desarrollo

December 4, 2024

[6]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Parámetros del sistema
padron = 109932
m = padron / 200
k = 25000
l_inicial = 0
c_inicial = 0.1
t_final = 5
h = 0.005
```

[7]:
```python
# Solución analítica de la ecuación diferencial
def sol_analitica(t):
        return 0.1 - 0.1 * np.cos(((k/m)**0.5)*t)

# Para el caso de beta = 1 aplico el método de Euler implícito

# Defino la inversa de la matriz A y el término independiente para tener un␣
↪sistema de tipo: Ax = b (termino_indep es b)

# Matriz inversa para la solución sin amortiguación
beta = 1
l = l_inicial
cp = 0 # c' (como el sistema no tiene amortiguación c' = 0)

divisor = (m + (h**2) * k * (beta**2) + (h * beta * l))
A_inversa = np.array([[((m+(h*beta*l))/divisor) , ((h*m*beta)/divisor)],
                        [(((-h*k*beta))/divisor) , (m/divisor)]])

termino_indep = np.array([0 , -1 * (h * beta) * (k * (c_inicial/m) + l * (cp/
↪m))])

# Vector de tiempo
t = np.arange(0, t_final+h, h)
```

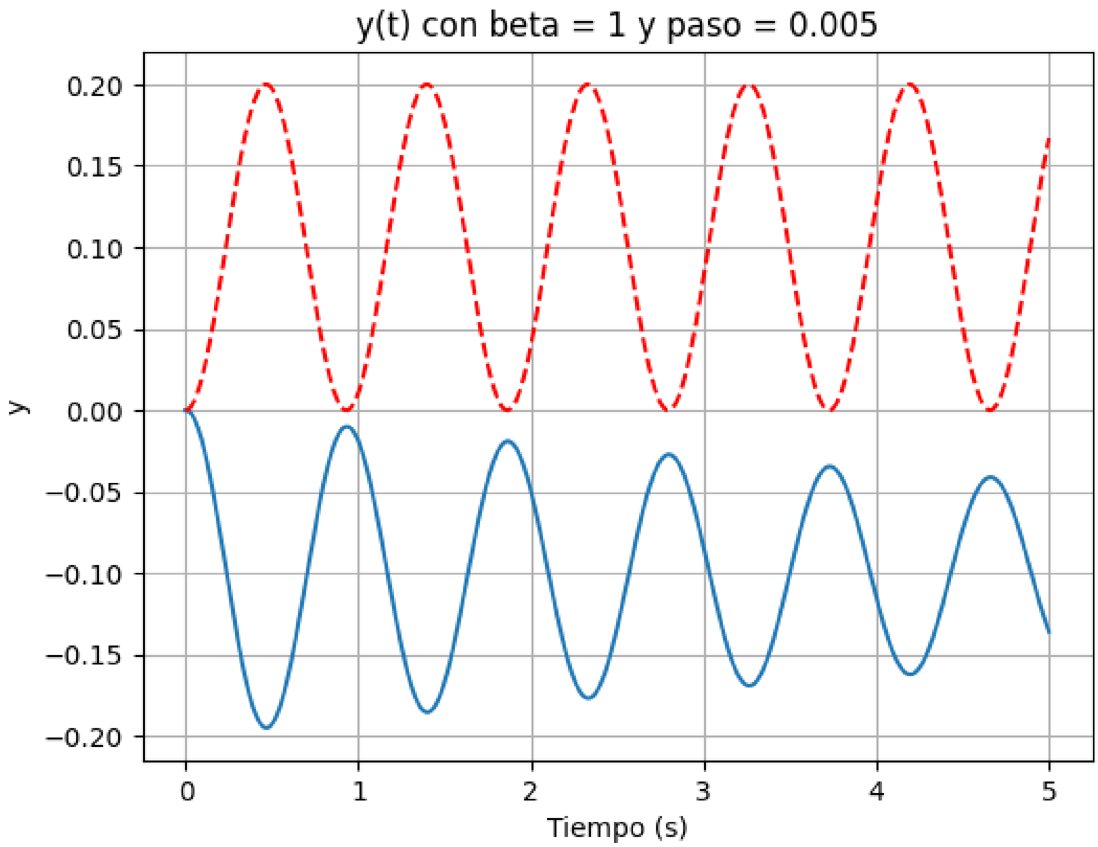
[8]:

```python
def euler_implicito(A_inversa, termino_indep, t):
        # número de pasos en 5 segundos
        cantidad = len(t)

        # Inicialización de variables
        u = np.zeros((2, cantidad))

        # Bucle para resolver sistema
        for n in range(cantidad - 1):
                aux = A_inversa @ u[:,n] + termino_indep
                u[:,(n+1)] = aux


        print("soluciones:", u[0])
        return u[0], t
```

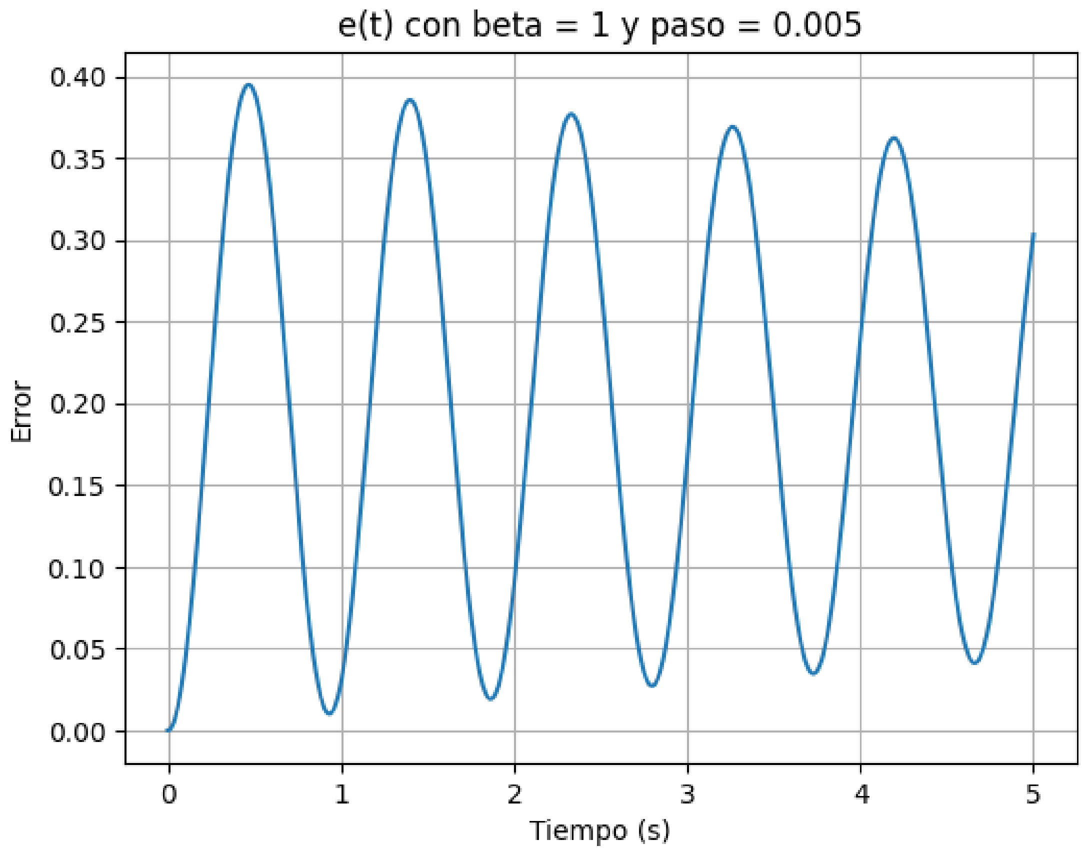
[9]:

```python
# Llamo al euler implícito
u, t = euler_implicito(A_inversa, termino_indep, t)

# Grafico la solución numérica obtenidad junto con la solución analítica
ax: plt.Axes
fig, ax = plt.subplots()
ax.plot(t, u, label='aproximación')
ax.plot(t, sol_analitica(t), 'r--', label='solucion analítica')
ax.set_title(f'y(t) con beta = {beta} y paso = {h}')
ax.set_xlabel('Tiempo (s)')
ax.set_ylabel('y')
ax.grid(True)
plt.show()

# Grafico el error de truncamiento entre la aproximación y la solución anlítica
paso = 0
aproximacion = np.copy(u)
for i in range(len(t)):
        aproximacion[i] = (sol_analitica(paso) - aproximacion[i])
        paso += h
ax: plt.Axes
fig, ax = plt.subplots()
ax.plot(t, aproximacion)
ax.set_title(f'e(t) con beta = {beta} y paso = {h}')
ax.set_xlabel('Tiempo (s)')
ax.set_ylabel('Error')
ax.grid(True)
plt.show()
```

```
soluciones: [ 0.00000000e+00  0.00000000e+00 -1.13577510e-04 … -1.32964778e-01
 -1.34483270e-01 -1.35960873e-01]
```



y(t) con beta = 1 y paso = 0.005
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
y
0
1
2
3
4
5
Tiempo (s)


e(t) con beta = 1 y paso = 0.005
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Error
0
1
2
3
4
5
Tiempo (s)