


Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

Máster Universitario en Sistemas Espaciales

Milestones 4

Ampliación de Matemáticas 1

6 de diciembre de 2022

Autor:

- Javier Zatón Miguel

Índice

1. Objetivo del software

El objetivo de este hito es el de integrar el problema de oscilador lineal, para unas condiciones iniciales, usando todos los esquemas numéricos usados en los hitos anteriores, añadiendo un esquema nuevo (Leap-Frog).

También se busca obtener las regiones de estabilidad de los esquemas numéricos de los hitos anteriores, para esto se va a crear un módulo nuevo denominado Regions.

2. Desarrollo de código

2.1. Modificaciones en módulos

Para poder integrar el problema del oscilador lineal se ha creado una nueva función dentro del módulo Functions denominada Oscilador_lin.

```
def Oscilator_lin(U, t):

    x = U[0]
    vx = U[1]

    return  array( [ vx,-x ] )
```

Con esta función desarrollada es muy sencillo, resolver el problema con el módulo de Cauchy y los esquemas numéricos creados de antemano, sin embargo, para implementar el método de Leap-Frog, es necesario realizar ciertas modificaciones en el módulo de Cauchy, ya que este método , para calcular el valor en un tiempo N+1 necesita el de N-1 , a excepción del primer cálculo, que se realizará con Euler. La función de Leap-Frog se añade en el módulo scheme_module:

```
def Leap_frog(U1,U2,F,dt,t):

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

Como ya se ha explicado, a diferencia del resto de esquemas, este requiere de un input de la función en tiempos distintos, se les ha denominado U1 y U2, el primero corresponde al valor calculado en el paso de integración anterior (N-1).

Para que Leap-Frog sea compatible con el módulo de Cauchy, se ha añadido la excepción dentro de la función, especificando que si se usa en concreto este esquema, se debe integrar el primer paso con Euler, y para el resto de casos, se especifica el valor de U1 y U2, para obtener para todos los pasos restantes la matriz de resultados U.

```
if scheme == Leap_frog:
    U[:,1] = U[:,0] + dt*F(U[:,0],lin_t[0])
    for i in range(1,N):

        U1 = U[:,i-1]
        U2 = U[:,i]
        U[:,i+1] = scheme(U1,U2,F,lin_t[i+1]-lin_t[i],lin_t[i])
else:
```

Con estas modificaciones queda resuelta la primera parte del hito, incorporar Leap-Frog y el problema del oscilador lineal.

Para representar las regiones de estabilidad de los esquemas numéricos se ha creado un nuevo módulo llamado Regions", este tiene tres inputs distintos, que consisten de dos vectores X e Y, que van a formar una malla compleja, para que al un esquema determinado represente su región de estabilidad.

La primera función del módulo, y la principal, es la que se ha denominado como Stab_regions.

```
def Stab_Regions(x,y,Scheme):
    #Z = meshgrid(x,y)
    p = size(x)
    py = size(y)
    if p == py:
        Z = zeros([p,p],dtype=complex)
        for i in range(p):
            for j in range(p):
                Z[j,i] = complex(x[i],y[j])
    else:
        print('WARNING: X partitions are not equal to Y, meshgrid incompatible with stab_Regions function')


    return absolute(Pol_scheme(Scheme, Z))
```

En esta función se define un parámetro interno p, que es el numero de particiones del vector X introducido, el parámetro py se ha creado con el objetivo de comprobar que ambos vectores tiene el mismo número de componentes, y evitar errores.

Una vez el módulo lee el número de particiones que va a tener la malla tanto en el eje X como en el eje y, se crea la malla compleja, como una matriz de ceros de p*p y se adapta con un bucle a los valores de los vectores introducidos x e y, en el programa principal, estos van a ser linspace y no tienen por qué tener los mismos valores iniciales y finales, pero si deberán tener el mismo número de particiones, como ya se ha especificado.

La función devuelve el valor absoluto de los valores del complejo Z en cada posición de la malla, este va a depender del polinomio del esquema numérico, esto viene definido en la segunda función del módulo Regions, con el nombre de Pol_scheme:

```
def Pol_scheme(Scheme, Z):
    if Scheme == Euler:
        return  1 + Z
    elif Scheme == Inverse_Euler:
        return 1/(1-Z)
    elif Scheme == Crank_Nicolson:
        return (1+Z/2)/(1-Z/2)
    elif Scheme == RK4:
        return 1 + Z + (Z**2)/2 + (Z**3)/(6) + (Z**4)/(24)
    elif Scheme == Leap_frog:
        return sqrt(1)
```

De esta forma, se define cada polinomio de estabilidad, solo queda representarlo en el programa principal y visualizar las fronteras de la región de estabilidad.

2.2. Programa principal

```python
from Modulos.Cauchy_module import Cauchy
from Modulos.Functions import Oscilator_lin
from Modulos.Scheme_module import RK4,Euler,Inverse_Euler,Crank_Nicolson,Leap_frog
from Modulos.Regions import Stab_Regions

from numpy import array, linspace, size
import matplotlib.pyplot as plt

U0 = array([1,0])
dt = array([0.01, 0.1, 0.5, 0.9, 1])
tf = 100

for i in range(0,size(dt)):
    N = ((100/dt[i])+1)
    N = round(N)
    linspace_t = linspace(0,tf,(N))

    y1 = Cauchy(Oscilator_lin,U0,linspace_t,Euler)
    y2 = Cauchy(Oscilator_lin,U0,linspace_t,Inverse_Euler)
    y3 = Cauchy(Oscilator_lin,U0,linspace_t,Crank_Nicolson)
    y4 = Cauchy(Oscilator_lin,U0,linspace_t,RK4)
    y5 = Cauchy(Oscilator_lin,U0,linspace_t,Leap_frog)

    fig, axs = plt.subplots(2, 3)
    axs[0, 0].plot(linspace_t, y1[0,:])
    axs[0, 0].set_title('Euler')
    axs[0, 1].plot(linspace_t, y2[0,:])
    axs[0, 1].set_title('Inverse Euler')
    axs[1, 0].plot(linspace_t, y3[0,:])
    axs[1, 0].set_title('Crank Nicolson')
    axs[1, 1].plot(linspace_t, y4[0,:])
    axs[1, 1].set_title('Runge Kutta 4')
    axs[1, 2].plot(linspace_t, y5[0,:])
    axs[1, 2].set_title('Leap-Frog')
    plt.show()
```

Para la primera parte del programa principal se definen las condiciones iniciales ('U0'),un tiempo final tf y un vector con distintos valores de dt, que se van a utilizar para evaluar los distintos métodos de integración numérica en el problema del oscilador lineal.

Para esto se ha creado un bucle que calcula las particiones dependiendo del dt usado y posteriormente se usa Cauchy para los 5 esquemas, y se representan en gráficas.

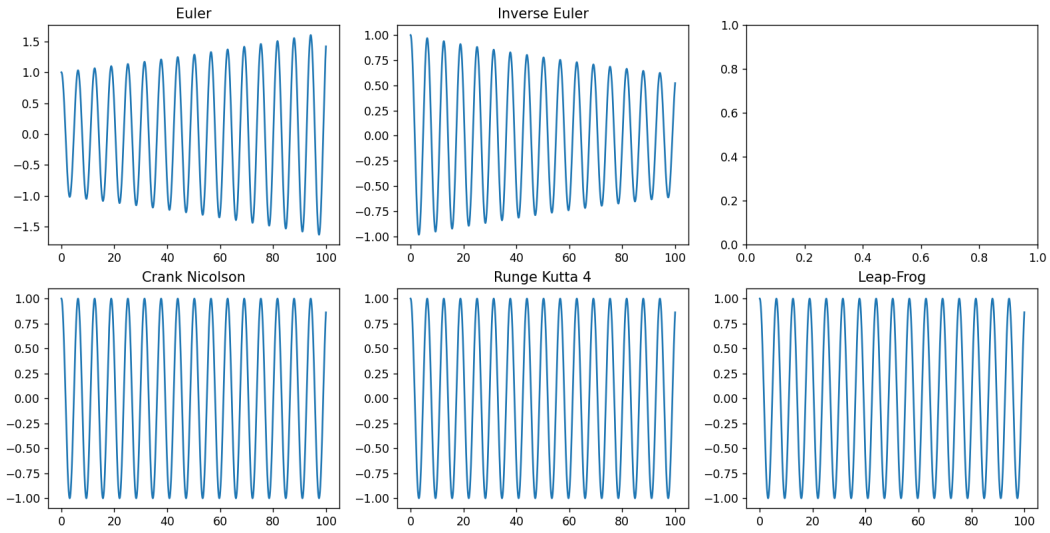
```
37
38  p = 1000 #precisión de malla de region
39  ejex = linspace(-5, 5, p)
40  ejey = linspace(-3.5, 3.5, p)
41  #Defino los ejes para representar los complejos
42  stab_Euler = Stab_Regions(ejex,ejey,Euler)
43  stab_Inverse_Euler = Stab_Regions(ejex,ejey,Inverse_Euler)
44  stab_CN = Stab_Regions(ejex,ejey,Crank_Nicolson)
45  stab_RK4 = Stab_Regions(ejex,ejey,RK4)
46  stab_LP = Stab_Regions(ejex,ejey,Leap_frog)
47
48  fig, axs = plt.subplots(2, 2)
49  axs[0, 0].contourf(ejex, ejey, stab_Euler, levels = [0, 1],  colors=['#C0C0C0'] )
50  axs[0, 0].contour(ejex, ejey, stab_Euler, levels = [0.5,1,1.5] )
51  axs[0, 0].plot([0,0],[dt,-dt], 'o', label = "Raíces del oscilador lineal con dt = " +str(dt))
52  axs[0, 0].set_title('Estabilidad Euler')
53  axs[0, 1].contourf(ejex, ejey, stab_Inverse_Euler, levels = [0, 1],  colors=['#C0C0C0'] )
54  axs[0, 1].contour(ejex, ejey, stab_Inverse_Euler, levels = [0.5,1,1.5] )
55  axs[0, 1].plot([0,0],[dt,-dt], 'o', label = "Raíces del oscilador lineal con dt = " +str(dt))
56  axs[0, 1].set_title('Estabilidad Inverse Euler')
57  axs[1, 0].contourf(ejex, ejey, stab_CN, levels = [0, 1],  colors=['#C0C0C0'] )
58  axs[1, 0].contour(ejex, ejey, stab_CN, levels = [0.5,1,1.5] )
59  axs[1, 0].plot([0,0],[dt,-dt], 'o', label = "Raíces del oscilador lineal con dt = " +str(dt))
60  axs[1, 0].set_title('Estabilidad Crank Nicolson')
61  axs[1, 1].contourf(ejex, ejey, stab_RK4, levels = [0, 1],  colors=['#C0C0C0'] )
62  axs[1, 1].contour(ejex, ejey, stab_RK4, levels = [0.5,1,1.5] )
63  axs[1, 1].plot([0,0],[dt,-dt], 'o', label = "Raíces del oscilador lineal con dt = " +str(dt))
64  axs[1, 1].set_title('Estabilidad Runge Kutta 4')
65
66  plt.show()
```

En la segunda mitad del programa se define una precisión de malla y se crean los vectores que actuarán de eje x y eje y, como el resto de procesos internos ya están explicados y realizados dentro del módulo Regions, solo queda calcular la estabilidad de los cinco esquemas, sin embargo, no se ha podido conseguir obtener la estabilidad para el Leap-Frog, ya que cuando se intenta graficar, aparece un error debido a que no es posible representar el contorno del mismo, a parte de representar la región de estabilidad, se han presentado distintas fronteras (0.5, 1, 1.5), la que nos interesa es la de valor 1 , tanto para el problema de oscilador lineal como a la hora de representar orbitas (Milestones 1 y 2).

Con estos contornos, solo queda representar los dt en el eje y, usando el vector que se ha definido en la linea 10 del programa.

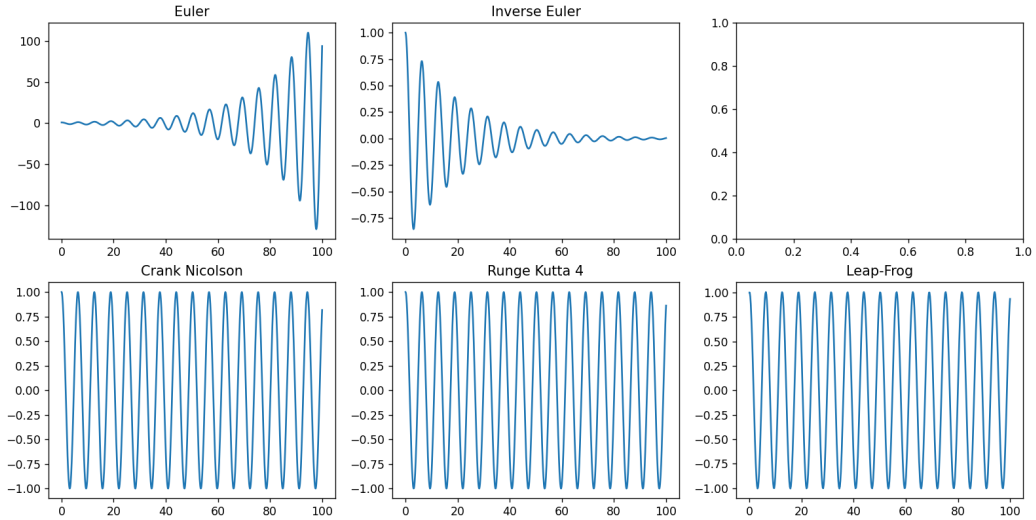
3. Discusión de Resultados

3.1. Oscilador lineal con dt = 0.01

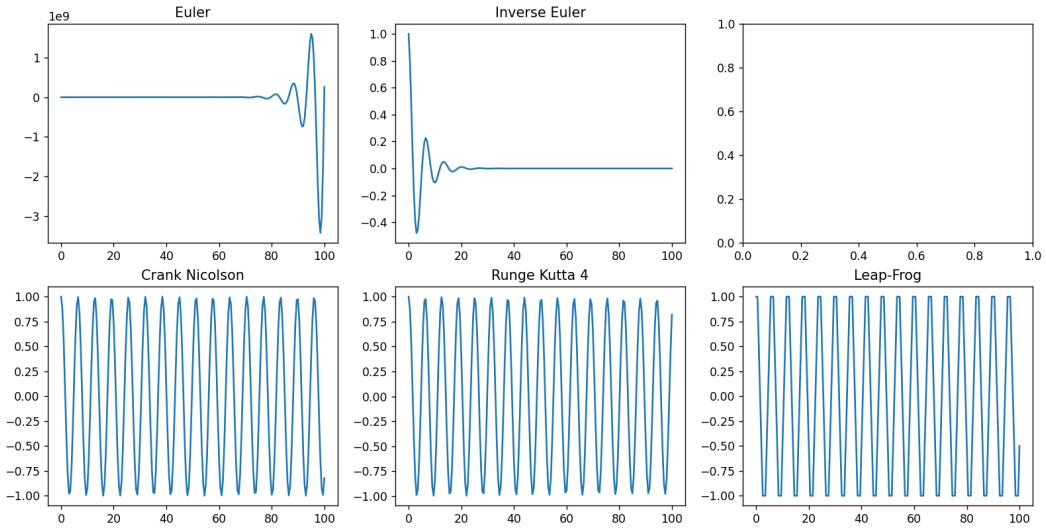


Para el caso de diferencial de tiempo mas pequeño, se puede ver que solo en Euler y Euler Inverso aparece incremento y reducción de la amplitud de la oscilación. En el resto de casos, donde los saltos en el tiempo son mayores, se espera que este cambio de amplitudes sean mucho mayores en Euler y Euler Inverso.

3.2. Oscilador lineal con dt = 0.1

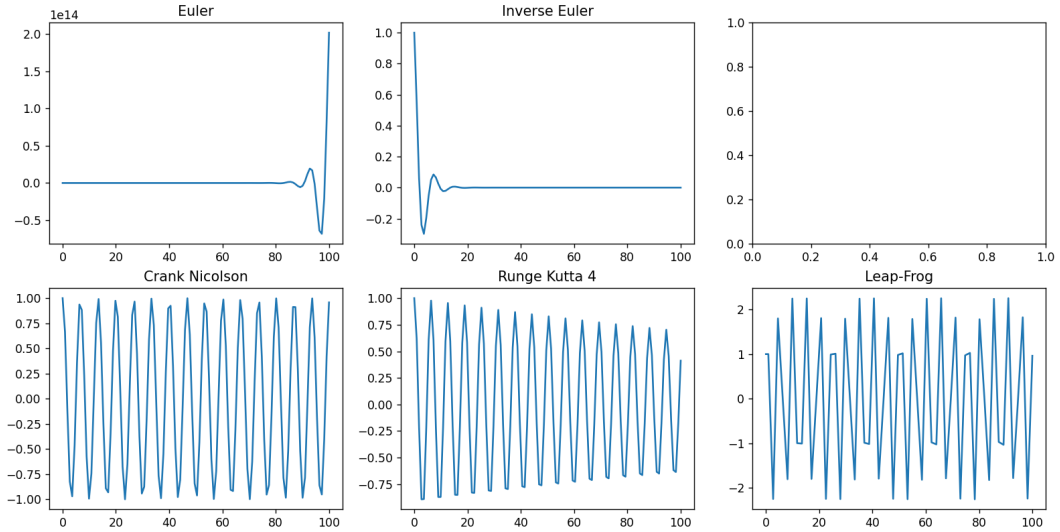


3.3. Oscilador lineal con dt = 0.5



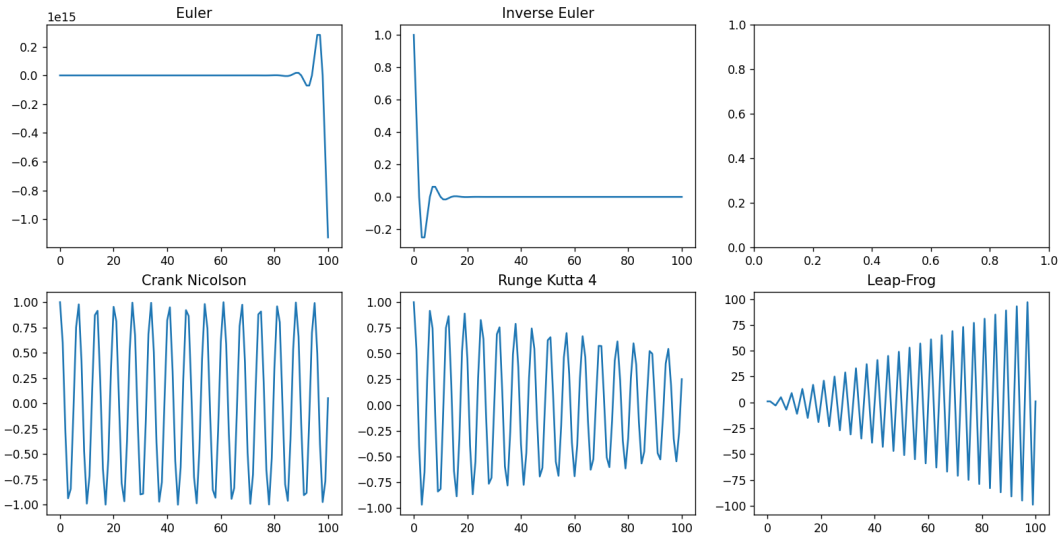
Si aumentamos el salto temporal a valores como 0.1 y 0.5, los primeros dos esquemas tienen la respuesta esperada, mientras que Crank-Nicolson, RK4 y Leap-Frog no parecen haber cambiado.

3.4. Oscilador lineal con dt = 0.9



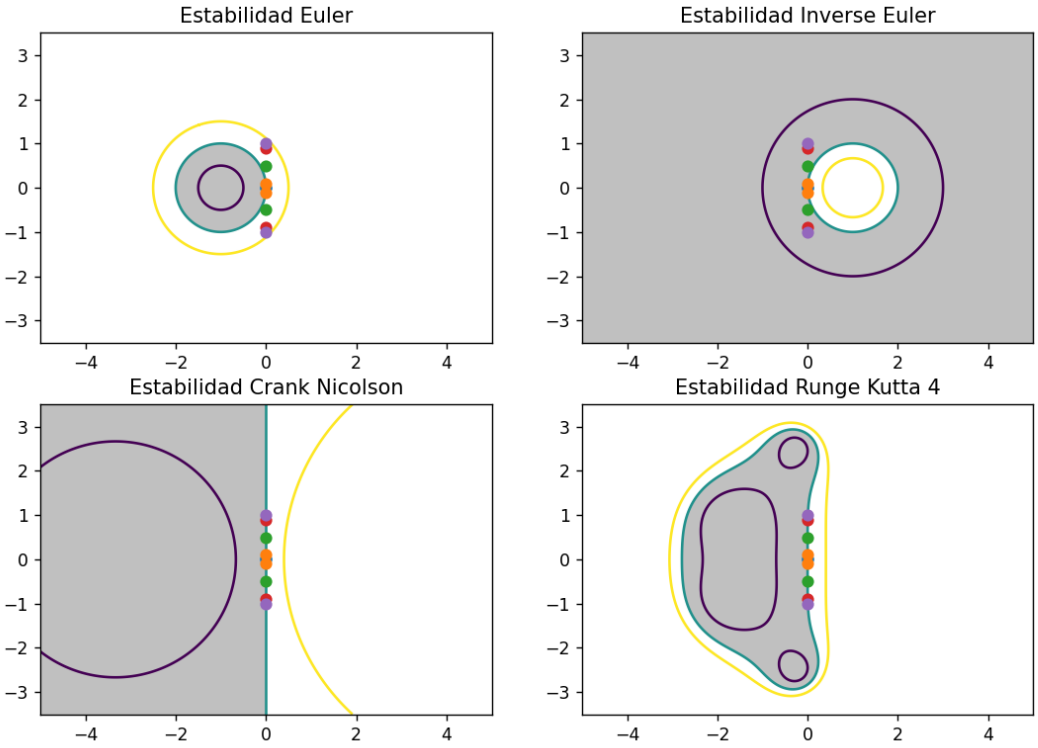
A valores cercanos e inferiores a 1, se empieza a apreciar efectos en el esquema RK4 y Leap-Frog, para Cranck-Nicolson sigue obteniéndose amplitudes casi-constantes de oscilación.

3.5. Oscilador lineal con dt = 1



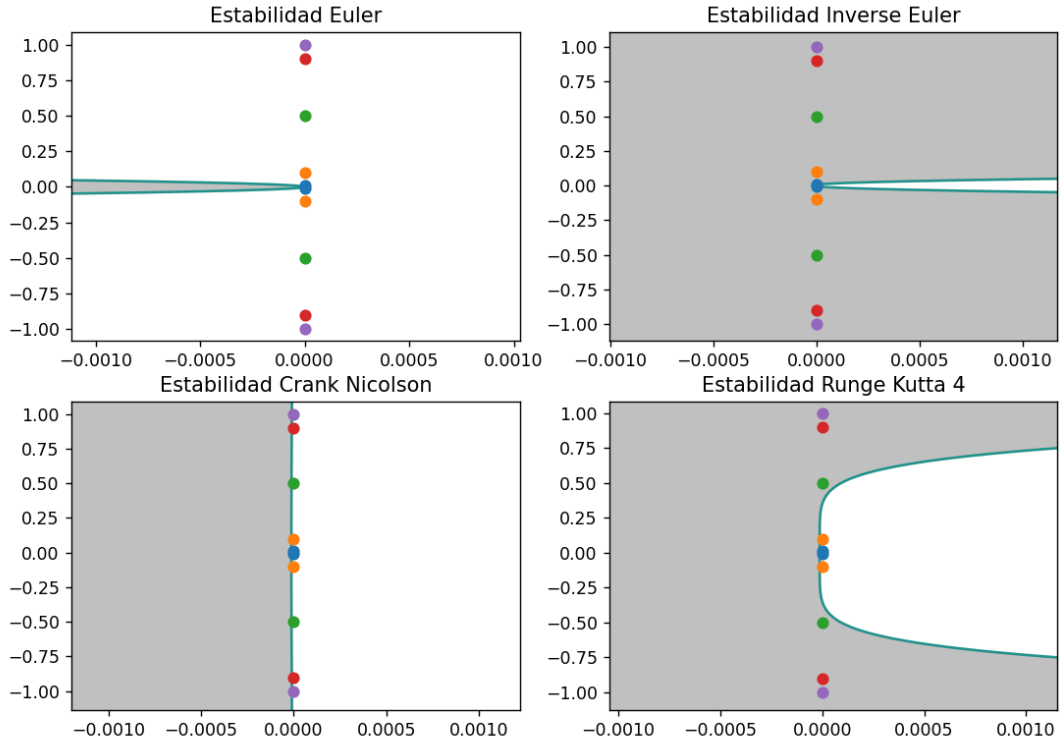
En un salto temporal unidad, el esquema Leap-Frog se dispara y diverge, Runge-kutta converge y Crank-Nicolson no parece verse afectado pese al dt elevado.

3.6. Regiones de Estabilidad



Las regiones de estabilidad son las siguientes, para los valores de dt que se encuentran dentro de la zona sombreada en gris, el oscilador lineal va a reducir su amplitud a lo largo

del tiempo, mientras que si se encuentra en la zona sombreada de blanco, esta amplitud va a incrementarse, esto explica los comportamientos de Euler y Euler Inverso.



Si se observa mas en detalle, se puede apreciar que para Crank Nicolson, todos los valores de dt, se encuentran muy próximos a la frontera, donde se conserva la amplitud del problema, para Runge Kutta 4 , se puede apreciar que para dt $= 0.01$ y dt $= 0.1$, estos se encuentran en la zona sombreada de blanco, por lo que la amplitud se debería de reducir, pero en las gráficas apenas se nota ya que son muy próximas a la frontera, para dt muy altos, como los dos últimos casos, los puntos se encuentran en la zona sombreada en gris, por lo que la amplitud del oscilador de reduce con el tiempo, como se ve en las gráficas.

La conclusión es que antes de elegir el esquema numérico para la integración numérica de un problema, sería ideal analizar primero las regiones de estabilidad para conocer su comportamiento antes de empezar a hacer cálculos, o para poder elegir un dt suficiente para que el programa no tarde demasiado en integrar.