Juan Carlos García-Taheño Hijes

1º MUSE ampliación de matemáticas I

Milestone 2:

Justificación de los Apartados:

1. Condiciones Iniciales y Definición de Función:

Este primer paso es establecer las condiciones iniciales del problema de Kepler, que involucra la posición y velocidad inicial de una partícula en el espacio. Además, se define la función **kepler_force**, que calcula la fuerza de Kepler en función de las posiciones y velocidades actuales de la partícula. Estas condiciones iniciales y la función de fuerza son fundamentales para llevar a cabo la integración numérica y resolver el problema.

```
        #Condiciones INICIALES

# Define condiciones iniciales

U0 = array([1.0, 0.0, 0.0, 1.0])  # Condiciones iniciales (x, y, vx, vy)
t0 = 0.0    # tiempo inicial
T = 10.0    # periodo o tiempo final
N = 1000    # numero de pasos
dt = 0.01   # paso de tiempo
```

```
# Funcion para la fuerza de Kepler
def kepler_force(U, t):
    x, y, vx, vy = U[0], U[1], U[2], U[3]
    r = (x**2 + y**2)**0.5
    return array([vx, vy, -x/(r**3), -y/(r**3)])
```

2. Método de Euler:

El método de Euler es una técnica numérica de integración simple pero menos preciso que algunos otros métodos, como veremos más adelante. Permite avanzar en el tiempo en pequeños pasos y calcular la nueva posición y velocidad de la partícula en cada paso.

```
def Euler(U, dt, t, F):

    return U + dt * F(U, t)

# los inputs parar la funcion Euler en un paso de Integracion N=1;
# tienen que ser el vector/matriz U
# el dt; paso de tiempo
# F la funcion del problema a integrar numericamente
# Por tanto; si el esquema es Euler y se busca el n+1, se busca la la definicion
# de como return Un+1= Un + dt*F(Un,t)
```

3. Método de Crank-Nicolson:

El método de Crank-Nicolson es un esquema implícito que mejora la precisión de la integración con respecto al método de Euler. Se busca encontrar una función que minimice la diferencia entre dos estados consecutivos en el tiempo.

```
# 2. Write a function called Crank_Nicolson to integrate one step.

#De la misma forma se busca la solucion en n+1 pero ahora el
# esquema es implicito luego si el esquema es Un+1=Un+0.5*(F(Un+1,t)+F(Un,t))
# para resolverlo se busca una funcion tal que Un+1-Un-0.5*(F(Un+1,t)+F(Un,t)) sea lo pedido
#y con ello se busque un 0 de la funcion

def Crank_Nicolson(U, dt, t, F ):

    def Residual_CN(X):

        return  X - a - dt/2 *  F(X, t + dt)

    a = U  +  dt/2 * F( U, t)
    return newton( Residual_CN, U )
#Esta seria la correcion hecha en clase del CN
```

4. Método de RK4 (Runge-Kutta de Cuarto Orden):

El método de RK4 es otro enfoque para la integración numérica que ofrece una mayor precisión en comparación con el método de Euler. Se basa en calcular cuatro pendientes ponderadas en varios puntos en el intervalo de tiempo y combinarlas para obtener una estimación más precisa del siguiente estado.

```
# 3. Write a function called RK4 to integrate one step
def RK4(U, dt, t, F ):

    k1 = F( U, t)
    k2 = F( U + dt * k1/2, t + dt/2 )
    k3 = F( U + dt * k2/2, t + dt/2 )
    k4 = F( U + dt * k3,   t + dt   )

    return  U + dt * ( k1 + 2*k2 + 2*k3 + k4 )/6
```

5. Método de Euler Inverso:

El método de Euler Inverso es otro esquema implícito que se utiliza para abordar problemas de integración numérica. Al igual que Crank-Nicolson, se enfoca en encontrar una función que minimice la diferencia entre dos estados en el tiempo.

```
# 4. Write a function called Inverse_Euler to integrate one step.

#pensando como en Crank Nicolson busco el implicito de  Euler Un+1=Un + dt*F(Un+1,t)
#llamo a otra funcion que me busque los ceros de la funcion como CN
# X-U-dt*F(X,t)=0
def Inverse_Euler(U, dt, t, F):

    def cerosINV(X):
        return X - U - dt * F(X, t)

    return newton(func = cerosINV, x0 = U )
```

6. Función para Integrar un Problema de Cauchy:

Se crea una función llamada **integrate_cauchy** que generaliza la integración numérica de los problemas de Cauchy. Esta función permite seleccionar un esquema temporal (cualquiera de los métodos mencionados) y resolver el problema de Kepler a lo largo de un intervalo de

tiempo especificado. Almacenará los resultados en matrices para su posterior visualización y análisis.

```
# 5. Write a function to integrate a Cauchy problem. Temporal scheme, initial
# condition and the function F(U, t) of the Cauchy problem should be input
# arguments.


def integrate_cauchy(EsquemaTemporal, U0, t0, T, dt, F):
    num_steps = int((T - t0) / dt) + 1      #numero pasos
    times = zeros(num_steps)                # tiempo; vector igual al numero de pasos
    states = zeros((len(U0), num_steps))    #matriz u por columnas, con n=numero de pasos de columnas

#condiciones iniciales
    t = t0
    U = U0
    step = 0

    while t <= T:
        times[step] = t # el tiempo va creciendo a T de dt en dt por cada bucle al igual que el paso
        states[:, step] = U          #almacen por columnas de las U soluciones

        U = EsquemaTemporal(U, dt, t, F)
        t =t+ dt
        step =step+ 1

    return times, states
```

```
# Elije un esquema temporal (por ejemplo, Euler) y realiza la integracion
times, states = integrate_cauchy(Euler, U0, t0, T, dt, kepler_force)
times_crank, states_crank = integrate_cauchy(Crank_Nicolson, U0, t0, T, dt, kepler_force)
times_rk4, states_rk4 = integrate_cauchy(RK4, U0, t0, T, dt, kepler_force)
times_inverse, states_inverse = integrate_cauchy(Inverse_Euler, U0, t0, T, dt, kepler_force)

# Graficos
plt.figure(1, figsize=(12, 4))

plt.subplot(141)
plt.plot(states[0, :], states[1, :], label='Euler')
plt.title('Orbita - Euler')
plt.axis('equal')

plt.subplot(142)
plt.plot(states_crank[0, :], states_crank[1, :], label='Crank-Nicolson')
plt.title('Orbita - Crank-Nicolson')
plt.axis('equal')

plt.subplot(143)
plt.plot(states_rk4[0, :], states_rk4[1, :], label='RK4')
plt.title('Orbita - RK4')
plt.axis('equal')

plt.subplot(144)
plt.plot(states_inverse[0, :], states_inverse[1, :], label='Inverse Euler')
plt.title('Orbita - Inverse Euler')
plt.axis('equal')

plt.tight_layout()
plt.show()
```

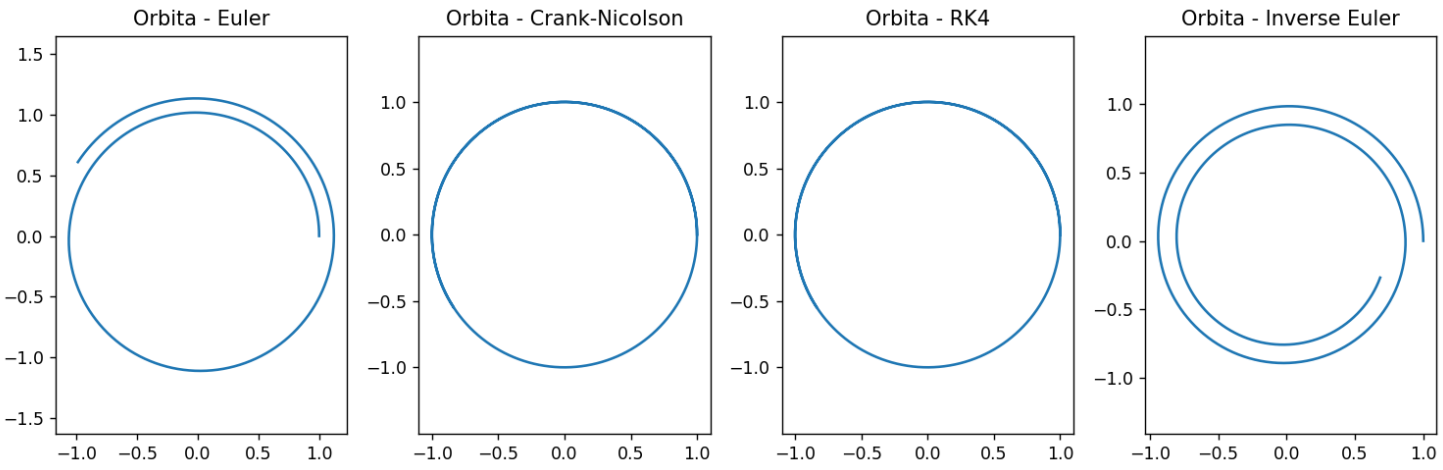

Imagen de la solución para dt=0.01

Juan Carlos García-Taheño Hijes

1º MUSE ampliación de matemáticas I

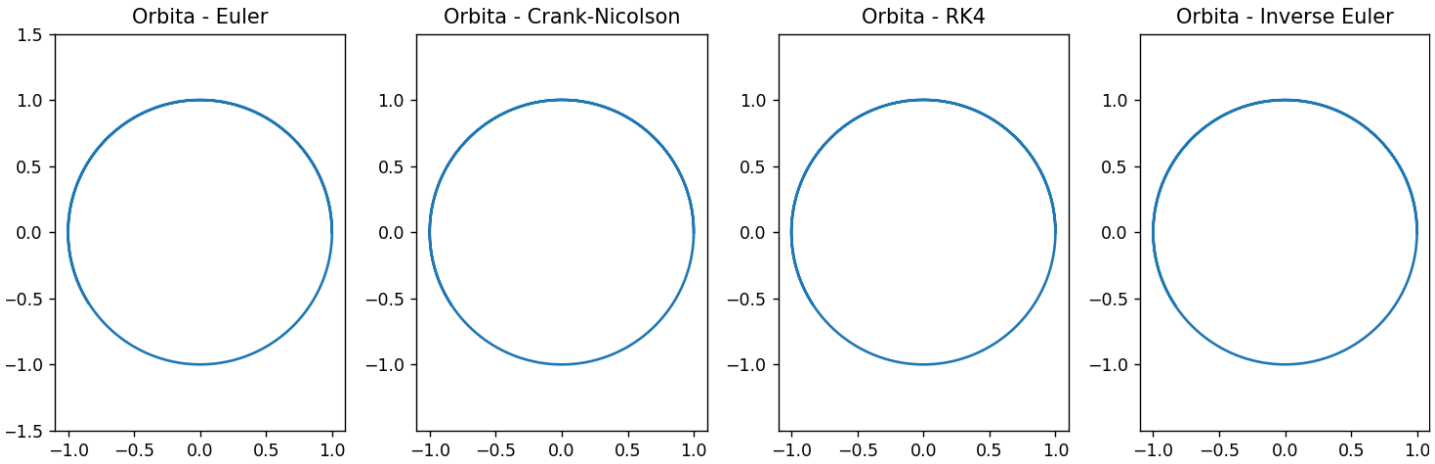

Imagen de la solución para dt=0.0001

7. Explicación de los Resultados al Integrar Kepler:

En los gráficos generados por la integración de Kepler con diferentes esquemas temporales (Euler, Crank-Nicolson, RK4 e Inverse Euler), podemos observar las trayectorias orbitales resultantes de cada método:

- Euler: Este esquema muestra una órbita que se aleja gradualmente del camino correcto debido a su naturaleza explícita. La energía total del sistema no se conserva, lo que resulta en una espiral que se aleja del origen.
- Crank-Nicolson: A diferencia de Euler, el método Crank-Nicolson es implícito y conserva mejor la energía total del sistema. La órbita resultante es mucho más precisa y estable en comparación con Euler.
- RK4: El método de Runge-Kutta de cuarto orden es altamente preciso. La órbita es casi idéntica a la esperada y se considera una mejora significativa sobre los métodos anteriores.
- Inverse Euler: Este esquema también es implícito, pero no del todo preciso.

8. Efecto de Variar el Paso de Tiempo:

Cambiar el valor del paso de tiempo (dt) tiene un impacto directo en la precisión y la eficiencia de la integración numérica.

Un paso de tiempo más pequeño produce resultados más precisos, pero con un mayor costo computacional, ya que se requieren más cálculos para avanzar en el tiempo.

Por otro lado, un paso de tiempo más grande puede generar resultados menos precisos, pero es más eficiente en términos de recursos computacionales. En la práctica, encontrar el equilibrio adecuado entre precisión y eficiencia al ajustar el valor de dt es esencial para obtener resultados confiables en problemas de integración numérica.

Juan Carlos García-Taheño Hijes

1º MUSE ampliación de matemáticas I

Resumen/Conclusión sobre la asignación de variables:

La asignación de variables es una técnica aumenta la legibilidad del código y simplifica la comprensión de las operaciones realizadas.

El costo computacional está mayormente determinado por la complejidad del algoritmo utilizado y la cantidad de operaciones ejecutadas en cada intervalo de tiempo. Además, la asignación de variables puede mejorar la eficiencia del código al evitar cálculos redundantes o innecesarios.

Luego el uso de este método es una práctica recomendable para mejorar la claridad del código; por ejemplo, respecto del hito 1.