ANÁLISIS NUMÉRICO I (75.12 – 95.04)

Resolución numérica de problemas de valores iniciales

Análisis de la suspensión de un vehículo: sistema oscilatorio amortiguado

Segundo cuatrimestre de 2023

Pelozo, Emanuel 99444 - epelozo@fi.uba.ar

Caporaletti, Tomás 108598 - tcaporaletti@fi.uba.ar

Introducción y Contexto:

El diseño de sistemas de suspensión en la industria automotriz es uno de los puntos críticos para garantizar la comodidad y seguridad de los pasajeros. En este informe se analiza el desafío de modelar numéricamente el sistema de suspensión de la rueda de un automóvil, centrándose en la dinámica del amortiguador y su capacidad de mantener la carrocería estable ante diferentes variaciones del terreno donde se desplaza.
El objetivo de esta investigación es aplicar y comparar métodos numéricos para la predicción de la respuesta de la suspensión bajo diferentes condiciones, en busca de una herramienta útil tanto para el diseño de nuevos vehículos cómo para la mejora de los existentes.

Marco teórico matemático y modelaciones numéricas elegidas.

Marco teórico:

El modelo matemático adoptado en este estudio considera un sistema dinámico unidimensional, representando la suspensión como un amortiguador y un resorte sujetos a una masa que simula la carrocería del vehículo. Este modelado se hará bajo la ley de Newton, la segunda ley del movimiento **F=ma** se aplica, donde **F** es la suma de las fuerzas actuantes en el sistema de suspensión, **m** la masa de una cuarta parte del vehículo (lo que correspondería a una rueda) y **a** la aceleración vertical a la que se somete el vehículo.
Las fuerzas actuantes a considerar serán las fuerzas elástica y de amortiguación:

- $F_{el} = k(c - y)$
- $F_{am} = \lambda(c' - y')$

donde k es la constante elástica del muelle [N/m], λ es la constante de amortiguación [Ns/m], c es la cota o elevación del terreno [m], y es la posición de la carrocería [m], c' e y' son derivadas de c e y con respecto al tiempo, es decir velocidades verticales [m/s]. La aceleración vertical de la carrocería puede expresarse como $a = y''$.

Teniendo en cuenta estas variables, la ecuación que utilizaremos para modelar el sistema será: $y`` = \frac{k}{m}(c - y) + \frac{\lambda}{m}(c` - y`)$

Modelaciones numéricas:

A través de la ecuación obtenida nos enfocaremos en métodos de resolución de ecuaciones diferenciales. Para este trabajo, se considerarán tres técnicas fundamentales: Euler implícito, Euler explícito y Runge-Kutta de orden 2.

A continuación, detallaremos el desarrollo de cada uno de los métodos que utilizaremos aplicados a este problema:

Euler explícito

Dada la ecuación diferencial:

$$y`` = \frac{k}{m}(c - y) + \frac{\lambda}{m}(c` - y`)$$

Aplicamos el siguiente cambio de variables:

- $U = Y$
- $U' = V$
- $V' = \frac{k}{m}(c - U) + \frac{\lambda}{m}(c' - V)$

Donde:

- $U' = f(u, v, t)$
- $V' = g(u, v, t)$

Finalmente, nos queda, por Euler explícito:

1. $U_{n+1} = U_n + h.V_n$
2. $V_{n+1} = V_n + h.(\frac{k}{m}(c - U_n) + \frac{\lambda}{m}(c' - V_n))$

(Con h siendo el paso entre punto y punto)

Euler implícito

A partir de cómo nos quedaron formuladas las ecuaciones de Euler explícito obtenemos las ecuaciones de Euler implícito, evaluando U_{n+1} y V_{n+1} en $f(u, v, t)$ y $g(u, v, t)$.

1. $U_{n+1} = U_n + h.V_{n+1}$
2. $V_{n+1} = V_n + h(\frac{k}{m}(c - U_{n+1}) + \frac{\lambda}{m}(c' - V_{n+1}))$

(Con h siendo el paso entre punto y punto)

Reemplazando 1 en 2, y despejando, obtenemos que:

$$V_{n+1} = \frac{V_n + h.\frac{k}{m}.(c - U_n) + \frac{\lambda}{m}.c'}{(1 + \frac{\lambda}{m} + h^2.\frac{k}{m})}$$

Lo cual es una expresión de V_{n+1} que depende de valores conocidos (explícita) y nos es útil para iterar, una vez obtenido su valor podemos también calcular U_{n+1}.

Runge Kutta

Dado que tenemos dos variables, para obtener sus respectivos valores mediante iteraciones hacemos:

- $U_{n+1} = U_n + \frac{1}{2}(q_{1u} + q_{2u})$
- $V_{n+1} = V_n + \frac{1}{2}(q_{1v} + q_{2v})$

con:

- $q_{1u} = h.f(u_n, v_n, t_n) = h.V_n$
- $q_{1v} = h.g(u_n, v_n, t_n) = h.(\frac{k}{m}(c - U_n) + \frac{\lambda}{m}(c' - V_n))$
- $q_{2u} = h.f(u_n + q_{1u}, v_n + q_{1v}, t_{n+1}) = h.(V_n + q_{1v})$
- $q_{2v} = h.g(u_n + q_{1u}, v_n + q_{1v}, t_{n+1}) = h.(\frac{k}{m}(c - (U_n + q_{1u})) + \frac{\lambda}{m}(c' - (V_n + q_{1v}))$

(Con f y g siendo las mismas funciones que utilizamos en Euler explícito e implícito)

Aplicación de los métodos

La teoría nos dice que la precisión de los métodos numéricos utilizados se ve significativamente influenciada por la elección del tamaño de paso h. Esta sección se dedica al análisis experimental de esta influencia para cada método en busca de la obtención del mejor.

Para la obtención del error, vamos a evaluar la desviación de los resultados numéricos de una solución que vamos a tomar como la “teórica”, la cual fue obtenida utilizando un tamaño de paso muy pequeño $1x10^{-6}$. Por otro lado, compararemos la frecuencia de oscilación del mejor resultado de cada método contra el resultado teórico brindado por la siguiente ecuación.

A través de este análisis, se busca entender cómo la granularidad de la simulación afecta la fidelidad de la representación del sistema real y determinar el equilibrio óptimo entre la precisión computacional y el costo de cálculo.

Este análisis se enfocará en diferentes escenarios de amortiguación, el primero sin amortiguación y el segundo con ella. Aprovecharemos el primer escenario para analizar cuál es el mejor método con el que trabajar y en base a la elección, seguiremos con ese método para el segundo escenario, en el cual también repetiremos el análisis teniendo en cuenta diferentes valores de amortiguación y constante elástica.

Escenario sin amortiguación

Para este primer escenario vamos a tomar los siguientes parámetros:

- masa m = 99444/200
- k = 25000 N/m
- λ = 0 Ns/m

- c = 0.1 m $\forall t$
- Intervalo de tiempo a evaluar: 5 segundos.

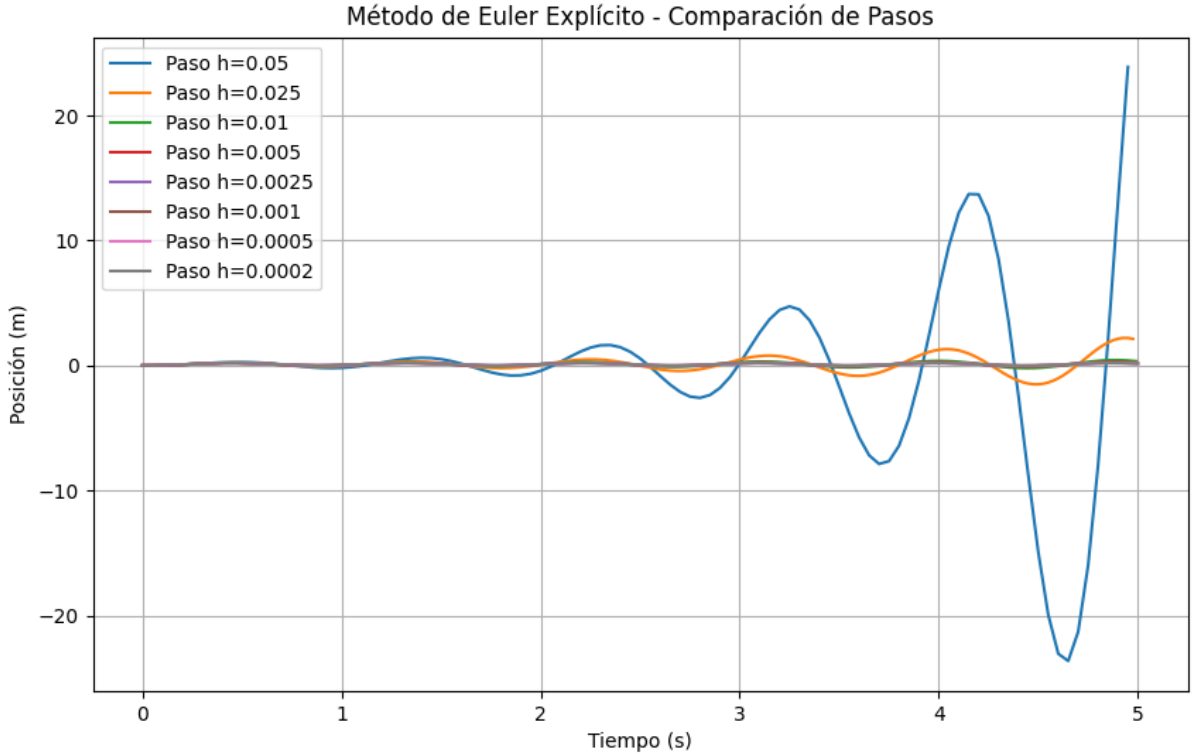
A través de estos valores vamos a evaluar cómo se comporta cada uno de los métodos numéricos. Cabe destacar que lo que se espera de este primer experimento es que las oscilaciones se mantengan con la misma frecuencia durante todo el tiempo de evaluación, esto es debido a la constante de amortiguación como λ = 0.

Por otro lado, asumimos los valores iniciales de posición y velocidad como 0 en el instante inicial. Finalmente, para todos los métodos utilizaremos los siguientes tamaños de paso: **0.05, 0.025, 0.01, 0.005, 0.0025, 0.001, 0.0005, 0.0002.**

Resultados

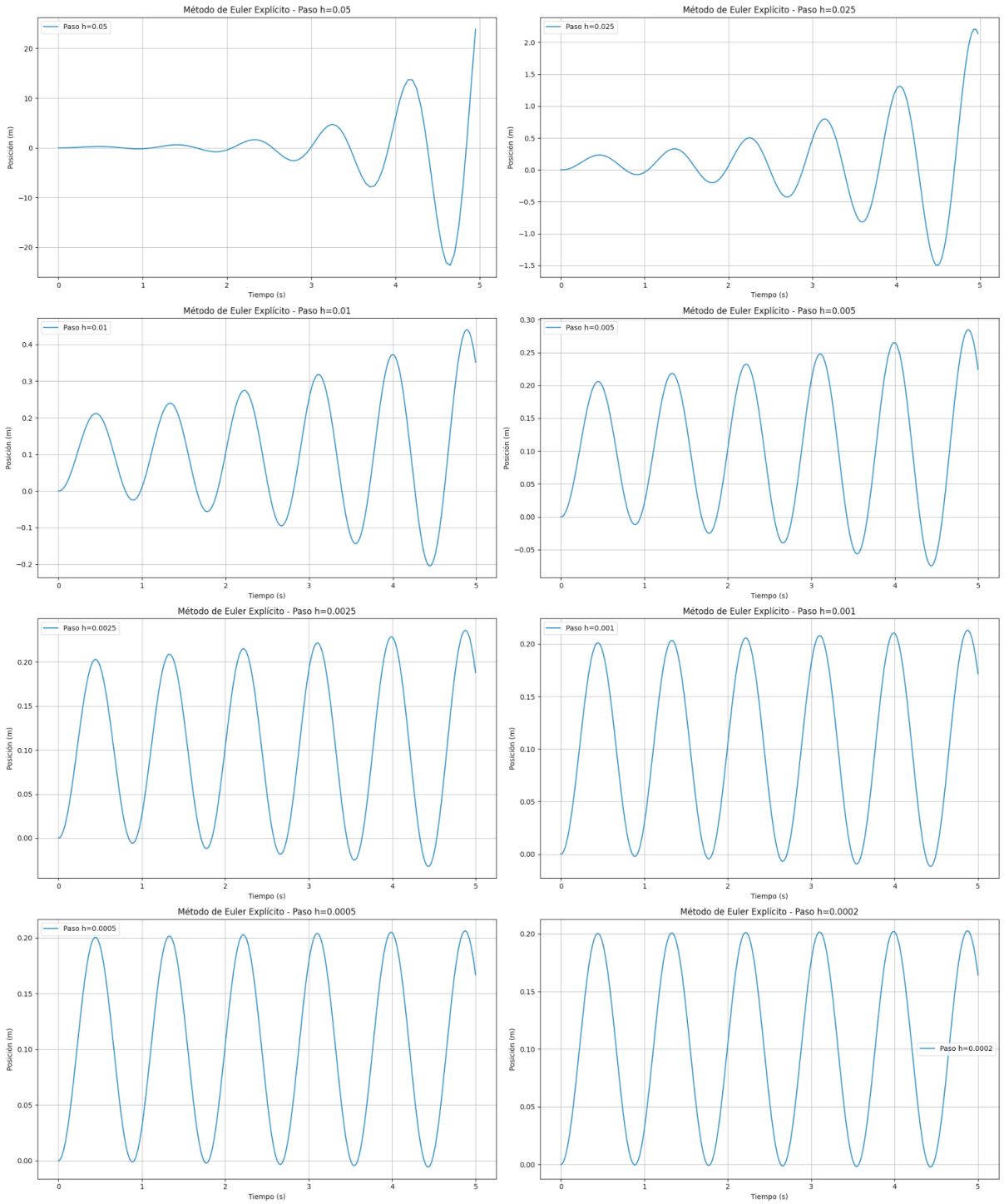
Euler explícito:

Evaluando el algoritmo de euler explícito en los pasos ya mencionados obtenemos lo siguiente:



De este gráfico, lo que se puede observar es que los dos pasos de mayor tamaño 0.05 y 0.025 son los que más tienden a aumentar la oscilación a medida que se alejan el el tiempo. Tal es la diferencia de escala que el resto de pasos parecen unirse en una misma línea cercana a 0.

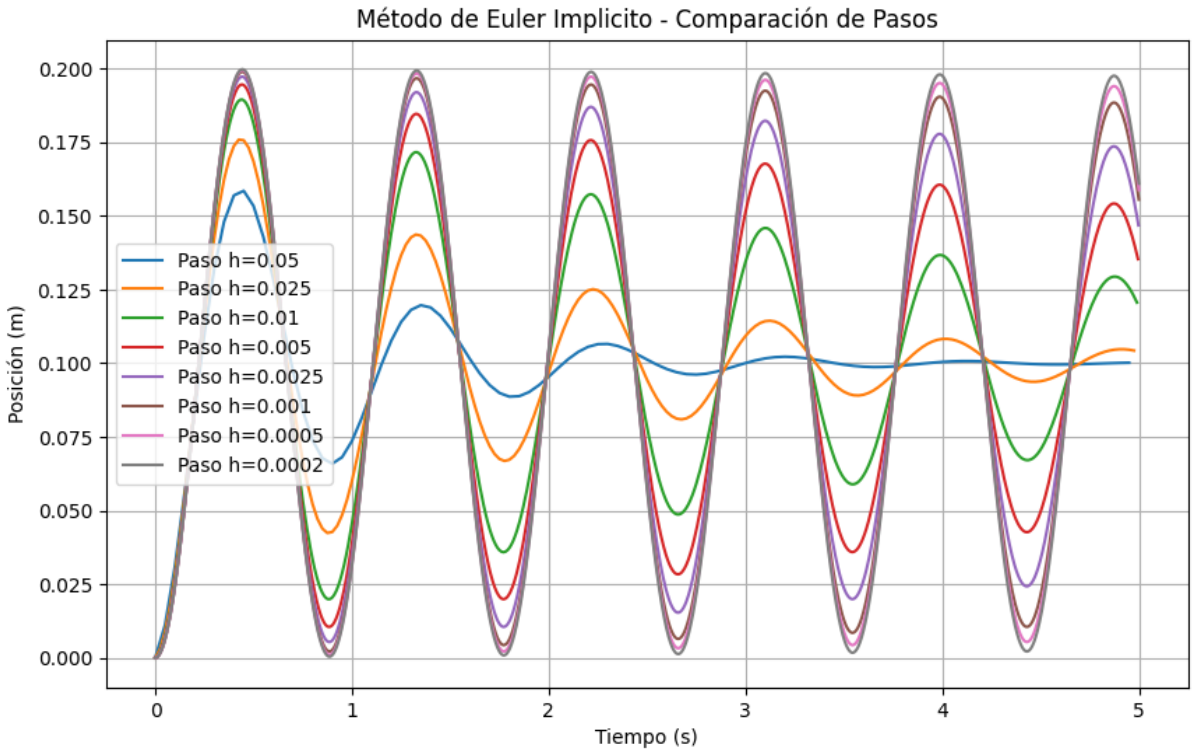
Separando el análisis para cada paso en su propia escala podemos observar su comportamiento más claramente:



Ahora podemos observar con mayor claridad que para pasos menores, la posición de los picos es mucho menor que para pasos mayores. Obtuvimos de conclusión a partir de los resultados experimentales, que el aumento en la oscilación a medida que avanza el tiempo es debido a que se está trabajando con un paso h menos estable, es decir, más cercano a sus límites de estabilidad. Podríamos repetirlo aumentando el valor h muy gradualmente durante muchas iteraciones y sacar un límite de estabilidad .

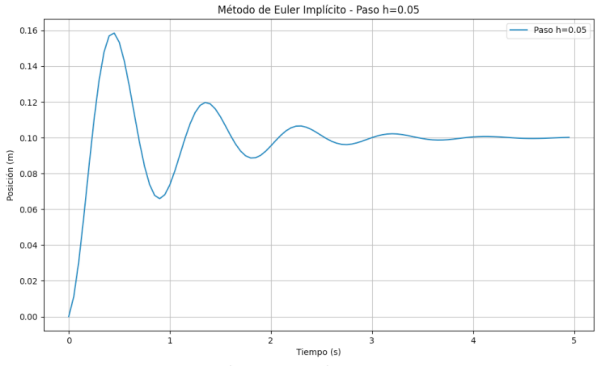
Euler implícito:

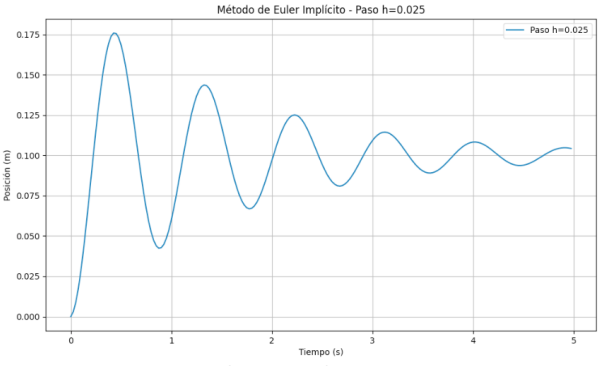
Comparación de pasos:

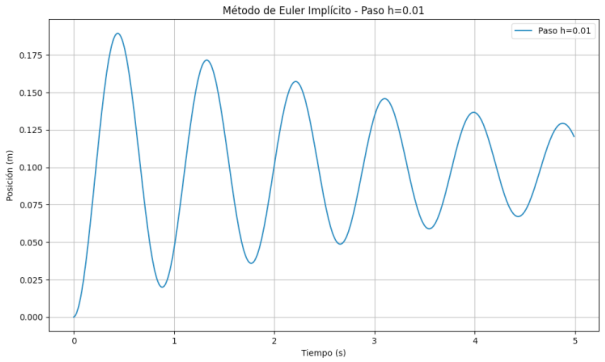


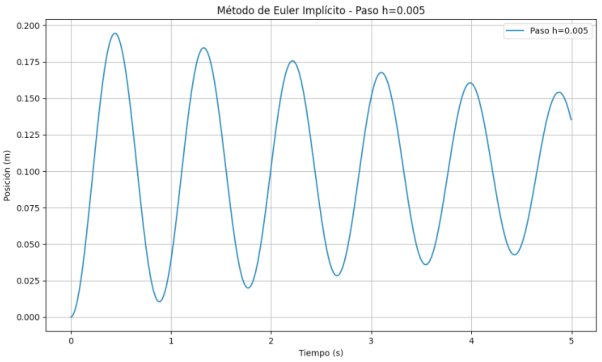
Para Euler Explícito vemos que también tenemos una imprecisión para los pasos más grandes, pero a diferencia de Euler Implícito el error se manifiesta en la tendencia a dejar de oscilar y establecerse hacia el valor **0.100.**

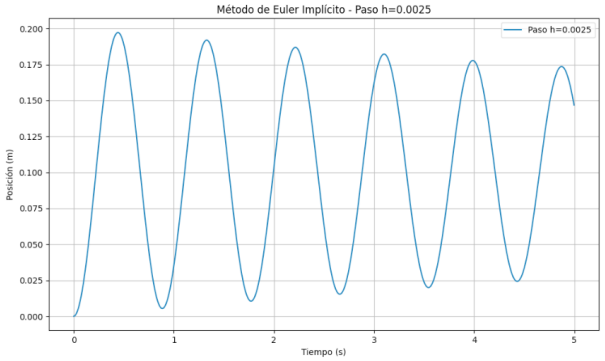
Separando el análisis para cada paso:

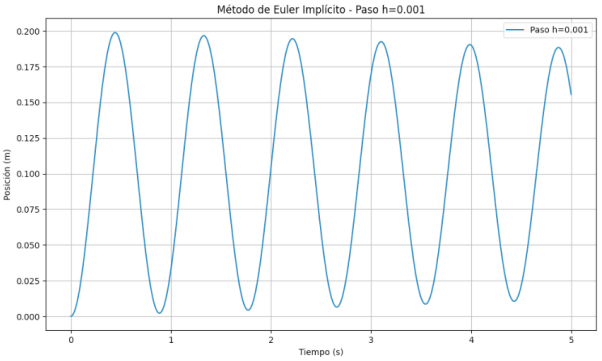

Método de Euler Implícito - Paso h=0.05
Paso h=0.05
Posición (m)
Tiempo (s)

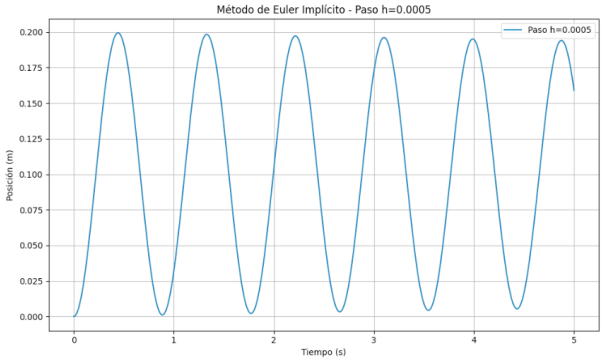

Método de Euler Implícito - Paso h=0.025
Paso h=0.025
Posición (m)
Tiempo (s)

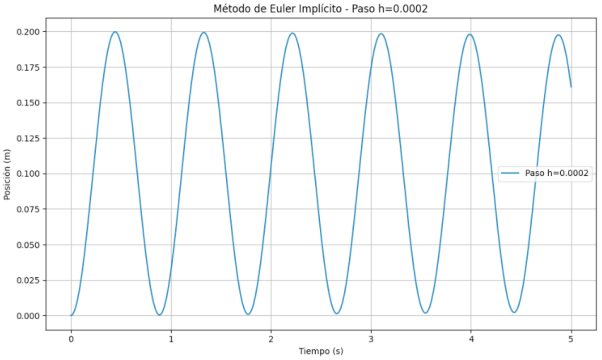

Método de Euler Implícito - Paso h=0.01
Paso h=0.01
Posición (m)
Tiempo (s)


Método de Euler Implícito - Paso h=0.005
Paso h=0.005
Posición (m)
Tiempo (s)


Método de Euler Implícito - Paso h=0.0025
Paso h=0.0025
Posición (m)
Tiempo (s)


Método de Euler Implícito - Paso h=0.001
Paso h=0.001
Posición (m)
Tiempo (s)

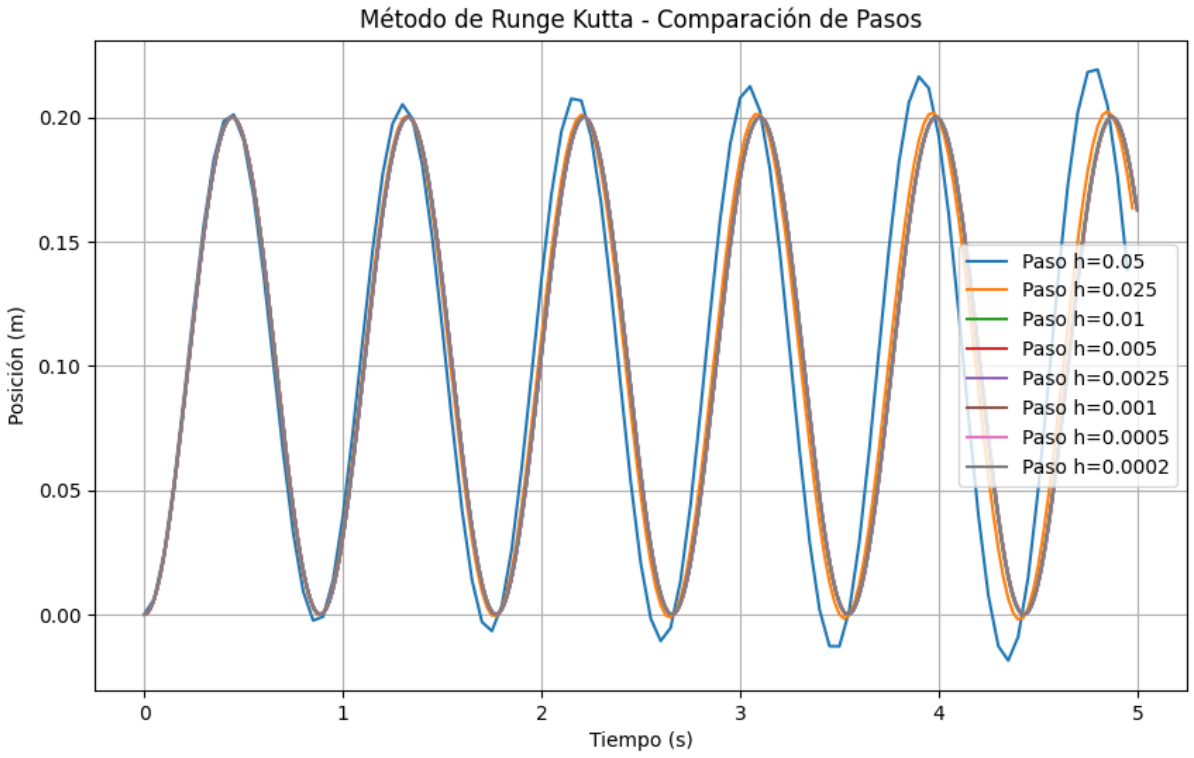

Método de Euler Implícito - Paso h=0.0005
Paso h=0.0005
Posición (m)
Tiempo (s)


Método de Euler Implícito - Paso h=0.0002
Paso h=0.0002
Posición (m)
Tiempo (s)

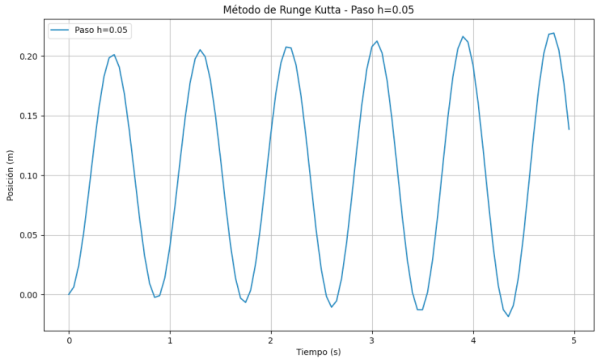
Runge Kutta:

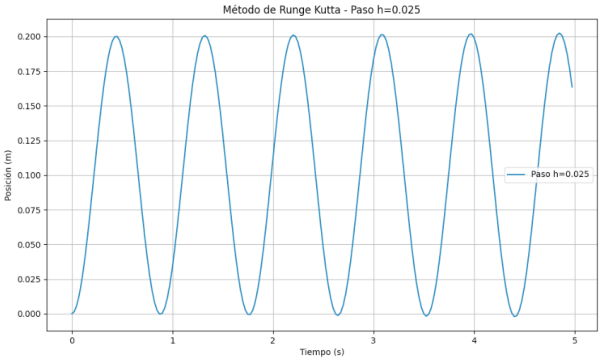
Para Runge Kutta de orden 2, lo que podemos observar es que, si bien hay una diferencia entre los pasos más grandes y los más pequeños, esta no es tan grande como en los métodos anteriores, el tamaño de paso que más difiere del resto es el 0.05, mientras que el resto se encuentra en una oscilación bastante similar.

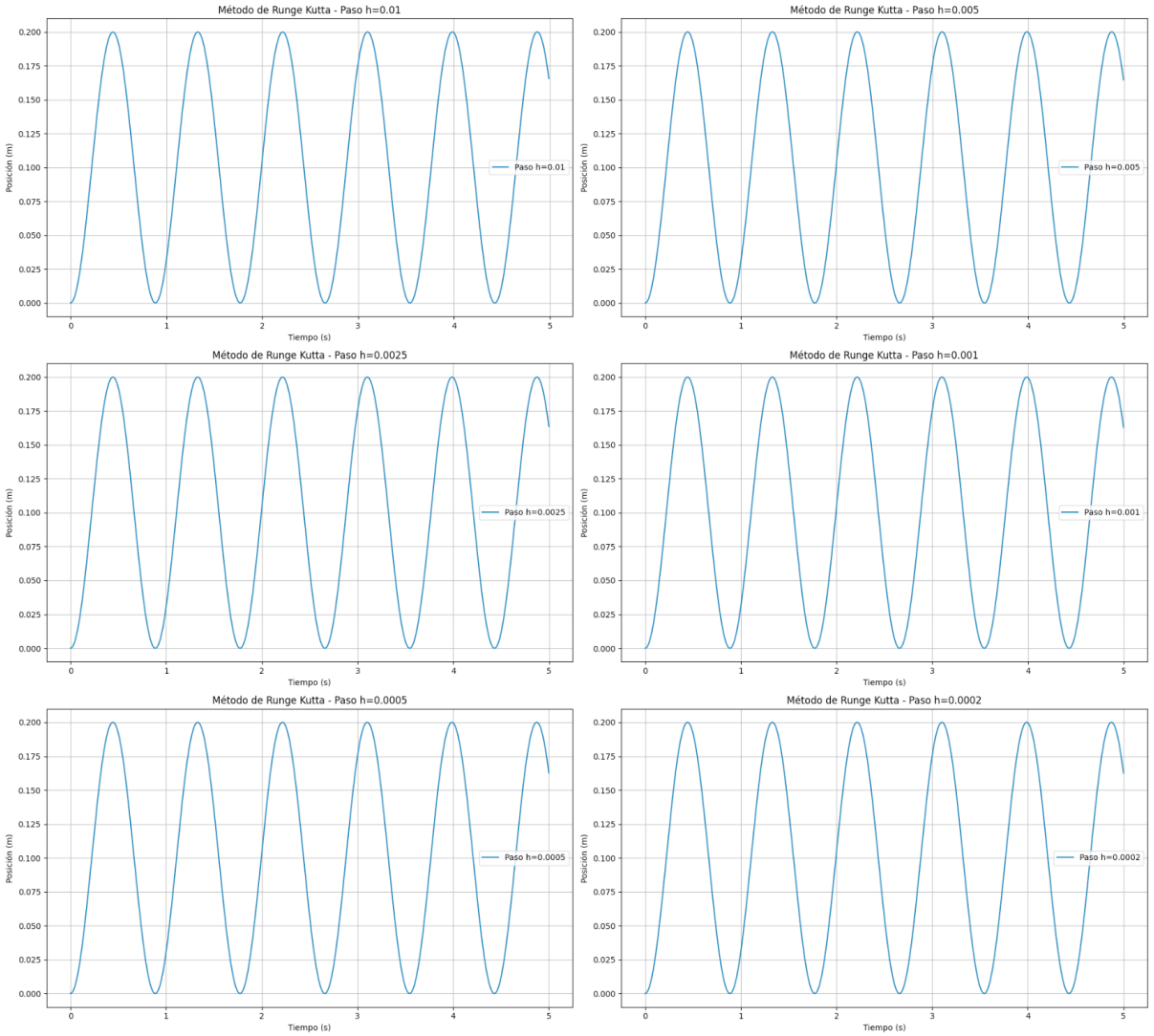
Comparación de pasos:



Separando el análisis para cada paso:



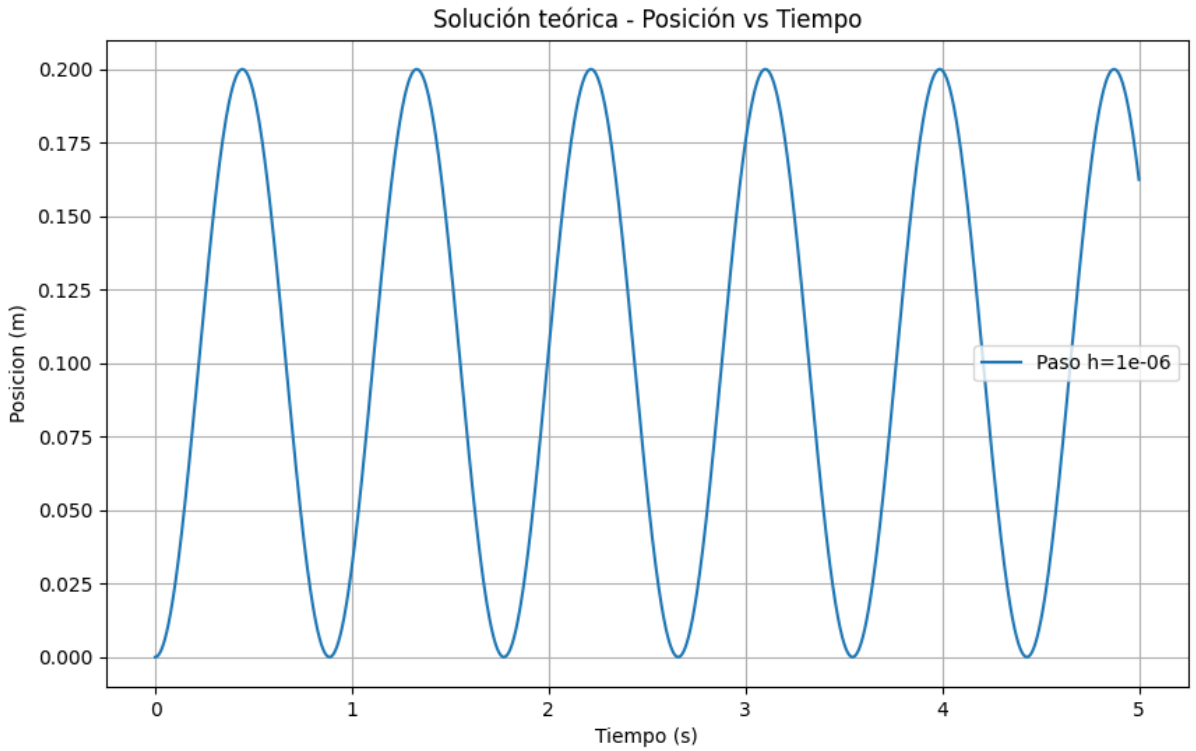




Para Runge Kutta a simple vista se puede ver que para la mayoría de los pasos probados, la aproximación obtenida es mejor y más similar entre todos.

Análisis del error

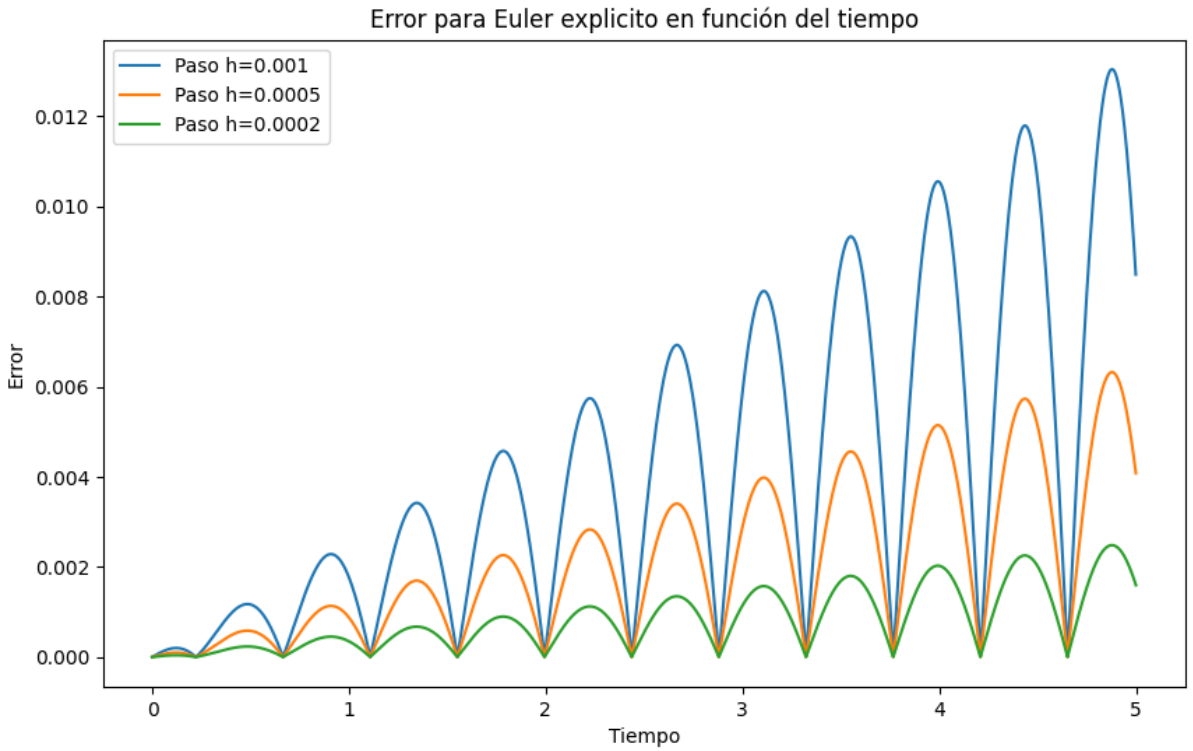
Para el análisis del error vamos a tomar como referencia la resolución de la misma ecuación para un tamaño de paso de $1x10^{-6}$. Es decir, tomaremos esta solución con un paso muy pequeño como la solución real para comparar y así calcular el error.



Evolución de los errores

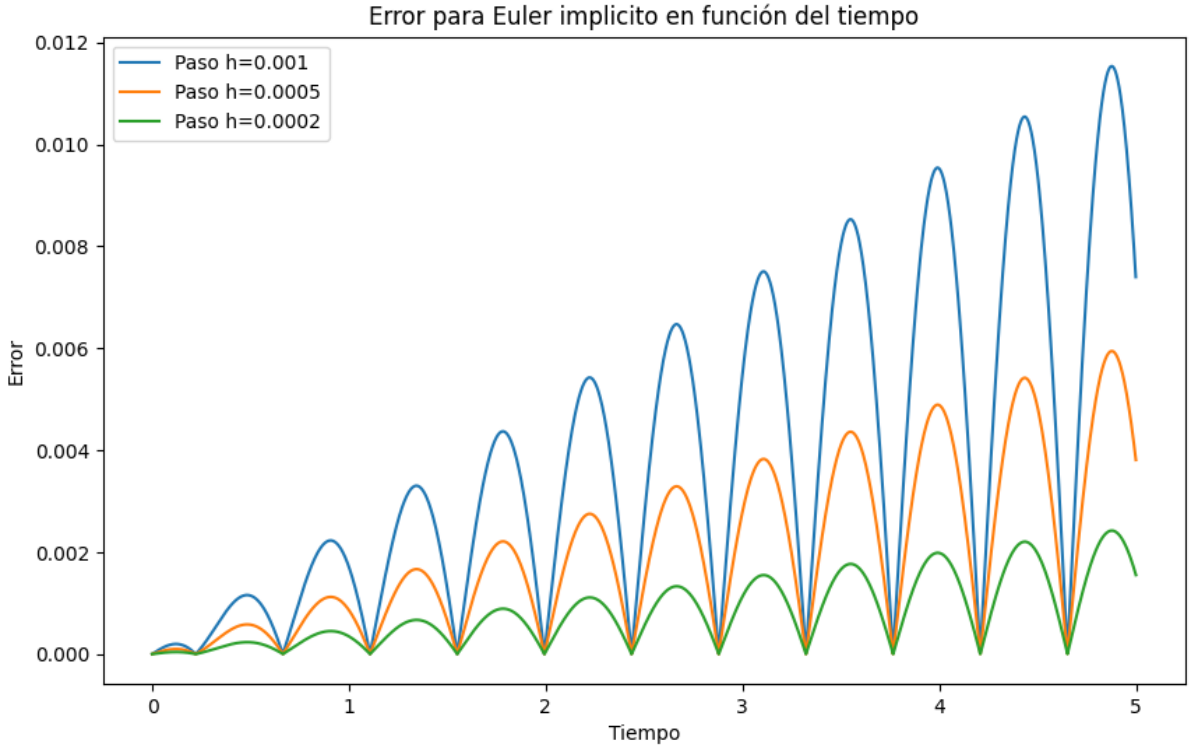
Euler explícito:

Vamos a comparar el error para los 3 menores tamaños de paso que se deberían encontrar más cercanos a la solución real: **0.01, 0.005, 0.0025.**



Euler implícito:

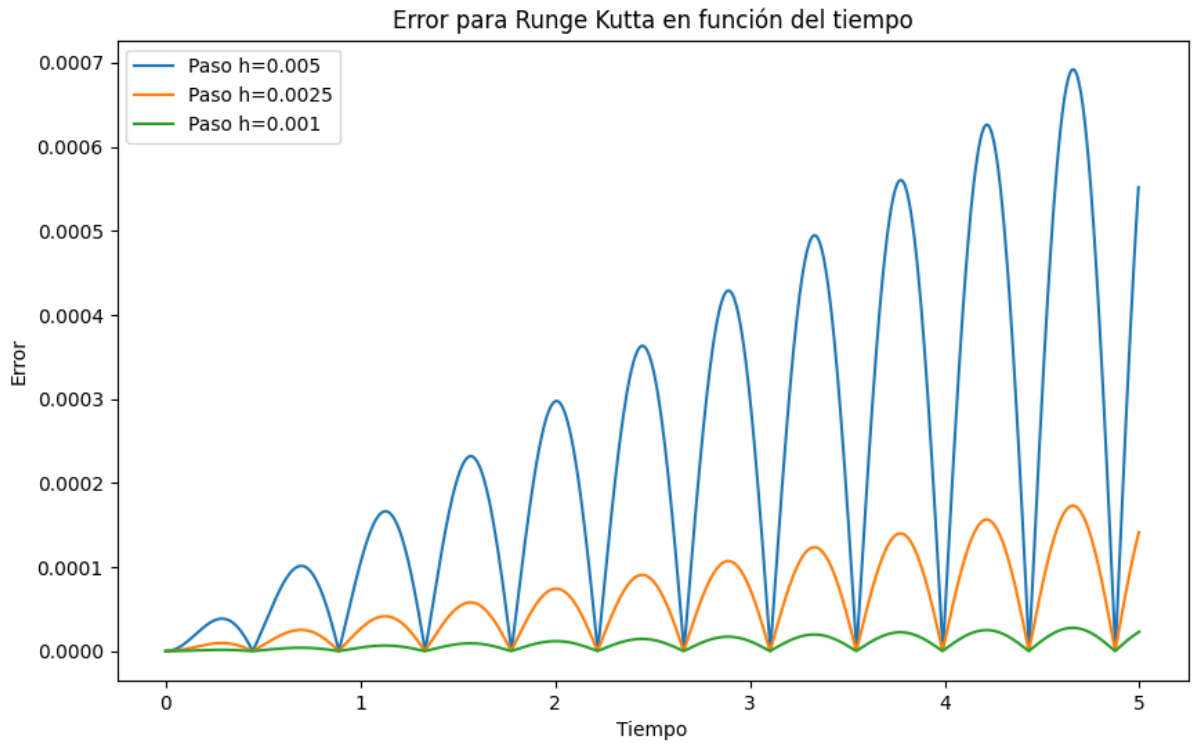
Para Euler implícito también vamos a comparar el error para los 3 menores tamaños de paso que se deberían encontrar más cercanos a la solución real: **0.001, 0.0005, 0.0002.**



Podemos observar una diferencia entre el error de ambos métodos a medida que aumentamos la distancia entre punto y punto (h).
Además, podemos ver que el error también tiene frecuencia similar a la de la oscilación de la posición ya que cuando se aproxima hacia la posición 0 no hay error, pero al aproximarse hacia los picos de la oscilación es cuando comienzan a aparecer los errores.

Runge Kutta:

Por lo visto en las sección anterior, para Runge Kutta a partir del paso 0.01 se puede observar que las soluciones siguientes no varían significativamente frente a su paso anterior, por lo que en este caso, vamos a comparar el error para los 3 tamaños de paso a siguientes al 0.01 inclusive: **0.005, 0.0025, 0.001.**



Orden de convergencia y frecuencia de oscilación.

Cómo etapa final del análisis del error vamos a comparar experimentalmente el orden de convergencia de cada método y su frecuencia de oscilación para compararla con su valor

teórico y así poder obtener variables de peso a la hora de decidir cuál es el método con el que vamos a quedarnos.

Orden de convergencia:

Para este análisis debemos tener en cuenta que el orden de convergencia de los métodos de euler es 1, mientras que para el método de Runge Kutta utilizado es de orden 2.
Teniendo en cuenta esto, un primer análisis se puede hacer en los gráficos de error, donde para los métodos de euler se puede observar como la disminución del paso de **0.001** hacia **0.0005 (**la mitad**)** disminuye el error en cada punto por el factor de ½ tal cual nos indica la teoría.

Lo mismo se puede evidenciar con el caso de Runge Kutta y los pasos **0.005** y **0.0025,** donde al dividir el paso por la mitad, el error en cada punto disminuye ¼.

Sin embargo, vamos a utilizar el siguiente cálculo para determinar experimentalmente cuál es el orden de convergencia p de cada método:

$$p = \frac{ln(\frac{E_1}{E_2})}{ln(\frac{h_1}{h_2})}$$

Donde E1 va a ser el error promedio del paso 1 y E2 va a ser el error promedio del paso 2, por otro lado h1 y h2 serán los pasos utilizados para la iteraciones que generaron el E1 y E2 respectivamente.

Aplicando esta fórmula para los pasos anteriormente mencionados en cada método tenemos lo siguiente:

Euler explícito:

Error promedio de paso **0.005:** 0.025419
Error promedio de paso **0.0025:** 0.011347
Orden de convergencia experimental: 1.163558
Euler implícito:

Error promedio de paso **0.005:** 0.016651
Error promedio de paso **0.0025:** 0.009185
Orden de convergencia experimental: 0.858298

Runge kutta:

Error promedio de paso **0.005:** 0.000232
Error promedio de paso **0.0025:** $5.806041x10^{-5}$
Orden de convergencia experimental: 1.999009

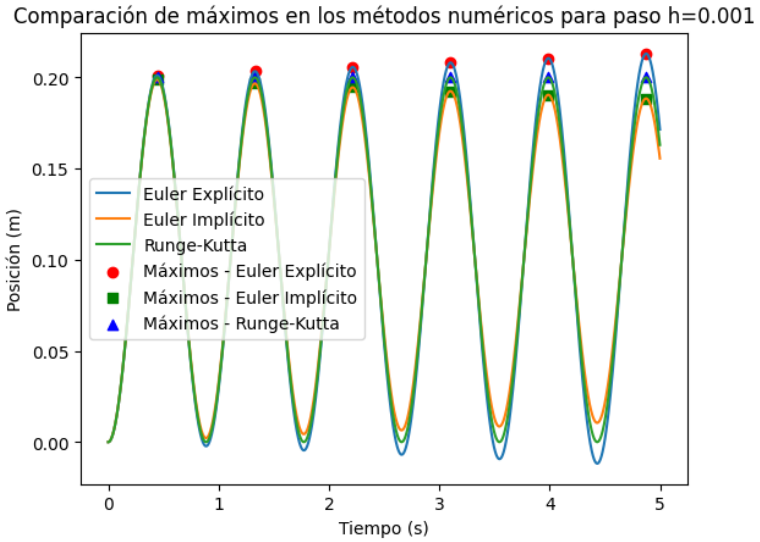
Observamos que el orden de convergencia experimental coincide con los órdenes de convergencia teórica .

Frecuencia de oscilación:

Finalmente, una última comparación a realizar va a ser la frecuencia de oscilación experimental de la posición contra su valor teórico, el cual es: $\omega_n = \sqrt{\frac{k}{m}}$. Para la comparación, vamos a obtener el periodo T, que en el contexto de oscilaciones armónicas es la duración de un ciclo completo de oscilación, es decir cuantas radianes se recorren en un segundo y utilizar este valor utilizar la fórmula de frecuencia angular: $\omega = \frac{T}{2\pi}$

Obtención del periodo T:

Vamos a encontrar los puntos máximos en oscilación y calcular la diferencia en tiempo entre los puntos que se encuentren consecutivos para obtener la duración de un periodo.



- Período promedio
    - Euler explicito: 0.886000
    - Euler implicito: 0.886000
    - Runge kutta: 0.886200

- Frecuencia angular experimental
    - Euler explicito: 7.091631
    - Euler implicito: 7.091631
    - Runge kutta: 7.090030

- Frecuencia natural de oscilación calculada analiticamente: 7.090808

- Relacion: frecuencia_analitica/frecuencia_experimental
    - Euler explicito: 0.999884
    - Euler implicito: 0.999884
    - Runge Kutta: 1.000110

Al hacer los cálculos se ve que los resultados de los euler son similares. Además se ve que el de Runge kutta comparte 4 decimales con la solución analítica (7.090). Pareciera ser el que se acerca más al resultado análitico

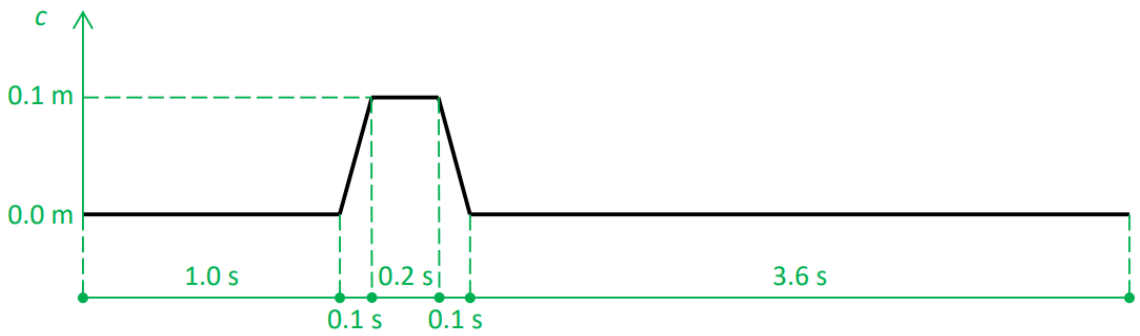
Conclusión parcial del mejor método

El mejor método ha resultado ser Runge Kutta de orden dos, dado que es un método de orden 2, es el método con el error más bajo, la frecuencia experimental calculada mediante RK fue la más cercana a la frecuencia analítica. Por otro lado, obtuvimos mejores resultados que en los métodos de Euler utilizando un paso más grande lo que nos ahorra tiempo de procesamiento, además, esta ventaja se obtiene sin tener que subir mucho la dificultad del algoritmo programado.

Escenarios con amortiguación

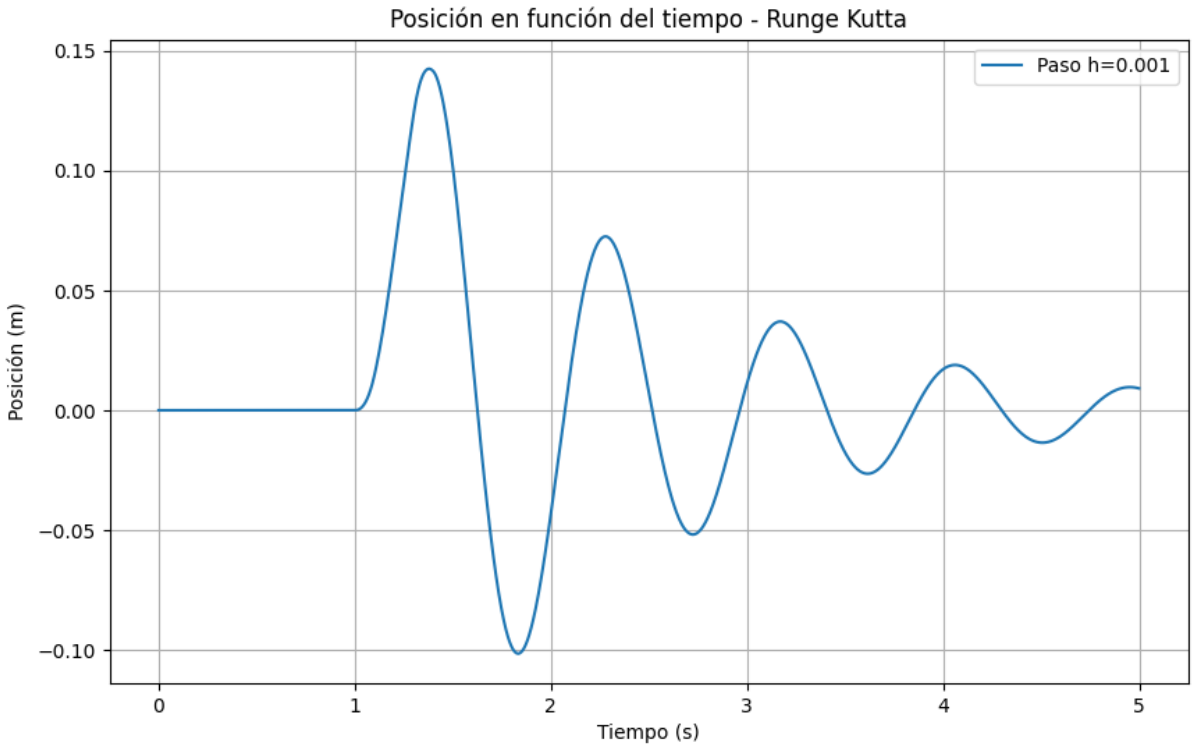
En este nuevo escenario trabajaremos con los siguientes parámetros:

- masa m = 99444/200
- k = 25000 N/m
- λ = 750 Ns/m
- Intervalo de tiempo a evaluar: 5 segundos.
- c de acuerdo a:



Además trabajaremos únicamente con el método de Runge-Kutta dado que hemos demostrado ser el mejor método para la situación en la que estamos trabajando.

Resolviendo el sistema obtenemos:



Podemos observar a primera vista que el amortiguador reduce el número de oscilaciones, evitando que el resorte se comprima y estire, sin parar.

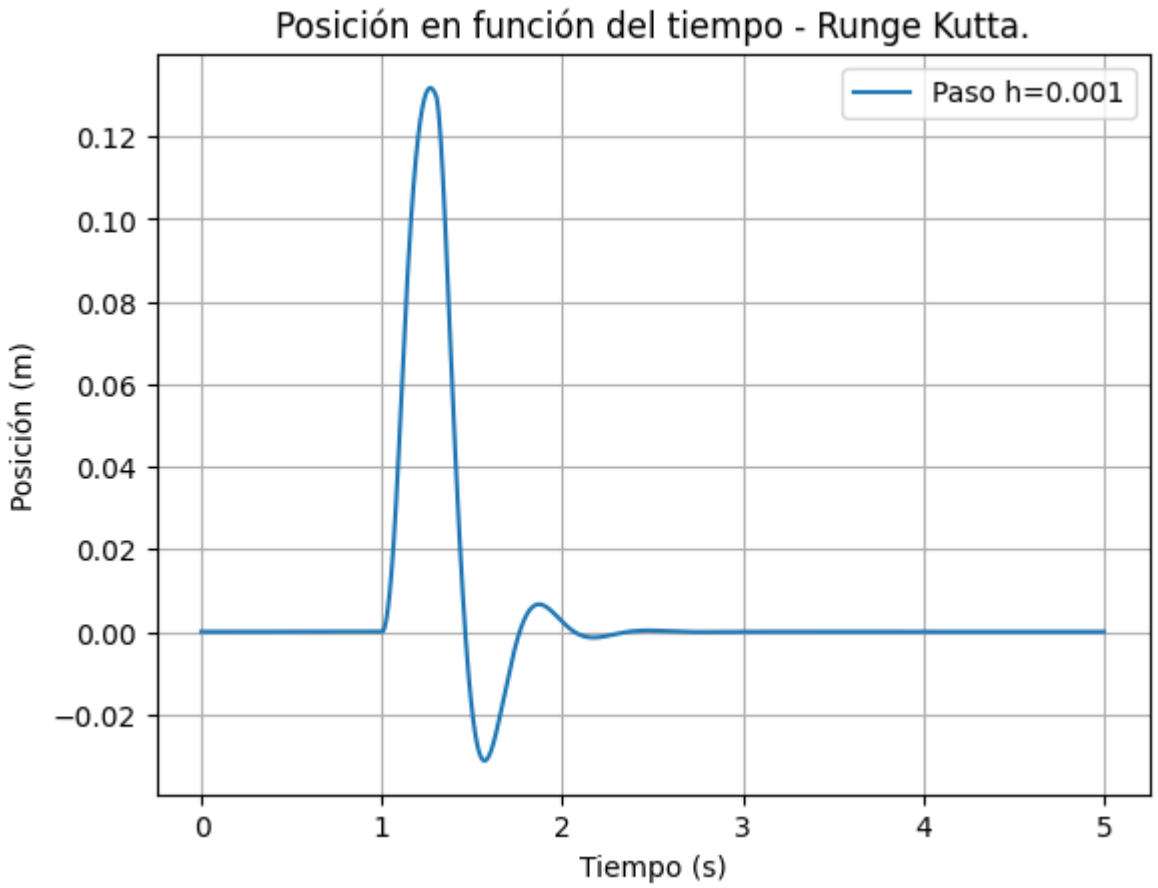
Ahora modifiquemos los valores de k y λ para optimizar la amortiguación del vehículo, en base a ciertas restricciones que impondremos y las compararemos con los valores originales.

Para la resolución de cada restricción que impondremos haremos un algoritmo que por fuerza bruta pruebe todas las distintas combinaciones entre k y λ que puede haber, con k tomando valores que van desde mil hasta 90 mil, con un paso de a mil y λ tomando valores que van desde 150 hasta 5100, con un paso de a 150.

<u>El sistema de amortiguación no puede comprimirse más de 0.05m</u>

Que el sistema de amortiguación no pueda comprimirse más de 0.05m es análogo a decir que $y - c \geq -0.05$. Buscamos la mayor compresión para cada combinación posible de k y λ, y luego buscamos en todos esos valores, cuál es el máximo y su k y λ asociado. Entonces, para esos valores k y λ Sabemos que lo máximo que puede llegar a comprimirse es menos de lo que puede comprimirse para cualquier otro valor. Resultado:

- k óptimo = 67000
- lambda óptimo = 5100



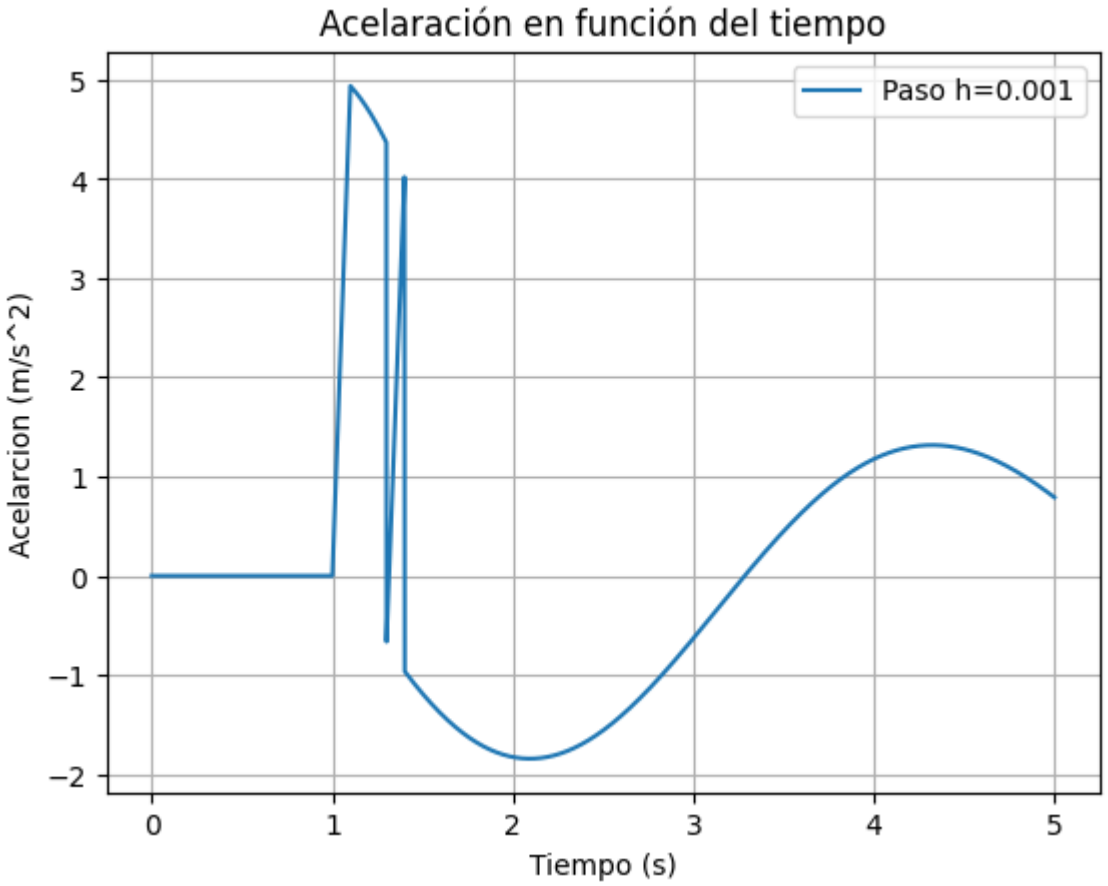
Podemos observar que la posición varía al igual que en el escenario principal, en función de la loma de burro, pero en este caso tiene una altura menor y muy poca cantidad de oscilaciones, es decir, se estabiliza brevemente pasada la loma de burro.

Minimizar la aceleración vertical para la loma de burro

Minimizar la aceleración vertical es minimizar el valor de $y``$ en el rango de tiempo [1.1 , 1.4]. Para esto nos quedaremos con el valor máximo de aceleración para cada combinación de k y λ en dicho intervalo, y luego nos quedaremos con la menor aceleración de todas esas aceleraciones máximas, así sabremos que tenemos la menor aceleración.

Resultado:

- k óptimo = 1000
- lambda óptimo =150

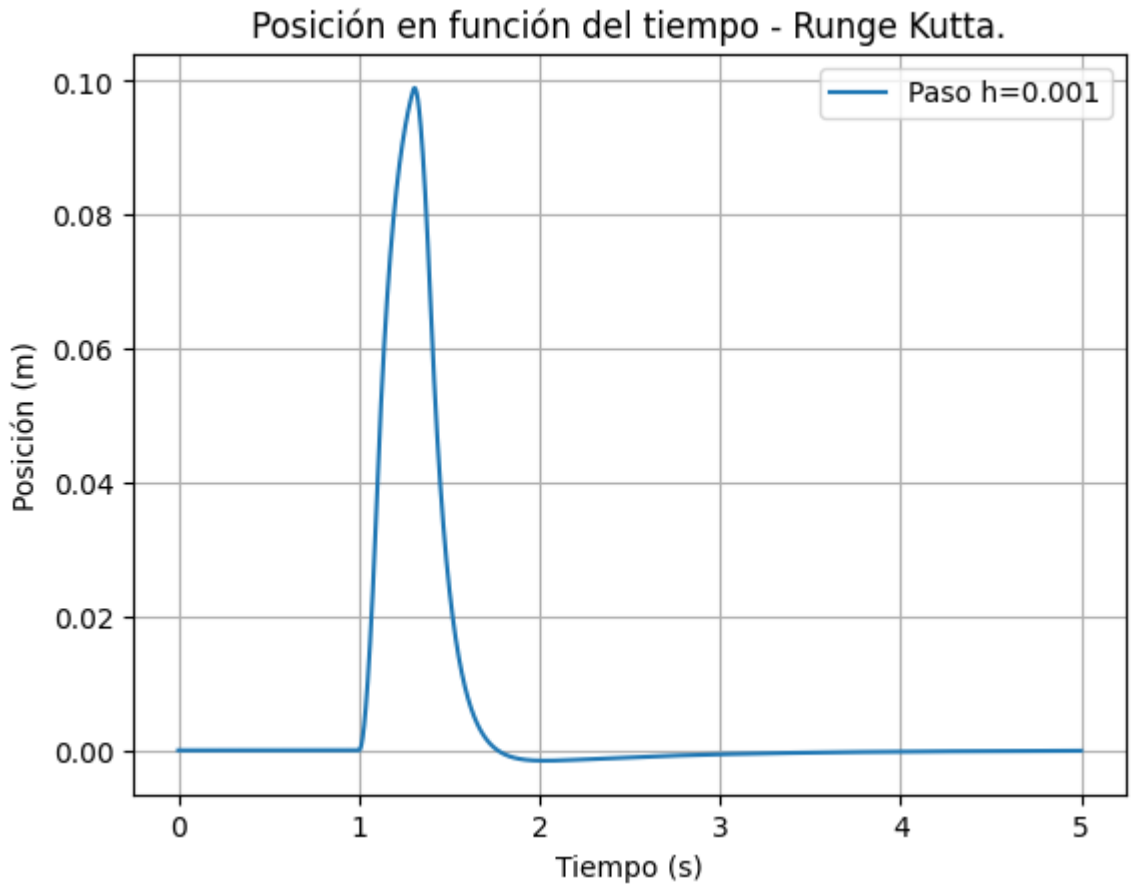


Minimizar las oscilaciones

Obtenemos el valor de U en cada instante de tiempo mayor a 1.4 (una vez pasada la loma de burro), luego hacemos el módulo de cada valor y los sumamos todos, nos quedamos con este resultado. Una vez hecho esto para cada par k y λ, nos fijamos en todos los resultados cuál fue el menor. El tener el menor significa que obtuviste la menor cantidad de oscilaciones.

Resultado:

- k óptimo = 5100
- lambda óptimo = 5100



Observamos que una vez pasada la loma de burro no ocurre ninguna oscilación en comparación con los valores iniciales.

Conclusión final

Los resultados de este informe subrayan la importancia crítica del tamaño del paso **h** en la precisión de las simulaciones de sistemas de suspensión, tanto en presencia como en ausencia de amortiguación. Se ha demostrado que un ajuste adecuado del paso es esencial para capturar con precisión la dinámica del sistema y para garantizar resultados confiables.

A través de los análisis realizados, hemos visto cómo los valores de k y λ han demostrado influir significativamente en el resultado, dependiendo de las restricciones específicas de los escenarios simulados. Estos hallazgos no solo proporcionan una base para estimar los parámetros óptimos que deben asumir k y λ para satisfacer una gama deseada de condiciones operativas, sino que también abren el camino para futuras investigaciones. Investigaciones posteriores podrían explorar la optimización de estos parámetros para mejorar el diseño y la respuesta de los sistemas de suspensión en aplicaciones de ingeniería real, contribuyendo así a la seguridad y comodidad vehicular.

Anexo

Links:

- Google Collab donde se encuentran todas las funciones: TP_numerico.ipynb (accesible con mail fiuba).
- Repositorio de trabajo: Link Github.

Código utilizado para resolver los problemas planteados:

Para el primer escenario, en el que no había amortiguación.

Euler explícito

```
# Aplico cambio de variable:
def sistema_cambio_de_variable(u, v):
    u_prima = v
    v_prima = (k / m) * (c - u) + (lambda_ / m) * (c_prima- v)
    return u_prima, v_prima

# Euler explicito
def euler_explicito(funcion, u0, v0, paso_h, intervalo_t):
    # Creamos vector tiempo desde 0 hasta intervalo_t, con un paso_h.
    # Aca guardamos los puntos en el tiempo donde vamos a evaluar la solución.
    tiempo = np.arange(0, intervalo_t, paso_h)

    valores_u = []
    valores_v = []

    # seteamos condiciones iniciales
    u = u0
    v = v0

    # Recorremos los puntos en el tiempo.
    for punto in tiempo:
        # Guardo el valor obtenido para el paso actual en los vectores.
        valores_u.append(u)
        valores_v.append(v)

        # Aplico la función recibida para calcular u' y v'
        u_prima, v_prima = funcion(u, v)
```

```
        # Calculo los valores n+1
        u += u_prima * paso_h
        v += v_prima * paso_h

    return tiempo, valores_u, valores_v
```

Euler implicito

```
def paso_siguiente(u,v,paso):
    """
    Calculo el paso siguiente de Un y Vn, es decir, Un+1 y Vn+1
    """
    v_sig = (v+(paso*k*(c-u)+lambda_*c_prima)*(1/m))/(1+(lambda_+(paso*paso)*k)*(1/m))
    u_sig = u + paso*v_sig

    return u_sig , v_sig


def euler_implicito(funcion, u0, v0, paso_h, intervalo_t):
    # Creamos vector tiempo desde 0 hasta intervalo_t, con un paso_h.
    # Aca guardamos los puntos en el tiempo donde vamos a evaluar la solución.
    tiempo = np.arange(0, intervalo_t, paso_h)

    valores_u = []
    valores_v = []

    # seteamos condiciones iniciales
    u = u0
    v = v0


    # Recorremos los puntos en el tiempo.
    for t in tiempo:
        # Guardo el valor obtenido para el paso actual en los vectores.
        valores_u.append(u)
        valores_v.append(v)

        u, v = funcion(u, v, paso_h)


    return tiempo, valores_u, valores_v
```

Runge Kutta

```
def calculo_q(u,v,paso):
    """
    Calculamos las q's
    """
    q1u = paso*v
    q1v = paso*((1/m)*(k*(c-u)+lambda_*(c_prima-v)))

    q2u = paso*(v+q1v)
    q2v = paso*((1/m)*(k*(c-(u+q1u))+lambda_*(c_prima-(v+q1v))))

    return q1u,q2u,q1v,q2v

def runge_kutta(funcion, u0, v0, paso_h, intervalo_t):
    # Creamos vector tiempo desde 0 hasta intervalo_t, con un paso_h.
    # Aca guardamos los puntos en el tiempo donde vamos a evaluar la solución.
    tiempo = np.arange(0, intervalo_t, paso_h)


    valores_u = []
    valores_v = []

    # seteamos condiciones iniciales
    u = u0
    v = v0

    # Recorremos los puntos en el tiempo.
    for t in tiempo:
        # Guardo el valor obtenido para el paso actual en los vectores.
        valores_u.append(u)
        valores_v.append(v)

        q1u,q2u,q1v,q2v = calculo_q(u,v,paso_h)

        u += 0.5*(q1u+q2u)
        v += 0.5*(q1v+q2v)

    return tiempo, valores_u, valores_v
```

Ahora en el segundo escenario utilizamos únicamente Runge-Kutta con un lambda constante y un c dinámico:

Runge Kutta

```
nuevo_lambda = 750
def calculo_q_c_dinamico(u,v,paso,punto_tiempo):
    """
    Calculamos las q's
    """
    c, c_prima = valor_de_c(punto_tiempo)
    q1u = paso*v
    q1v = paso*((1/m)*(k*(c-u)+nuevo_lambda*(c_prima-v)))

    q2u = paso*(v+q1v)
    q2v = paso*((1/m)*(k*(c-(u+q1u))+nuevo_lambda*(c_prima-(v+q1v))))

    return q1u,q2u,q1v,q2v

def runge_kutta_c_dinamico(funcion, u0, v0, paso_h, intervalo_t):
    # Creamos vector tiempo desde 0 hasta intervalo_t, con un paso_h.
    # Aca guardamos los puntos en el tiempo donde vamos a evaluar la solución.
    tiempo = np.arange(0, intervalo_t, paso_h)


    valores_u = []
    valores_v = []

    # seteamos condiciones iniciales
    u = u0
    v = v0

    # Recorremos los puntos en el tiempo.
    for t in tiempo:
        # Guardo el valor obtenido para el paso actual en los vectores.
        valores_u.append(u)
        valores_v.append(v)

        q1u,q2u,q1v,q2v = calculo_q_c_dinamico(u,v,paso_h, t)

        u += 0.5*(q1u+q2u)
        v += 0.5*(q1v+q2v)

    return tiempo, valores_u, valores_v
```

Valor de c en funcion del tiempo

```
def valor_de_c(tiempo):
    """
    Devuelve el valor de c y c' en funcion del tiempo, en forma de una tupla (c , c').
    """

    if (tiempo <= 1.1 and tiempo >1.0 ) :
        return (tiempo-1,1)

    elif (tiempo<1.4 and tiempo>=1.3):
        return (tiempo-1.3,-1)

    elif (tiempo<1.3 and tiempo>1.1):
        return(0.1,0)

    else:
        return (0,0)
```

No hemos agregado las funciones de Fuerza Bruta ni los errores o oscilaciones, pero se encuentran todas en el Google Collabs.