Escuela Politécnica Nacional

[Tarea 12] Ejercicios Unidad 05-A | ODE Método de Euler

Nombre: Sebastián Morales

Fecha: 06/08/2025

Curso: GR1CC

Repositorio:

https://github.com/SebastianMoralesEpn/Github1.0/tree/83509df7cb39786d75f69cd619d4786b6ac5d319/Tareas/%5BTarea%2012%5D%20Ejercicios%20Unidad%2005-A%20ODE%20M%C3%A9

CONJUNTO DE EJERCICIOS

In [1]:
```
%load_ext autoreload
```

1. Use el método de Euler para aproximar las soluciones para cada uno de los siguientes problemas de valor inicial

a. $y' = te^{3t} - 2y, 0 \leq t \leq 1, y(0) = 0, \text{con } h = 0.5$
b. $y' = 1 + (t-y)^2, 2 \leq t \leq 3, y(2) = 1, \text{con } h = 0.5$
c. $y' = 1 + {}^{y}/_{t}, 1 \leq t \leq 2, y(1) = 2, \text{con } h = 0.25$
d. $y' = \cos 2t + \operatorname{sen} 3t, 0 \leq t \leq 1, y(0) = 1, \text{con } h = 0.25$

EJERICICIO A)

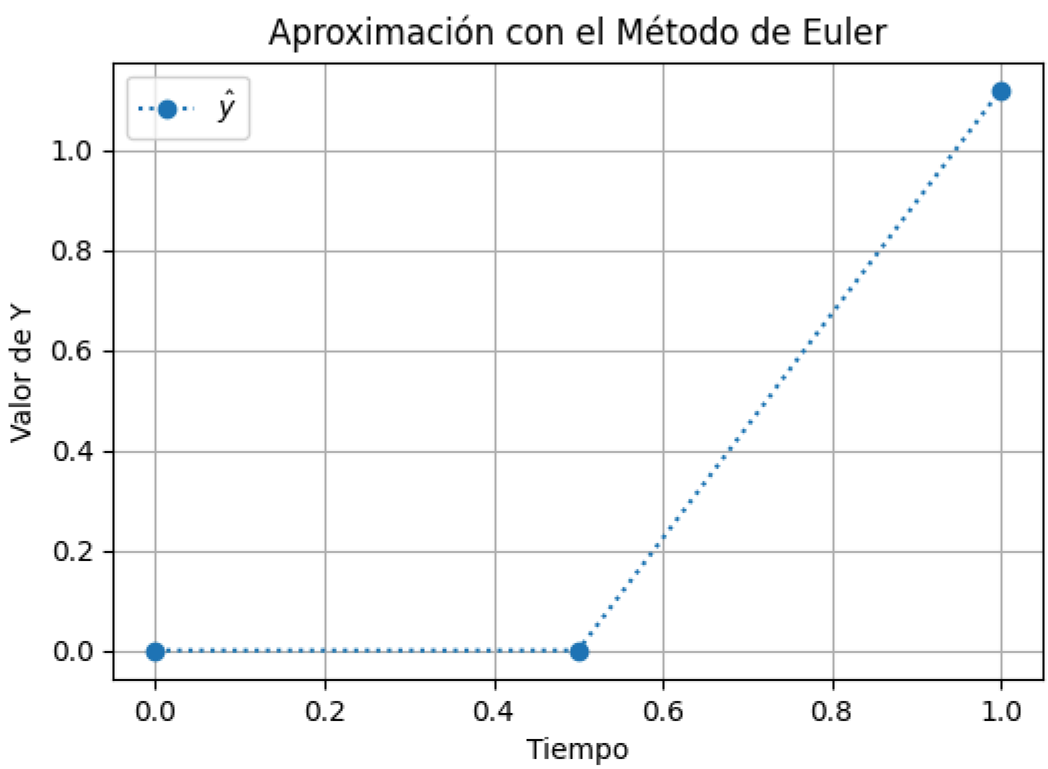
In [ ]:
```
from math import exp

def euler_method(f, y0, a, b, N):
    h = (b - a) / N
    t_values = [a + i * h for i in range(N + 1)]
    y_values = [y0]

    for i in range(N):
        y_next = y_values[-1] + h * f(t_values[i], y_values[-1])
        y_values.append(round(y_next, 4))
```

```python
    return y_values, t_values, h

f_a = lambda t, y: t * exp(3 * t) - 2 * y

a, b, y0, step_size = 0, 1, 0, 0.5
N = int((b - a) / step_size)

ys, ts, h = euler_method(f_a, y0, a, b, N)
ys
```

Out[ ]: [0, 0.0, 1.1204]

Gráfico:

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

def plot_euler_solution(t_values, y_values):
    plt.figure(figsize=(6, 4))
    plt.plot(t_values, y_values, marker="o", linestyle=":", label=r"$\hat{y}$"
    plt.xlabel("Tiempo")
    plt.ylabel("Valor de Y")
    plt.title("Aproximación con el Método de Euler")
    plt.legend()
    plt.grid(True)
    plt.show()

plot_euler_solution(ts, ys)
```



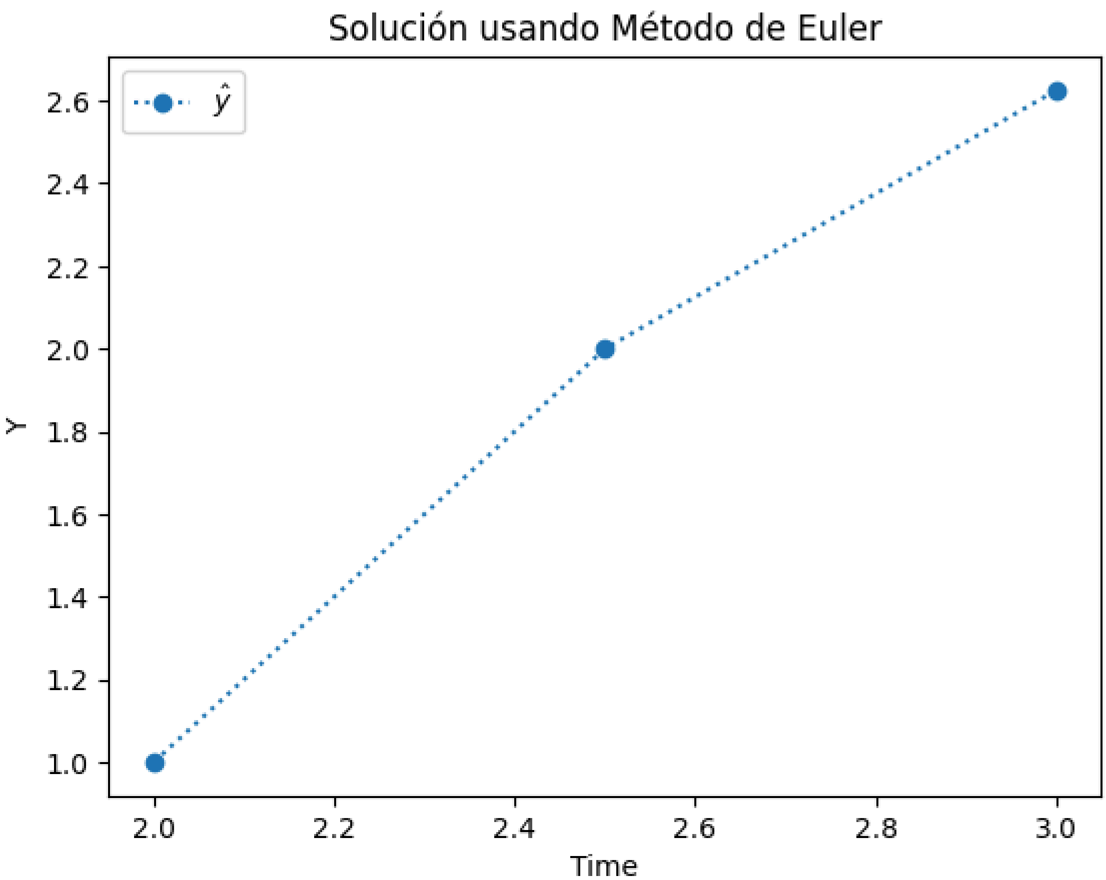
EJERICICO B)

In [7]:
```
def euler_method(f, y0, a, b, N):
    h = (b - a) / N
    t_values = [a + i * h for i in range(N + 1)]
    y_values = [y0]

    for i in range(N):
        y_next = y_values[-1] + h * f(t_values[i], y_values[-1])
        y_values.append(round(y_next, 4))

    return y_values, t_values, h

f_b = lambda t, y: 1 + pow(t - y, 2)

a, b, y0, step_size = 2, 3, 1, 0.5
N = int((b - a) / step_size)

ys, ts, h = euler_method(f_b, y0, a, b, N)
ys
```

Out[7]:
```
[1, 2.0, 2.625]
```

Gráfico:

In [8]:
```
plt.plot(ts, ys, marker="o", linestyle=":", label=r"$\hat{y}$")
plt.xlabel("Time")
plt.ylabel("Y")
plt.title("Solución usando Método de Euler")
plt.legend()
plt.show()
```



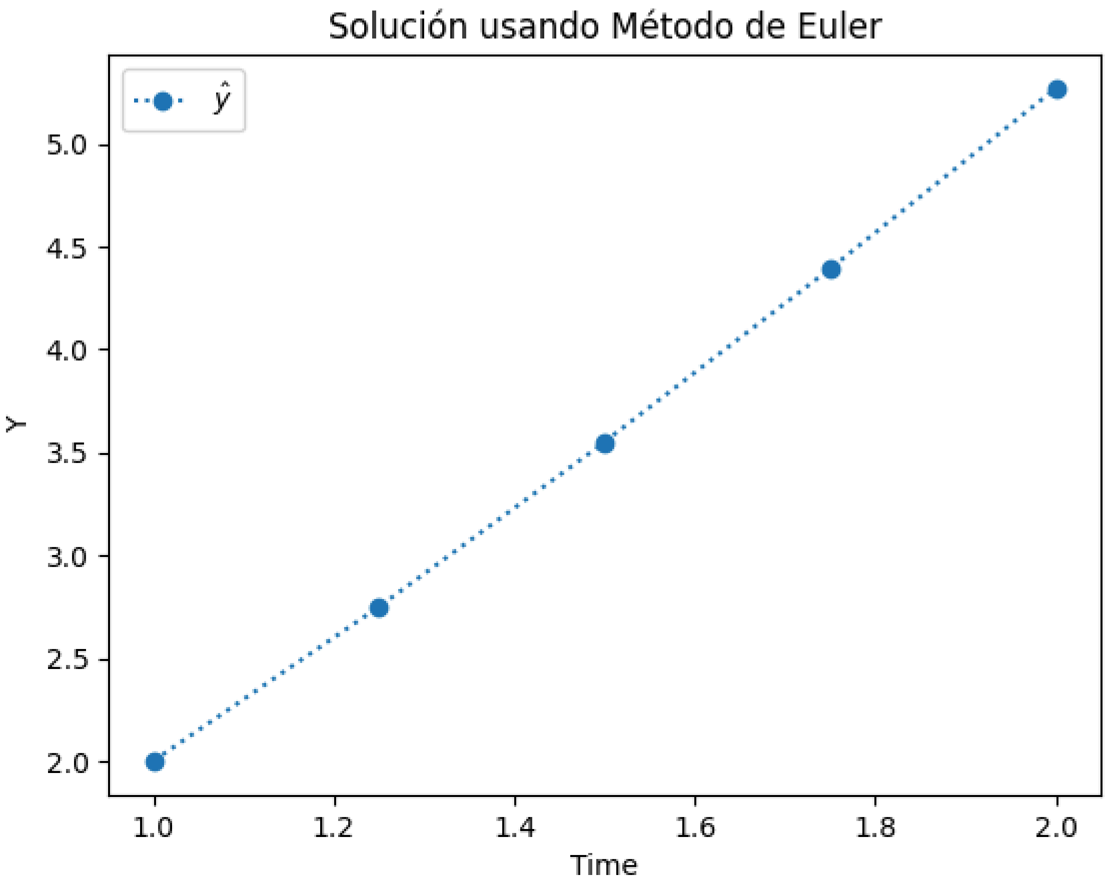
EJERCICIO C)

In [9]:
```
def euler_method(f, y0, a, b, N):
    h = (b - a) / N
    t_values = [a + i * h for i in range(N + 1)]
    y_values = [y0]

    for i in range(N):
        y_next = y_values[-1] + h * f(t_values[i], y_values[-1])
        y_values.append(round(y_next, 4))

    return y_values, t_values, h

f_c = lambda t, y: 1 + y / t

a, b, y0, step_size = 1, 2, 2, 0.25
N = int((b - a) / step_size)

ys, ts, h = euler_method(f_c, y0, a, b, N)
ys
```

Out[9]:
```
[2, 2.75, 3.55, 4.3917, 5.2691]
```

Gráfico:

In [10]:
```python
plt.plot(ts, ys, marker="o", linestyle=":", label=r"$\hat{y}$")
plt.xlabel("Time")
plt.ylabel("Y")
plt.title("Solución usando Método de Euler")
plt.legend()
plt.show()
```



EJRCICIO D)

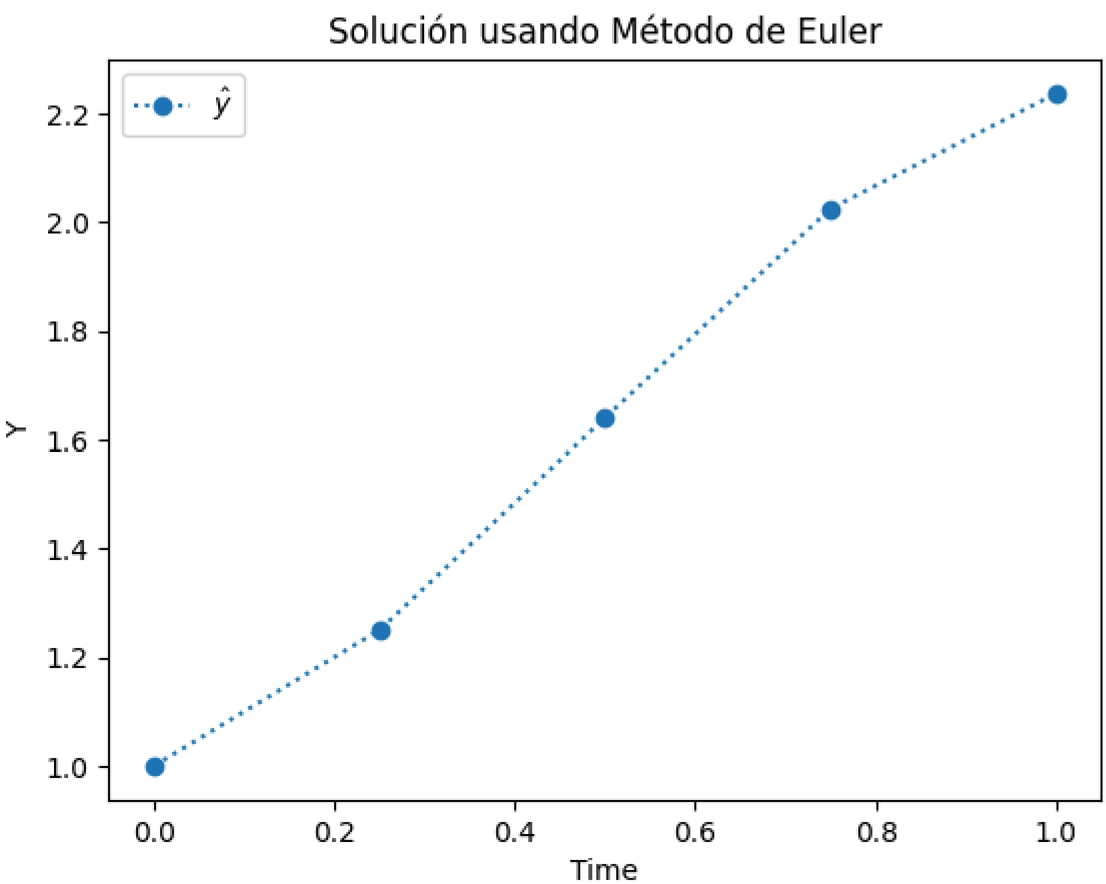
In [12]:
```python
from math import cos, sin

def euler_method(f, y0, a, b, N):
    h = (b - a) / N
    t_values = [a + i * h for i in range(N + 1)]
    y_values = [y0]

    for i in range(N):
        y_next = y_values[-1] + h * f(t_values[i], y_values[-1])
        y_values.append(round(y_next, 4))

    return y_values, t_values, h

f_d = lambda t, y: cos(2 * t) + sin(3 * t)

a, b, y0, step_size = 0, 1, 1, 0.25
N = int((b - a) / step_size)
```

```
ys, ts, h = euler_method(f_d, y0, a, b, N)
ys
```

Out[12]: `[1, 1.25, 1.6398, 2.0242, 2.2364]`

Gráfico:

In [13]:
```
plt.plot(ts, ys, marker="o", linestyle=":", label=r"$\hat{y}$")
plt.xlabel("Time")
plt.ylabel("Y")
plt.title("Solución usando Método de Euler")
plt.legend()
plt.show()
```



2. Las soluciones reales para los problemas de valor inicial en el ejercicio 1 se proporcionan aquí. Compare el error real en cada paso.

a. $y(t) = \frac{1}{5}te^{3t} - \frac{1}{25}e^{3t} + \frac{1}{25}e^{-2t}$

b. $y(t) = t + \frac{1}{1-t}$

c. $y(t) = t \ln t + 2t$

d. $y(t) = \frac{1}{2}\text{sen}\,2t - \frac{1}{3}\cos 3t + \frac{4}{3}$

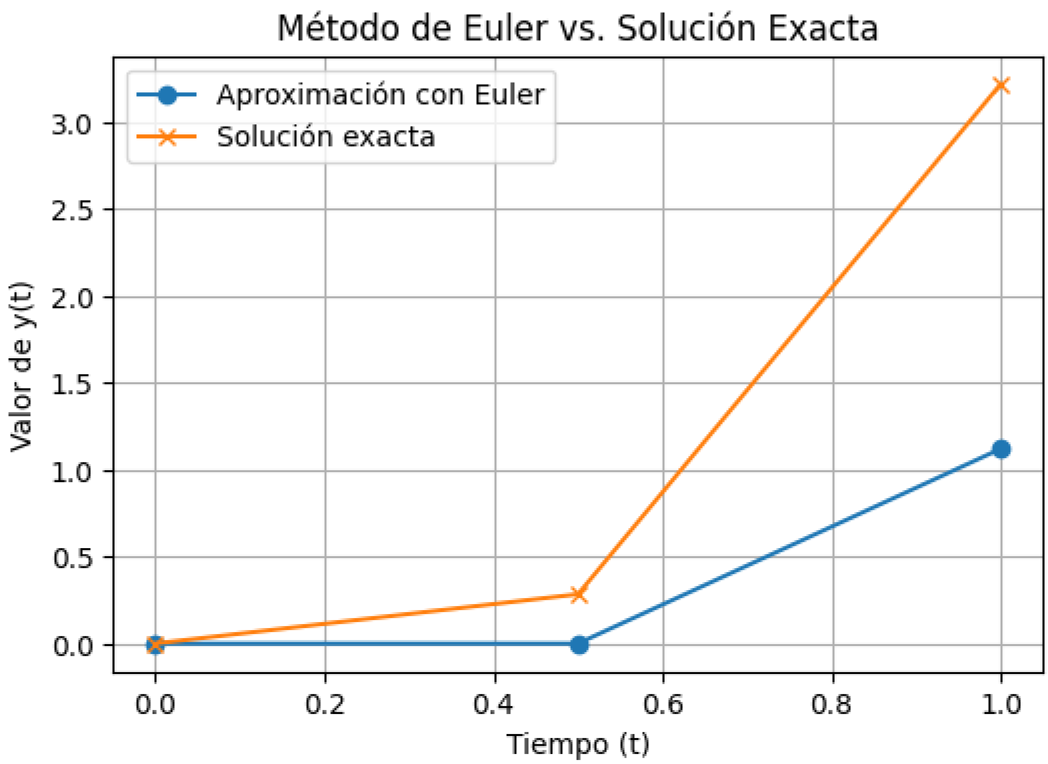
EJERCICIO A)

In [14]:
```python
def euler_method(f, y0, a, b, N):
    h = (b - a) / N
    t_values = [round(a + i * h, 4) for i in range(N + 1)]
    y_values = [y0]

    for i in range(N):
        y_next = y_values[-1] + h * f(t_values[i], y_values[-1])
        y_values.append(round(y_next, 4))

    return y_values, t_values, h

f_a = lambda t, y: t * exp(3 * t) - 2 * y
f_ex_a = lambda t: (1 / 5) * t * exp(3 * t) - (1 / 25) * exp(3 * t) + (1 / 25)

a, b, y0, step_size = 0, 1, 0, 0.5
N = int((b - a) / step_size)

ys, ts, h = euler_method(f_a, y0, a, b, N)
ys_exact = [round(f_ex_a(t), 4) for t in ts]
errors = [round(abs(ys_exact[i] - ys[i]), 4) for i in range(len(ts))]

print("Aproximación:", ys)
print("Exacta:", ys_exact)
print("Errores en cada paso:", errors)
```

```
Aproximación: [0, 0.0, 1.1204]
Exacta: [0.0, 0.2836, 3.2191]
Errores en cada paso: [0.0, 0.2836, 2.0987]
```

Gráfico:

In [15]:
```python
def plot_euler_vs_exact(t_values, y_approx, y_exact):
    plt.figure(figsize=(6, 4))
    plt.plot(t_values, y_approx, 'o-', label='Aproximación con Euler')
    plt.plot(t_values, y_exact, 'x-', label='Solución exacta')
    plt.xlabel('Tiempo (t)')
    plt.ylabel('Valor de y(t)')
    plt.title("Método de Euler vs. Solución Exacta")
    plt.legend()
    plt.grid(True)
    plt.show()

plot_euler_vs_exact(ts, ys, ys_exact)
```



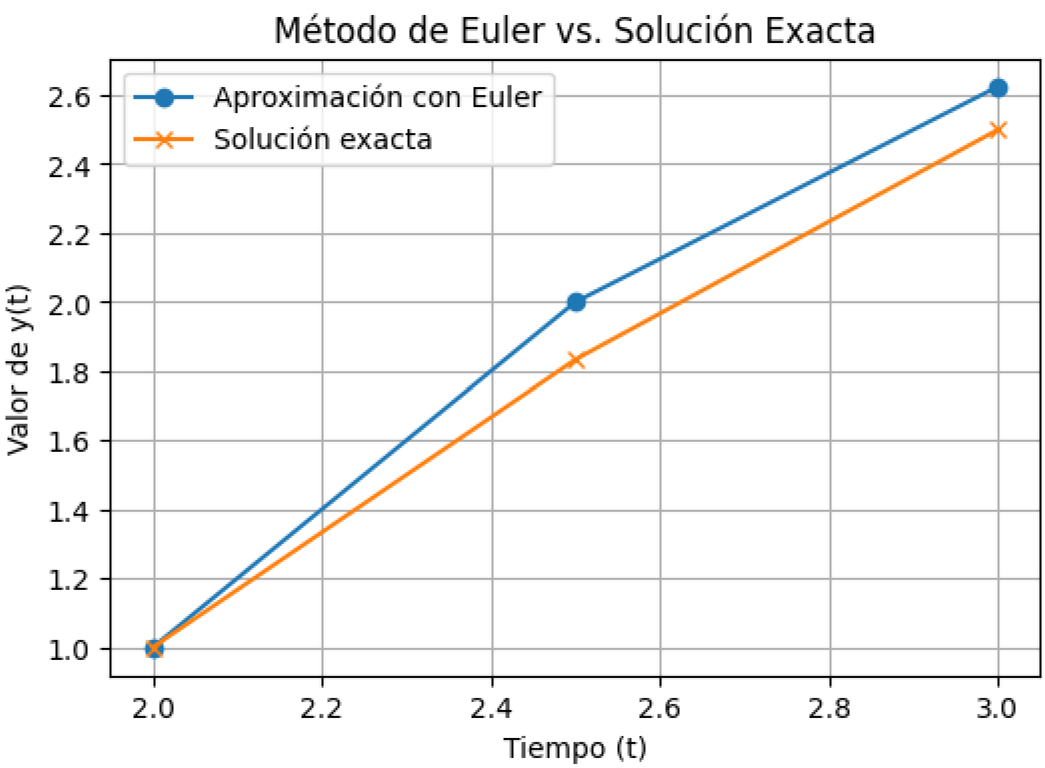
EJERICICIO B)

In [16]:
```python
def euler_method(f, y0, a, b, N):
    h = (b - a) / N
    t_values = [round(a + i * h, 4) for i in range(N + 1)]
    y_values = [y0]

    for i in range(N):
        y_next = y_values[-1] + h * f(t_values[i], y_values[-1])
        y_values.append(round(y_next, 4))

    return y_values, t_values, h

f_b = lambda t, y: 1 + (t - y) ** 2
f_ex_b = lambda t: t + 1 / (1 - t)

a, b, y0, step_size = 2, 3, 1, 0.5
N = int((b - a) / step_size)

ys, ts, h = euler_method(f_b, y0, a, b, N)
ys_exact = [round(f_ex_b(t), 4) for t in ts]
errors = [round(abs(ys_exact[i] - ys[i]), 4) for i in range(len(ts))]

print("Aproximación:", ys)
print("Exacta:", ys_exact)
print("Errores en cada paso:", errors)
```

```
Aproximación: [1, 2.0, 2.625]
Exacta: [1.0, 1.8333, 2.5]
Errores en cada paso: [0.0, 0.1667, 0.125]
```

Gráfico:

In [17]:
```
def plot_euler_vs_exact(t_values, y_approx, y_exact):
    plt.figure(figsize=(6, 4))
    plt.plot(t_values, y_approx, 'o-', label='Aproximación con Euler')
    plt.plot(t_values, y_exact, 'x-', label='Solución exacta')
    plt.xlabel('Tiempo (t)')
    plt.ylabel('Valor de y(t)')
    plt.title("Método de Euler vs. Solución Exacta")
    plt.legend()
    plt.grid(True)
    plt.show()

plot_euler_vs_exact(ts, ys, ys_exact)
```



EJERICICIO C)

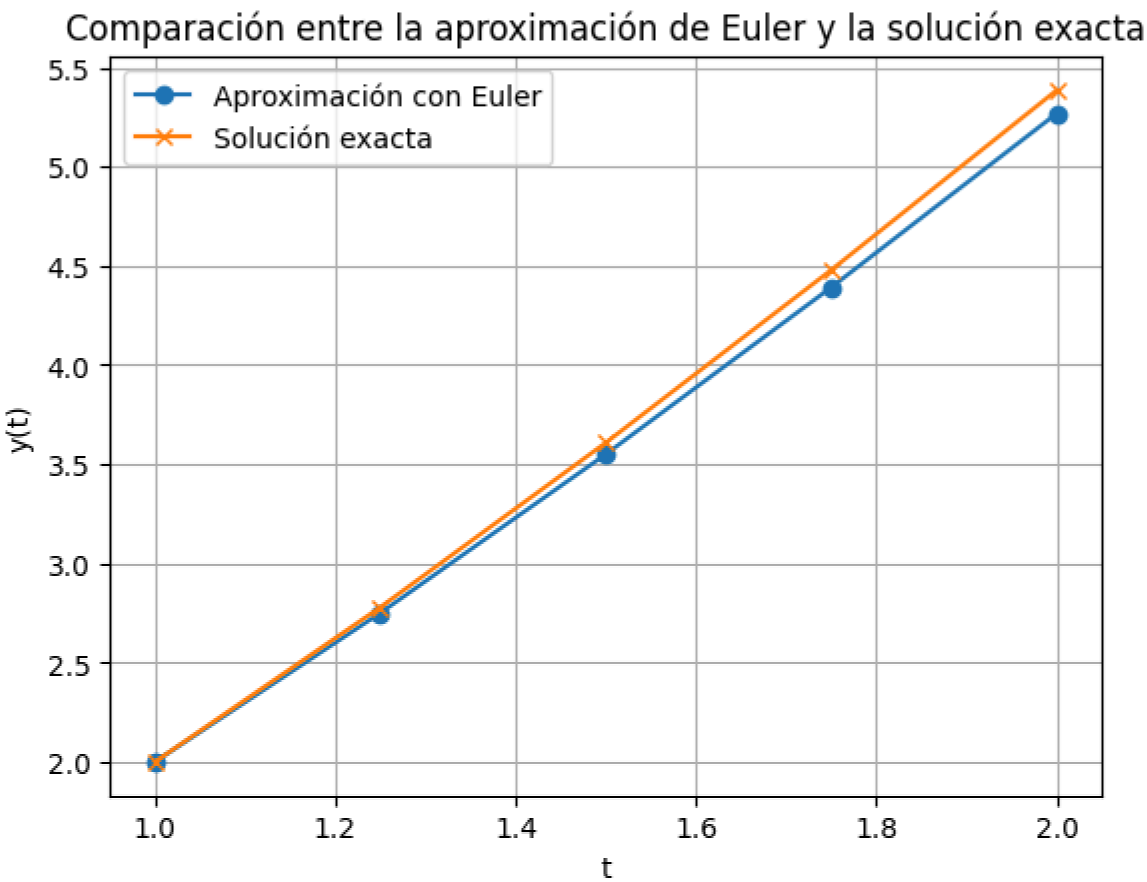
In [19]:
```
from math import log

def euler_method(f, y0, a, b, N):
    h = (b - a) / N
    t_values = [round(a + i * h, 4) for i in range(N + 1)]
    y_values = [y0]

    for i in range(N):
        y_next = y_values[-1] + h * f(t_values[i], y_values[-1])
        y_values.append(round(y_next, 4))

    return y_values, t_values, h
```

```python
f_c = lambda t, y: 1 + y / t
f_ex_c = lambda t: t * log(t) + 2 * t

a, b, y0, step_size = 1, 2, 2, 0.25
N = int((b - a) / step_size)

ys, ts, h = euler_method(f_c, y0, a, b, N)
ys_exact = [round(f_ex_c(t), 4) for t in ts]
errors = [round(abs(ys_exact[i] - ys[i]), 4) for i in range(len(ts))]

print("Aproximación:", ys)
print("Exacta:", ys_exact)
print("Errores en cada paso:", errors)
```

```
Aproximación: [2, 2.75, 3.55, 4.3917, 5.2691]
Exacta: [2.0, 2.7789, 3.6082, 4.4793, 5.3863]
Errores en cada paso: [0.0, 0.0289, 0.0582, 0.0876, 0.1172]
```

Gráfico:

In [20]:
```python
plt.plot(ts, ys, 'o-', label='Aproximación con Euler')
plt.plot(ts, ys_exact, 'x-', label='Solución exacta')
plt.xlabel('t')
plt.ylabel('y(t)')
plt.title("Comparación entre la aproximación de Euler y la solución exacta")
plt.legend()
plt.grid(True)
plt.show()
```



EJERCICIO D)

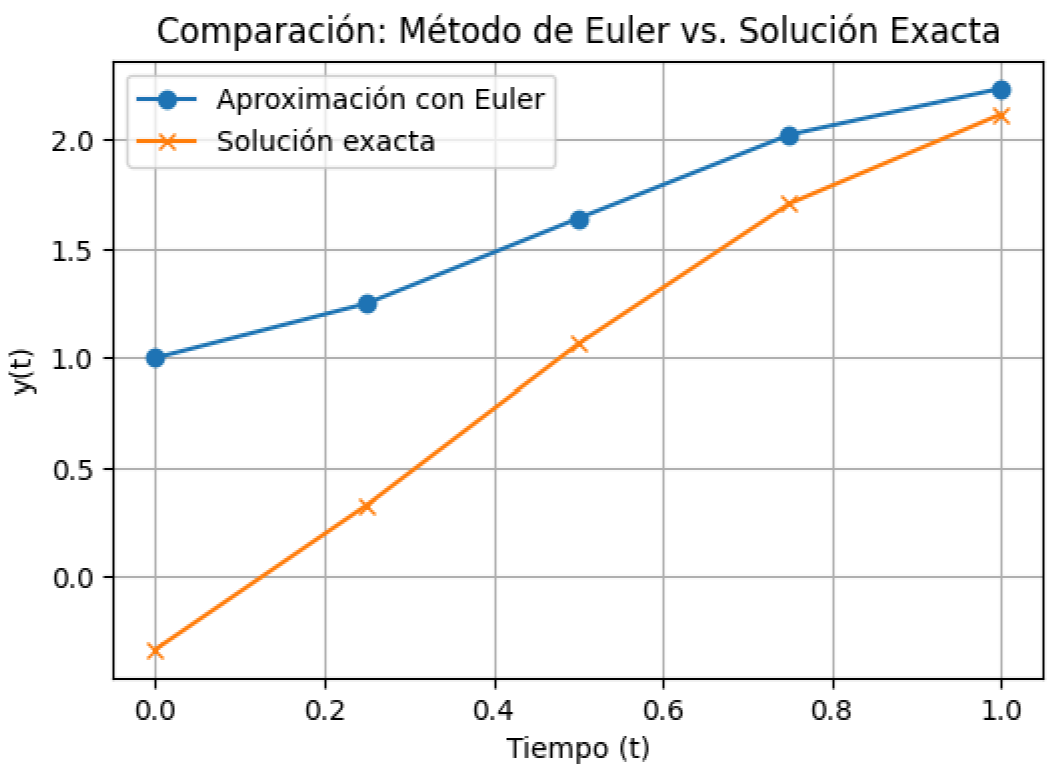
In [21]:
```
def euler_method(f, y0, a, b, N):
    h = (b - a) / N
    t_values = [round(a + i * h, 4) for i in range(N + 1)]
    y_values = [y0]

    for i in range(N):
        y_next = y_values[-1] + h * f(t_values[i], y_values[-1])
        y_values.append(round(y_next, 4))

    return y_values, t_values, h

f_d = lambda t, y: cos(2 * t) + sin(3 * t)
f_ex_d = lambda t: (1 / 2) * sin(2 * t) - (1 / 3) * cos(3 * t) + (4 / 3) * t

a, b, y0, step_size = 0, 1, 1, 0.25
N = int((b - a) / step_size)

ys, ts, h = euler_method(f_d, y0, a, b, N)
ys_exact = [round(f_ex_d(t), 4) for t in ts]
errors = [round(abs(ys_exact[i] - ys[i]), 4) for i in range(len(ts))]

print("Aproximación:", ys)
print("Exacta:", ys_exact)
```

```
print("Errores en cada paso:", errors)
```

```
Aproximación: [1, 1.25, 1.6398, 2.0242, 2.2364]
Exacta: [-0.3333, 0.3291, 1.0638, 1.7081, 2.118]
Errores en cada paso: [1.3333, 0.9209, 0.576, 0.3161, 0.1184]
```

Gráfico:

In [22]:
```
def plot_euler_vs_exact(t_values, y_approx, y_exact):
    plt.figure(figsize=(6, 4))
    plt.plot(t_values, y_approx, 'o-', label='Aproximación con Euler')
    plt.plot(t_values, y_exact, 'x-', label='Solución exacta')
    plt.xlabel('Tiempo (t)')
    plt.ylabel('y(t)')
    plt.title("Comparación: Método de Euler vs. Solución Exacta")
    plt.legend()
    plt.grid(True)
    plt.show()

plot_euler_vs_exact(ts, ys, ys_exact)
```



3. Utilice el método de Euler para aproximar las soluciones para cada uno de los siguientes problemas de valor inicial.

a. $y' = {}^{y}/_{t} - \left({}^{y}/_{t}\right)^2, 1 \le t \le 2, y(1) = 1, \text{con } h = 0.1$

b. $y' = 1 + {}^{y}/_{t} + \left({}^{y}/_{t}\right)^2, 1 \le t \le 3, y(1) = 0, \text{con } h = 0.2$

c. $y' = -(y+1)(y+3), 0 \le t \le 2, y(0) = -2, \text{con } h = 0.2$

d. $y' = -5y + 5t^2 + 2t, 0 \le t \le 1, y(0) = \frac{1}{3}, \text{con } h = 0.1$

EJERCICIO A)

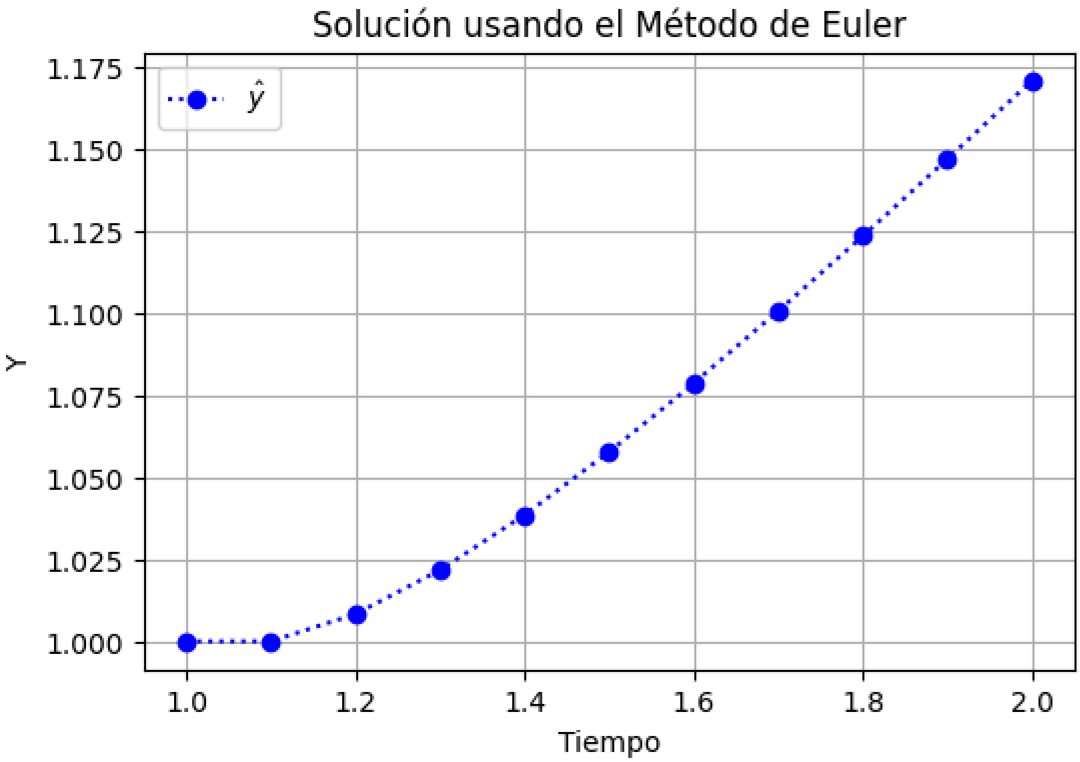
In [23]:
```
def euler_method(f, y0, a, b, N):
    h = (b - a) / N
    t_values = [round(a + i * h, 4) for i in range(N + 1)]
    y_values = [y0]

    for i in range(N):
        y_next = y_values[-1] + h * f(t_values[i], y_values[-1])
        y_values.append(round(y_next, 4))

    return y_values, t_values, h

f_a = lambda t, y: y / t - (y / t) ** 2

a, b, y0, step_size = 1, 2, 1, 0.1
N = int((b - a) / step_size)

ys, ts, h = euler_method(f_a, y0, a, b, N)
ys
```

Out[23]:
```
[1,
 1.0,
 1.0083,
 1.0217,
 1.0385,
 1.0577,
 1.0785,
 1.1005,
 1.1233,
 1.1468,
 1.1707]
```

Gráfico:

In [24]:
```
def plot_euler_solution(t_values, y_values):
    plt.figure(figsize=(6, 4))
    plt.plot(t_values, y_values, marker="o", linestyle=":", color="blue", labe
    plt.xlabel("Tiempo")
    plt.ylabel("Y")
    plt.title("Solución usando el Método de Euler")
    plt.legend()
    plt.grid(True)
```

```
    plt.show()

plot_euler_solution(ts, ys)
```



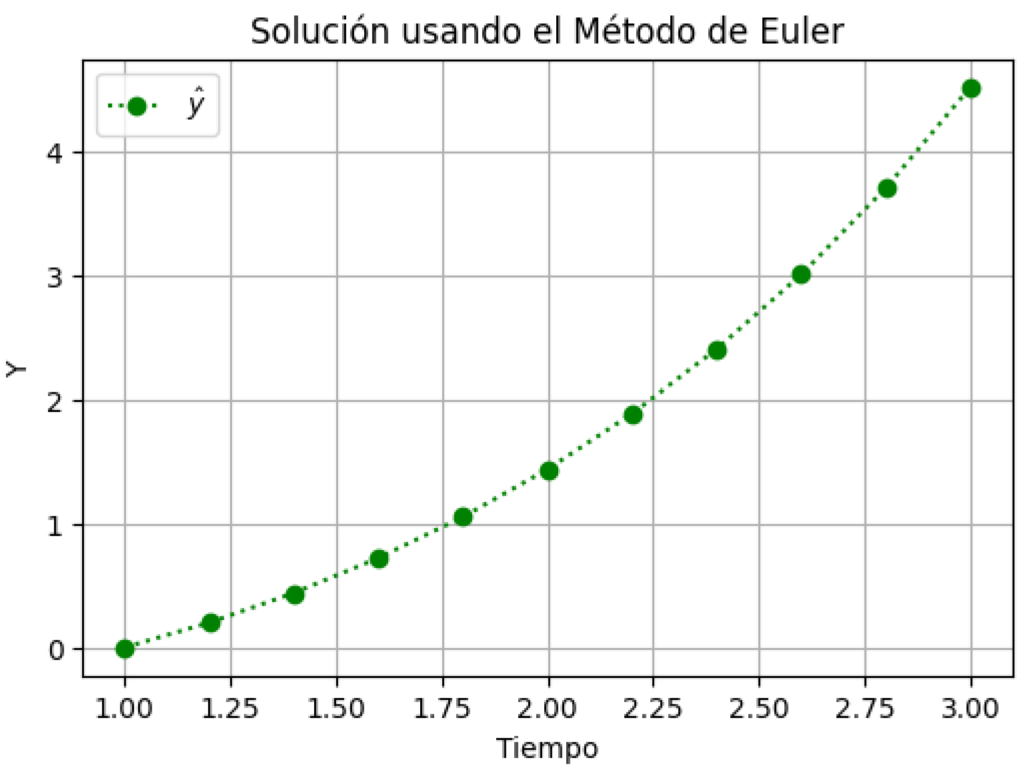
EJERCICIO B)

In [25]:
```
def euler_method(f, y0, a, b, N):
    h = (b - a) / N
    t_values = [round(a + i * h, 4) for i in range(N + 1)]
    y_values = [y0]

    for i in range(N):
        y_next = y_values[-1] + h * f(t_values[i], y_values[-1])
        y_values.append(round(y_next, 4))

    return y_values, t_values, h

f_b = lambda t, y: 1 + y / t + (y / t) ** 2

a, b, y0, step_size = 1, 3, 0, 0.2
N = int((b - a) / step_size)

ys, ts, h = euler_method(f_b, y0, a, b, N)
ys
```

Out[25]:
```
[0,
 0.2,
 0.4389,
 0.7213,
 1.0521,
 1.4373,
 1.8843,
 2.4023,
 3.0029,
 3.7007,
 4.5144]
```

Gráfico:

In [26]:
```python
def plot_euler_solution(t_values, y_values):
    plt.figure(figsize=(6, 4))
    plt.plot(t_values, y_values, marker="o", linestyle=":", color="green", lab
    plt.xlabel("Tiempo")
    plt.ylabel("Y")
    plt.title("Solución usando el Método de Euler")
    plt.legend()
    plt.grid(True)
    plt.show()

plot_euler_solution(ts, ys)
```



EJERCICIO C)

In [27]:
```python
def euler_method(f, y0, a, b, N):
```

```python
    h = (b - a) / N
    t_values = [round(a + i * h, 4) for i in range(N + 1)]
    y_values = [y0]

    for i in range(N):
        y_next = y_values[-1] + h * f(t_values[i], y_values[-1])
        y_values.append(round(y_next, 4))

    return y_values, t_values, h

f_c = lambda t, y: -(y + 1) * (y + 3)

a, b, y0, step_size = 0, 2, -2, 0.2
N = int((b - a) / step_size)

ys, ts, h = euler_method(f_c, y0, a, b, N)
ys
```

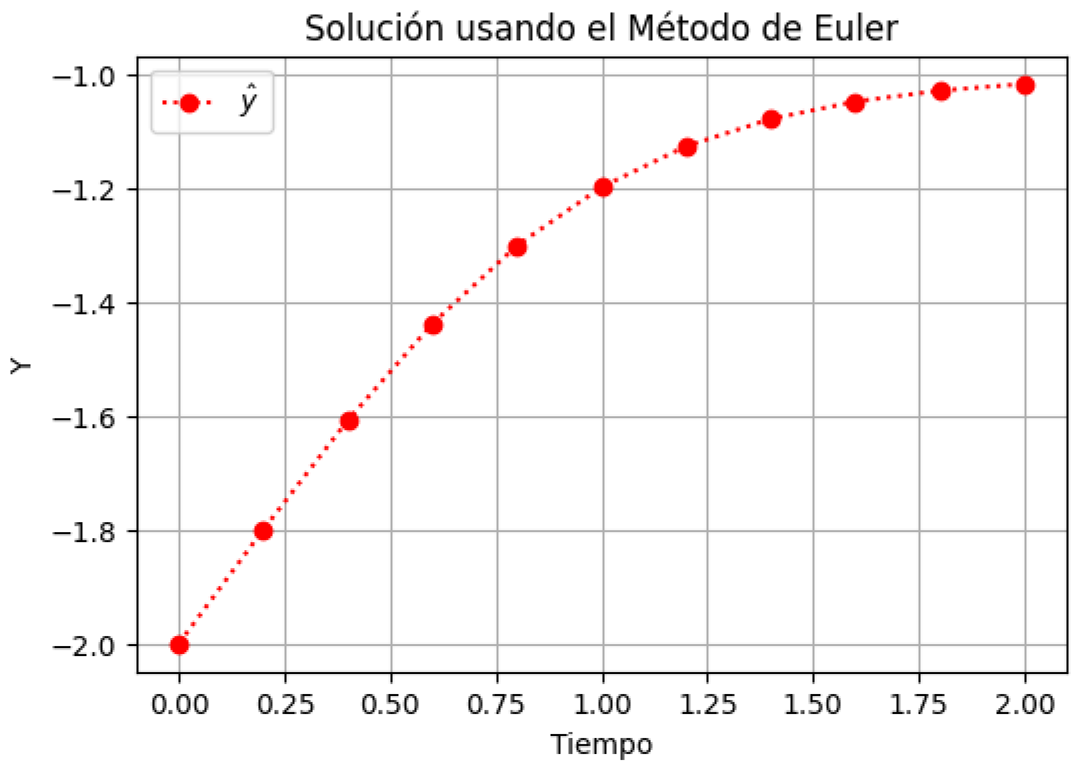
Out[27]:

```
[-2,
 -1.8,
 -1.608,
 -1.4387,
 -1.3017,
 -1.1992,
 -1.1275,
 -1.0798,
 -1.0492,
 -1.03,
 -1.0182]
```

Gráfico:

In [28]:

```python
def plot_euler_solution(t_values, y_values):
    plt.figure(figsize=(6, 4))
    plt.plot(t_values, y_values, marker="o", linestyle=":", color="red", label
    plt.xlabel("Tiempo")
    plt.ylabel("Y")
    plt.title("Solución usando el Método de Euler")
    plt.legend()
    plt.grid(True)
    plt.show()

plot_euler_solution(ts, ys)
```



EJERCICIO D)

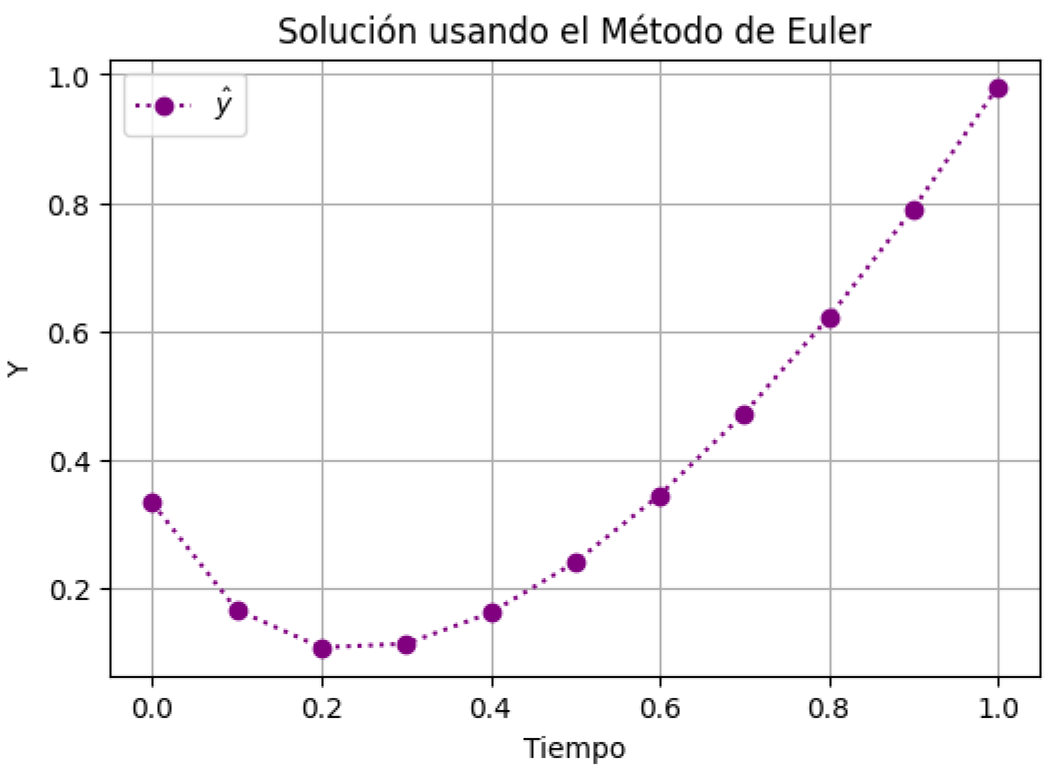
In [29]:

```
def euler_method(f, y0, a, b, N):
    h = (b - a) / N
    t_values = [round(a + i * h, 4) for i in range(N + 1)]
    y_values = [y0]

    for i in range(N):
        y_next = y_values[-1] + h * f(t_values[i], y_values[-1])
        y_values.append(round(y_next, 4))

    return y_values, t_values, h

f_d = lambda t, y: -5 * y + 5 * t**2 + 2 * t

a, b, y0, step_size = 0, 1, 1/3, 0.1
N = int((b - a) / step_size)

ys, ts, h = euler_method(f_d, y0, a, b, N)
ys
```

```
Out[29]: [0.3333333333333333,
 0.1667,
 0.1084,
 0.1142,
 0.1621,
 0.2411,
 0.3456,
 0.4728,
 0.6214,
 0.7907,
 0.9804]
```

Gráfico:

```
In [30]: def plot_euler_solution(t_values, y_values):
    plt.figure(figsize=(6, 4))
    plt.plot(t_values, y_values, marker="o", linestyle=":", color="purple", la
    plt.xlabel("Tiempo")
    plt.ylabel("Y")
    plt.title("Solución usando el Método de Euler")
    plt.legend()
    plt.grid(True)
    plt.show()

plot_euler_solution(ts, ys)
```



4. Aquí se dan las soluciones reales para los problemas de valor inicial en el ejercicio 3. Calcule el error real en las aproximaciones del ejercicio 3.

a. $y(t) = \frac{t}{1+\ln t}$

b. $y(t) = t\tan(\ln t)$

c. $y(t) = -3 + \frac{2}{1+e^{-2t}}$

d. $y(t) = t^2 + \frac{1}{3}e^{-5t}$

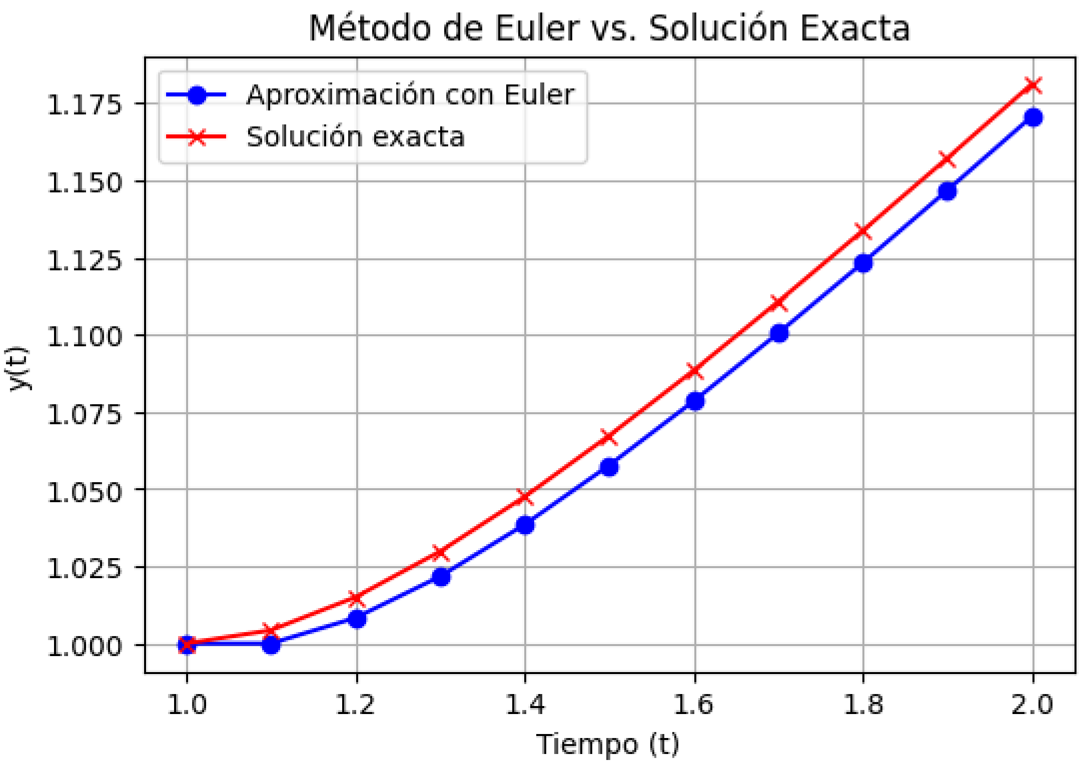
EJERCICIO A)

In [31]:
```
def euler_method(f, y0, a, b, N):
    h = (b - a) / N
    t_values = [round(a + i * h, 4) for i in range(N + 1)]
    y_values = [y0]

    for i in range(N):
        y_next = y_values[-1] + h * f(t_values[i], y_values[-1])
        y_values.append(round(y_next, 4))

    return y_values, t_values, h

f_a = lambda t, y: y / t - (y / t) ** 2
f_ex_a = lambda t: t / (1 + log(t))

a, b, y0, step_size = 1, 2, 1, 0.1
N = int((b - a) / step_size)

ys, ts, h = euler_method(f_a, y0, a, b, N)
ys_exact = [round(f_ex_a(t), 4) for t in ts]
errors = [round(abs(ys_exact[i] - ys[i]), 4) for i in range(len(ts))]

print("Aproximación:", ys)
print("Exacta:", ys_exact)
print("Errores en cada paso:", errors)
```

```
Aproximación: [1, 1.0, 1.0083, 1.0217, 1.0385, 1.0577, 1.0785, 1.1005, 1.1233,
1.1468, 1.1707]
Exacta: [1.0, 1.0043, 1.015, 1.0298, 1.0475, 1.0673, 1.0884, 1.1107, 1.1337,
1.1572, 1.1812]
Errores en cada paso: [0.0, 0.0043, 0.0067, 0.0081, 0.009, 0.0096, 0.0099, 0.01
02, 0.0104, 0.0104, 0.0105]
```

Gráfico:

In [32]:
```
def plot_euler_vs_exact(t_values, y_approx, y_exact):
    plt.figure(figsize=(6, 4))
    plt.plot(t_values, y_approx, 'o-', color="blue", label='Aproximación con E
    plt.plot(t_values, y_exact, 'x-', color="red", label='Solución exacta')
    plt.xlabel('Tiempo (t)')
    plt.ylabel('y(t)')
    plt.title("Método de Euler vs. Solución Exacta")
    plt.legend()
    plt.grid(True)
    plt.show()

plot_euler_vs_exact(ts, ys, ys_exact)
```



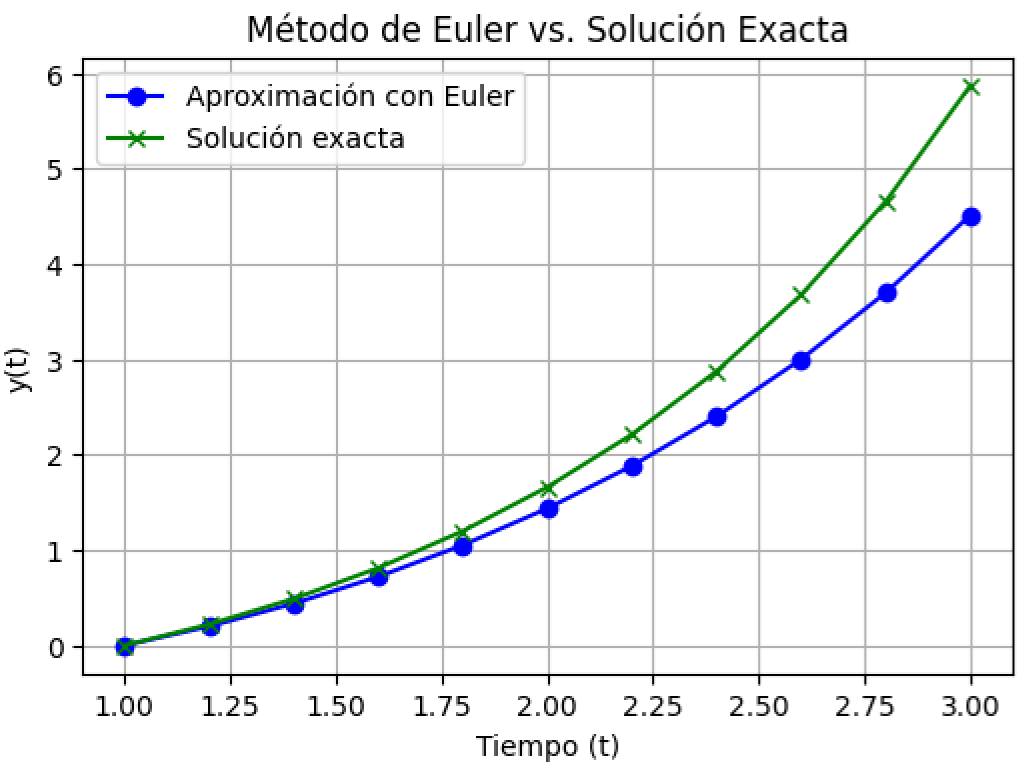
EJERCICIO B)

In [34]:
```
from math import tan
def euler_method(f, y0, a, b, N):
    h = (b - a) / N
    t_values = [round(a + i * h, 4) for i in range(N + 1)]
    y_values = [y0]

    for i in range(N):
        y_next = y_values[-1] + h * f(t_values[i], y_values[-1])
        y_values.append(round(y_next, 4))

    return y_values, t_values, h

f_b = lambda t, y: 1 + y / t + (y / t) ** 2
f_ex_b = lambda t: t * tan(log(t))

a, b, y0, step_size = 1, 3, 0, 0.2
N = int((b - a) / step_size)

ys, ts, h = euler_method(f_b, y0, a, b, N)
ys_exact = [round(f_ex_b(t), 4) for t in ts]
errors = [round(abs(ys_exact[i] - ys[i]), 4) for i in range(len(ts))]

print("Aproximación:", ys)
print("Exacta:", ys_exact)
print("Errores en cada paso:", errors)
```

```
Aproximación: [0, 0.2, 0.4389, 0.7213, 1.0521, 1.4373, 1.8843, 2.4023, 3.0029,
3.7007, 4.5144]
Exacta: [0.0, 0.2212, 0.4897, 0.8128, 1.1994, 1.6613, 2.2135, 2.8766, 3.6785,
4.6587, 5.8741]
Errores en cada paso: [0.0, 0.0212, 0.0508, 0.0915, 0.1473, 0.224, 0.3292, 0.47
43, 0.6756, 0.958, 1.3597]
```

Gráfico:

In [35]:
```
def plot_euler_vs_exact(t_values, y_approx, y_exact):
    plt.figure(figsize=(6, 4))
    plt.plot(t_values, y_approx, 'o-', color="blue", label='Aproximación con E
    plt.plot(t_values, y_exact, 'x-', color="green", label='Solución exacta')
    plt.xlabel('Tiempo (t)')
    plt.ylabel('y(t)')
    plt.title("Método de Euler vs. Solución Exacta")
    plt.legend()
    plt.grid(True)
    plt.show()

plot_euler_vs_exact(ts, ys, ys_exact)
```



EJERCICIO C)

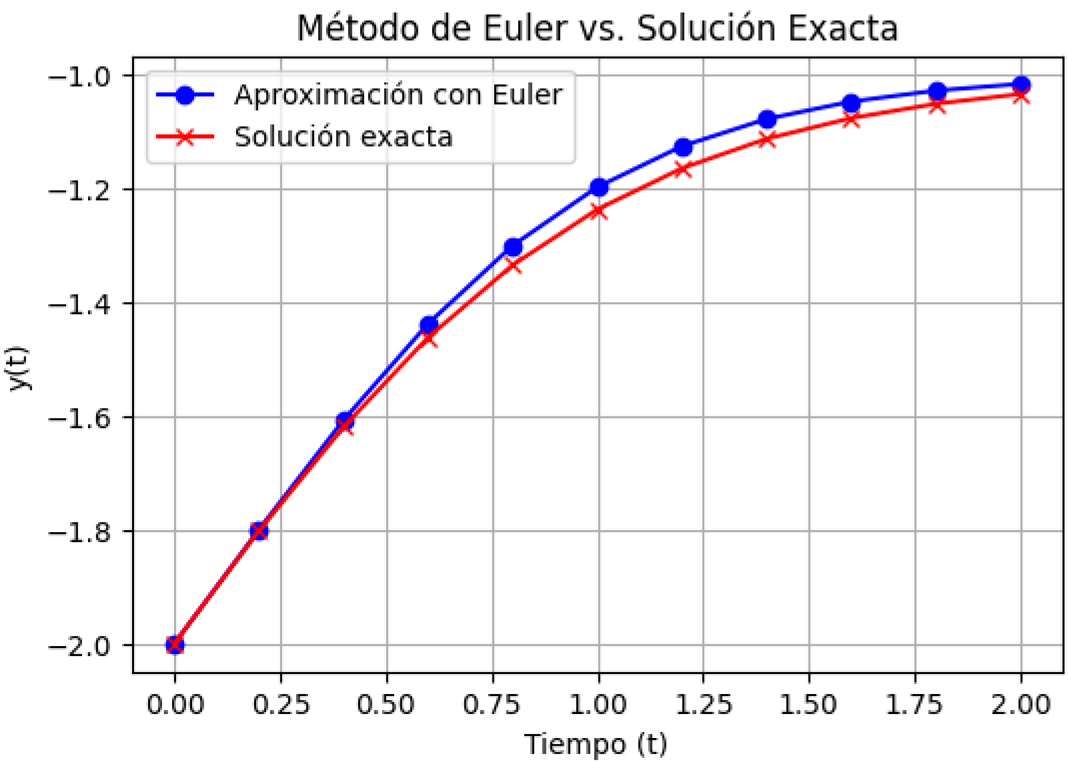
In [36]:
```
from math import exp

def euler_method(f, y0, a, b, N):
    h = (b - a) / N
    t_values = [round(a + i * h, 4) for i in range(N + 1)]
```

```python
    y_values = [y0]

    for i in range(N):
        y_next = y_values[-1] + h * f(t_values[i], y_values[-1])
        y_values.append(round(y_next, 4))

    return y_values, t_values, h

f_c = lambda t, y: -(y + 1) * (y + 3)
f_ex_c = lambda t: -3 + 2 / (1 + exp(-2 * t))

a, b, y0, step_size = 0, 2, -2, 0.2
N = int((b - a) / step_size)

ys, ts, h = euler_method(f_c, y0, a, b, N)
ys_exact = [round(f_ex_c(t), 4) for t in ts]
errors = [round(abs(ys_exact[i] - ys[i]), 4) for i in range(len(ts))]

print("Aproximación:", ys)
print("Exacta:", ys_exact)
print("Errores en cada paso:", errors)
```

```
Aproximación: [-2, -1.8, -1.608, -1.4387, -1.3017, -1.1992, -1.1275, -1.0798,
-1.0492, -1.03, -1.0182]
Exacta: [-2.0, -1.8026, -1.6201, -1.463, -1.336, -1.2384, -1.1663, -1.1146,
-1.0783, -1.0532, -1.036]
Errores en cada paso: [0.0, 0.0026, 0.0121, 0.0243, 0.0343, 0.0392, 0.0388, 0.0
348, 0.0291, 0.0232, 0.0178]
```

Gráfico:

In [37]:
```python
def plot_euler_vs_exact(t_values, y_approx, y_exact):
    plt.figure(figsize=(6, 4))
    plt.plot(t_values, y_approx, 'o-', color="blue", label='Aproximación con E
    plt.plot(t_values, y_exact, 'x-', color="red", label='Solución exacta')
    plt.xlabel('Tiempo (t)')
    plt.ylabel('y(t)')
    plt.title("Método de Euler vs. Solución Exacta")
    plt.legend()
    plt.grid(True)
    plt.show()

plot_euler_vs_exact(ts, ys, ys_exact)
```



EJERCICIO D)

In [38]:

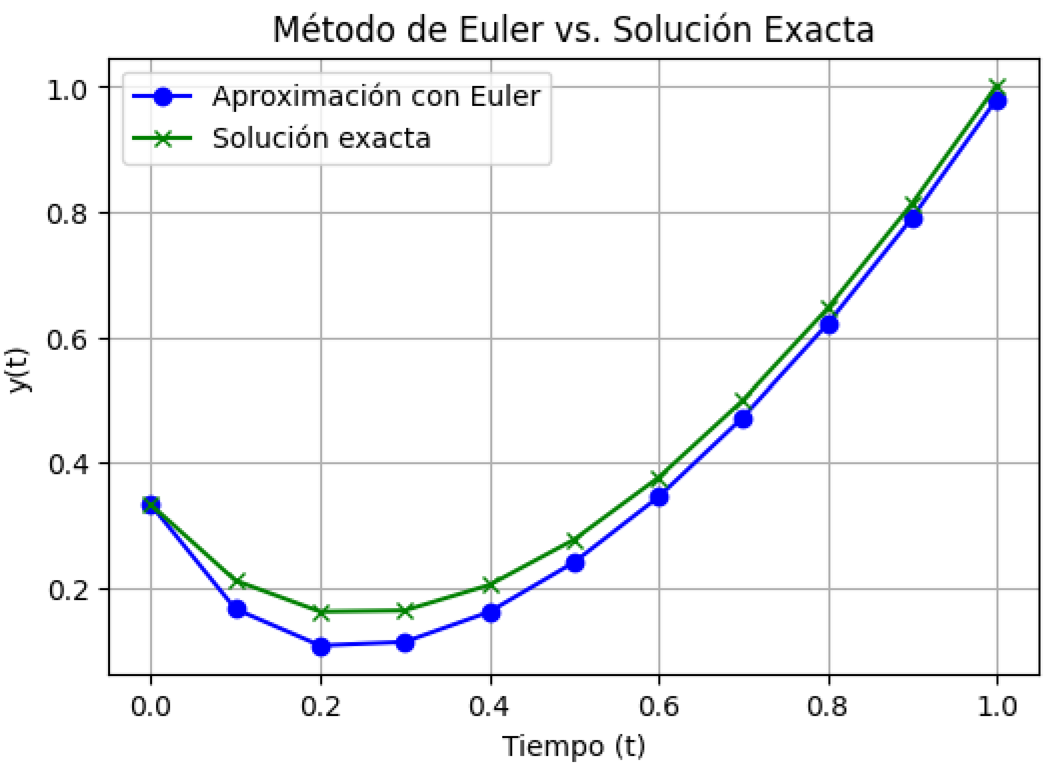
```
def euler_method(f, y0, a, b, N):
    h = (b - a) / N
    t_values = [round(a + i * h, 4) for i in range(N + 1)]
    y_values = [y0]

    for i in range(N):
        y_next = y_values[-1] + h * f(t_values[i], y_values[-1])
        y_values.append(round(y_next, 4))

    return y_values, t_values, h

f_d = lambda t, y: -5 * y + 5 * t**2 + 2 * t
f_ex_d = lambda t: t**2 + (1 / 3) * exp(-5 * t)

a, b, y0, step_size = 0, 1, 1/3, 0.1
N = int((b - a) / step_size)

ys, ts, h = euler_method(f_d, y0, a, b, N)
ys_exact = [round(f_ex_d(t), 4) for t in ts]
errors = [round(abs(ys_exact[i] - ys[i]), 4) for i in range(len(ts))]

print("Aproximación:", ys)
print("Exacta:", ys_exact)
print("Errores en cada paso:", errors)
```

```
Aproximación: [0.3333333333333333, 0.1667, 0.1084, 0.1142, 0.1621, 0.2411, 0.34
56, 0.4728, 0.6214, 0.7907, 0.9804]
Exacta: [0.3333, 0.2122, 0.1626, 0.1644, 0.2051, 0.2774, 0.3766, 0.5001, 0.646
1, 0.8137, 1.0022]
Errores en cada paso: [0.0, 0.0455, 0.0542, 0.0502, 0.043, 0.0363, 0.031, 0.027
3, 0.0247, 0.023, 0.0218]
```

Gráfico:

In [39]:

```python
def plot_euler_vs_exact(t_values, y_approx, y_exact):
    plt.figure(figsize=(6, 4))
    plt.plot(t_values, y_approx, 'o-', color="blue", label='Aproximación con E
    plt.plot(t_values, y_exact, 'x-', color="green", label='Solución exacta')
    plt.xlabel('Tiempo (t)')
    plt.ylabel('y(t)')
    plt.title("Método de Euler vs. Solución Exacta")
    plt.legend()
    plt.grid(True)
    plt.show()

plot_euler_vs_exact(ts, ys, ys_exact)
```



5. Utilice los resultados del ejercicio 3 y la interpolación lineal para aproximar los siguientes valores de �(�). Compare las aproximaciones asignadas para los valores reales obtenidos mediante las funciones determinadas en el ejercicio 4.

a. $y(0.25)$ y $y(0.93)$

b. $y(t) = y(1.25)$ y $y(1.93)$

c. $y(2.10)$ y $y(2.75)$

d. $y(t) = y(0.54)$ y $y(0.94)$

EJERCICIO A)

In [40]:
```
def interpolacion_lineal(y_1, y_2, x_val):

    interpolated_values = []

    for x in x_val:
        idx = int(x * (len(y_1) - 1))  # Aproxima el índice correspondiente en
        if idx >= len(y_1) - 1:
            idx = len(y_1) - 2  # Evita índices fuera del rango

        # Interpolación lineal: y = y1 + (y2 - y1) * (x - x1) / (x2 - x1)
        y_interp = y_1[idx] + (y_2[idx] - y_1[idx]) * (x - idx) / ((idx + 1) -
        interpolated_values.append(round(y_interp, 4))  # Redondeo a 4 decimal

    return interpolated_values

# Datos de entrada
y_1 = [1, 1.0, 1.0083, 1.0217, 1.0385, 1.0577, 1.0785, 1.1004, 1.1233, 1.1467,
y_2 = [1.0, 1.0043, 1.0150, 1.0298, 1.0475, 1.0673, 1.0884, 1.1107, 1.1337, 1.

# Valores donde se desea interpolar
x_val = [0.25, 0.93]

# Cálculo de la interpolación
y_val = interpolacion_lineal(y_1, y_2, x_val)

# Salida de resultados
print(f"Valores interpolados en x = {x_val}: {y_val}")
```

```
Valores interpolados en x = [0.25, 0.93]: [0.9966, 1.062]
```

EJERCICIO B)

In [41]:
```
def interpolacion_lineal(y_1, y_2, x_val):

    interpolated_values = []
    n = len(y_1) - 1  # Número de puntos en la tabla

    for x in x_val:
        idx = int(x)  # Índice entero más cercano a x

        if idx >= n:
            idx = n - 1  # Evita que el índice salga del rango

        # Interpolación lineal: y = y1 + (y2 - y1) * (x - x1) / (x2 - x1)
        y_interp = y_1[idx] + (y_2[idx] - y_1[idx]) * (x - idx) / ((idx + 1) -
        interpolated_values.append(round(y_interp, 4))  # Redondeo a 4 decimal
```

```python
    return interpolated_values

# Datos de entrada
y_1 = [0, 0.2, 0.4389, 0.7212, 1.0520, 1.4373, 1.8843, 2.4023, 3.0028, 3.7006,
y_2 = [0.0, 0.2212, 0.4897, 0.8128, 1.1994, 1.6613, 2.2135, 2.8766, 3.6785, 4.

# Valores donde se desea interpolar
x_val = [1.25, 1.93]

# Cálculo de la interpolación
y_val = interpolacion_lineal(y_1, y_2, x_val)

# Salida de resultados
print(f"Valores interpolados en x = {x_val}: {y_val}")
```

```
Valores interpolados en x = [1.25, 1.93]: [0.2053, 0.2197]
```

EJERCICIO C)

In [42]:

```python
def interpolacion_lineal(y_1, y_2, x_val):

    interpolated_values = []
    n = len(y_1) - 1  # Número de puntos en la tabla

    for x in x_val:
        idx = int(x)  # Índice entero más cercano a x

        if idx >= n:
            idx = n - 1  # Evita que el índice salga del rango

        # Interpolación lineal: y = y1 + (y2 - y1) * (x - x1) / (x2 - x1)
        y_interp = y_1[idx] + (y_2[idx] - y_1[idx]) * (x - idx) / ((idx + 1) -
        interpolated_values.append(round(y_interp, 4))  # Redondeo a 4 decimal

    return interpolated_values

# Datos de entrada
y_1 = [-2, -1.8, -1.608, -1.4387, -1.3017, -1.1993, -1.1275, -1.0797, -1.0491,
y_2 = [-2.0, -1.8026, -1.6201, -1.4629, -1.3359, -1.2384, -1.1663, -1.1146, -1

# Valores donde se desea interpolar
x_val = [2.10, 2.75]

# Cálculo de la interpolación
y_val = interpolacion_lineal(y_1, y_2, x_val)

# Salida de resultados
print(f"Valores interpolados en x = {x_val}: {y_val}")
```

```
Valores interpolados en x = [2.1, 2.75]: [-1.6092, -1.6171]
```

EJERCICIO D)

In [43]:

```python
def interpolacion_lineal(y_1, y_2, x_val):
```

```
    interpolated_values = []
    n = len(y_1) - 1  # Número de puntos en la tabla

    for x in x_val:
        idx = int(x * n)  # Escalar x al índice correspondiente en la lista

        if idx >= n:
            idx = n - 1  # Evita que el índice salga del rango

        # Interpolación lineal: y = y1 + (y2 - y1) * (x - x1) / (x2 - x1)
        y_interp = y_1[idx] + (y_2[idx] - y_1[idx]) * (x - idx / n) / (1 / n)
        interpolated_values.append(round(y_interp, 4))  # Redondeo a 4 decimal

    return interpolated_values

# Datos de entrada
y_1 = [0.3333, 0.1667, 0.1083, 0.1142, 0.1621, 0.2410, 0.3455, 0.4728, 0.6214,
y_2 = [0.3333, 0.2122, 0.1626, 0.1644, 0.2051, 0.2774, 0.3766, 0.5001, 0.6461,

# Valores donde se desea interpolar
x_val = [0.54, 0.94]

# Cálculo de la interpolación
y_val = interpolacion_lineal(y_1, y_2, x_val)

# Salida de resultados
print(f"Valores interpolados en x = {x_val}: {y_val}")
```

```
Valores interpolados en x = [0.54, 0.94]: [0.2556, 0.7999]
```

6. Use el método de Taylor de orden 2 para aproximar las soluciones para cada uno de los siguientes problemas de valor inicial.

a. $y' = te^{3t} - 2y, 0 \leq t \leq 1, y(0) = 0$, con $h = 0.5$

b. $y' = 1 + (t - y)^2, 2 \leq t \leq 3, y(2) = 1$, con $h = 0.5$

c. $y' = 1 + {}^{y}/_{t}\,, 1 \leq t \leq 2, y(1) = 2$, con $h = 0.25$

d. $y' = \cos 2t + \operatorname{sen} 3t\,, 0 \leq t \leq 1, y(0) = 1$, con $h = 0.25$

EJERCICIOS

In [44]:
```
import numpy as np
import pandas as pd

def taylor_order2(f, df, a, b, y0, h):
    """ Método de Taylor de orden 2 para resolver ED ordinarias """
    N = int((b - a) / h)  # Número de pasos
    t_values = np.arange(a, b + h, h)
    y_values = np.zeros(len(t_values))
```

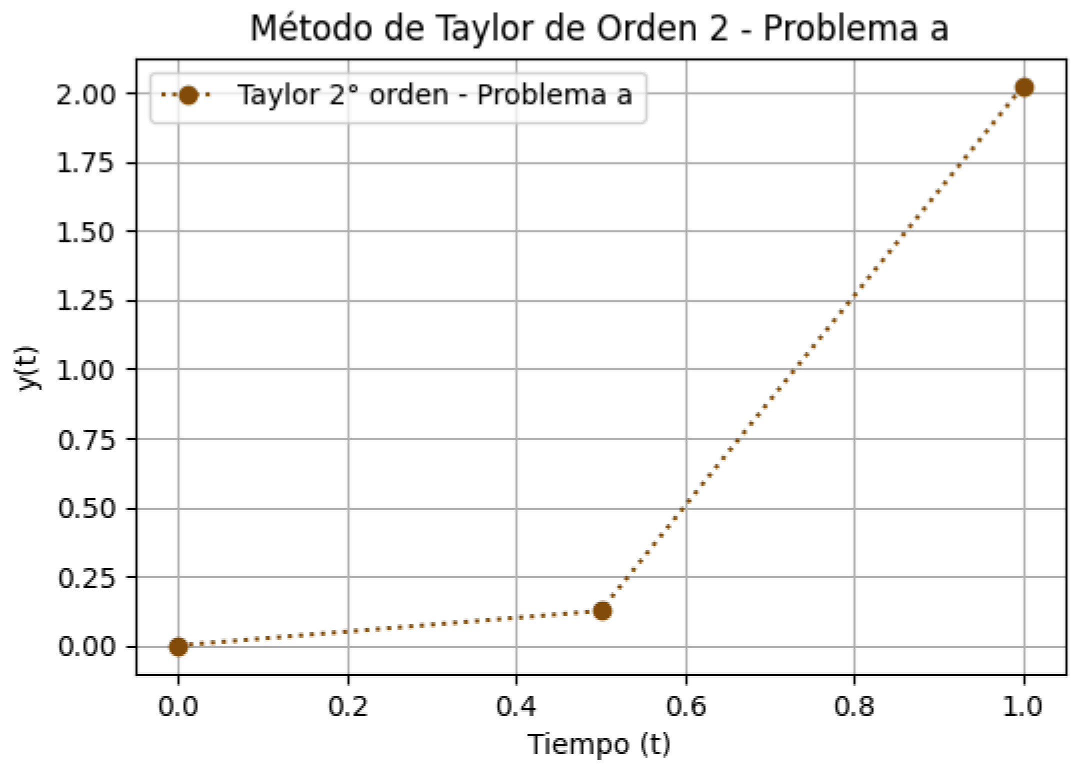
```python
    y_values[0] = y0

    for i in range(N):
        t, y = t_values[i], y_values[i]
        y_values[i + 1] = y + h * f(t, y) + (h**2 / 2) * df(t, y)

    return t_values, y_values

# Definición de las ecuaciones diferenciales y sus derivadas
problems = {
    "a": {
        "f": lambda t, y: t * np.exp(3*t) - 2*y,
        "df": lambda t, y: np.exp(3*t) * (1 + 3*t) - 2 * (t * np.exp(3*t) - 2*
        "a": 0, "b": 1, "y0": 0, "h": 0.5
    },
    "b": {
        "f": lambda t, y: 1 + (t - y)**2,
        "df": lambda t, y: 2 * (t - y) * (1 - (1 + (t - y)**2)),
        "a": 2, "b": 3, "y0": 1, "h": 0.5
    },
    "c": {
        "f": lambda t, y: 1 + y/t,
        "df": lambda t, y: (-y / t**2) + (1 + y/t) / t,
        "a": 1, "b": 2, "y0": 2, "h": 0.25
    },
    "d": {
        "f": lambda t, y: np.cos(2*t) + np.sin(3*t),
        "df": lambda t, y: -2*np.sin(2*t) + 3*np.cos(3*t),
        "a": 0, "b": 1, "y0": 1, "h": 0.25
    }
}

# Cálculo de las soluciones aproximadas
results = {}
for key, params in problems.items():
    t_values, y_values = taylor_order2(
        params["f"], params["df"], params["a"], params["b"], params["y0"], par
    )
    results[key] = pd.DataFrame({"t": t_values, "y_aprox": np.round(y_values,

# Imprimir resultados en la consola
for key, df in results.items():
    print(f"\nResultados para el problema {key}:")
    print(df.to_string(index=False))
```

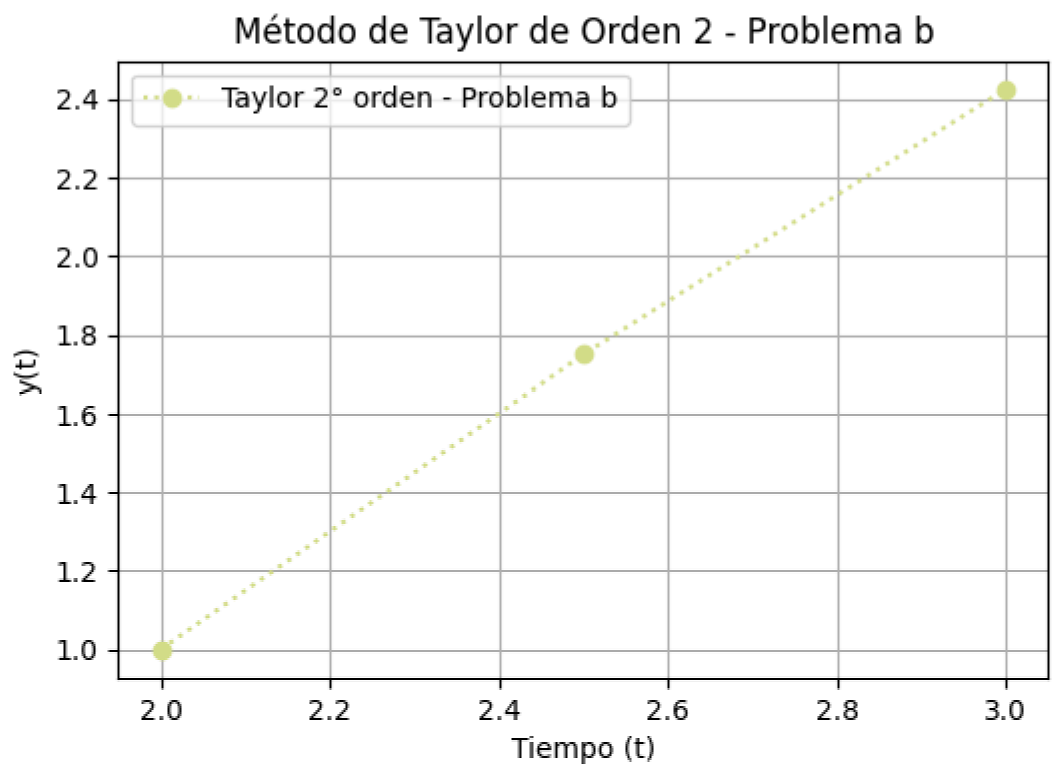
```
Resultados para el problema a:
     t  y_aprox
0.0   0.0000
0.5   0.1250
1.0   2.0232

Resultados para el problema b:
     t  y_aprox
2.0   1.0000
2.5   1.7500
3.0   2.4258

Resultados para el problema c:
      t  y_aprox
1.00   2.0000
1.25   2.7812
1.50   3.6125
1.75   4.4854
2.00   5.3940

Resultados para el problema d:
      t  y_aprox
0.00   1.0000
0.25   1.3438
0.50   1.7722
0.75   2.1107
1.00   2.2016
```

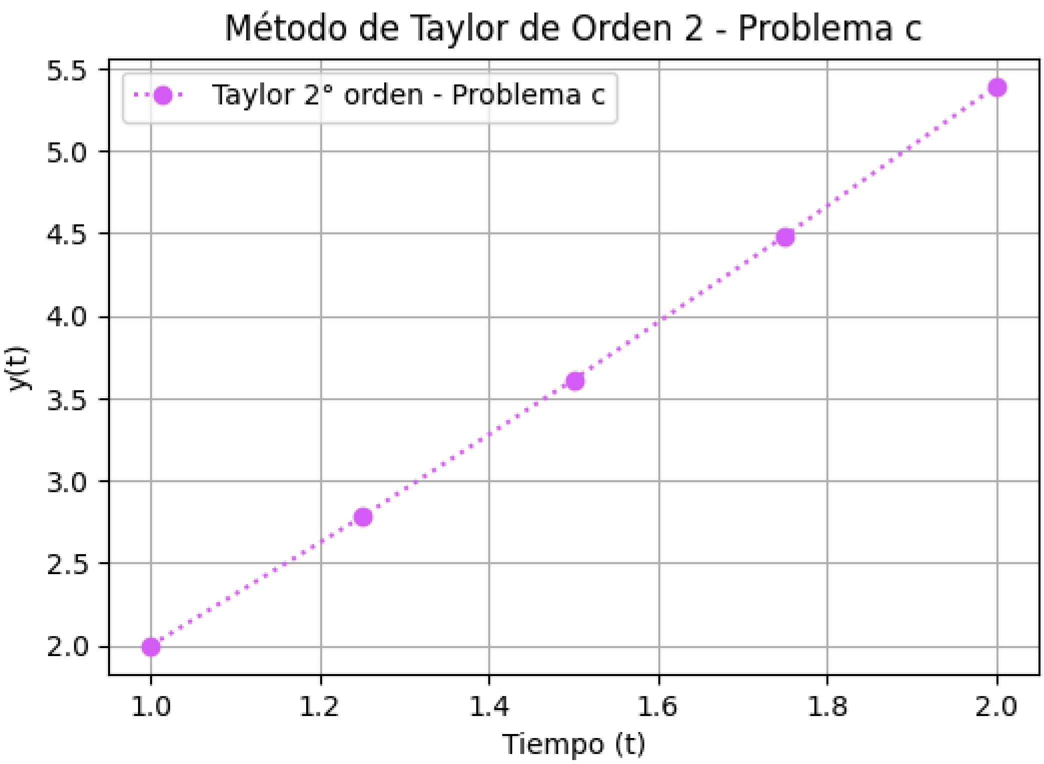
Gráfico de cada Ejercicio:

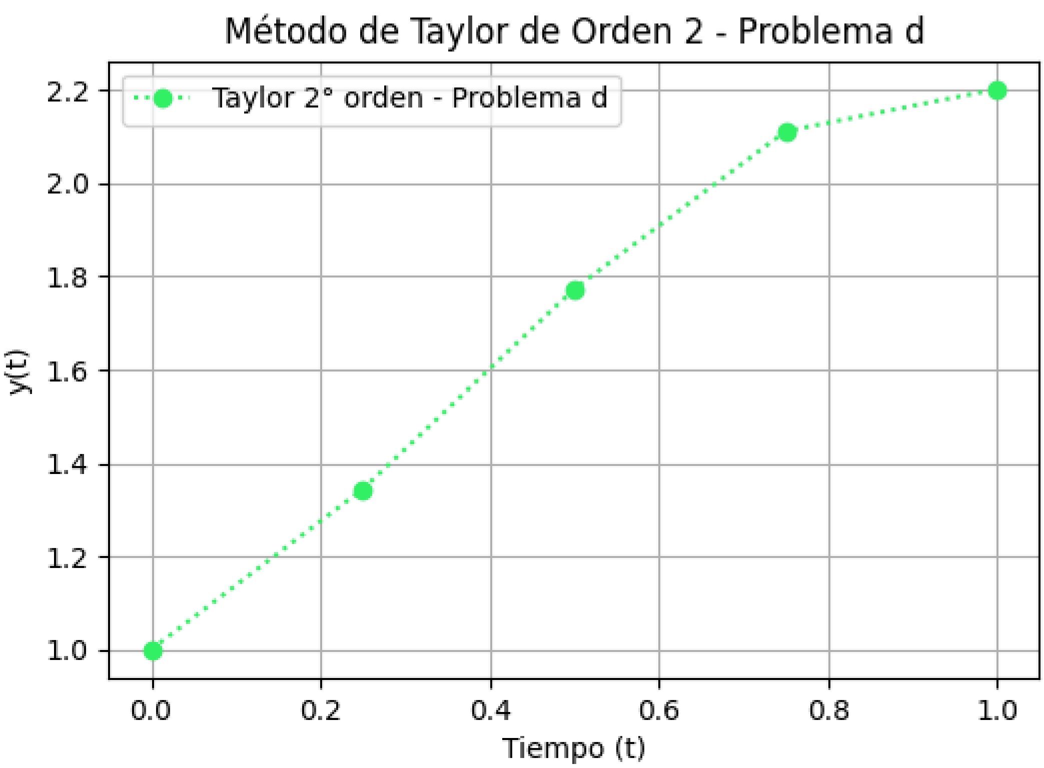
In [45]:

```python
# Graficar los resultados de cada problema
for key, df in results.items():
    plt.figure(figsize=(6, 4))
    plt.plot(df["t"], df["y_aprox"], marker="o", linestyle=":", label=f"Taylor
    plt.xlabel("Tiempo (t)")
    plt.ylabel("y(t)")
    plt.title(f"Método de Taylor de Orden 2 - Problema {key}")
    plt.legend()
    plt.grid(True)
    plt.show()
```


Método de Taylor de Orden 2 - Problema a
Taylor 2° orden - Problema a
y(t)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0
Tiempo (t)


Método de Taylor de Orden 2 - Problema b
Taylor 2° orden - Problema b
y(t)
2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
2.0
2.2
2.4
2.6
2.8
3.0
Tiempo (t)





7. Repita el ejercicio 6 con el método de Taylor de orden 4

a. $y' = te^{3t} - 2y, 0 \leq t \leq 1, y(0) = 0,$ con $h = 0.5$
b. $y' = 1 + (t - y)^2, 2 \leq t \leq 3, y(2) = 1,$ con $h = 0.5$
c. $y' = 1 + {}^{y}/_{t}, 1 \leq t \leq 2, y(1) = 2,$ con $h = 0.25$
d. $y' = \cos 2t + \operatorname{sen} 3t, 0 \leq t \leq 1, y(0) = 1,$ con $h = 0.25$

EJERCICIOS

In [46]:
```
def taylor_order4(f, df1, df2, df3, a, b, y0, h):
    """ Método de Taylor de orden 4 para resolver ED ordinarias """
    N = int((b - a) / h)  # Número de pasos
    t_values = np.arange(a, b + h, h)
    y_values = np.zeros(len(t_values))
    y_values[0] = y0

    for i in range(N):
        t, y = t_values[i], y_values[i]
        y_values[i + 1] = y + h * f(t, y) + (h**2 / 2) * df1(t, y) + (h**3 / 6

    return t_values, y_values

# Definición de las ecuaciones diferenciales y sus derivadas
problems_taylor4 = {
    "a": {
        "f": lambda t, y: t * np.exp(3*t) - 2*y,
        "df1": lambda t, y: np.exp(3*t) * (1 + 3*t) - 2 * (t * np.exp(3*t) - 2
        "df2": lambda t, y: 3 * np.exp(3*t) * (3*t + 2) - 2 * (np.exp(3*t) * (
        "df3": lambda t, y: 9 * np.exp(3*t) * (3*t + 3) - 2 * (3 * np.exp(3*t)
        "a": 0, "b": 1, "y0": 0, "h": 0.5
    },
    "b": {
        "f": lambda t, y: 1 + (t - y)**2,
        "df1": lambda t, y: 2 * (t - y) * (1 - (1 + (t - y)**2)),
        "df2": lambda t, y: -2 * (1 - (1 + (t - y)**2))**2 - 2 * (t - y) * 2 *
        "df3": lambda t, y: -6 * (1 - (1 + (t - y)**2))**3 - 12 * (t - y) * (1
        "a": 2, "b": 3, "y0": 1, "h": 0.5
    },
    "c": {
        "f": lambda t, y: 1 + y/t,
        "df1": lambda t, y: (-y / t**2) + (1 + y/t) / t,
        "df2": lambda t, y: (2 * y / t**3) - (1 + y/t) / t**2 + (-y / t**2 + (
        "df3": lambda t, y: (-6 * y / t**4) + (3 * (1 + y/t) / t**3) - (2 * y
        "a": 1, "b": 2, "y0": 2, "h": 0.25
    },
    "d": {
        "f": lambda t, y: np.cos(2*t) + np.sin(3*t),
        "df1": lambda t, y: -2*np.sin(2*t) + 3*np.cos(3*t),
        "df2": lambda t, y: -4*np.cos(2*t) - 9*np.sin(3*t),
        "df3": lambda t, y: 8*np.sin(2*t) - 27*np.cos(3*t),
        "a": 0, "b": 1, "y0": 1, "h": 0.25
```

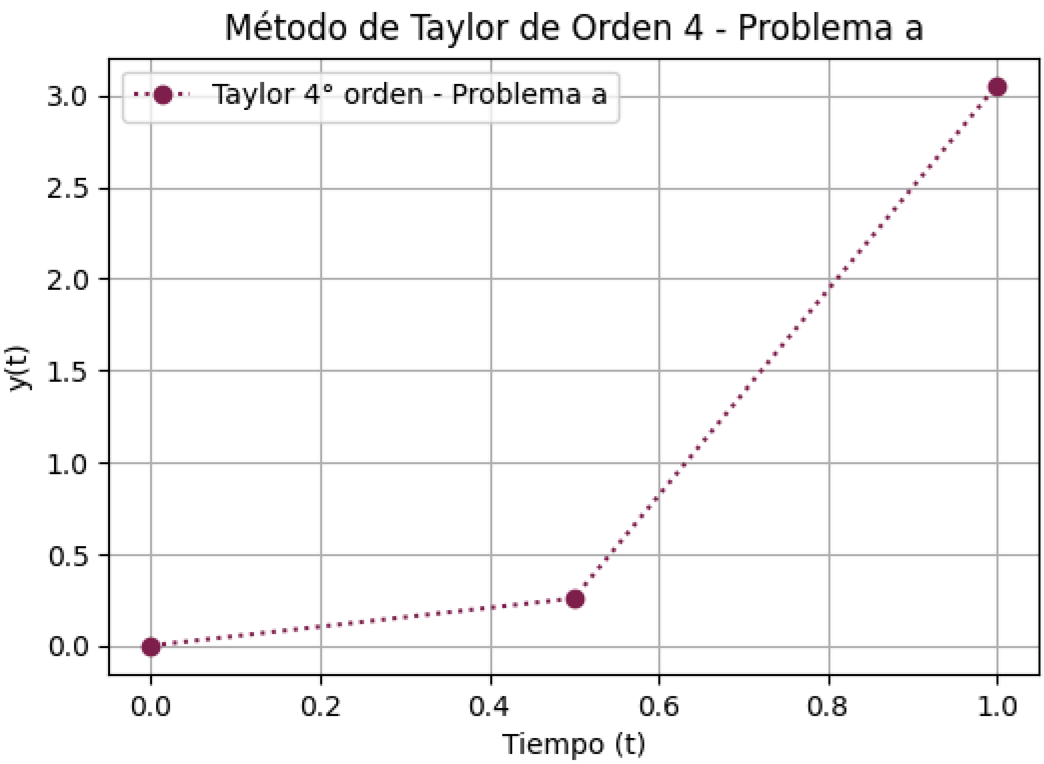
```
    }
}

# Cálculo de las soluciones aproximadas con Taylor de orden 4
results_taylor4 = {}
for key, params in problems_taylor4.items():
    t_values, y_values = taylor_order4(
        params["f"], params["df1"], params["df2"], params["df3"],
        params["a"], params["b"], params["y0"], params["h"]
    )
    results_taylor4[key] = pd.DataFrame({"t": t_values, "y_aprox": np.round(y_

# Imprimir resultados en la consola
for key, df in results_taylor4.items():
    print(f"\nResultados para el problema {key} - Método de Taylor de Orden 4:
    print(df.to_string(index=False))
```

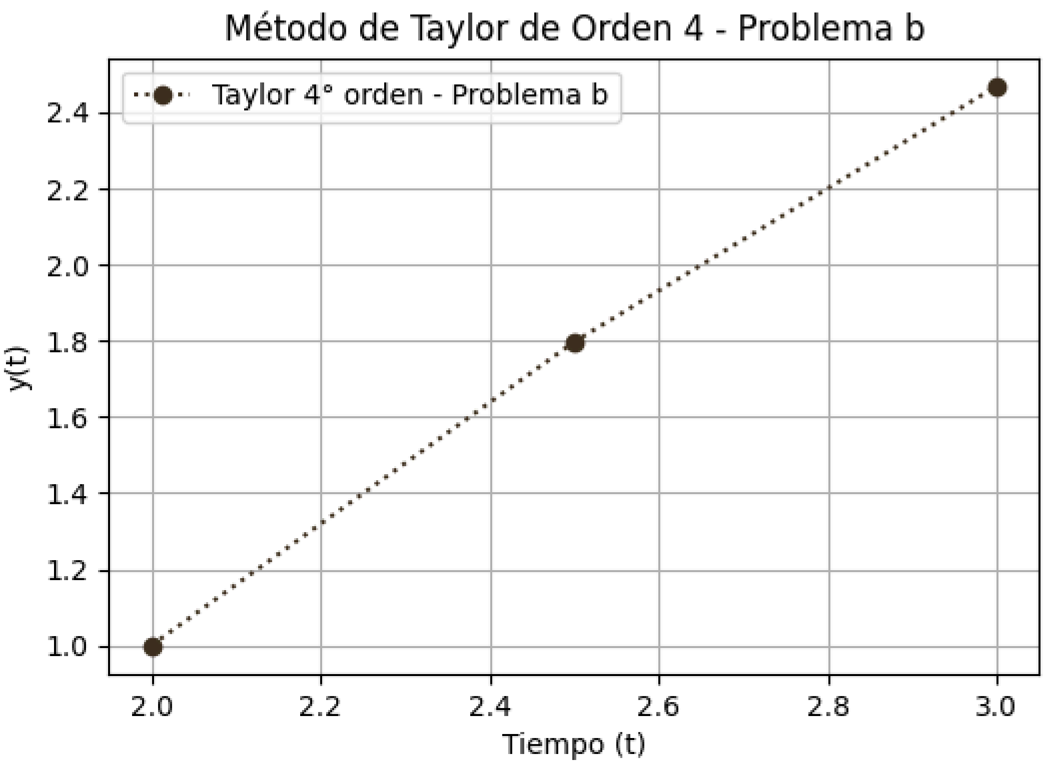
```
Resultados para el problema a - Método de Taylor de Orden 4:
  t  y_aprox
0.0   0.0000
0.5   0.2578
1.0   3.0553

Resultados para el problema b - Método de Taylor de Orden 4:
  t  y_aprox
2.0   1.0000
2.5   1.7969
3.0   2.4690

Resultados para el problema c - Método de Taylor de Orden 4:
   t  y_aprox
1.00   2.0000
1.25   2.7856
1.50   3.6210
1.75   4.4978
2.00   5.4102

Resultados para el problema d - Método de Taylor de Orden 4:
   t  y_aprox
0.00   1.0000
0.25   1.3289
0.50   1.7297
0.75   2.0399
1.00   2.1160
```

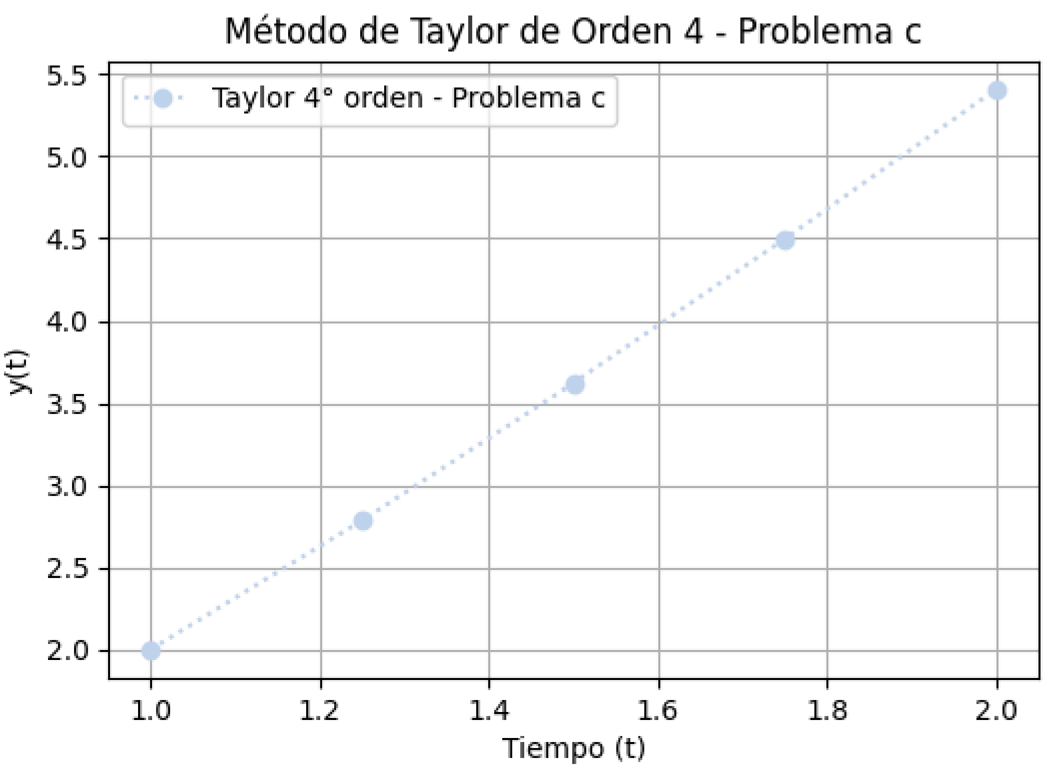
Gráfico de cada Ejercicio:

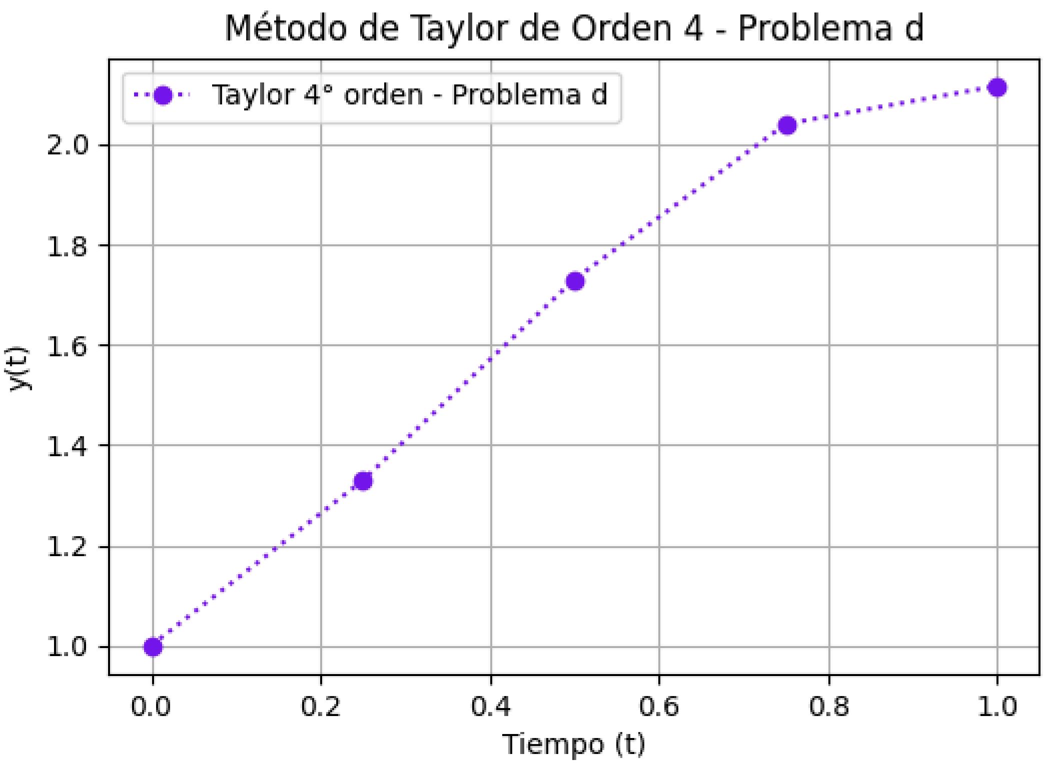
In [47]:
```
def plot_taylor4_solutions(results):
    """
    Genera una gráfica para cada uno de los problemas resueltos con el Método
    """
    for key, df in results.items():
        plt.figure(figsize=(6, 4))
        plt.plot(df["t"], df["y_aprox"], marker="o", linestyle=":", color=np.r
```

```python
                label=f"Taylor 4° orden - Problema {key}")
        plt.xlabel("Tiempo (t)")
        plt.ylabel("y(t)")
        plt.title(f"Método de Taylor de Orden 4 - Problema {key}")
        plt.legend()
        plt.grid(True)
        plt.show()

# Llamar la función para graficar los resultados
plot_taylor4_solutions(results_taylor4)
```




Método de Taylor de Orden 4 - Problema b
Taylor 4° orden - Problema b
y(t)
Tiempo (t)
2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
2.0
2.2
2.4
2.6
2.8
3.0


Método de Taylor de Orden 4 - Problema c
Taylor 4° orden - Problema c
y(t)
Tiempo (t)
5.5
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.0
1.2
1.4
1.6
1.8
2.0


Método de Taylor de Orden 4 - Problema d
Taylor 4° orden - Problema d
y(t)
2.0
1.8
1.6
1.4
1.2
1.0
0.0
0.2
0.4
0.6
0.8
1.0
Tiempo (t)