


Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

Máster Universitario en Sistemas Espaciales

Milestone IV Report

Ampliación de Matemáticas I

13 de noviembre de 2022

Autora:

- Rosa Martínez Rubiella

1. Introducción

En este Hito, se resuelve el problema del oscilador lineal armónico mediante los esquemas numéricos tratados anteriormente y se añade un nuevo esquema temporal, el **Leap-Frog**. Se analiza el error obtenido respecto de la solución analítica, así como el comportamiento del nuevo esquema de integración a distintos problemas de entrada.

Posteriormente, se analizan las regiones de estabilidad absoluta de dichos esquemas, y se relacionan con el comportamiento de la solución del oscilador armónico al ser integrada por los mismos.

Finalmente, se añaden las actualizaciones del código de Python, así como una explicación del proceso de implementación seguido para este Hito.

2. Oscilador Armónico

Se integra el problema del oscilador armónico simple con condiciones iniciales. Para ello se utilizan los diferentes esquemas numéricos implementados hasta ahora: Euler explícito, Euler Inverso, Crank-Nicolson y Runge-Kutta-4.

$$\ddot{x} + x = 0 \quad con \quad x(0) = 1\,, \quad \dot{x}(0) = 0$$

Además, se obtiene la solución analítica del problema para poder realizar una comparación con los resultados obtenidos.

$$x(t) = \cos t \quad y \quad \dot{x}(t) = -\sin t$$

Se ha utilizado un paso de integración de 0.1 para la simulación, ya que de esta forma es posible visualizar los errores cometidos por la aproximación en casi todos los casos. En la **Figura 1** se representa la solución del oscilador armónico para el Euler explícito, tanto en posición como en velocidad. Se observa como existe un error tanto en amplitud como en fase, que se amplifica conforme aumenta el tiempo de simulación. Sin embargo al reducir el paso de integración, es decir $\Delta t \ll 1$, el error de fase tiende a ser nulo a la vez que el de amplitud se reduce de manera más progresiva. El oscilador va ganando energía conforme aumenta el tiempo de simulación, una divergencia característica de este esquema temporal, tal y como se vio para el problema de Kepler. Más adelante se explica este fenómeno mediante las regiones de estabilidad absoluta.

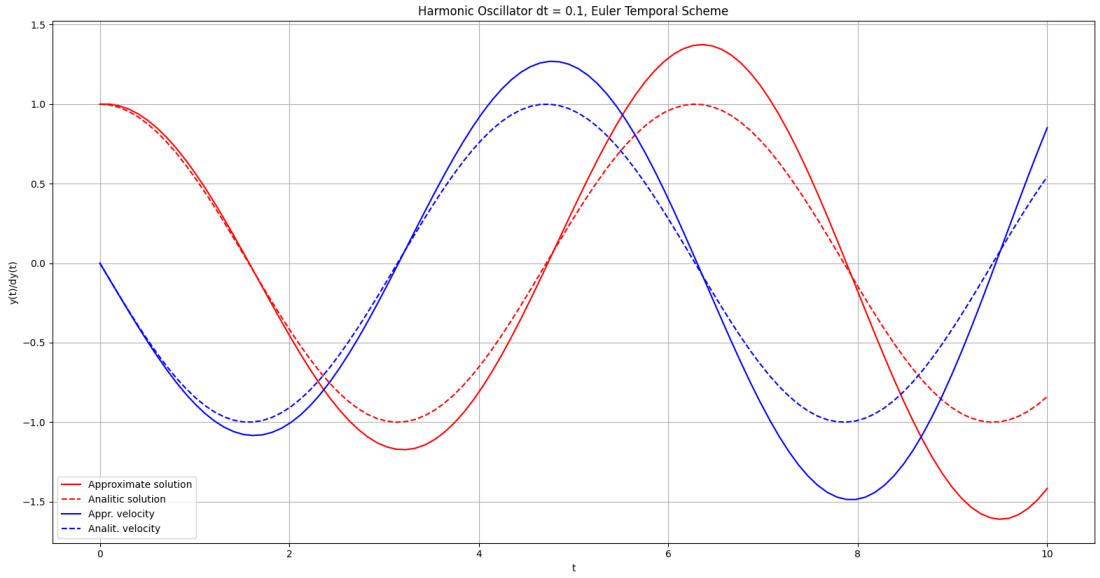


Figura 1: Euler TS Harmonic Oscillator integration, $\Delta t = 0{,}1$

Sin embargo, en el caso del Euler Inverso (**Figura 2**), se pierde energía conforme pasa el tiempo. Tiene carácter disipativo, al igual que con la órbita de Kepler. Tiene también error en fase y amplitud, con un comportamiento similar al esquema anterior en módulo. La solución aproximada tenderá a 0 en el infinito, lo que simularía el comportamiento real del oscilador si existieran fuerzas disipativas de fricción o gravedad (muelle o péndulo).

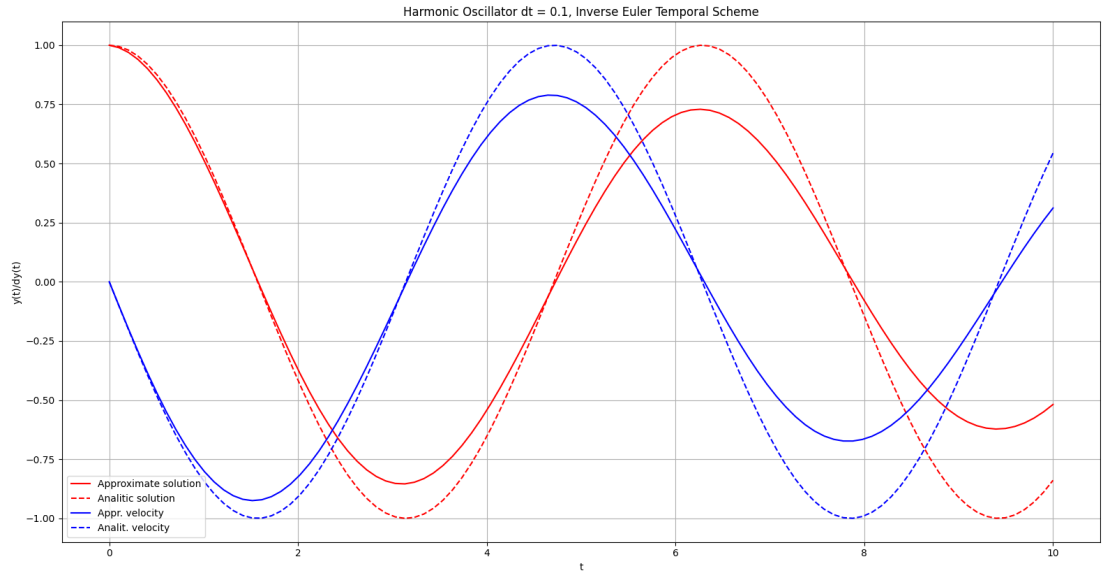


Figura 2: Inverse Euler TS Harmonic Oscillator integration, $\Delta t = 0{,}1$

En el caso del Crank-Nicolson (**Figura 3**), la solución tiene un leve error en fase, pero no en amplitud. Este error de fase tenderá a 0 con $\Delta t \ll 1$. Se puede relacionar con el caso de la órbita de Kepler, donde el satélite va más rápido/lento que en la realidad. Para el Δt elegido no se aprecia mucho este fenómeno, pero aumenta con el tiempo de simulación.

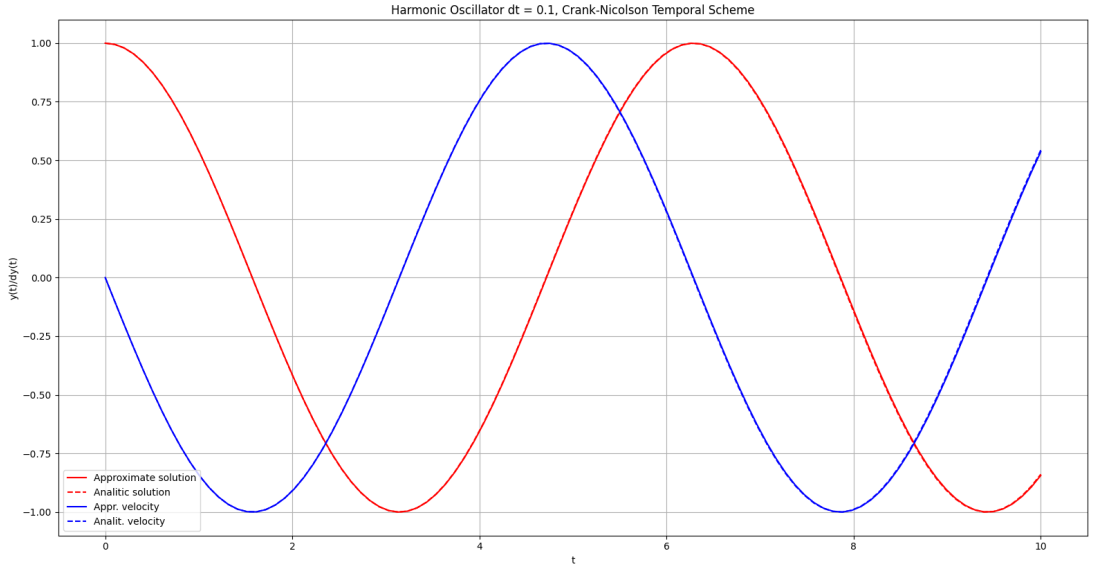


Figura 3: Crank-Nicolson TS Harmonic Oscillator integration, $\Delta t = 0{,}1$

Finalmente, el esquema de Runge-Kutta de 4º orden proporciona una buena aproximación de la solución analítica, como era esperado. Esto se aprecia en la **Figura 4**

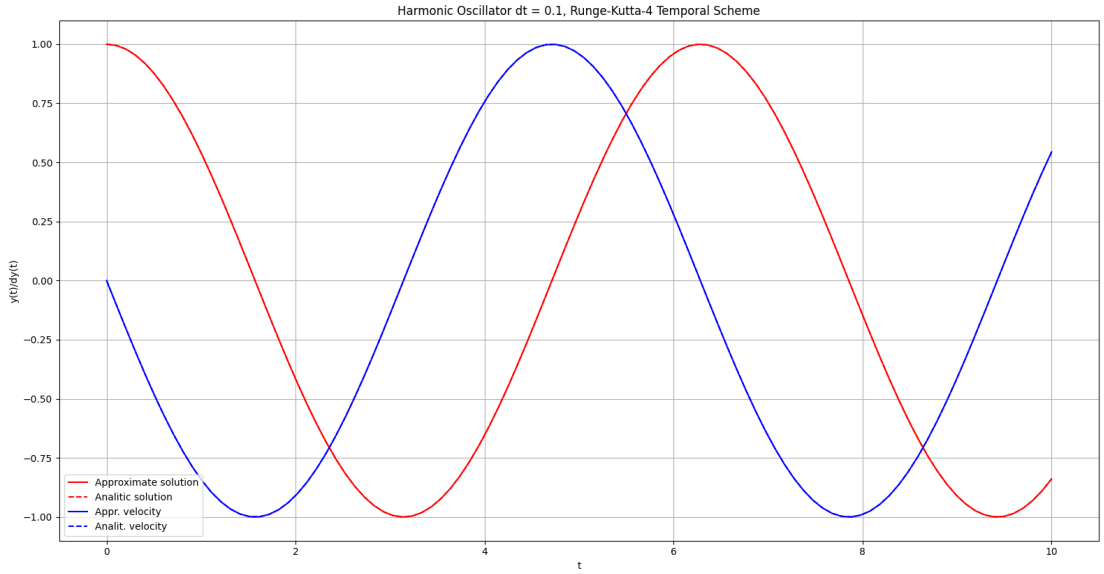


Figura 4: Runge-Kutta-4 TS Harmonic Oscillator integration, $\Delta t = 0{,}1$

Además, se implementa un nuevo esquema numérico de integración (ver sección 4), que presenta una solución aproximada muy similar a la proporcionada por el Crank-Nicolson. Tiene un pequeño error en fase que disminuye con el paso de integración (**Figura 5**). Ambos esquemas numéricos son de orden 2, como se puede observar en la regresión lineal del ratio de convergencia (**Figura 6**). Además, se ha representado el error numérico tanto del problema de Kepler como del oscilador armónico simple utilizando el Leap-Frog como integrador para un $\Delta t = 0{,}01s$. Se observa la tendencia progresiva con el tiempo de simulación, y un orden de error de 10^{-4} aproximadamente, que se corresponde con $O(\Delta t^2)$.

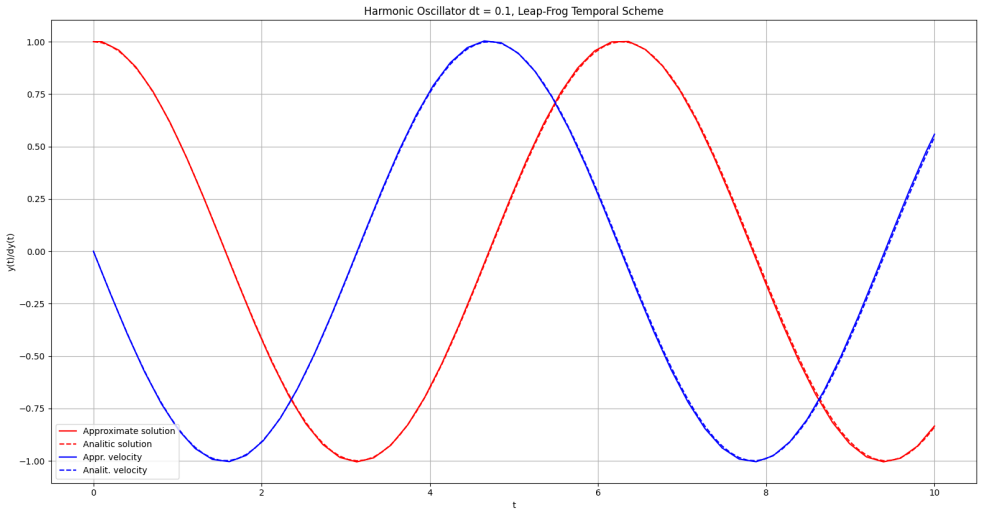
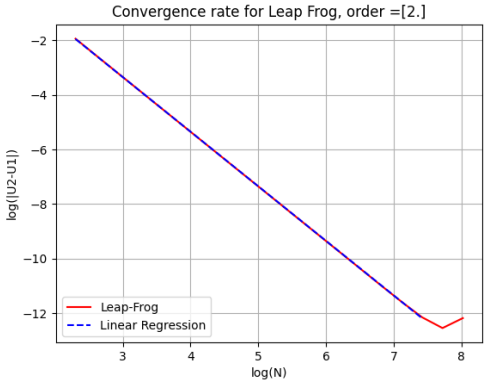
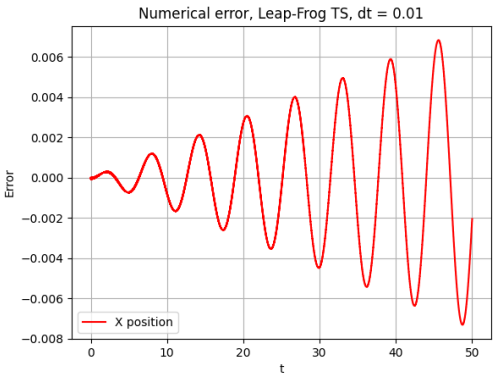


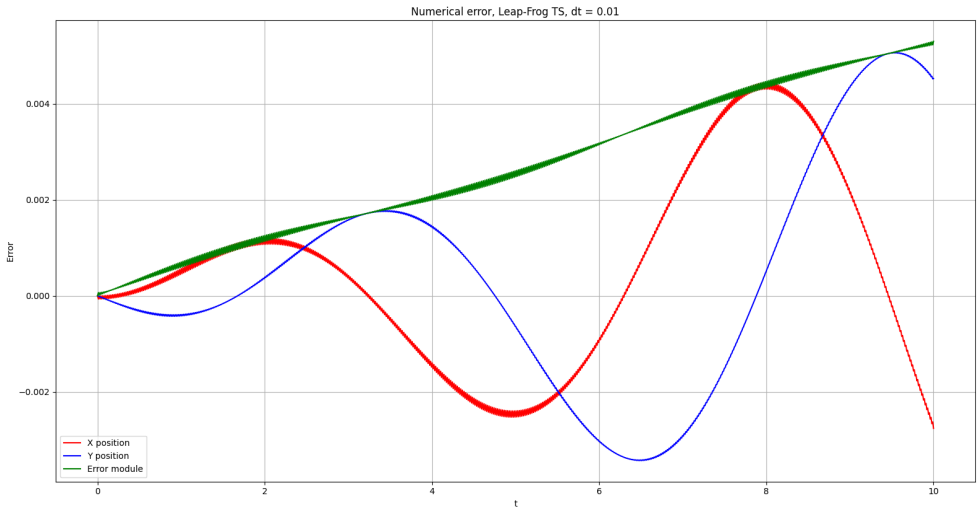
Figura 5: Leap-Frog TS Harmonic Oscillator integration, $\Delta t = 0{,}1$



(a) Leap-Frog convergence rate



(b) Leap-Frog Numerical Error for the Oscillator problem



(c) Leap-Frog Numerical Error for the Kepler problem

Figura 6: Leap Frog temporal scheme analysis

3. Regiones de estabilidad absoluta

En esta sección se representan las regiones de estabilidad absoluta de los esquemas temporales utilizados (ver sección 4). De esta forma es posible realizar un análisis del comportamiento de la solución aproximada generada por cada esquema de integración, y la tendencia de su error con el paso de tiempo elegido (**Figura 7**).

En el caso del Euler explícito, la región de estabilidad se corresponde con el interior de una circunferencia centrada en (-1, 0). Es decir, en los puntos de su frontera el módulo del espectro de B es 1, con $U^{n+1} = BU^n$. Si los autovalores del problema asociado, en este caso:

$$A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

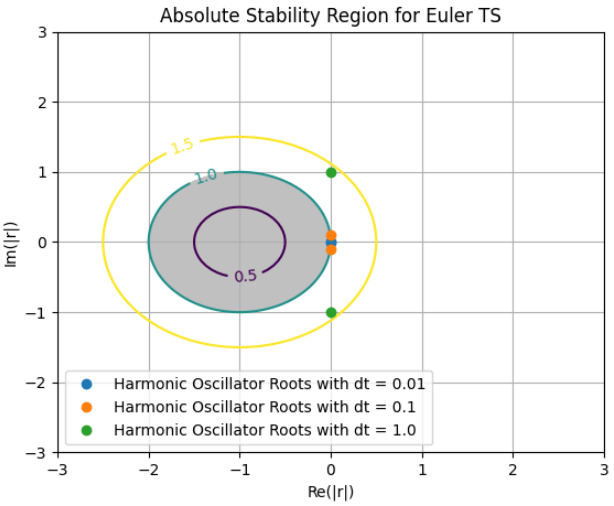
con $\lambda_1 = i$ y $\lambda_2 = -i$, multiplicados por el Δt pertenecen a la región de estabilidad, el error de la solución numérica estaría acotado. Se puede observar como para dicho esquema temporal, las raíces se encuentran fuera de la región de estabilidad y convergen a su contorno cuando el paso de tiempo tiende a 0. Esto quiere decir que el error de la solución no va a estar acotado, diverge para todo paso de tiempo seleccionado; y esto se corresponde con el comportamiento de la solución observado previamente.

Para el esquema de Euler Inverso, se da una situación contraria; la región de estabilidad se corresponde con el exterior de una circunferencia centrada en (1,0). Conforme se reduce el paso de integración, las raíces tienden al contorno de la región de estabilidad. Las raíces se sitúan en la zona con radio espectral menor de 1, donde el error está acotado, característico de los esquemas disipativos, como se ha visto anteriormente.

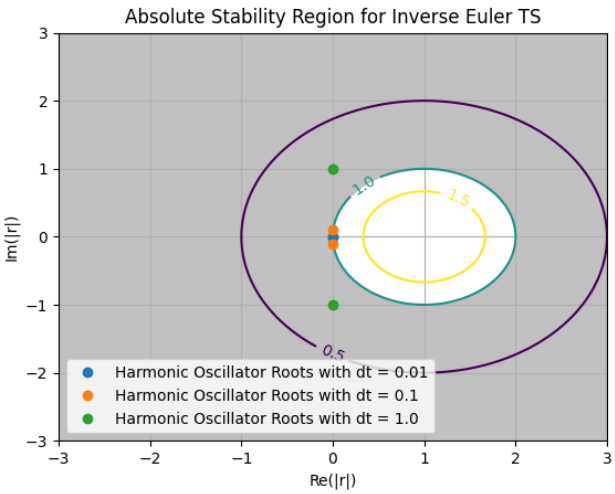
El esquema Runge-Kutta de cuarto orden presenta unas características un poco diferentes. Si Δt está entre $|(0,1)|$ s, las raíces se sitúan sobre el contorno de la región de estabilidad. Conforme se aumenta el paso de tiempo $|(1, 2.78)|$, las raíces pasan brevemente por el interior de la región de estabilidad, para salir de ella cuando tiende a infinito $|(2.78, \infty)|$.

En el caso del Crank-Nicolson, se consideran estables todas las raíces con parte real negativa, por lo que las soluciones al problema del oscilador armónico también van a quedar situadas sobre el contorno de la región de estabilidad. En este caso, sin embargo, no depende del paso de integración que se elija, siempre se va a cumplir que el radio espectral es la unidad.

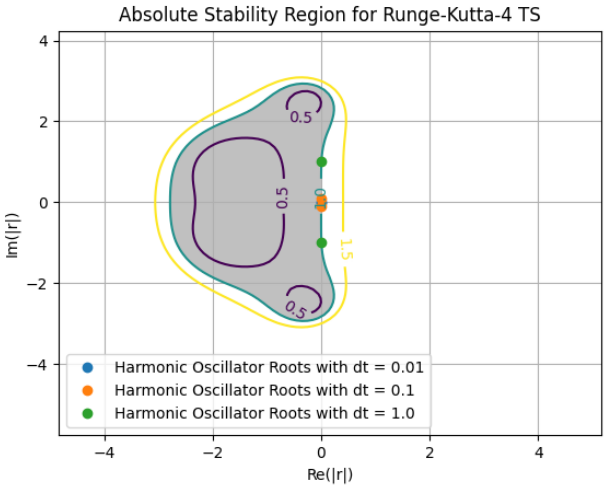
Finalmente, para el Leap-Frog, la región de estabilidad absoluta se corresponde con una única línea uniendo los puntos (1,0) y (-1,0). Por tanto, las soluciones están a su vez sobre el contorno de dicha región. Para $\Delta t > 1$, el error no estará acotado y tenderá a infinito con el número de puntos de integración. Se puede observar cómo las soluciones del Leap-Frog, Crank-Nicolson y Runge-Kutta-4 tienen un comportamiento similar, ya que para valores de Δt entre (0,1) las raíces se sitúan sobre el contorno de la región.



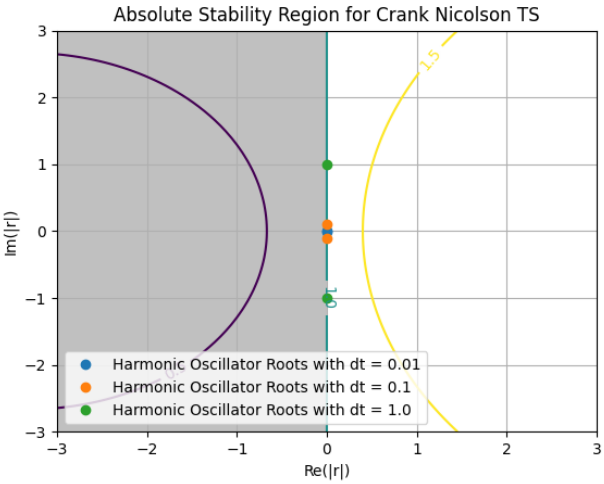
(a) Euler



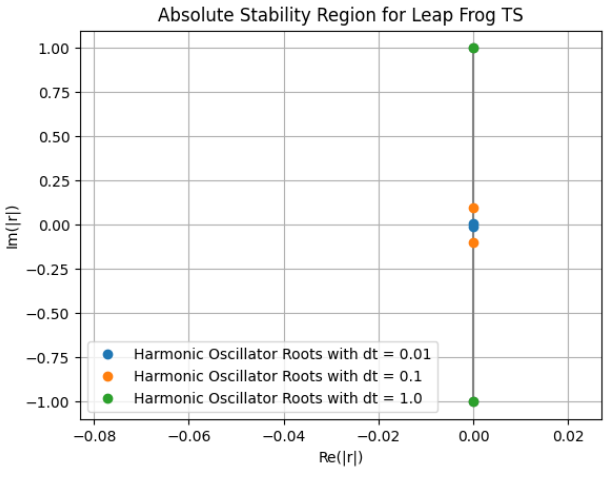
(b) Inverse Euler



(c) Runge-Kutta-4



(d) Crank-Nicolson



(e) Leap-Frog

Figura 7: Absolute Stability Regions, harmonic oscillator analysis

4. Code Review

Para el problema del oscilador armónico, se crea un módulo que contiene su función de integración [F(U,t)] o derivada del vector de estado. Se añade una constante que generaliza la función para diferentes casos del oscilador armónico.

```
def F_oscillator(U,t):
    a = 1
    return array([U[1], -a*U[0]])
```

Listing 1: Harmonic Oscillator Integration Function

Para la implementación del esquema Leap-Frog se ha realizado una modificación del código utilizado para el Problema de Cauchy, ya que se necesita una condición inicial adicional que en el resto de esquemas temporales. Es por ello que el bucle que integra el problema necesita comenzar en U^2, ya que U^1 se obtiene mediante un esquema de Euler explícito. Se realiza esta modificación para poder utilizar la misma sintaxis que con el resto de esquemas a a la hora de llamar al programa, pasando por el problema de Cauchy inicial.

A continuación, se implementa el código del esquema temporal dentro del módulo de *Temporal Schemes*, donde se añaden como argumentos U^{n-1} y U^n, que caracteriza el funcionamiento de este integrador, haciendo uso de U^{n-1}, U^n y U^{n+1} en cada paso. Además, se le ingresan los típicos argumentos de entrada como son el paso de tiempo, la derivada del vector de estado y el tiempo donde se evalúa la función.

```
def Cauchy_Problem(F, t, U_0, Temporal_Scheme):

    N = len(t) - 1
    N_0 = len(U_0)
    U = array(zeros([N+1, N_0]))

    delta_t = t[2] - t[1]

    U[0,:] = U_0

    if Temporal_Scheme == ts.Leap_Frog: # Modificacion para el Leap-Frog

        U[1,:] = U[0,:] + delta_t*F(U[0,:], t[0]) # Condicion inicial
    dada por un esquema de Euler explicito

        for i in range(1,N): # El bucle comienza en 1
            U1 = U[i-1, :]
            U2 = U[i, :]
            U[i+1, :] = Temporal_Scheme(U2, U1, delta_t, F, t[i]) # Se
    anade el argumento del paso antepenultimo y ultimo de integracion

    else: # Para el resto de esquemas numericos
        for i in range(N):
            U[i+1, :] = Temporal_Scheme(U[i, :], delta_t, F, t[i])

    return U
```

Listing 2: Cauchy Problem modification

```
def Leap_Frog(U2, U1, delta_t, F, t):

```

```
    return U1 + 2*delta_t*F(U2,t)
```

Listing 3: Harmonic Oscillator Integration Function

Para representar las regiones de estabilidad absoluta, se ha generado un módulo que contiene dos funciones: una que devuelve el polinomio característico de estabilidad de cada esquema y otra que evalúa dicho polinomio en el plano complejo que el usuario requiera.

```

import Resources.Temporal_Schemes  as ts
from numpy import meshgrid, array, zeros, size, absolute, sqrt

def Stability_Region(x, y, Temporal_Scheme): # Los argumentos de entrada
    son el esquema temporal, y el plano complejo generado por sus
    coordenadas x (parte real) e y (parte compleja) en las que se quiere
    evaluar dicho polinomio

    N = size(x)
    Z = zeros([N,N],dtype=complex)

    for i in range(N):
        for j in range(N):
            Z[N-1-j,i] = complex(x[i],y[j]) # Z es la matriz de
    coordenadas que define el plano complejo

    return absolute(array(Stability_Polinomial(Temporal_Scheme, Z))) #
    Se evalua el polinomio para dicha matriz de coordenadas y se calcula
    el valor absoluto de cada termino. Resultando de esta forma un array
    que puede ser representado mediante contornos de estabilidad en el
    plano complejo.



def Stability_Polinomial(Temporal_Scheme, w): # Los polinomios de
    estabilidad han sido generados de forma analitica y son implementados
    directamente en el programa. Se escriben igualados a r, ya que al
    realizar el valor absoluto de los mismos se puede obtener el contorno
    de estabilidad |r| = 1.

    if Temporal_Scheme == ts.Euler:
        r = 1 + w
    elif Temporal_Scheme == ts.Inverse_Euler:
        r = 1/(1-w)
    elif Temporal_Scheme == ts.Crank_Nicolson:
        r = (1+w/2)/(1-w/2)
    elif Temporal_Scheme == ts.Runge_Kutta_4:
        r = 1 + w + (w**2)/2 + (w**3)/6 + (w**4)/(4*3*2)
    elif Temporal_Scheme == ts.Leap_Frog:
        r = sqrt(1)

    return r
```

Listing 4: Stability Regions module

Finalmente, se genera un script "main"desde el cual se llama a las funciones implementadas y se generan las gráficas obtenidas para el informe. Además, para el error analítico y el ratio de convergencia se han utilizado los códigos del Hito III, donde solo ha sido necesario introducir la nueva función del oscilador armónico y el esquema de integración Leap-Frog.

```
1
2 import Resources.Temporal_Schemes  as ts
3 import matplotlib.pyplot as plt
4 import Resources.Stability_Regions as sr
5
6 from Resources.Cauchy_Problem import Cauchy_Problem
7 from Resources.Harmonic_Oscillator import F_oscillator
8
9 from numpy import array, linspace, cos, sin
10
11
12 N = 100 # Puntos de integraci n
13
14 U_0 = array([1,0])  # Condiciones iniciales
15
16 t = linspace(0, 10, N) # Tiempo final de la simulacion
17
18 U = Cauchy_Problem( F_oscillator, t, U_0, ts.Leap_Frog) #Problema
      deCauchy, donde se incluye la funcion del oscilador arm nico , el
      tiempo de simulacion, las condiciones iniciales y el esquema temporal
19
20 plt.plot(t, U[:,0],"r", label = "Approximate solution") # Plotea la
      solucion aproximada de la posicion
21 plt.plot(t, cos(t),"--",color = "r", label = "Analitic solution") #
      Solucion analitica de la posicion
22 plt.plot(t, U[:,1], "b", label = "Appr. velocity") # Solucion aproximada
      de la velocidad
23 plt.plot(t, -sin(t),"--", color = "b", label = "Analit. velocity") #
      Solucion analitica de la velocidad
24 plt.title("Harmonic Oscillator dt = 0.1, Leap-Frog Temporal Scheme")
25 plt.xlabel("t")
26 plt.ylabel("y(t)/dy(t)")
27 plt.legend(loc = "lower left")
28 plt.grid()
29 plt.show()
30
31
32 x = linspace(-5, 5, num=100) # Parte real del plano complejo a
      representar, discretizada
33 y = linspace(-5, 5, num=100) # Parte imaginaria del plano complejo a
      representar, discretizada
34
35 dt = array([0.01, 0.1, 1]) # Valores de dt donde quiero evaluar las
      raices del problema del oscilador armonico, ya que landa_1 = idt y
      landa_2 = -idt . Estas raices son los autovalores de la matriz
      asociada a la ecuacion diferencial multiplicados por el delta de
      tiempo.
36
37 stab_region = sr.Stability_Region(x,y,ts.Runge_Kutta_4) # Matriz que
      contiene el valor absoluto del polinomio caracteristico de
      estabilidad evaluado en el plano complejo
38
39 CSF = plt.contourf(x, y1, stab_region, levels = [0, 1],
      colors=['#C0C0C0'] ) #levels = [0.5, 1, 1.5] # Plotea la region de
      estabilidad pintada de gris, donde el valor absoluto de stab_region
      va de (0,1)
40 CS = plt.contour(x, y1, stab_region, levels = [0.5, 1, 1.5] )  # Plotea
      el contorno de la region de estabilidad (modulo = 1), asi como una
      zona exterior e interior para observar su variacion
```

```
for t in range(3): # Plotea las raices del problema del oscilador armonico para analizar su tendencia con cada esquema de integracion y con el paso de integracion seleccionado
    plt.plot([0,0],[1*dt[t],-1*dt[t]], 'o', label = "Harmonic Oscillator Roots with dt = " +str(dt[t]))
plt.clabel(CS, inline=1, fontsize=10) # Escribe el valor correpondiente al contorno ploteado p.e. |r| = 1
plt.title('Absolute Stability Region for Leap Frog TS')
plt.xlabel("Re(|r|)")
plt.ylabel("Im(|r|)")
plt.legend(loc = "lower left")
plt.grid()
plt.show()
```

Listing 5: Main Milestone-4 Module

Update del código de estabilidad Para sacar los polinomios de estabilidad de manera automática, se utiliza una regla mnemotécnica. De esta forma, todos los esquemas numéricos, exceptuando los multipaso como el Leap-Frog donde nos encontramos con ecuaciones implícitas de variable compleja, pueden resolverse muy fácilmente. A continuación se adjunta el código actualizado de la región de estabilidad. Las gráficas obtenidas, así como los resultados numéricos son semejantes. Se ha intentado resolver dichas ecuaciones implícitas mediante un método de Newton en variable compleja, pero sin éxito.

```
def Stability_Region(x, y, Temporal_Scheme): # x = linspace(x0,y0,N) e igual para la y
    N = size(x)
    rho = zeros([N,N], dtype=float64)

    for i in range(N):
        for j in range(N):
            Z = complex(x[i],y[j])
            r = Temporal_Scheme(1., 1., lambda u, t: Z*u, 0.) # Los argumentos de entrada son U, delta_t, F y t
            rho[i,j] = abs(r)

    return rho
```

Listing 6: Updated Stability Region code

5. Conclusiones y sugerencias

Estaría bien poder resolver una ecuación implícita en variable compleja de forma directa, mediante una función tipo fsolve o un método de newton de cualquier biblioteca estándar. Además, plotear contornos en python mediante la biblioteca matplotlib es un tanto largo y engorroso, aunque los resultados gráficos son buenos.