25 DE SEPTIEMBRE DE 2022

MILESTONES 1

JAVIER ZATÓN MIGUEL

El objetivo de este programa es comparar los resultados del problema de Cauchy para una órbita Kepleriana, dadas unas condiciones iniciales de posición y velocidad.

$$\vec{U} = \begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{pmatrix}$$

$$\vec{U}(0) = \vec{U}_0$$

$$\frac{d\vec{U}}{dt} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \end{pmatrix} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dfrac{-x}{(x^2+y^2)^{3/2}} \\ \dfrac{-y}{(x^2+y^2)^{3/2}} \end{pmatrix}$$

Las condiciones iniciales del problema y el salto de tiempo vienen definidos al principio del programa.

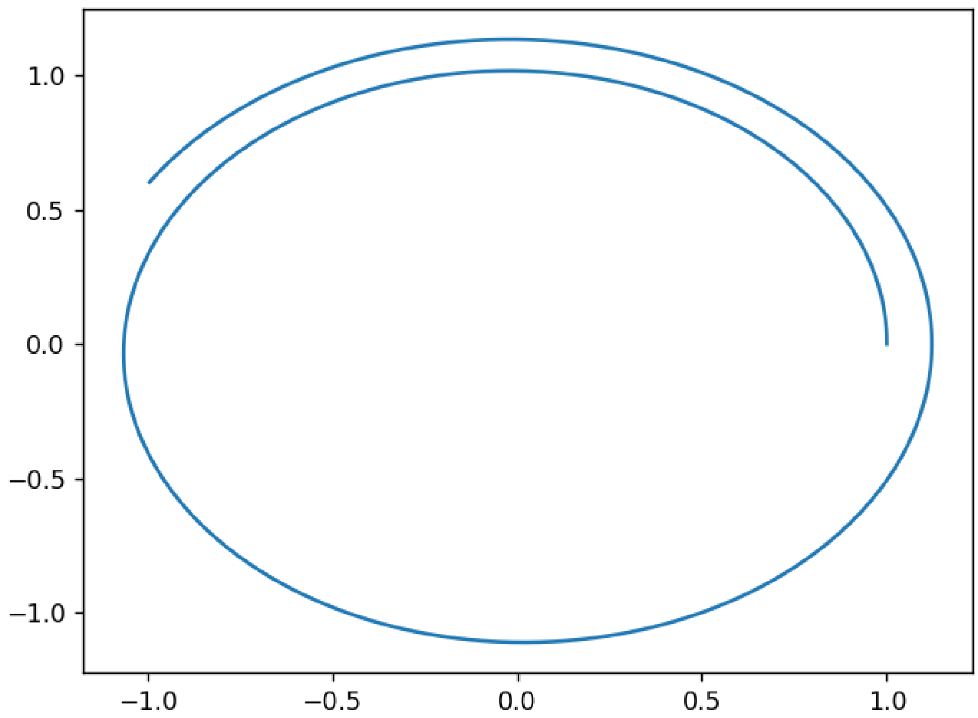
```
#Condiciones iniciales
x0 = 1.
vx0 = 0.
y0 = 0.
vy0 = 1.
U0 = np.array([[x0],[y0],[vx0],[vy0]])
U0.shape = (4,1)
#saltos de tiempo y tiempo final
t0=0
DeltaT= 0.01
tf=10
```

El bucle de la imagen inferior corresponde al esquema tipo Euler explícito.

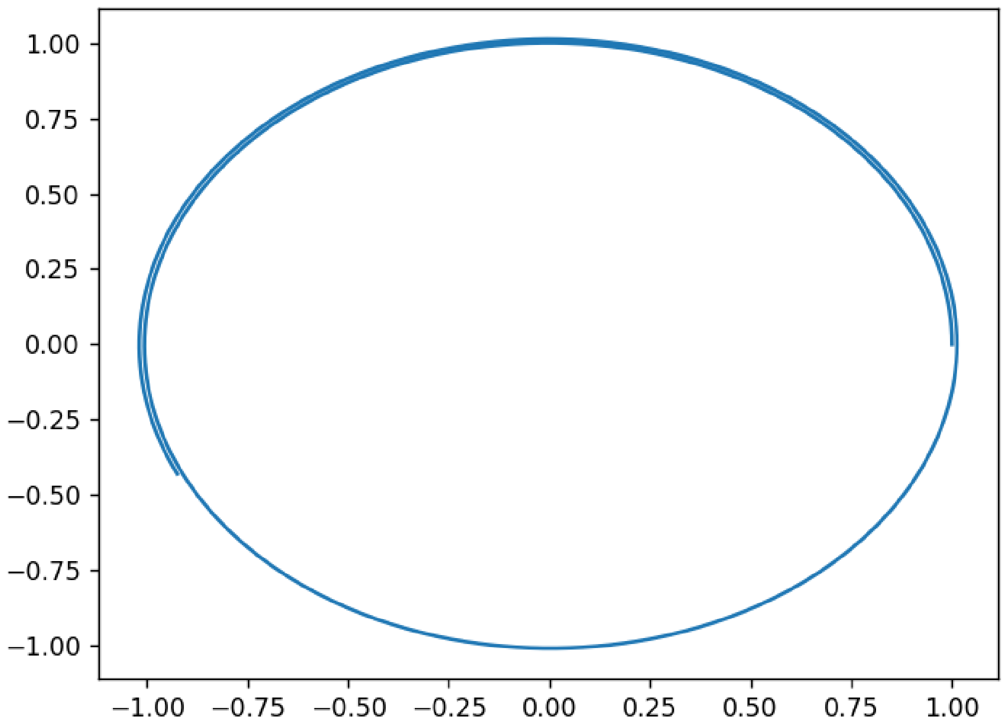
```
while (t < tf):
    Fpre = np.array([[Upre[2,0]],[Upre[3,0]],
                     [-(Upre[0,0])/(((Upre[0,0])**2+(Upre[1,0])**2.)**(3./2.))],
                     [-(Upre[1,0])/(((Upre[0,0])**2.+(Upre[1,0])**2.)**(3./2.))]])
    U = Upre + DeltaT*Fpre
    t=t+DeltaT
    Orbita2D = np.append(Orbita2D,U,axis = 1)
    Upre = U
```

Con los valores **x,y** se puede representar las órbitas en un plano 2D.

Para las condiciones iniciales (1,0,0,1) y un dt=0.01:



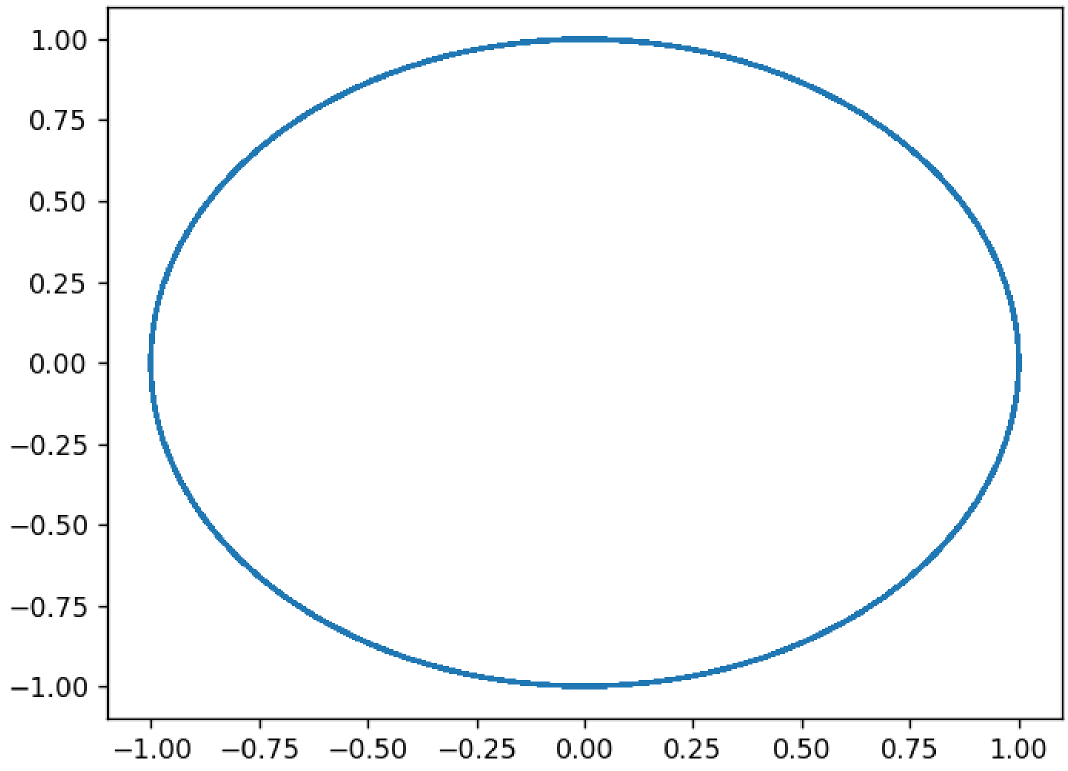
Si disminuimos el dt a un valor de 0.001, se obtiene un valor más preciso:



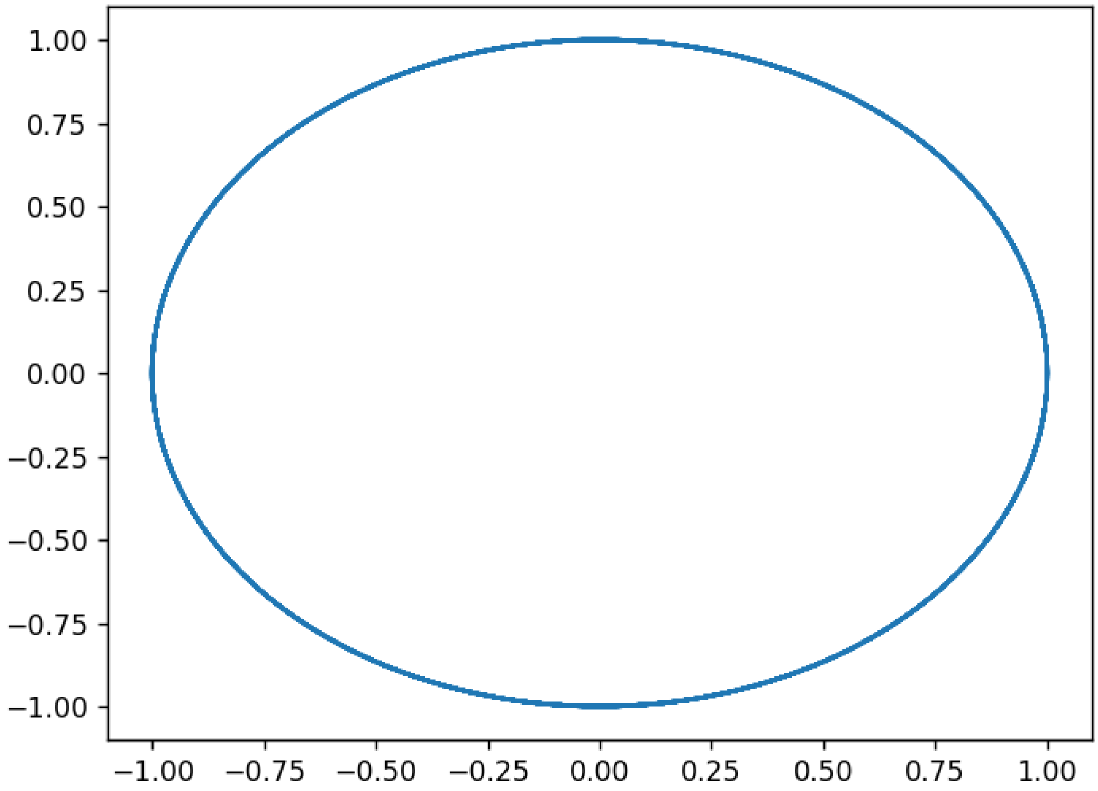
El caso del esquema de Runge Kutta de orden 4 tiene mayor complejidad que el anterior:

```
while (t < tf):
    k1 = np.array([[Upre[2,0]],[Upre[3,0]],
                   [-(Upre[0,0])/(((Upre[0,0])**2+(Upre[1,0])**2.)**(3./2.))],
                   [-(Upre[1,0])/(((Upre[0,0])**2.+(Upre[1,0])**2.)**(3./2.))]])
    U2 = Upre + k1*DeltaT/2
    k2 = np.array([[U2[2,0]],[U2[3,0]],
                   [-(U2[0,0])/(((U2[0,0])**2+(U2[1,0])**2.)**(3./2.))],
                   [-(U2[1,0])/(((U2[0,0])**2.+(U2[1,0])**2.)**(3./2.))]])
    U3 = Upre + k2*DeltaT/2
    k3 = np.array([[U3[2,0]],[U3[3,0]],
                   [-(U3[0,0])/(((U3[0,0])**2+(U3[1,0])**2.)**(3./2.))],
                   [-(U3[1,0])/(((U3[0,0])**2.+(U3[1,0])**2.)**(3./2.))]])
    U4 = Upre + k3*DeltaT
    k4 = np.array([[U4[2,0]],[U4[3,0]],
                   [-(U4[0,0])/(((U4[0,0])**2+(U4[1,0])**2.)**(3./2.))],
                   [-(U4[1,0])/(((U4[0,0])**2.+(U4[1,0])**2.)**(3./2.))]])
    k0 = (1/6.)*(k1 + 2.*k2 + 2.*k3 + k4)
    U = Upre + DeltaT*k0
    t=t+DeltaT
    Orbita2D = np.append(Orbita2D,U,axis = 1)
    Upre = U
```

De forma equivalente se calcula para las mismas condiciones iniciales y un dt=0.01:



De la misma forma, reduciendo el dt a un valor menor de 0.001:



Conclusiones:

El método Runge Kutta-4 ofrece un menor error acumulado con t, para las mismas condiciones iniciales que el esquema tipo Euler.

Otra forma de aumentar la precisión es reduciendo el salto de tiempo entre puntos , aunque esto aumenta el tiempo de proceso del programa.