(t), y_2(t), \ldots, y_m(t), t) \tag{4.2b}$$
$$\vdots$$
$$y_m'(t) = f_m(\mathbf{y}(t), t) = f_m(y_1(t), y_2(t), \ldots, y_m(t), t) \tag{4.2c}$$

Ahora aplicamos las definiciones en (2.2) a cada una de las ecuaciones diferenciales de primer orden en (4.2). Como cada ecuación diferencial tiene cuatro k-es asociados, necesitaremos cuatro para cada una, $4m$ en total. Definiendo $k_j^{(i)}$ como el j-ésimo k asociado a f_i ($j = 1, 2, 3, 4$; $i = 1, 2, \ldots, m$) tenemos

$$k_1^{(i)} = f_i\left({y_n}^{(1)}, {y_n}^{(2)}, \ldots, {y_n}^{(m)}, t_n\right) \tag{4.3a}$$
$$k_2^{(i)} = f_i\left({y_n}^{(1)} + \frac{\tau}{2}k_1^{(1)}, {y_n}^{(2)} + \frac{\tau}{2}k_1^{(2)}, \ldots, {y_n}^{(m)} + \frac{\tau}{2}k_1^{(m)}, t_n + \frac{\tau}{2}\right) \tag{4.3b}$$
$$k_3^{(i)} = f_i\left({y_n}^{(1)} + \frac{\tau}{2}k_2^{(1)}, {y_n}^{(2)} + \frac{\tau}{2}k_2^{(2)}, \ldots, {y_n}^{(m)} + \frac{\tau}{2}k_2^{(m)}, t_n + \frac{\tau}{2}\right) \tag{4.3c}$$
$$k_4^{(i)} = f_i\left({y_n}^{(1)} + \tau k_3^{(1)}, {y_n}^{(2)} + \tau k_3^{(2)}, \ldots, {y_n}^{(m)} + \tau k_3^{(m)}, t_n + \tau\right) \tag{4.3d}$$
$${y_{n+1}}^{(i)} = {y_n}^{(i)} + \frac{\tau}{6}\left[k_1^{(i)} + 2k_2^{(i)} + 2k_3^{(i)} + k_4^{(i)}\right]. \tag{4.3e}$$

De este modo, hemos encontrado $\mathbf{y}_{n+1} = ({y_1}^{(n+1)}, {y_2}^{(n+1)}, \ldots, {y_m}^{(n+1)})$. Es importante notar que el orden en que estas ecuaciones son declaradas: todos los k_1 deben ser declarados antes de los k_2, ya que para obtener $k_2^{(i)}$, es necesario conocer el valor de todos los $k_1^{(i)}$. Lo mismo para k_3 y k_4. **Es un error común** en la implementación del código definir primero los cuatro k asociados a f_1 $(k_1^{(1)}, k_2^{(1)}, k_3^{(1)}, k_4^{(1)})$ y luego los k asociados a f_2, f_3, etc.

Para resumir estos resultados, definimos $\mathbf{k}_1 \equiv (k_1^{(1)}, k_1^{(2)}, \ldots, k_1^{(m)})$, $\mathbf{k}_2 \equiv (k_2^{(1)}, k_2^{(2)}, \ldots, k_2^{(m)})$, $\mathbf{k}_3 \equiv (k_3^{(1)}, k_3^{(2)}, \ldots, k_3^{(m)})$, $\mathbf{k}_4 \equiv (k_4^{(1)}, k_4^{(2)}, \ldots, k_4^{(m)})$, las ecuaciones en (4.3) se reducen a

$$\mathbf{k}_1 = \mathbf{f}\left(\mathbf{y}_n, t_n\right) \tag{4.4a}$$
$$\mathbf{k}_2 = \mathbf{f}\left(\mathbf{y}_n + \frac{\tau}{2}\mathbf{k_1}, t + \frac{\tau}{2}\right) \tag{4.4b}$$
$$\mathbf{k}_3 = \mathbf{f}\left(\mathbf{y}_n + \frac{\tau}{2}\mathbf{k_2}, t + \frac{\tau}{2}\right) \tag{4.4c}$$
$$\mathbf{k}_4 = \mathbf{f}\left(\mathbf{y}_n + \tau\mathbf{k_3}, t + \tau\right) \tag{4.4d}$$
$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{\tau}{6}[\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4], \tag{4.4e}$$

Con esto tenemos, básicamente, el mismo set que en (2.2). Lo único que cambia en el método es que ahora tanto los k_j como los y_n son vectores con m coordenadas escalares.

4.1 Ejemplo: ecuación diferencial vectorial de primer orden

Consideremos la ecuación diferencial

$$\mathbf{r}'(t) = \mathbf{B}(t) \times \mathbf{r}(t),$$

donde $\mathbf{B} : \mathbb{R} \to \mathbb{R}^3$, y $\mathbf{r} : \mathbb{R} \to \mathbb{R}^3$. Consideremos la condición incial $\mathbf{r}(t_0) \equiv \mathbf{r}_0 = ({r_0}^{(1)}, {r_0}^{(2)}, {r_0}^{(3)})$ para cierto $t_0 \in \mathbb{R}$. Escribamos la ecuación diferencial por componentes:

$$\begin{aligned} r_1'(t) &= B_2(t)r_3(t) - B_3(t)r_2(t) \\ r_2'(t) &= B_3(t)r_1(t) - B_1(t)r_3(t) \\ r_3'(t) &= B_1(t)r_2(t) - B_2(t)r_1(t) \end{aligned}$$

En este caso, la función $\mathbf{f}$ está dada por $\mathbf{f}((a,b,c),t) = \mathbf{B}(t) \times (a,b,c)$, cuyas componentes son

$$\begin{aligned} f_1(a,b,c,t) &= B_2(t)c - B_3(t)b, \\ f_2(a,b,c,t) &= B_3(t)a - B_1(t)c, \\ f_3(a,b,c,t) &= B_1(t)b - B_2(t)a, \end{aligned}$$

para las cuales se tiene

$$\begin{aligned} f_1(r_1(t), r_2(t), r_3(t), t) &= B_2(t)r_3(t) - B_3(t)r_2(t) = r_1'(t), \\ f_2(r_1(t), r_2(t), r_3(t), t) &= B_3(t)r_1(t) - B_1(t)r_3(t) = r_2'(t), \\ f_3(r_1(t), r_2(t), r_3(t), t) &= B_1(t)r_2(t) - B_2(t)r_1(t) = r_3'(t). \end{aligned}$$

Usando (4.4), tenemos que $\mathbf{k}_1 = \mathbf{f}\left(\mathbf{r}_0, t_0\right)$, es decir,

$$\begin{aligned}
k_1^{(1)} &= f_1\left({r_0}^{(1)}, {r_0}^{(2)}, {r_0}^{(3)}, t_0\right) = B_2(t){r_0}^{(3)} - B_3(t){r_0}^{(2)},\\
k_1^{(2)} &= f_2\left({r_0}^{(1)}, {r_0}^{(2)}, {r_0}^{(3)}, t_0\right) = B_3(t){r_0}^{(1)} - B_1(t){r_0}^{(3)},\\
k_1^{(3)} &= f_3\left({r_0}^{(1)}, {r_0}^{(2)}, {r_0}^{(3)}, t_0\right) = B_1(t){r_0}^{(2)} - B_2(t){r_0}^{(1)}.
\end{aligned}$$

Para $\mathbf{k}_2 = \mathbf{f}\left(\mathbf{r}_0 + \frac{\tau}{2}\mathbf{k_1}, t + \frac{\tau}{2}\right)$ se tiene

$$\begin{aligned}
k_2^{(1)} &= f_1\left({r_0}^{(1)} + \frac{\tau}{2}k_1^{(1)}, {r_0}^{(2)} + \frac{\tau}{2}k_1^{(2)}, {r_0}^{(3)} + \frac{\tau}{2}k_1^{(3)}, t_0 + \frac{\tau}{2}\right) = B_2(t)\left({r_0}^{(3)} + \frac{\tau}{2}k_1^{(3)}\right) - B_3(t)\left({r_0}^{(2)} + \frac{\tau}{2}k_1^{(2)}\right),\\
k_2^{(2)} &= f_2\left({r_0}^{(1)} + \frac{\tau}{2}k_1^{(1)}, {r_0}^{(2)} + \frac{\tau}{2}k_1^{(2)}, {r_0}^{(3)} + \frac{\tau}{2}k_1^{(3)}, t_0 + \frac{\tau}{2}\right) = B_3(t)\left({r_0}^{(1)} + \frac{\tau}{2}k_1^{(1)}\right) - B_1(t)\left({r_0}^{(3)} + \frac{\tau}{2}k_1^{(3)}\right),\\
k_2^{(3)} &= f_3\left({r_0}^{(1)} + \frac{\tau}{2}k_1^{(1)}, {r_0}^{(2)} + \frac{\tau}{2}k_1^{(2)}, {r_0}^{(3)} + \frac{\tau}{2}k_1^{(3)}, t_0 + \frac{\tau}{2}\right) = B_1(t)\left({r_0}^{(2)} + \frac{\tau}{2}k_1^{(2)}\right) - B_2(t)\left({r_0}^{(1)} + \frac{\tau}{2}k_1^{(1)}\right).
\end{aligned}$$

Finalmente, $\mathbf{k}_3 = \mathbf{f}\left(\mathbf{r}_0 + \frac{\tau}{2}\mathbf{k_2}, t + \frac{\tau}{2}\right)$, es decir,

$$\begin{aligned}
k_3^{(1)} &= f_1\left({r_0}^{(1)} + \frac{\tau}{2}k_2^{(1)}, {r_0}^{(2)} + \frac{\tau}{2}k_2^{(2)}, {r_0}^{(3)} + \frac{\tau}{2}k_2^{(3)}, t_n + \frac{\tau}{2}\right) = B_2(t)\left({r_0}^{(3)} + \frac{\tau}{2}k_2^{(3)}\right) - B_3(t)\left({r_0}^{(2)} + \frac{\tau}{2}k_2^{(2)}\right),\\
k_3^{(2)} &= f_2\left({r_0}^{(1)} + \frac{\tau}{2}k_2^{(1)}, {r_0}^{(2)} + \frac{\tau}{2}k_2^{(2)}, {r_0}^{(3)} + \frac{\tau}{2}k_2^{(3)}, t_n + \frac{\tau}{2}\right) = B_3(t)\left({r_0}^{(1)} + \frac{\tau}{2}k_2^{(1)}\right) - B_1(t)\left({r_0}^{(3)} + \frac{\tau}{2}k_2^{(3)}\right),\\
k_3^{(3)} &= f_3\left({r_0}^{(1)} + \frac{\tau}{2}k_2^{(1)}, {r_0}^{(2)} + \frac{\tau}{2}k_2^{(2)}, {r_0}^{(3)} + \frac{\tau}{2}k_2^{(3)}, t_n + \frac{\tau}{2}\right) = B_1(t)\left({r_0}^{(2)} + \frac{\tau}{2}k_2^{(2)}\right) - B_2(t)\left({r_0}^{(1)} + \frac{\tau}{2}k_2^{(1)}\right),
\end{aligned}$$

y $\mathbf{k}_4 = \mathbf{f}\left(\mathbf{y}_n + \tau\mathbf{k_3}, t + \tau\right)$, luego

$$\begin{aligned}
k_4^{(1)} &= f_1\left({r_0}^{(1)} + \tau k_3^{(1)}, {r_0}^{(2)} + \tau k_3^{(2)}, {r_0}^{(3)} + \tau k_3^{(3)}, t_n + \tau\right) = B_2(t)\left({r_0}^{(3)} + \tau k_3^{(3)}\right) - B_3(t)\left({r_0}^{(2)} + \tau k_3^{(2)}\right),\\
k_4^{(2)} &= f_2\left({r_0}^{(1)} + \tau k_3^{(1)}, {r_0}^{(2)} + \tau k_3^{(2)}, {r_0}^{(3)} + \tau k_3^{(3)}, t_n + \tau\right) = B_3(t)\left({r_0}^{(1)} + \tau k_3^{(1)}\right) - B_1(t)\left({r_0}^{(3)} + \tau k_3^{(3)}\right),\\
k_4^{(3)} &= f_3\left({r_0}^{(1)} + \tau k_3^{(1)}, {r_0}^{(2)} + \tau k_3^{(2)}, {r_0}^{(3)} + \tau k_3^{(3)}, t_n + \tau\right) = B_1(t)\left({r_0}^{(2)} + \tau k_3^{(2)}\right) - B_2(t)\left({r_0}^{(1)} + \tau k_3^{(1)}\right).
\end{aligned}$$

Con esto, $\mathbf{r}_1 = \mathbf{r}_0 + \frac{\tau}{6}[\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4]$, es decir,

$$\begin{aligned}
{r_1}^{(1)} &= {r_0}^{(1)} + \frac{\tau}{6}\left[k_1^{(1)} + 2k_2^{(1)} + 2k_3^{(1)} + k_4^{(1)}\right]\\
{r_1}^{(2)} &= {r_0}^{(2)} + \frac{\tau}{6}\left[k_1^{(2)} + 2k_2^{(2)} + 2k_3^{(2)} + k_4^{(2)}\right]\\
{r_1}^{(3)} &= {r_0}^{(3)} + \frac{\tau}{6}\left[k_1^{(3)} + 2k_2^{(3)} + 2k_3^{(3)} + k_4^{(3)}\right]
\end{aligned}$$

Ahora que tenemos $\mathbf{r}_1 = ({r_1}^{(1)}, {r_1}^{(2)}, {r_1}^{(3)}) \approx \mathbf{r}(t_1)$, con $t_1 = t_0 + \tau$, podemos repetir el proceso y obtener $\mathbf{r}_2 = ({r_2}^{(1)}, {r_2}^{(2)}, {r_2}^{(3)}) \approx \mathbf{r}(t_2)$, donde $t_2 = t_1 + \tau$. Así,

$$\begin{aligned}
k_1^{(i)} &= f_i\left({r_1}^{(1)}, {r_1}^{(2)}, {r_1}^{(3)}, t_1\right),\\
k_2^{(i)} &= f_i\left({r_1}^{(1)} + \frac{\tau}{2}k_1^{(1)}, {r_1}^{(2)} + \frac{\tau}{2}k_1^{(2)}, {r_1}^{(3)} + \frac{\tau}{2}k_1^{(3)}, t_1 + \frac{\tau}{2}\right),
\end{aligned}$$

etc. Podemos iterar este proceso hasta llegar a algún valor deseado de t.

4.2 Programa para ecuación diferencial vectorial de primer orden

Existen muchas formas de programar el método de Runge-Kutta, pero lo esencial es la definicion de k_1, k_2, k_3, k_4 y la función f. A continuación presento un programa hecho en `C++` que resuelve la ecuación diferencial vectorial de primer orden del ejemplo anterior,

$$\mathbf{r}'(t) = \mathbf{B}(t) \times \mathbf{r}(t),$$

para distintos $\mathbf{B}(t) = B_1(t)\hat{\mathbf{x}} + B_2(t)\hat{\mathbf{y}} + B_3(t)\hat{\mathbf{z}}$, con la condición inicial $\mathbf{r}(t_0) = \mathbf{r}_0$ usando el método Runge-Kutta. La forma en que el programa está hecho pretende ser lo más clara posible, separando el método de integración de la ecuación a resolver y del programa principal.

El programa consiste en lo siguiente:

1. Mediante `typedef` se define el elemento `Funcion` como un puntero a una función tipo `double` de 4 argumentos tipo `double`. Esto permite crear un nuevo tipo de variable `Funcion` que nos permite declarar funciones de 4 argumentos o arreglos de las mismas.

2. Se define la función `RK4` que toma un `t` (que al principio vale t_0) y un arreglo `y[]` que al principio vale $\mathbf{y}_0 = (y_0^{(1)}, y_0^{(2)}, y_0^{(3)})$, un arreglo de funciones `f[]` de 4 argumentos, que representa nuestra función

   $$\mathbf{f}(a, b, c, t) = (f_1(a, b, c, t), f_2(a, b, c, t), f_3(a, b, c, t)),$$

   que es un arreglo de 3 funciones de 4 variables. Esta función es declarada más adelante como `Ec_1er_Orden`. Por último, toma un número `tau` que representa el paso de tiempo τ.

3. Al llamar a esta función (`Ec_1er_Orden`) en el programa, con un `tau`, un `t` y un `y[]` dados, la función encuentra cuánto vale $\mathbf{r}$(`t+tau`).

4. Cada vez que esta función es llamada, ésta cambia el valor de `y` por $\mathbf{r}$(`t+tau`).

5. En el programa principal simplemente pedimos al usuario las condiciones iniciales (`xo`, θ(`xo`) = `yo` y θ'(`xo`) = `ypo`), y llamamos a la función `Ec_2o_Orden` con un paso de tiempo `tau`= 0.01 dentro de un ciclo `for`. El ciclo se hace 2000 veces, por lo que obtendremos $\theta(t)$ para los t entre t_0 y $t_0 + 2000\tau$.

6. En cada *loop*, el programa arroja `x` e `y` (que representan a t y $\theta(t)$, respectivamente) al archivo `Solucion-2oOrden.txt`.

```
1  /*******************************************************/
2  /* INTEGRADOR DE ECUACIONES DIFERENCIALES              */
3  /*******************************************************/
4  // AUTOR: FELIPE GONZALEZ CATALDO, Mayo 2011.
5  // www.gnm.cl/fgonzalez
6  #include<iostream>
7  #include<cmath>
8  #include<fstream>
9  using namespace std;
10
11 // Funcion de 4 variables: Puntero a una funcion de 4 argumentos double, que retorna un double
12 typedef double (*Funcion) (double a, double b, double c, double t);
13
```

```
/* Funcion Runge Kutta para ecuaciones de primer orden , como y'(x) = f(y,x) */
void RK4(double & t, double y[], Funcion f[], double tau)
{
 double k1[] = {
  f[0](y[0],y[1],y[2],t),
  f[1](y[0],y[1],y[2],t),
  f[2](y[0],y[1],y[2],t) };
 double k2[] = {
  f[0](y[0]+tau*k1[0]/2.0,y[1]+tau*k1[1]/2.0,y[2]+tau*k1[2]/2.0,t+tau/2.0),
  f[1](y[0]+tau*k1[0]/2.0,y[1]+tau*k1[1]/2.0,y[2]+tau*k1[2]/2.0,t+tau/2.0),
  f[2](y[0]+tau*k1[0]/2.0,y[1]+tau*k1[1]/2.0,y[2]+tau*k1[2]/2.0,t+tau/2.0) };
 double k3[] = {
  f[0](y[0]+tau*k2[0]/2.0,y[1]+tau*k2[1]/2.0,y[2]+tau*k2[2]/2.0,t+tau/2.0),
  f[1](y[0]+tau*k2[0]/2.0,y[1]+tau*k2[1]/2.0,y[2]+tau*k2[2]/2.0,t+tau/2.0),
  f[2](y[0]+tau*k2[0]/2.0,y[1]+tau*k2[1]/2.0,y[2]+tau*k2[2]/2.0,t+tau/2.0) };
 double k4[] = {
  f[0](y[0]+tau*k3[0],y[1]+tau*k3[1],y[2]+tau*k3[2],t+tau),
  f[1](y[0]+tau*k3[0],y[1]+tau*k3[1],y[2]+tau*k3[2],t+tau),
  f[2](y[0]+tau*k3[0],y[1]+tau*k3[1],y[2]+tau*k3[2],t+tau) };

 for (int q=0; q<3; ++q) y[q]= y[q] + (tau/6.0)*(k1[q]+2*k2[q]+2*k3[q]+k4[q]);
}

//----------------- Que ecuacion desea resolver? -----------------//

// Componentes de B
double B1(double t){ return 0;}
double B2(double t){ return cos(t);}
double B3(double t){ return sin(t);}

// Componentes de la funcion f
double f1(double a, double b, double c, double t){ return B2(t)*c-B3(t)*b; }
double f2(double a, double b, double c, double t){ return B3(t)*a-B1(t)*c; }
double f3(double a, double b, double c, double t){ return B1(t)*b-B2(t)*a; }

Funcion Ec_1er_Orden[] = {f1, f2, f3};
//---------------------------------------------------------------------//
int main()
{
 cout << "|-----------------------------------------|" << endl;
 cout << "|        Resuelva ecuaciones del tipo     |" << endl;
 cout << "|              y'(t) = f(y(t),t),         |" << endl;
 cout << "|        donde y(t) es un vector          |" << endl;
 cout << "|-----------------------------------------|" << endl;
 ofstream file("Solucion-1erOrdenN.txt");
 double t, y[3], tmax=100, tau=0.01 ; // Condiciones iniciales: t = to, y[] = (y1(to),y2(to),y3(to
    ))
 cout << "Su ecuacion necesita una condicion inicial: y(to)=(y1(to),y2(to),y3(to))=yo.\n";
 cout << "Ingrese el valor de to: "; cin >> t;
 cout << "Ingrese el valor de y1(to): "; cin >> y[0];
 cout << "Ingrese el valor de y2(to): "; cin >> y[1];
 cout << "Ingrese el valor de y3(to): "; cin >> y[2];
 for(t=0; t<tmax; t+=tau)
 {
  RK4(t,y,Ec_1er_Orden,tau);
  file << t << " " << y[0] <<  " " << y[1] <<  " " << y[2] << endl;
 }
 file.close();
 cout << "Ecuacion resuelta. Su solucion se encuentra en el archivo Solucion-1erOrdenN.txt" <<
    endl;
 return 0;
}
```

Listing 4.1: Programa `rk4-1ordN.cc` en lenguaje `C++`.

Gráfico de la solución

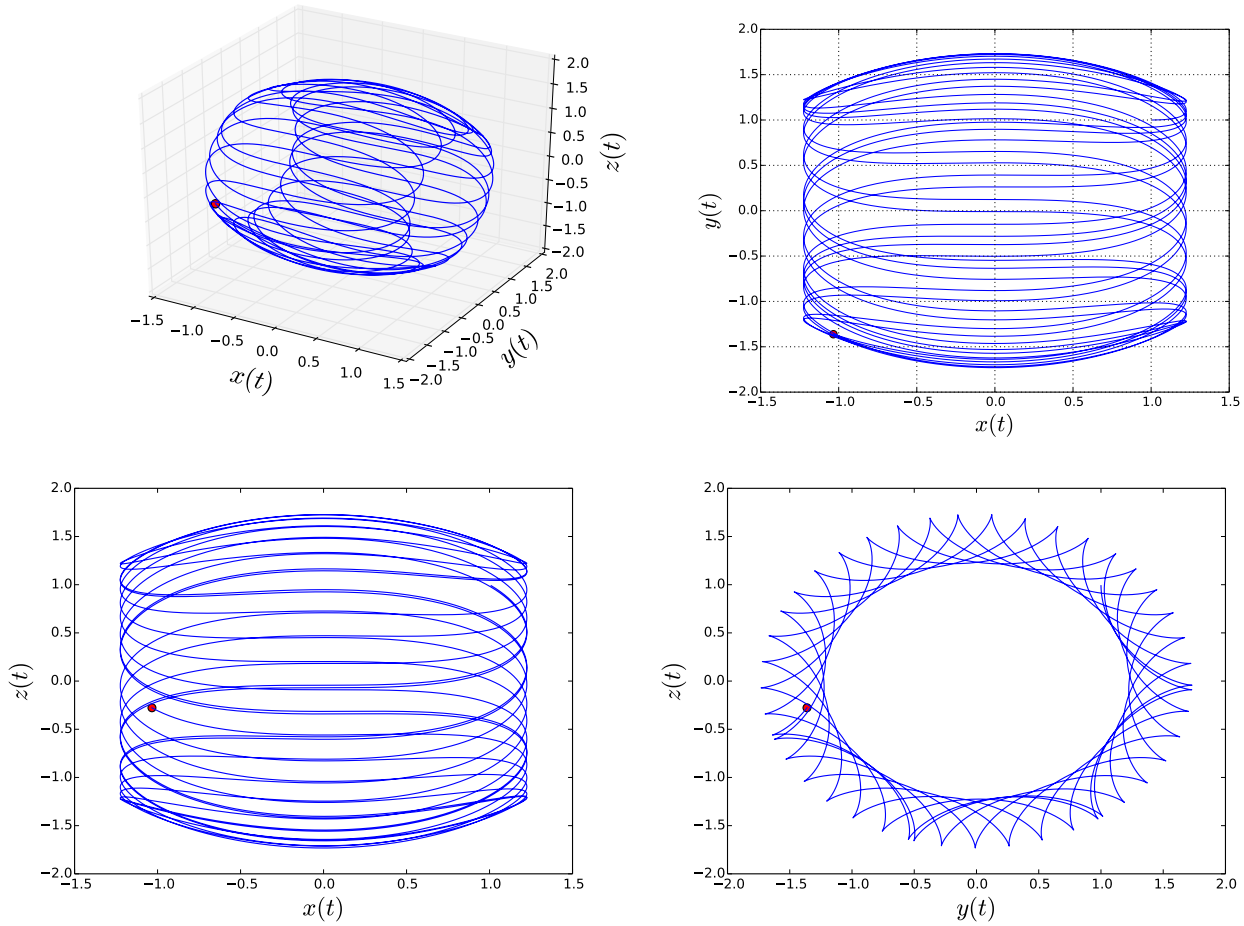

Figure 4.1: Gráfico de la trayectoria $\mathbf{r}(t)$, solución a la ecuación $\mathbf{r}'(t) = \mathbf{B}(t) \times \mathbf{r}(t)$, con $\mathbf{B}(t) = (0, \cos t, \sin t)$ y condición inicial $\mathbf{r}(0) = (1, 1, 1)$.

Chapter 5

Ecuaciones Diferenciales Vectoriales de Segundo Orden

Considere una función diferenciable $y : \mathbb{R} \to \mathbb{R}^m$,

$$y(t) \equiv \mathbf{y}(t) = (y_1(t), y_2(t), \ldots, y_m(t)),$$

que cumple una ecuación diferencial de segundo orden del tipo

$$\mathbf{y}''(t) = \mathbf{F}(\mathbf{y}(t), \mathbf{y}'(t), t) \equiv (F_1(\mathbf{y}(t), \mathbf{y}'(t), t), F_2(\mathbf{y}(t), \mathbf{y}'(t), t), \ldots, F_m(\mathbf{y}(t), \mathbf{y}'(t), t)), \tag{5.1}$$

donde $\mathbf{F} : \mathbb{R}^m \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^m$ es un campo vectorial que depende de la función $\mathbf{y}$, de su derivada $\mathbf{y}'$ y el parámetro $t \in \mathbb{R}$. Conociendo el valor de $\mathbf{y}$ e $\mathbf{y}'$ en algún punto $t = t_n$, digamos $\mathbf{y}(t_n) \equiv \mathbf{y}_n = (y_n{}^{(1)}, y_n{}^{(2)}, \ldots, y_n{}^{(m)})$, $\mathbf{y}'(t_n) \equiv \mathbf{v}_n = (v_n{}^{(1)}, v_n{}^{(2)}, \ldots, v_n{}^{(m)})$, el método de Runge-Kutta permite aproximar, para un $\tau \in \mathbb{R}$ dado, el valor tanto de $\mathbf{y}$ como de $\mathbf{y}'$ en $t_n + \tau$, con un error del orden de τ^5. La idea es convertir esta ecuación diferencial vectorial de segundo orden en dos ecuaciones diferenciales vectoriales de primer orden y aplicar el método anterior. Para ello, definimos

$$\mathbf{x}_1(t) \equiv \mathbf{y}(t) \tag{5.2a}$$
$$\mathbf{x}_2(t) \equiv \mathbf{y}'(t) \tag{5.2b}$$

para las cuales se tiene

$$\mathbf{x}_1'(t) = \mathbf{x}_2(t) \tag{5.3a}$$
$$\mathbf{x}_2'(t) = \mathbf{y}''(t) = \mathbf{F}(\mathbf{x}_1(t), \mathbf{x}_2(t), t) \tag{5.3b}$$

Ahora nuestro problema no es la ecuación (5.1), sino el conjunto de ecuaciones acopladas en (5.3). Sobre este conjunto, aplicamos el método de Runge-Kutta: al igual que en el capítulo anterior, definimos una función (campo vectorial) $\mathbf{f}$, uno para cada ecuación diferencial:

$$\mathbf{x}_1'(t) = \mathbf{f}_1(\mathbf{x}_1(t), \mathbf{x}_2(t), t) \equiv \mathbf{x}_2(t), \tag{5.4a}$$
$$\mathbf{x}_2'(t) = \mathbf{f}_2(\mathbf{x}_1(t), \mathbf{x}_2(t), t) \equiv \mathbf{F}(\mathbf{x}_1(t), \mathbf{x}_2(t), t). \tag{5.4b}$$

Ahora aplicamos las definiciones en (2.2) a cada una de las ecuaciones diferenciales de primer orden en (5.4). Como vimos en el capítulo anterior, cada ecuación diferencial vectorial de primer orden

tiene asociados 4 vectores $\mathbf{k}$, por lo tanto necesitaremos ocho $\mathbf{k}$-es en total. Definiendo $\mathbf{k}_j^{(i)}$ como el j-ésimo $\mathbf{k}$ asociado a $\mathbf{f}_i$ $(j = 1, 2, 3, 4;\ i = 1, 2)$ tenemos

$$\mathbf{k}_1^{(1)} = \mathbf{f}_1\left(\mathbf{x}_1{}^{(n)}, \mathbf{x}_2{}^{(n)}, t_n\right) \tag{5.5a}$$
$$\mathbf{k}_1^{(2)} = \mathbf{f}_2\left(\mathbf{x}_1{}^{(n)}, \mathbf{x}_2{}^{(n)}, t_n\right) \tag{5.5b}$$
$$\mathbf{k}_2^{(1)} = \mathbf{f}_1\left(\mathbf{x}_1{}^{(n)} + \frac{\tau}{2}\mathbf{k}_1^{(1)}, \mathbf{x}_2{}^{(n)} + \frac{\tau}{2}\mathbf{k}_1^{(2)}, t_n + \frac{\tau}{2}\right) \tag{5.5c}$$
$$\mathbf{k}_2^{(2)} = \mathbf{f}_2\left(\mathbf{x}_1{}^{(n)} + \frac{\tau}{2}\mathbf{k}_1^{(1)}, \mathbf{x}_2{}^{(n)} + \frac{\tau}{2}\mathbf{k}_1^{(2)}, t_n + \frac{\tau}{2}\right) \tag{5.5d}$$
$$\mathbf{k}_3^{(1)} = \mathbf{f}_1\left(\mathbf{x}_1{}^{(n)} + \frac{\tau}{2}\mathbf{k}_2^{(1)}, \mathbf{x}_2{}^{(n)} + \frac{\tau}{2}\mathbf{k}_2^{(2)}, t_n + \frac{\tau}{2}\right) \tag{5.5e}$$
$$\mathbf{k}_3^{(2)} = \mathbf{f}_2\left(\mathbf{x}_1{}^{(n)} + \frac{\tau}{2}\mathbf{k}_2^{(1)}, \mathbf{x}_2{}^{(n)} + \frac{\tau}{2}\mathbf{k}_2^{(2)}, t_n + \frac{\tau}{2}\right) \tag{5.5f}$$
$$\mathbf{k}_4^{(1)} = \mathbf{f}_1\left(\mathbf{x}_1{}^{(n)} + \tau\mathbf{k}_3^{(1)}, \mathbf{x}_2{}^{(n)} + \tau\mathbf{k}_3^{(2)}, t_n + \tau\right) \tag{5.5g}$$
$$\mathbf{k}_4^{(2)} = \mathbf{f}_2\left(\mathbf{x}_1{}^{(n)} + \tau\mathbf{k}_3^{(1)}, \mathbf{x}_2{}^{(n)} + \tau\mathbf{k}_3^{(2)}, t_n + \tau\right) \tag{5.5h}$$
$$\mathbf{x}_1{}^{(n+1)} = \mathbf{x_1}{}^{(n)} + \frac{\tau}{6}\left[\mathbf{k}_1^{(1)} + 2\mathbf{k}_2^{(1)} + 2\mathbf{k}_3^{(1)} + \mathbf{k}_4^{(1)}\right] \tag{5.5i}$$
$$\mathbf{x}_2{}^{(n+1)} = \mathbf{x_2}{}^{(n)} + \frac{\tau}{6}\left[\mathbf{k}_1^{(2)} + 2\mathbf{k}_2^{(2)} + 2\mathbf{k}_3^{(2)} + \mathbf{k}_4^{(2)}\right]. \tag{5.5j}$$

donde $\mathbf{x}_1{}^{(n)} \equiv \mathbf{y}_n$ y $\mathbf{x}_2{}^{(n)} \equiv \mathbf{v}_n = \mathbf{y}'(t_n)$. Es importante recalcar nuevamente que el orden en que estas ecuaciones son declaradas es muy importante: todos los $\mathbf{k}_1$ deben ser declarados antes de los $\mathbf{k}_2$, ya que para obtener $\mathbf{k}_2^{(i)}$, es necesario conocer el valor de todos los $\mathbf{k}_1^{(i)}$. Lo mismo para $\mathbf{k}_3$ y $\mathbf{k}_4$. **Es un error común** en la implementación del código definir primero los cuatro $\mathbf{k}$ asociados a $\mathbf{f}_1$ $(\mathbf{k}_1^{(1)}, \mathbf{k}_2^{(1)}, \mathbf{k}_3^{(1)}$ y $\mathbf{k}_4^{(1)})$ y luego los $\mathbf{k}$ asociados a $\mathbf{f}_2$.

5.1 Ejemplo 1: $\mathbf{r}''(t) = -\frac{GM}{r^3}\mathbf{r}(t)$,

Consideremos la ecuación diferencial que rige el movimiento angular de un planeta girando en torno al sol:

$$\mathbf{r}''(t) = -\frac{GM}{r^3}\mathbf{r}(t),$$

donde G es la constante de gravitación universal, M la masa del sol y Consideremos las condiciones inciales $\mathbf{r}(t_0) = (x_0, y_0, z_0) = \mathbf{x}_1{}^{(0)}$ y $\mathbf{r}'(t_0) = (v_{x0}, v_{y0}, v_{z0}) = \mathbf{x}_2{}^{(0)}$ para cierto $t_0 \in \mathbb{R}$. En este caso, de acuerdo a (5.4), las funciones $\mathbf{f}$ están dadas por

$$\mathbf{f}_1(\mathbf{x}_1(t), \mathbf{x}_2(t), t) = \mathbf{x}_2(t), \quad \mathbf{f}_2(\mathbf{x}_1(t), \mathbf{x}_2(t), t) = -\frac{GM}{\|\mathbf{x}_1(t)\|^3}\mathbf{x}_1(t),$$

donde $a = \|\mathbf{a}\|$, $\mathbf{x}_1(t) = \mathbf{r}(t)$ y $\mathbf{x}_2(t) = \mathbf{r}'(t)$. Para un valor de τ suficientemente pequeño, tenemos:

$$
\begin{aligned}
\mathbf{k}_1^{(1)} &= \mathbf{f}_1\left({\mathbf{x}_1}^{(0)}, {\mathbf{x}_2}^{(0)}, t_0\right) = {\mathbf{x}_2}^{(0)} \\
\mathbf{k}_1^{(2)} &= \mathbf{f}_2\left({\mathbf{x}_1}^{(0)}, {\mathbf{x}_2}^{(0)}, t_0\right) = -\frac{GM}{\|{\mathbf{x}_1}^{(0)}\|^3}\mathbf{x}_1^{(0)} = -\frac{GM}{(x_0^2+y_0^2+z_0^2)^{\frac{3}{2}}}(x_0, y_0, z_0) \\
\mathbf{k}_2^{(1)} &= \mathbf{f}_1\left({\mathbf{x}_1}^{(0)} + \frac{\tau}{2}\mathbf{k}_1^{(1)}, {\mathbf{x}_2}^{(0)} + \frac{\tau}{2}\mathbf{k}_1^{(2)}, t_0 + \frac{\tau}{2}\right) = {\mathbf{x}_2}^{(0)} + \frac{\tau}{2}\mathbf{k}_1^{(2)} \\
\mathbf{k}_2^{(2)} &= \mathbf{f}_2\left({\mathbf{x}_1}^{(0)} + \frac{\tau}{2}\mathbf{k}_1^{(1)}, {\mathbf{x}_2}^{(0)} + \frac{\tau}{2}\mathbf{k}_1^{(2)}, t_0 + \frac{\tau}{2}\right) = -\frac{GM}{\|{\mathbf{x}_1}^{(0)} + \frac{\tau}{2}\mathbf{k}_1^{(1)}\|^3}\left({\mathbf{x}_1}^{(0)} + \frac{\tau}{2}\mathbf{k}_1^{(1)}\right) \\
\mathbf{k}_3^{(1)} &= \mathbf{f}_1\left({\mathbf{x}_1}^{(0)} + \frac{\tau}{2}\mathbf{k}_2^{(1)}, {\mathbf{x}_2}^{(0)} + \frac{\tau}{2}\mathbf{k}_2^{(2)}, t_0 + \frac{\tau}{2}\right) = {\mathbf{x}_2}^{(0)} + \frac{\tau}{2}\mathbf{k}_2^{(2)} \\
\mathbf{k}_3^{(2)} &= \mathbf{f}_2\left({\mathbf{x}_1}^{(0)} + \frac{\tau}{2}\mathbf{k}_2^{(1)}, {\mathbf{x}_2}^{(0)} + \frac{\tau}{2}\mathbf{k}_2^{(2)}, t_0 + \frac{\tau}{2}\right) = -\frac{GM}{\|{\mathbf{x}_1}^{(0)} + \frac{\tau}{2}\mathbf{k}_2^{(1)}\|^3}\left({\mathbf{x}_1}^{(0)} + \frac{\tau}{2}\mathbf{k}_2^{(1)}\right) \\
\mathbf{k}_4^{(1)} &= \mathbf{f}_1\left({\mathbf{x}_1}^{(0)} + \tau\mathbf{k}_3^{(1)}, {\mathbf{x}_2}^{(0)} + \tau\mathbf{k}_3^{(2)}, t_0 + \tau\right) = {\mathbf{x}_2}^{(0)} + \tau\mathbf{k}_3^{(2)} \\
\mathbf{k}_4^{(2)} &= \mathbf{f}_2\left({\mathbf{x}_1}^{(0)} + \tau\mathbf{k}_3^{(1)}, {\mathbf{x}_2}^{(0)} + \tau\mathbf{k}_3^{(2)}, t_0 + \tau\right) = -\frac{GM}{\|{\mathbf{x}_1}^{(0)} + \frac{\tau}{2}\mathbf{k}_3^{(1)}\|^3}\left({\mathbf{x}_1}^{(0)} + \frac{\tau}{2}\mathbf{k}_3^{(1)}\right) \\
{\mathbf{x}_1}^{(1)} &= {\mathbf{x}_1}^{(0)} + \frac{\tau}{6}\left[\mathbf{k}_1^{(1)} + 2\mathbf{k}_2^{(1)} + 2\mathbf{k}_3^{(1)} + \mathbf{k}_4^{(1)}\right] \\
{\mathbf{x}_2}^{(1)} &= {\mathbf{x}_2}^{(0)} + \frac{\tau}{6}\left[\mathbf{k}_1^{(2)} + 2\mathbf{k}_2^{(2)} + 2\mathbf{k}_3^{(2)} + \mathbf{k}_4^{(2)}\right].
\end{aligned}
$$

Ahora que tenemos ${\mathbf{x}_1}^{(1)} \approx \mathbf{r}(t_1)$, con $t_1 = t_0+\tau$, podemos repetir el proceso y obtener ${\mathbf{x}_1}^{(2)} \approx \mathbf{r}(t_2)$, donde $t_2 = t_1 + \tau$. Así, $\mathbf{k}_1^{(1)} = \mathbf{f}_1({\mathbf{x}_1}^{(1)}, {\mathbf{x}_2}^{(1)}, t_1)$, $\mathbf{k}_2^{(1)} = \mathbf{f}_1({\mathbf{x}_1}^{(1)} + \frac{\tau}{2}\mathbf{k}_1^{(1)}, {\mathbf{x}_2}^{(1)} + \frac{\tau}{2}\mathbf{k}_1^{(2)}, t_1 + \tau/2)$, etc. Podemos iterar este proceso hasta llegar a algún valor deseado de t.

5.2 Código fuente Ejemplo 1

A continuación se muestra una implementación particular en `C++` del método de Runge-Kutta que resuelve la ecuación diferencial vectorial de segundo orden del ejemplo anterior, correspondiente a un planeta orbitando al sol:

$$\mathbf{r}''(t) = -\frac{GM}{r^3}\mathbf{r}(t),$$

donde $G = 4\pi^2 a^3/(T^2 M)$, $a = 1$ AU (unidad astronómica), $T = 1$ año y M la masa del sol. Consideraremos las condiciones iniciales $\mathbf{r}(0) = (0.7a, 0, 0)$ y $\mathbf{r}'(0) = (0, 6a/T, 0)$. Usando una clase de vectores, podemos implementar el código de manera casi idéntica al código 3.2. Basta cambiar `double` por `Vector`. De esta manera, las ecuaciones 5.5 conservan la notación y pueden ser escritas de manera casi literal en el código.

El programa consiste en lo siguiente:

1. Creamos una clase de vectores, de modo que podamos operar con ellos (suma, multiplicación por escalar, dividir por escalar, extraer componentes, etc.). Si el usuario lo necesita, puede añadir nuevas operaciones entre los vectores, tales como el producto interno, que pueden ser necesarios para aceleraciones más complicadas.

2. Se define la función `RK4` que representa al integrador. Esta función se encarga de calcular será la posición y velocidad en el siguiente paso. Toma un valor de `t` (que al principio vale $t_0 = 0.0$), un `Vector x1` que representa a $\mathbf{r}(t)$ y un `Vector x2` que representa a $\mathbf{r}'(t)$.

3. Toma también una función `f2` de tres parámetros, que representa nuestra función $\mathbf{f}_2(\mathbf{x}_1, \mathbf{x}_2, t) = -\frac{GM}{r^3}\mathbf{r}(t)$, que más adelante se declara como `Ec_2o_Orden`, y un número `tau` que representa el paso de tiempo τ.

4. Usando esto, encuentra cuánto vale $\mathbf{r}$(`t+tau`) y $\mathbf{r}'$(`t+tau`).

5. Cada vez que esta función es llamada, se actualiza el valor de `x1` por $\mathbf{r}$(`t+tau`) y el de `x2` por $\mathbf{r}'$(`t+tau`), es decir, hace avanzar `x1` y `x2` un paso `tau`. El ciclo `for` se encarga de cambiar `t` por `t+tau`.

6. En el programa principal, `main`, simplemente ingresamos las condiciones iniciales por coordenada ($\mathbf{r}(0) = (0.7a, 0, 0)$, $\mathbf{r}'(0) = (0, 6a/T, 0)$), y llamamos a la función `Ec_2o_Orden` con un paso de tiempo `tau`$= 0.01$ dentro de un ciclo `for`. El ciclo se hace hasta que $t = 2$ años.

7. En cada *loop*, el programa arroja `t`, las componentes de `r` y `v` (que representan a t, $\mathbf{r}(t)$ y $\mathbf{r}'(t)$, respectivamente) al archivo `Solucion-2oOrdenV.txt`.

```
/*********************************************************/
/* INTEGRADOR DE ECUACIONES DIFERENCIALES                */
/*********************************************************/
// AUTOR: FELIPE GONZALEZ CATALDO, Mayo 2011.
// www.gnm.cl/fgonzalez
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;

//////////////////  CLASS VECTOR ////////////////////////////////
class Vector{     // Definimos el nombre de la clase: "Vector".
 private:
 double elem[3];  // Valor del elemento en la posicion i

 public:
 Vector(void) {  for(long i=0; i<3; i++) elem[i]=0.0e0; } // Constructor: Vector a;
 inline double & operator[](int q)  { return elem[q]; }  // Vector a; a[0]=...

};
Vector operator + (Vector x, Vector y){     // Suma de vectores
 Vector z;
 for (int i=0; i<3; i++) z[i]=x[i]+y[i];
 return z;
}
Vector operator - (Vector x){               // Vector -v
 Vector z;
 for (int i=0; i<3; i++) z[i]=-x[i];
 return z;
}
Vector operator * (double z, Vector y){    //  double*Vector
  Vector x=y;
```

```
  for (int i=0; i<3; i++) x[i] = z*x[i];
  return x;
}
Vector operator / (Vector y, double z){   //  Vector/double
  Vector x=y;
  for (int i=0; i<3; i++) x[i] = x[i]/z;
  return x;
}
double abs(Vector x){ return sqrt(x[0]*x[0]+x[1]*x[1]+x[2]*x[2]);}  // modulo

Vector Cross(Vector a, Vector b){                                    // producto cruz
 Vector c;
 c[0]=a[1]*b[2]-a[2]*b[1];
 c[1]=-a[0]*b[2]+a[2]*b[0];
 c[2]=a[0]*b[1]-a[1]*b[0];
 return c;
}
////////////////// END OF CLASS VECTOR ///////////////////////

double G= 4.0*M_PI*M_PI;    // en (AU^3)/(sunmass*year^2)
double M = 1.0;             // en masas solares
double a = 1.0;             // en AU
double T = 1.0;             // en years

//----------------- Que ecuacion desea resolver? -------------------//
Vector Ec_2o_Orden(Vector r, Vector v,double t){ double mod=pow(abs(r),3); return -G*M*r/mod;}
//------------------------------------------------------------------//

/* Funcion Runge Kutta para ecuaciones de segundo orden , como y''(x) = f(y,y',x) */
Vector f1(Vector x1, Vector x2, double t){ return x2;}
void RK4(double & t, Vector & x1, Vector & x2, Vector f2(Vector,Vector,double), double tau)
{
 Vector k1[2],k2[2],k3[2],k4[2];
 k1[0] = f1(x1,x2,t);
 k1[1] = f2(x1,x2,t);
 k2[0] = f1(x1+tau*k1[0]/2.0, x2+tau*k1[1]/2.0, t+tau/2.0);
 k2[1] = f2(x1+tau*k1[0]/2.0, x2+tau*k1[1]/2.0, t+tau/2.0);
 k3[0] = f1(x1+tau*k2[0]/2.0, x2+tau*k2[1]/2.0, t+tau/2.0);
 k3[1] = f2(x1+tau*k2[0]/2.0, x2+tau*k2[1]/2.0, t+tau/2.0);
 k4[0] = f1(x1+tau*k3[0], x2+tau*k3[1], t+tau);
 k4[1] = f2(x1+tau*k3[0], x2+tau*k3[1], t+tau);
 x1 = x1 + (tau/6.0)*(k1[0]+2*k2[0]+2*k3[0]+k4[0]);
 x2 = x2 + (tau/6.0)*(k1[1]+2*k2[1]+2*k3[1]+k4[1]);
}

int main()
{
 cout << "|---------------------------------|" << endl;
 cout << "| Resuelva ecuaciones del tipo    |" << endl;
 cout << "|       y''(t)=f(y,y',t)          |" << endl;
 cout << "|con y un vector.                 |" << endl;
 cout << "|---------------------------------|" << endl;
 ofstream file("Solucion-2oOrdenV.txt");
 double t, tmax=2.0, tau=0.01; // Condiciones iniciales: xo=x, y(xo)=y, y'(xo)=yp
 Vector  r, v;
 r[0]=0.7*a; r[1]=0.0;     // r[2]= 0.0 by default
 v[0]=0.0; v[1]=6.0*a/T;
 for(t=0; t<tmax; t+=tau)
 {
  RK4(t,r,v,Ec_2o_Orden,tau);
  file << t << " " << r[0] << " " << r[1] << " " << r[2] << " ";
  file             << v[0] << " " << v[1] << " " << v[2] << " " << endl;
 }
 file.close();
 cout << "Su solucion se encuentra en el archivo Solucion-2oOrdenV.txt" << endl;
 return 0;
}
```

Listing 5.1: Programa `rk4-2ordV.cc` para resolver ecuaciones diferenciales vectoriales de 2º orden.

Gráfico de la solución

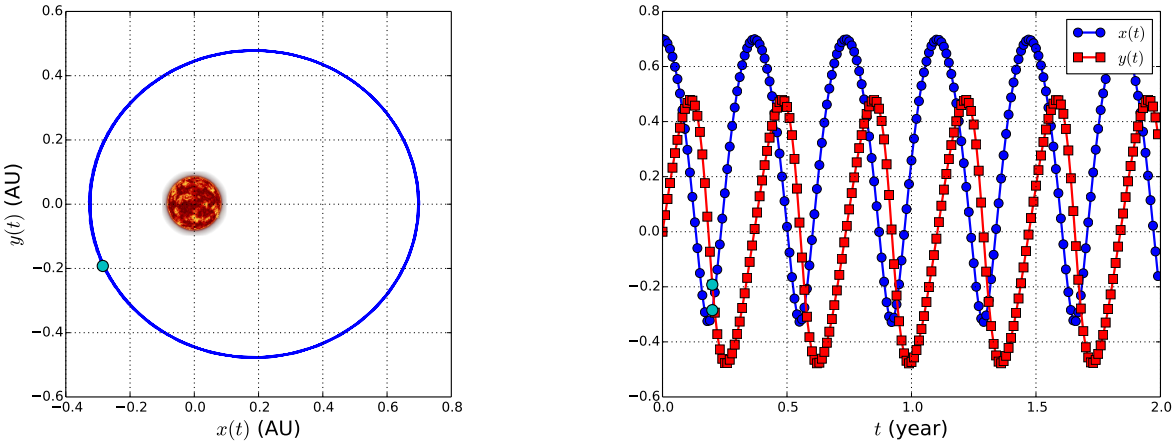

Figure 5.1: Trayectoria de un planeta en torno al sol. Condiciones iniciales: $\mathbf{r}(0) = (0.7a, 0, 0)$, $\mathbf{r}'(0) = (0, 6a/T, 0)$. A la izquierda, la trayectoria descrita por $\mathbf{r}(t)$. A la derecha, la evolución de sus componentes, $x(t)$ e $y(t)$ con el tiempo t.

De nuestra simulación, podemos inferir que el semieje mayor es $a_0 = (0.700a + 0.325a)/2 = 0.3669$ AU (Figura 5.2, izquierda), y que el periodo es $T_1 \approx 0.37$ años (Figura 5.2, derecha). Podemos comprobar nuestros resultados usando la teoría de gravitación de Newton, la cual predice que

$$T_1 = \sqrt{\frac{4\pi^2 a_0^3}{GM}} \approx 0.37 \text{ años.}$$

Esto nos permite comprobar que nuestro integrador funciona, por lo que podemos agregar considerar una fuerza más complejas, como por ejemplo una estrella sometida a un campo gravitacional de una galaxia, cuya solución no se conozca analíticamente; o planetas que ganan masa con el tiempo para modelar acreción.

5.3 Propiedades físicas

Podemos pedirle al programa que calcule propiedades físicas, como la energía y el momento angular. Basta agregar a nuestro código las siguientes funciones:

```
double Kin(Vector v){ return 0.5*m*(v[0]*v[0]+v[1]*v[1]+v[2]*v[2]); }
double Pot(Vector r){ return -G*M*m/abs(r); }
Vector L(Vector r, Vector p){ return Cross(r,p); }
```

Listing 5.2: Programa `rk4-2ordV.cc` en lenguaje `C++` para resolver ecuaciones diferenciales vectoriales de segundo orden.

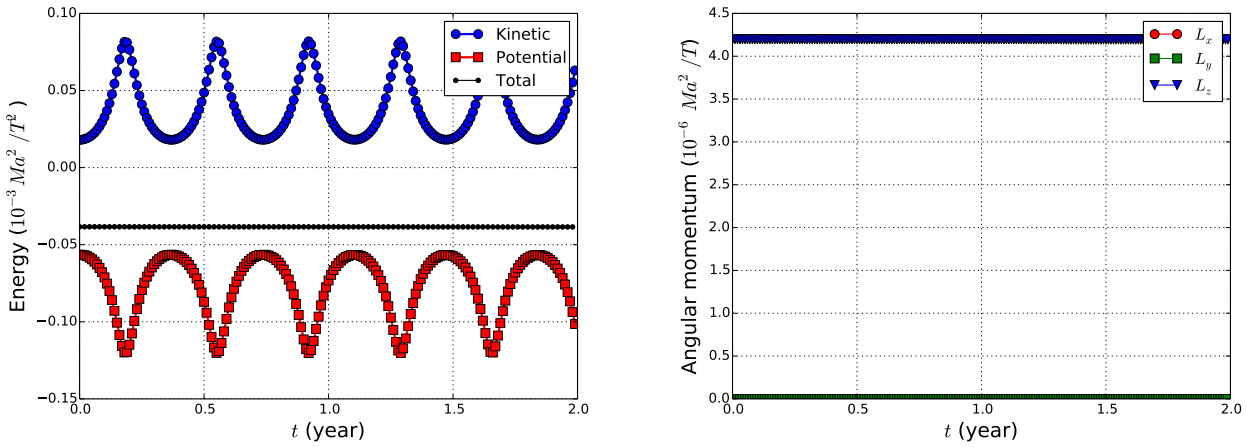

Figure 5.2: A la izquierda, energías cinética y potencial del planeta girando en torno al sol. A la derecha, componentes del momento angular en función del tiempo. Condiciones iniciales: $\mathbf{r}(0) = (0.7a, 0, 0)$, $\mathbf{r}'(0) = (0, 6a/T, 0)$. Notamos que tanto la energía total como el momento angular se conservan, como predice la teoría. Más aún, las componentes x e y son cero, y la componente z es positiva, lo que muestra que el planeta gira en sentido antehorario.

Chapter 6

Ecuaciones Diferenciales Vectoriales Acopladas de Segundo Orden: El problema de N cuerpos

Consideremos por un momento el problema de 3 cuerpos, de masas m_1, m_2 y m_3, interactuando bajo la acción de la gravedad. Las ecuaciones que rigen el movimiento son:

$$m_1\mathbf{r}_1''(t) = -\frac{Gm_1m_2}{\|\mathbf{r}_1-\mathbf{r}_2\|^3}(\mathbf{r}_1(t)-\mathbf{r}_2(t)) - \frac{Gm_1m_3}{\|\mathbf{r}_1-\mathbf{r}_3\|^3}(\mathbf{r}_1(t)-\mathbf{r}_3(t)), \tag{6.1a}$$

$$m_2\mathbf{r}_2''(t) = -\frac{Gm_2m_1}{\|\mathbf{r}_2-\mathbf{r}_1\|^3}(\mathbf{r}_2(t)-\mathbf{r}_1(t)) - \frac{Gm_2m_1}{\|\mathbf{r}_2-\mathbf{r}_3\|^3}(\mathbf{r}_2(t)-\mathbf{r}_3(t)), \tag{6.1b}$$

$$m_3\mathbf{r}_3''(t) = -\frac{Gm_3m_1}{\|\mathbf{r}_3-\mathbf{r}_1\|^3}(\mathbf{r}_3(t)-\mathbf{r}_1(t)) - \frac{Gm_3m_2}{\|\mathbf{r}_3-\mathbf{r}_2\|^3}(\mathbf{r}_3(t)-\mathbf{r}_2(t)). \tag{6.1c}$$

donde G es la constante de gravitación universal. Como vemos, las ecuaciones muestran que $\mathbf{r}_i''(t)$ depende de todos los demás $\mathbf{r}_j$ y podría, en general, depender de las velocidades $\mathbf{r}_j'$, como ocurre cuando hay disipación en el sistema. Por lo tanto, el problema de múltiples ecuaciones diferenciales vectoriales, acopladas, y de segundo orden, puede ser planteado como

$$\mathbf{r}_1''(t) \equiv \mathbf{F}_1(\mathbf{r}_1(t), \mathbf{r}_2(t), \ldots, \mathbf{r}_N(t), \mathbf{r}_1'(t), \mathbf{r}_2'(t), \ldots, \mathbf{r}_N'(t), t), \tag{6.2a}$$

$$\mathbf{r}_2''(t) \equiv \mathbf{F}_2(\mathbf{r}_1(t), \mathbf{r}_2(t), \ldots, \mathbf{r}_N(t), \mathbf{r}_1'(t), \mathbf{r}_2'(t), \ldots, \mathbf{r}_N'(t), t), \tag{6.2b}$$

$$\mathbf{r}_3''(t) \equiv \mathbf{F}_3(\mathbf{r}_1(t), \mathbf{r}_2(t), \ldots, \mathbf{r}_N(t), \mathbf{r}_1'(t), \mathbf{r}_2'(t), \ldots, \mathbf{r}_N'(t), t),$$

$$\vdots$$

$$\mathbf{r}_N''(t) \equiv \mathbf{F}_N(\mathbf{r}_1(t), \mathbf{r}_2(t), \ldots, \mathbf{r}_N(t), \mathbf{r}_1'(t), \mathbf{r}_2'(t), \ldots, \mathbf{r}_N'(t), t), \tag{6.2c}$$

donde $\mathbf{r}_i(t) = (r_{i1}, r_{i2}, \ldots, r_{ik})$, donde k es la cantidad de componentes de los vectores a considerar. Por simplicidad, consideraremos $k = 3$, es decir, $\mathbf{r}_i(t) = (x_i, y_i, z_i)$. Las funciones $\mathbf{F}_i$, aunque tomen varios vectores como argumento, entregan un vector del mismo espacio, es decir, $\mathbf{F} : \mathbb{R}^k \times \mathbb{R}^k \times \ldots \times \mathbb{R}^k \times \mathbb{R} \to \mathbb{R}^k$ es un campo vectorial que depende de las funciones $\mathbf{r}_i$, de sus derivadas $\mathbf{r}_i'$ y el parámetro $t \in \mathbb{R}$. Conociendo el valor de todas los $\mathbf{r}_i$ y $\mathbf{r}_i'$ en algún punto $t = t_n$, digamos $\mathbf{r}_i(t_n) \equiv \mathbf{r}_n = (x_i^{(n)}, y_i^{(n)}, z_i^{(n)})$, $\mathbf{r}_i'(t_n) \equiv \mathbf{v}_i^{(n)} = (x_i{}^{(n)}, y_i{}^{(n)}, z_i{}^{(n)})$, el método de Runge-Kutta permite aproximar, para un $\tau \in \mathbb{R}$ dado, el valor tanto de cada $\mathbf{r}_i$ y $\mathbf{r}_i'$ en $t_n + \tau$, con un error del orden de τ^5. La idea es convertir cada ecuación diferencial vectorial de segundo orden en dos

ecuaciones diferenciales vectoriales de primer orden y aplicar el método como lo hemos hecho en los previos capítulos. Para ello, definimos

$$\mathbf{x}_1^{(1)}(t) \equiv \mathbf{r}_1(t) \tag{6.3a}$$
$$\mathbf{x}_2^{(1)}(t) \equiv \mathbf{r}_1'(t) \tag{6.3b}$$
$$\mathbf{x}_1^{(2)}(t) \equiv \mathbf{r}_2(t) \tag{6.3c}$$
$$\mathbf{x}_2^{(2)}(t) \equiv \mathbf{r}_2'(t) \tag{6.3d}$$
$$\vdots \tag{6.3e}$$
$$\mathbf{x}_1^{(N)}(t) \equiv \mathbf{r}_N(t) \tag{6.3f}$$
$$\mathbf{x}_2^{(N)}(t) \equiv \mathbf{r}_N'(t), \tag{6.3g}$$
$$\tag{6.3h}$$

es decir, $\mathbf{x}_1^{(i)}$ es el $\mathbf{x}_1$ asociado a $\mathbf{r}_i(t)$, y $\mathbf{x}_2^{(i)}$ es el $\mathbf{x}_2$ asociado a $\mathbf{r}_i'(t)$. De esta manera, la notación es coherente con la usada en los capítulos anteriores, y obtenemos $2N$ ecuaciones acopladas de primer orden muy similares a las anteriormente obtenidas:

$$\mathbf{x}_1^{(j)\prime}(t) = \mathbf{x}_2^{(j)}(t) \tag{6.4a}$$
$$\mathbf{x}_2^{(j)\prime}(t) = \mathbf{r}_j''(t) = \mathbf{F}_j(\mathbf{x}_1^{(1)}(t), \mathbf{x}_1^{(2)}(t), \ldots, \mathbf{x}_1^{(N)}(t), \mathbf{x}_2^{(1)}(t), \mathbf{x}_2^{(2)}(t), \ldots, \mathbf{x}_2^{(N)}(t), t), \tag{6.4b}$$

para cada $j = 1, 2, \ldots, N$. Podemos definir la colección de posiciones $\{\mathbf{x}_1^{(j)}\}_{j=1}^N$ como un arreglo de vectores $\mathbf{X}_1$, al igual que la colección de velocidades $\{\mathbf{x}_2^{(j)}\}_{j=1}^N$ y funciones vectoriales $\{\mathbf{F}_j\}_{j=1}^N$ como arreglos $\mathbf{X}_2$ y $\mathbf{\Phi}$, respectivamente, de la siguiente manera:

$$\mathbf{X}_1(t) \equiv (\mathbf{x}_1^{(1)}(t), \mathbf{x}_1^{(2)}(t), \ldots, \mathbf{x}_1^{(N)}(t)), \tag{6.5a}$$
$$\mathbf{X}_2(t) \equiv (\mathbf{x}_2^{(1)}(t), \mathbf{x}_2^{(2)}(t), \ldots, \mathbf{x}_2^{(N)}(t)), \tag{6.5b}$$
$$\mathbf{F}(\mathbf{X}_1(t), \mathbf{X}_2(t), t) \equiv (\mathbf{F}_1(\mathbf{X}_1(t), \mathbf{X}_2(t), t), \mathbf{F}_2(\mathbf{X}_1(t), \mathbf{X}_2(t), t), \ldots, \mathbf{F}_N(\mathbf{X}_1(t), \mathbf{X}_2(t), t)). \tag{6.5c}$$

Esto nos ayuda a simplificar la notación, ya que podemos compactar todas las ecuaciones en (6.4) en dos simples ecuaciones:

$$\mathbf{X}_1'(t) = \mathbf{X}_2(t) \tag{6.6a}$$
$$\mathbf{X}_2'(t) = \mathbf{F}(\mathbf{X}_1(t), \mathbf{X}_2(t), t), \tag{6.6b}$$

las cuales toman la misma forma que en los casos anteriores, es decir, (3.3) y (5.3). Ahora nuestro problema ya no es las ecuaciones (6.2), sino el conjunto de ecuaciones vectoriales acopladas en (6.6). Sobre este conjunto, aplicamos el método de Runge-Kutta: definimos una función (campo vectorial) $\mathbf{f}$, uno para cada ecuación diferencial:

$$\mathbf{X}_1'(t) = \mathbf{f}_1(\mathbf{X}_1(t), \mathbf{X}_2(t), t) \equiv \mathbf{X}_2(t), \tag{6.7a}$$
$$\mathbf{X}_2'(t) = \mathbf{f}_2(\mathbf{X}_1(t), \mathbf{X}_2(t), t) \equiv \mathbf{F}(\mathbf{X}_1(t), \mathbf{X}_2(t), \mathbf{X}_2(t), t), \tag{6.7b}$$

Ahora aplicamos las definiciones en (2.2) a cada una de las ecuaciones diferenciales de primer orden en (6.7). Como vimos en el capítulo anterior, cada ecuación diferencial vectorial de primer orden tiene asociados 4 vectores $\mathbf{k}$, por lo tanto necesitaremos ocho $\mathbf{k}$-es en total por cada partícula ($8N$

en total). Definiendo $\mathbf{K}_j^{(i)}$ como el arreglo de vectores $\mathbf{k}_j$ asociado a $\mathbf{f}_i$ ($j = 1, 2, 3, 4;\ i = 1, 2$), tenemos

$$\mathbf{K}_1^{(1)} = \mathbf{f}_1\left(\mathbf{X}_1{}^{(n)}, \mathbf{X}_2{}^{(n)}, t_n\right) \tag{6.8a}$$

$$\mathbf{K}_1^{(2)} = \mathbf{f}_2\left(\mathbf{X}_1{}^{(n)}, \mathbf{X}_2{}^{(n)}, t_n\right) \tag{6.8b}$$

$$\mathbf{K}_2^{(1)} = \mathbf{f}_1\left(\mathbf{X}_1{}^{(n)} + \frac{\tau}{2}\mathbf{K}_1^{(1)}, \mathbf{X}_2{}^{(n)} + \frac{\tau}{2}\mathbf{K}_1^{(2)}, t_n + \frac{\tau}{2}\right) \tag{6.8c}$$

$$\mathbf{K}_2^{(2)} = \mathbf{f}_2\left(\mathbf{X}_1{}^{(n)} + \frac{\tau}{2}\mathbf{K}_1^{(1)}, \mathbf{X}_2{}^{(n)} + \frac{\tau}{2}\mathbf{K}_1^{(2)}, t_n + \frac{\tau}{2}\right) \tag{6.8d}$$

$$\mathbf{K}_3^{(1)} = \mathbf{f}_1\left(\mathbf{X}_1{}^{(n)} + \frac{\tau}{2}\mathbf{K}_2^{(1)}, \mathbf{X}_2{}^{(n)} + \frac{\tau}{2}\mathbf{K}_2^{(2)}, t_n + \frac{\tau}{2}\right) \tag{6.8e}$$

$$\mathbf{K}_3^{(2)} = \mathbf{f}_2\left(\mathbf{X}_1{}^{(n)} + \frac{\tau}{2}\mathbf{K}_2^{(1)}, \mathbf{X}_2{}^{(n)} + \frac{\tau}{2}\mathbf{K}_2^{(2)}, t_n + \frac{\tau}{2}\right) \tag{6.8f}$$

$$\mathbf{K}_4^{(1)} = \mathbf{f}_1\left(\mathbf{X}_1{}^{(n)} + \tau\mathbf{K}_3^{(1)}, \mathbf{X}_2{}^{(n)} + \tau\mathbf{K}_3^{(2)}, t_n + \tau\right) \tag{6.8g}$$

$$\mathbf{K}_4^{(2)} = \mathbf{f}_2\left(\mathbf{X}_1{}^{(n)} + \tau\mathbf{K}_3^{(1)}, \mathbf{X}_2{}^{(n)} + \tau\mathbf{K}_3^{(2)}, t_n + \tau\right) \tag{6.8h}$$

$$\mathbf{X}_1{}^{(n+1)} = \mathbf{X_1}^{(n)} + \frac{\tau}{6}\left[\mathbf{K}_1^{(1)} + 2\mathbf{K}_2^{(1)} + 2\mathbf{K}_3^{(1)} + \mathbf{K}_4^{(1)}\right] \tag{6.8i}$$

$$\mathbf{X}_2{}^{(n+1)} = \mathbf{X_2}^{(n)} + \frac{\tau}{6}\left[\mathbf{K}_1^{(2)} + 2\mathbf{K}_2^{(2)} + 2\mathbf{K}_3^{(2)} + \mathbf{K}_4^{(2)}\right]. \tag{6.8j}$$

donde $\mathbf{X}_1{}^{(n)} \equiv \{\mathbf{r}_n^{(j)}\}_{j=1}^N$ y $\mathbf{X}_2{}^{(n)} \equiv \mathbf{V}_n = \{\mathbf{r}'(t_n)^{(j)}\}_{j=1}^N$ son el conjunto de posiciones y velocidades en el paso n, respectivamente. Es importante recalcar nuevamente que el orden en que estas ecuaciones son declaradas es muy importante: todos los $\mathbf{K}_1$ deben ser declarados antes de los $\mathbf{K}_2$, ya que para obtener $\mathbf{K}_2^{(i)}$, es necesario conocer el valor de todos los $\mathbf{K}_1^{(i)}$. Lo mismo para $\mathbf{K}_3$ y $\mathbf{K}_4$. **Es un error común** en la implementación del código definir primero los cuatro $\mathbf{K}$ asociados a $\mathbf{f}_1$ ($\mathbf{K}_1^{(1)}, \mathbf{K}_2^{(1)}, \mathbf{K}_3^{(1)}$ y $\mathbf{K}_4^{(1)}$) y luego los $\mathbf{K}$ asociados a $\mathbf{f}_2$. Se advierte al programador no cometer ese error.

6.1 Ejemplo 1: N planetas interactuantes

Consideremos entonces el ejemplo propuesto, es decir, 3 cuerpos, de masas m_1, m_2 y m_3, interactuando bajo la acción de la gravedad. Las ecuaciones que rigen el movimiento son:

$$m_1\mathbf{r}_1''(t) = -\frac{Gm_1m_2}{\|\mathbf{r}_1 - \mathbf{r}_2\|^3}(\mathbf{r}_1(t) - \mathbf{r}_2(t)) - \frac{Gm_1m_3}{\|\mathbf{r}_1 - \mathbf{r}_3\|^3}(\mathbf{r}_1(t) - \mathbf{r}_3(t)), \tag{6.9a}$$

$$m_2\mathbf{r}_2''(t) = -\frac{Gm_2m_1}{\|\mathbf{r}_2 - \mathbf{r}_1\|^3}(\mathbf{r}_2(t) - \mathbf{r}_1(t)) - \frac{Gm_2m_1}{\|\mathbf{r}_2 - \mathbf{r}_3\|^3}(\mathbf{r}_2(t) - \mathbf{r}_3(t)), \tag{6.9b}$$

$$m_3\mathbf{r}_3''(t) = -\frac{Gm_3m_1}{\|\mathbf{r}_3 - \mathbf{r}_1\|^3}(\mathbf{r}_3(t) - \mathbf{r}_1(t)) - \frac{Gm_3m_2}{\|\mathbf{r}_3 - \mathbf{r}_2\|^3}(\mathbf{r}_3(t) - \mathbf{r}_2(t)). \tag{6.9c}$$

donde G es la constante de gravitación universal. Como vemos, las ecuaciones muestran que $\mathbf{r}_i''(t)$ depende de todos los demás $\mathbf{r}_j$ y podría, en general, depender de las velocidades $\mathbf{r}_j'$, como ocurre cuando hay disipación en el sistema. Por lo tanto, el problema de múltiples ecuaciones diferenciales vectoriales, acopladas, y de segundo orden, puede ser planteado como Consideremos las condiciones

inciales

$$\mathbf{r}_1(t_0) = (x_0, y_0, z_0) = \mathbf{x}_1^{(1)^{(0)}}, \tag{6.10a}$$
$$\mathbf{r}_2(t_0) = (x_0, y_0, z_0) = \mathbf{x}_1^{(2)^{(0)}}, \tag{6.10b}$$
$$\mathbf{r}_3(t_0) = (x_0, y_0, z_0) = \mathbf{x}_1^{(3)^{(0)}}, \tag{6.10c}$$
$$\mathbf{r}_1'(t_0) = (v_{x0}, v_{y0}, v_{z0}) = \mathbf{x}_2^{(1)^{(0)}}, \tag{6.10d}$$
$$\mathbf{r}_2'(t_0) = (v_{x0}, v_{y0}, v_{z0}) = \mathbf{x}_2^{(2)^{(0)}}, \tag{6.10e}$$
$$\mathbf{r}_3'(t_0) = (v_{x0}, v_{y0}, v_{z0}) = \mathbf{x}_2^{(3)^{(0)}} \tag{6.10f}$$

para cierto $t_0 \in \mathbb{R}$. En este caso, de acuerdo a (6.7), las funciones $\mathbf{f}$ están dadas por

$$\mathbf{f}_1(\mathbf{X}_1(t), \mathbf{X}_2(t), t) = \mathbf{X}_2(t), \quad \mathbf{f}_2(\mathbf{X}_1(t), \mathbf{X}_2(t), t) = \mathbf{F}(\mathbf{X}_1(t), \mathbf{X}_2(t), t),$$

donde las $N = 3$ componentes de $\mathbf{F}$ son

$$\mathbf{F}_1(\mathbf{X}_1(t), \mathbf{X}_2(t), t) = -\frac{Gm_1m_2}{\left\|\mathbf{x}_1^{(1)} - \mathbf{x}_1^{(2)}\right\|^3}(\mathbf{x}_1^{(1)} - \mathbf{x}_1^{(2)}) - \frac{Gm_1m_3}{\left\|\mathbf{x}_1^{(1)} - \mathbf{x}_1^{(3)}\right\|^3}(\mathbf{x}_1^{(1)} - \mathbf{x}_1^{(3)}), \tag{6.11a}$$
$$\mathbf{F}_2(\mathbf{X}_1(t), \mathbf{X}_2(t), t) = -\frac{Gm_2m_1}{\left\|\mathbf{x}_1^{(2)} - \mathbf{x}_1^{(1)}\right\|^3}(\mathbf{x}_1^{(2)} - \mathbf{x}_1^{(1)}) - \frac{Gm_2m_1}{\left\|\mathbf{x}_1^{(2)} - \mathbf{x}_1^{(3)}\right\|^3}(\mathbf{x}_1^{(2)} - \mathbf{x}_1^{(3)}), \tag{6.11b}$$
$$\mathbf{F}_3(\mathbf{X}_1(t), \mathbf{X}_2(t), t) = -\frac{Gm_3m_1}{\left\|\mathbf{x}_1^{(3)} - \mathbf{x}_1^{(1)}\right\|^3}(\mathbf{x}_1^{(3)} - \mathbf{x}_1^{(1)}) - \frac{Gm_3m_2}{\left\|\mathbf{x}_1^{(3)} - \mathbf{x}_1^{(2)}\right\|^3}(\mathbf{x}_1^{(3)} - \mathbf{x}_1^{(2)}). \tag{6.11c}$$

Para un valor de τ suficientemente pequeño, tenemos:

$$\begin{aligned}
\mathbf{K}_1^{(1)} &= \mathbf{f}_1\left(\mathbf{X}_1{}^{(0)}, \mathbf{X}_2{}^{(0)}, t_0\right) = \mathbf{X}_2{}^{(0)} \\
\mathbf{K}_1^{(2)} &= \mathbf{f}_2\left(\mathbf{X}_1{}^{(0)}, \mathbf{X}_2{}^{(0)}, t_0\right) = \mathbf{F}(\mathbf{X}_1(t), \mathbf{X}_2(t), t) \\
&= \left(-\frac{Gm_1m_2}{\left\|\mathbf{x}_1^{(1)^{(0)}} - \mathbf{x}_1^{(2)^{(0)}}\right\|^3}(\mathbf{x}_1^{(1)^{(0)}} - \mathbf{x}_1^{(2)^{(0)}}) - \frac{Gm_1m_3}{\left\|\mathbf{x}_1^{(1)^{(0)}} - \mathbf{x}_1^{(3)^{(0)}}\right\|^3}(\mathbf{x}_1^{(1)^{(0)}} - \mathbf{x}_1^{(3)^{(0)}}), \right. \\
&\quad - \frac{Gm_2m_1}{\left\|\mathbf{x}_1^{(2)^{(0)}} - \mathbf{x}_1^{(1)^{(0)}}\right\|^3}(\mathbf{x}_1^{(2)^{(0)}} - \mathbf{x}_1^{(1)^{(0)}}) - \frac{Gm_2m_1}{\left\|\mathbf{x}_1^{(2)^{(0)}} - \mathbf{x}_1^{(3)^{(0)}}\right\|^3}(\mathbf{x}_1^{(2)^{(0)}} - \mathbf{x}_1^{(3)^{(0)}}), \\
&= \left. -\frac{Gm_3m_1}{\left\|\mathbf{x}_1^{(3)^{(0)}} - \mathbf{x}_1^{(1)^{(0)}}\right\|^3}(\mathbf{x}_1^{(3)^{(0)}} - \mathbf{x}_1^{(1)^{(0)}}) - \frac{Gm_3m_2}{\left\|\mathbf{x}_1^{(3)^{(0)}} - \mathbf{x}_1^{(2)^{(0)}}\right\|^3}(\mathbf{x}_1^{(3)^{(0)}} - \mathbf{x}_1^{(2)^{(0)}}), \right)
\end{aligned}$$

De manera análoga se van definiendo los demás $\mathbf{K}_j^{(1)}$ y $\mathbf{K}_j^{(2)}$, en donde las operaciones como $\mathbf{X}_1{}^{(0)} + \frac{\tau}{2}\mathbf{K}_1^{(1)}$ simplemente se van realizando por componentes. Iterando el proceso, es posible avanzar las posiciones en pasos de τ.

6.2 Código fuente N-planetas interactuantes

A continuación se muestra una implementación particular en `C++` del método de Runge-Kutta que resuelve la ecuación diferencial vectorial de segundo orden del ejemplo anterior, correspondiente a 3 cuerpos interactuando: bajo la acción de la gravedad. Las ecuaciones que rigen el movimiento son:

$$m_1\mathbf{r}_1''(t) = -\frac{Gm_1m_2}{\|\mathbf{r}_1-\mathbf{r}_2\|^3}(\mathbf{r}_1(t)-\mathbf{r}_2(t)) - \frac{Gm_1m_3}{\|\mathbf{r}_1-\mathbf{r}_3\|^3}(\mathbf{r}_1(t)-\mathbf{r}_3(t)), \tag{6.12a}$$

$$m_2\mathbf{r}_2''(t) = -\frac{Gm_2m_1}{\|\mathbf{r}_2-\mathbf{r}_1\|^3}(\mathbf{r}_2(t)-\mathbf{r}_1(t)) - \frac{Gm_2m_1}{\|\mathbf{r}_2-\mathbf{r}_3\|^3}(\mathbf{r}_2(t)-\mathbf{r}_3(t)), \tag{6.12b}$$

$$m_3\mathbf{r}_3''(t) = -\frac{Gm_3m_1}{\|\mathbf{r}_3-\mathbf{r}_1\|^3}(\mathbf{r}_3(t)-\mathbf{r}_1(t)) - \frac{Gm_3m_2}{\|\mathbf{r}_3-\mathbf{r}_2\|^3}(\mathbf{r}_3(t)-\mathbf{r}_2(t)). \tag{6.12c}$$

donde $G = 4\pi^2a^3/(T^2M)$ is la constante de gravitación universal, $a = 1$ AU (unidad astronómica), $T = 1$ año y M la masa del sol. Consideremos las condiciones inciales

$$\mathbf{r}_1(0) = (-2a, 0, 0), \tag{6.13a}$$
$$\mathbf{r}_2(0) = (2a, 0, 0), \tag{6.13b}$$
$$\mathbf{r}_3(0) = (4a, 0, 0), \tag{6.13c}$$
$$\mathbf{r}_1'(0) = (0, -2, 0)a/T, \tag{6.13d}$$
$$\mathbf{r}_2'(0) = (0, 2, 0)a/T, \tag{6.13e}$$
$$\mathbf{r}_3'(0) = (0, 7, 0)a/T \tag{6.13f}$$

Usando la clase de vectores que usamos anteriormente, y definiendo arreglos de estos vectores, podemos implementar el código de manera casi idéntica al código 5.2. Basta cambiar los vectores por `Vector a[N]`, con `N` es la cantidad de partículas, o punteros a vector `Vector *a`, para las variables $\mathbf{K}_i$, $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{f}_1$, $\mathbf{f}_2$.
El programa consiste en lo siguiente:

1. Creamos una clase de vectores, de modo que podamos operar con ellos (suma, multiplicación por escalar, dividir por escalar, extraer componentes, etc.). Si el usuario lo necesita, puede añadir nuevas operaciones entre los vectores, tales como el producto interno, que pueden ser necesarios para aceleraciones más complicadas.

2. Se define la función `RK4` que representa al integrador. Esta función se encarga de calcular será la posición y velocidad en el siguiente paso. Toma un valor de `t` (que al principio vale $t_0 = 0.0$), un `Vector x1` que representa a $\mathbf{r}(t)$ y un `Vector x2` que representa a $\mathbf{r}'(t)$.

3. Toma también una función `f2` de tres parámetros, que representa nuestra función $\mathbf{f}_2(\mathbf{x}_1, \mathbf{x}_2, t) = -\frac{GM}{r^3}\mathbf{r}(t)$, que más adelante se declara como `Ec_2o_Orden`, y un número `tau` que representa el paso de tiempo τ.

4. Usando esto, encuentra cuánto vale $\mathbf{r}$(`t+tau`) y $\mathbf{r}'$(`t+tau`).

5. Cada vez que esta función es llamada, se actualiza el valor de `x1` por $\mathbf{r}$(`t+tau`) y el de `x2` por $\mathbf{r}'$(`t+tau`), es decir, hace avanzar `x1` y `x2` un paso `tau`. El ciclo `for` se encarga de cambiar `t` por `t+tau`.

6. En el programa principal, `main`, simplemente ingresamos las condiciones iniciales por coordenada ($\mathbf{r}(0) = (0.7a, 0, 0)$, $\mathbf{r}'(0) = (0, 6a/T, 0)$), y llamamos a la función `Ec_2o_Orden` con un paso de tiempo `tau`$= 0.01$ dentro de un ciclo `for`. El ciclo se hace hasta que $t = 2$ años.

7. En cada *loop*, el programa arroja `t`, las componentes de `r` y `v` (que representan a t, $\mathbf{r}(t)$ y $\mathbf{r}'(t)$, respectivamente) al archivo `Solucion-2oOrdenV.txt`.

```
/*********************************************************/
/* INTEGRADOR DE ECUACIONES DIFERENCIALES                */
/*********************************************************/
// AUTOR: FELIPE GONZALEZ CATALDO, Octubre 2017.
// www.gnm.cl/fgonzalez
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;


//////////////////  CLASS VECTOR ///////////////////////////////
class Vector{     // Definimos el nombre de la clase: "Vector".
 private:
 double elem[3];  // Valor del elemento en la posicion i

 public:
 Vector(void) {  for(long i=0; i<3; i++) elem[i]=0.0e0; }
 Vector(double x,double y,double z) {  elem[0]=x; elem[1]=y; elem[2]=z; }
 inline double & operator[](int q)  { return elem[q]; }

};
Vector operator + (Vector x, Vector y){
 Vector z;
 for (int i=0; i<3; i++) z[i]=x[i]+y[i];
 return z;
}
Vector operator - (Vector x, Vector y){
 Vector z;
 for (int i=0; i<3; i++) z[i]=x[i]-y[i];
 return z;
}
Vector operator - (Vector x){
 Vector z;
 for (int i=0; i<3; i++) z[i]=-x[i];
 return z;
}
Vector operator * (double z, Vector y){
  Vector x=y;
  for (int i=0; i<3; i++) x[i] = z*x[i];
  return x;
}
Vector operator / (Vector y, double z){
  Vector x=y;
  for (int i=0; i<3; i++) x[i] = x[i]/z;
  return x;
}

double abs(Vector x){ return sqrt(x[0]*x[0]+x[1]*x[1]+x[2]*x[2]);}

Vector Cross(Vector a, Vector b){
 Vector c;
 c[0]=a[1]*b[2]-a[2]*b[1];
 c[1]=-a[0]*b[2]+a[2]*b[0];
 c[2]=a[0]*b[1]-a[1]*b[0];
 return c;
}
////////////////////// END OF CLASS VECTOR ////////////////////////

double G= 4.0*M_PI*M_PI;    // en (AU^3)/(sunmass*year^2)
double M = 1.0;             // en masas solares
```

```
double a = 1.0;             // en AU
double T = 1.0;             // en years
const int N=3;              // numero de particulas

Vector x1[N], x2[N];
Vector K1[N][2],K2[N][2],K3[N][2],K4[N][2];
Vector acc[N];
double m[N];

//------------------ Que ecuacion desea resolver? --------------------//
Vector *Ec_2o_Orden(Vector *X1, Vector *X2,double t)
{
 double mod, mod3;
 for (int i=0; i<N; i++)
 {
  acc[i]= 0*acc[i];
  for (int j=0; j<N; j++)
    if (i!=j)
    {
      mod=abs(X1[i]-X1[j]);
      mod3=mod*mod*mod;
      acc[i] = acc[i] -G*m[j]*(X1[i]-X1[j])/mod3;
    }
 }
 return  acc;
}
//--------------------------------------------------------------------//

double Kin(Vector *v, double *m){
 double k=0.0;
 for (int j=0; j<N; j++) k+= 0.5*m[j]*(v[j][0]*v[j][0]+v[j][1]*v[j][1]+v[j][2]*v[j][2]);
 return k;
}
double Pot(Vector *r, double *m){
  double p=0.0;
  for (int i=0; i<N; i++)
   for (int j=i+1; j<N; j++)
     p += -G*m[i]*m[j]/abs(r[i]-r[j]);
  return p;
}


/* Funcion Runge Kutta para ecuaciones de segundo orden , como y''(x) = f(y,y',x) */
Vector *f1(Vector *X1, Vector *X2, double t){ return X2;}
void RK4(double & t, Vector *X1, Vector *X2, Vector *f2(Vector*,Vector*,double), double tau)
{
 for (int j=0; j<N; j++)
 {
  K1[j][0] = f1(X1,X2,t)[j];
  K1[j][1] = f2(X1,X2,t)[j];
  // x1 y x2 seran argumentos para f1 y f2
  x1[j]= X1[j]+tau*K1[j][0]/2.0;
  x2[j]= X2[j]+tau*K1[j][1]/2.0;
 }
 for (int j=0; j<N; j++)
 {
  K2[j][0] = f1(x1, x2, t+tau/2.0)[j];
  K2[j][1] = f2(x1, x2, t+tau/2.0)[j];
  x1[j]= X1[j]+tau*K2[j][0]/2.0;
  x2[j]= X2[j]+tau*K2[j][1]/2.0;
 }
 for (int j=0; j<N; j++)
 {
  K3[j][0] = f1(x1, x2, t+tau/2.0)[j];
  K3[j][1] = f2(x1, x2, t+tau/2.0)[j];
  x1[j]= X1[j]+tau*K3[j][0];
  x2[j]= X2[j]+tau*K3[j][1];
 }
 for (int j=0; j<N; j++)
 {
  K4[j][0] = f1(x1, x2, t+tau)[j];
```

```
133    K4[j][1] = f2(x1, x2, t+tau)[j];
134   }
135   for (int j=0; j<N; j++)
136   {
137    X1[j] = X1[j] + (tau/6.0)*(K1[j][0]+2*K2[j][0]+2*K3[j][0]+K4[j][0]);
138    X2[j] = X2[j] + (tau/6.0)*(K1[j][1]+2*K2[j][1]+2*K3[j][1]+K4[j][1]);
139   }
140  }
141
142
143
144  int main()
145  {
146   cout << "|---------------------------------|" << endl;
147   cout << "| Resuelva ecuaciones del tipo    |" << endl;
148   cout << "|       y''(t)=f(y,y',t)          |" << endl;
149   cout << "|con y un vector.                 |" << endl;
150   cout << "|---------------------------------|" << endl;
151   ofstream file("Solucion-2oOrdenVN.txt");
152
153   double tmax=30.0, tau=0.0001; // Condiciones iniciales: xo=x, y(xo)=y, y'(xo)=yp
154   Vector  r[N], v[N];
155   m[0]=M;
156   m[1]=M;
157   m[2]=M*1e-6;
158
159   r[0]=Vector(-2,0,0); v[0]=Vector(0,-2,0);
160   r[1]=Vector(2,0,0);  v[1]=Vector(0,2,0);
161   r[2]=Vector(4,0,0);  v[2]=Vector(0,7,0);
162
163   for(double t=0; t<tmax; t+=tau)
164   {
165    file << t << " ";
166    for(int j=0;j<N;j++) file << r[j][0] << " " << r[j][1]<< " " << r[j][2] << " ";
167    file << Kin(v,m) << " " << Pot(r,m) << "\n";
168    RK4(t,r,v,Ec_2o_Orden,tau);
169   }
170   file.close();
171   cout << "Su solucion se encuentra en el archivo Solucion-2oOrdenVN.txt" << endl;
172
173   return 0;
174  }
```

Listing 6.1: Programa `rk4-2ordVN.cc` en lenguaje `C++` para resolver ecuaciones diferenciales vectoriales de segundo orden.

6.2.1 Gráfico de la solución

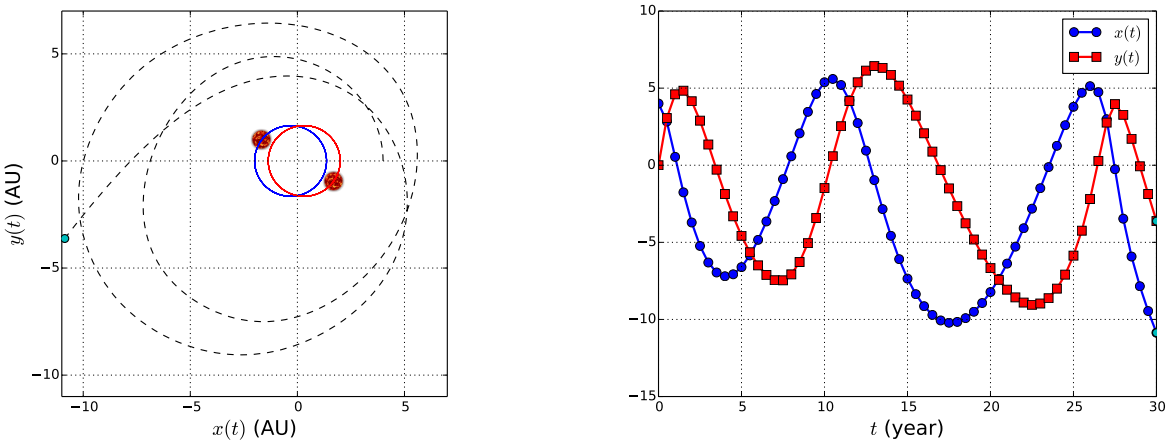

Figure 6.1: Trayectoria de un planeta en torno a un par de estrellas binarias. Ambas estrellas rotan en torno al centro de masas en (0,0). Condiciones iniciales: $\mathbf{r}_1(0) = (-2a, 0, 0), \mathbf{r}_2(0) = (2a, 0, 0), \mathbf{r}_3(0) = (4a, 0, 0), \mathbf{r}_1'(0) = (0, -2, 0)a/T, \mathbf{r}_2'(0) = (0, 2, 0)a/T, \mathbf{r}_3'(0) = (0, 7, 0)a/T$.

A pesar que las estrellas se mantienen rotando en torno a su centro de masa de manera estable, la trayectoria del planeta en torno a la binaria no es estable, y es eyectado del sistema luego de 30 años.

6.2.2 Propiedades físicas

Podemos pedirle al programa que calcule propiedades físicas, como la energía y el momento angular. Basta agregar a nuestro código las siguientes funciones:

```
1
2  double Kin(Vector *v, double *m){
3   double k=0.0;
4   for (int j=0; j<N; j++) k+= 0.5*m[j]*(v[j][0]*v[j][0]+v[j][1]*v[j][1]+v[j][2]*v[j][2]);
5   return k;
6  }
7  double Pot(Vector *r, double *m){
8     double p=0.0;
9     for (int i=0; i<N; i++)
10     for (int j=i+1; j<N; j++)
11       p += -G*m[i]*m[j]/abs(r[i]-r[j]);
12    return p;
13 }
14 Vector L(Vector *r, Vector *v, double *m){
15  Vector ll;
16  for (int i=0; i<N; i++) ll = ll + Cross(r[i],m[i]*v[i]);
17  return ll;
18 }
```

Listing 6.2: Programa `rk4-2ordV.cc` en lenguaje `C++` para resolver ecuaciones diferenciales vectoriales de segundo orden.

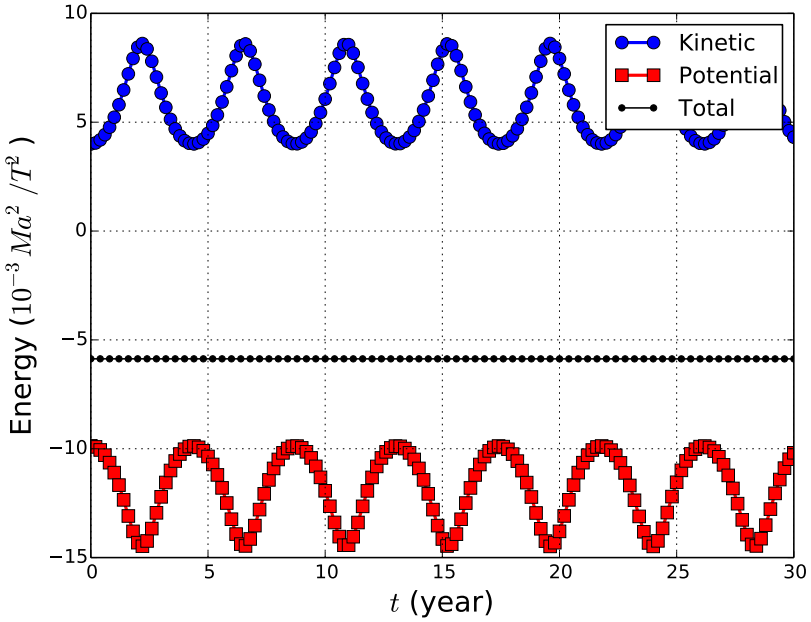

Figure 6.2: Energías cinética, potencial y total (conservada) del sistema.