In [ ]:
```
import warnings
import numpy as np
import matplotlib.pyplot as plt

plt.rc('font',family='Times New Roman')
plt.rcParams['xtick.direction'] = 'in'
plt.rcParams['ytick.direction'] = 'in'
warnings.filterwarnings('ignore')

# %matplotlib inline
%config InlineBackend.figure_format = 'retina'
```

Test the Euler, Improved Euler and Runge-Kutta method on the initial value problem $\dot{x} = -x, x(0) = 1$.

Solve the problem analytically.

We separate the variables and then integrate:

$$\frac{dx}{x} = -dt,$$

which implies

$$\begin{aligned} ln|x| &= -t + C \\ x &= C_1 e^{-t}. \end{aligned}$$

Plug $x(0) = 1$ in. Then $C_1 = 1$. Hence the solution is

$$x = e^{-t}.$$

So the exact value of $x(1)$ is e^{-1}.

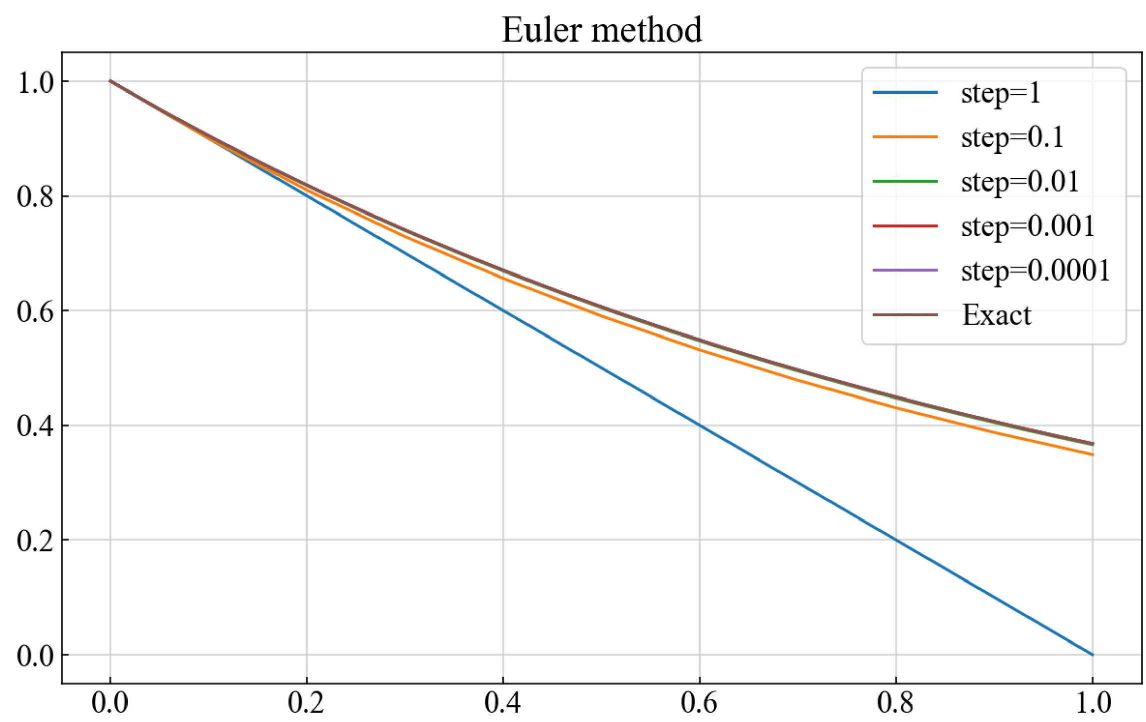
In [ ]:
```
start = 0
end = 1
initial = 1

def exact_function(t: float, step: float):
    return np.exp(-t)

def euler(x: float, step: float):
    return x + step * -x

def t_x(step: float, function: object):
    t = [start]
    x = [initial]
    for iter in np.arange(start + step, end + step, step):
        newX = function(x[-1], step)
        x.append(newX)
        t.append(iter)
    return t, x

def t_x_exact(step: float, function: object):
    t = [start]
    x = [initial]
    for iter in np.arange(start + step, end + step, step):
```

```
        newX = function(iter, step)
        x.append(newX)
        t.append(iter)
    return t, x

def t_error(step: float, algorithm_function: object, exact_function: object):
    t,x = t_x(step, algorithm_function)
    error = [abs(exact_function(t[i], step) - x[i]) for i in range(len(t))]
    return t, error
```

Euler method

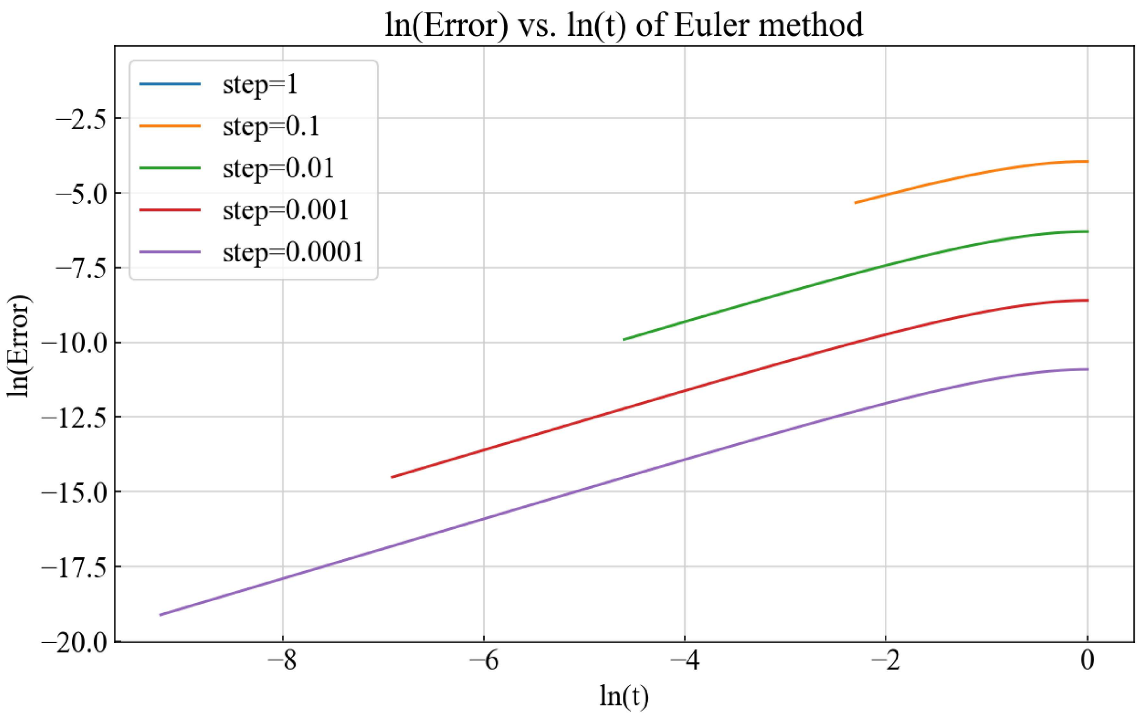
$$x_{n+1} = x_n + f\left(x_n\right)\Delta t$$

In [ ]:
```
plt.rcParams.update({'font.size': 16})
plt.figure(figsize=(10, 6))
plt.title("Euler method")

for step_power in [0, 1, 2, 3, 4]:
    step = pow(10, -step_power)
    t, x = t_x(step, euler)
    plt.plot(t, x, label=f"step={step}")

t, x = t_x_exact(step, exact_function)
plt.plot(t, x, label="Exact")

plt.legend()
plt.grid(color="#D1D1D1")
```



In [ ]:

```python
plt.rcParams.update({'font.size': 16})
plt.figure(figsize=(10, 6))
plt.title("ln(Error) vs. ln(t) of Euler method")

for step_power in [0, 1, 2, 3, 4]:
    step = pow(10, -step_power)
    t, error = t_error(step, euler, exact_function)
    plt.plot(np.log(t), np.log(error), label=f"step={step}")

plt.xlabel("ln(t)")
plt.ylabel("ln(Error)")
plt.legend()
plt.grid(color="#D1D1D1")
plt.show()
```



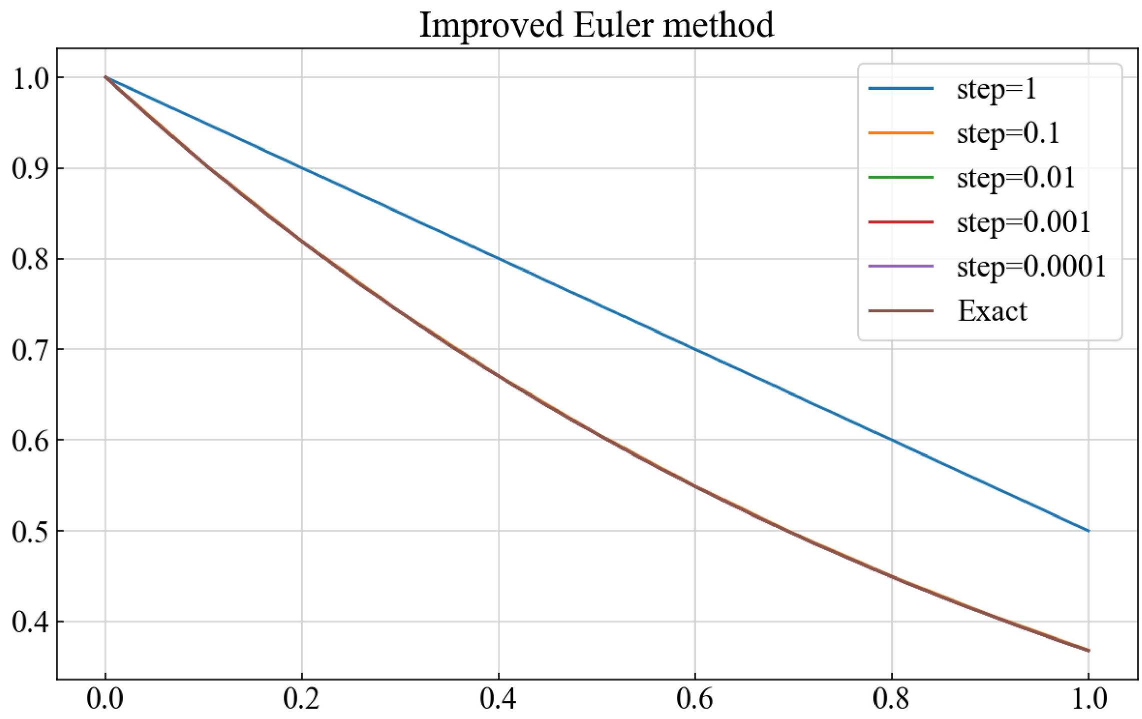
Improved Euler method

$$\tilde{x}_{n+1} = x_n + f\left(x_n\right)\Delta t$$
$$x_{n+1} = x_n + \frac{1}{2}\left[f\left(x_n\right) + f\left(\tilde{x}_{n+1}\right)\right]\Delta t$$

In [ ]:

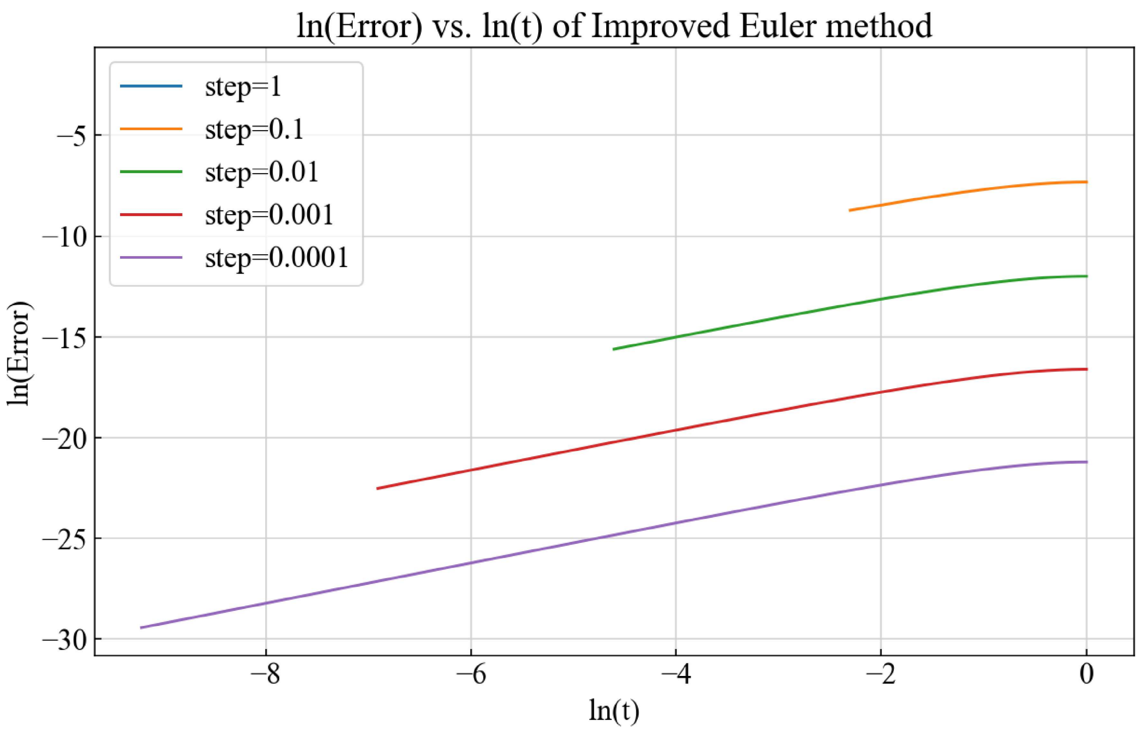
```python
def improved_euler(x: float, step: float):
    k1 = -x
    k2 = -(x + step * k1)
    return x + step / 2 * (k1 + k2)

plt.rcParams.update({'font.size': 16})
plt.figure(figsize=(10, 6))
plt.title("Improved Euler method")
```

```python
for step_power in [0, 1, 2, 3, 4]:
    step = pow(10, -step_power)
    t, x = t_x(step, improved_euler)
    plt.plot(t, x, label=f"step={step}")

t, x = t_x_exact(step, exact_function)
plt.plot(t, x, label="Exact")

plt.legend()
plt.grid(color="#D1D1D1")
```



In [ ]:

```python
plt.rcParams.update({'font.size': 16})
plt.figure(figsize=(10, 6))
plt.title("ln(Error) vs. ln(t) of Improved Euler method")

for step_power in [0, 1, 2, 3, 4]:
    step = pow(10, -step_power)
    t, error = t_error(step, improved_euler, exact_function)
    plt.plot(np.log(t), np.log(error), label=f"step={step}")

plt.xlabel("ln(t)")
plt.ylabel("ln(Error)")
plt.legend()
plt.grid(color="#D1D1D1")
plt.show()
```



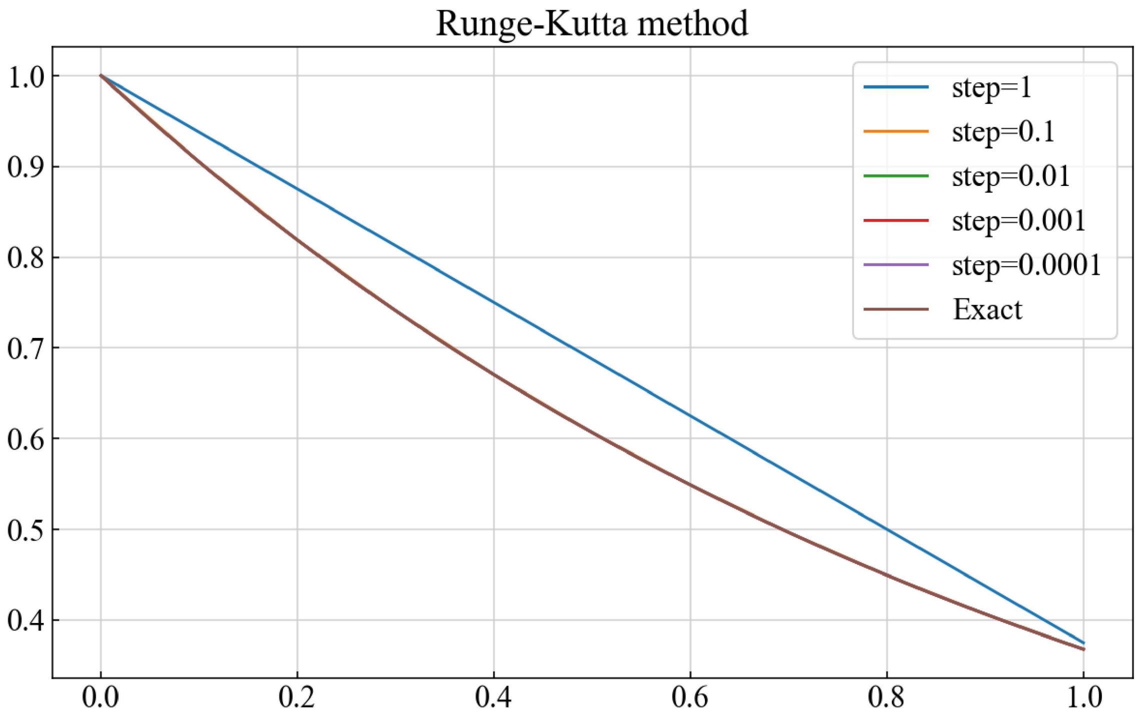
Runge-Kutta method

$$
\begin{aligned}
k_1 &= f\left(x_n\right)\Delta t \\
k_2 &= f\left(x_n + \tfrac{1}{2}k_1\right)\Delta t \\
k_3 &= f\left(x_n + \tfrac{1}{2}k_2\right)\Delta t \\
k_4 &= f\left(x_n + k_3\right)\Delta t
\end{aligned}
$$

$$x_{n+1} = x_n + \tfrac{1}{6}(k_1 + 2k_2 + 3k_3 + k_4)$$

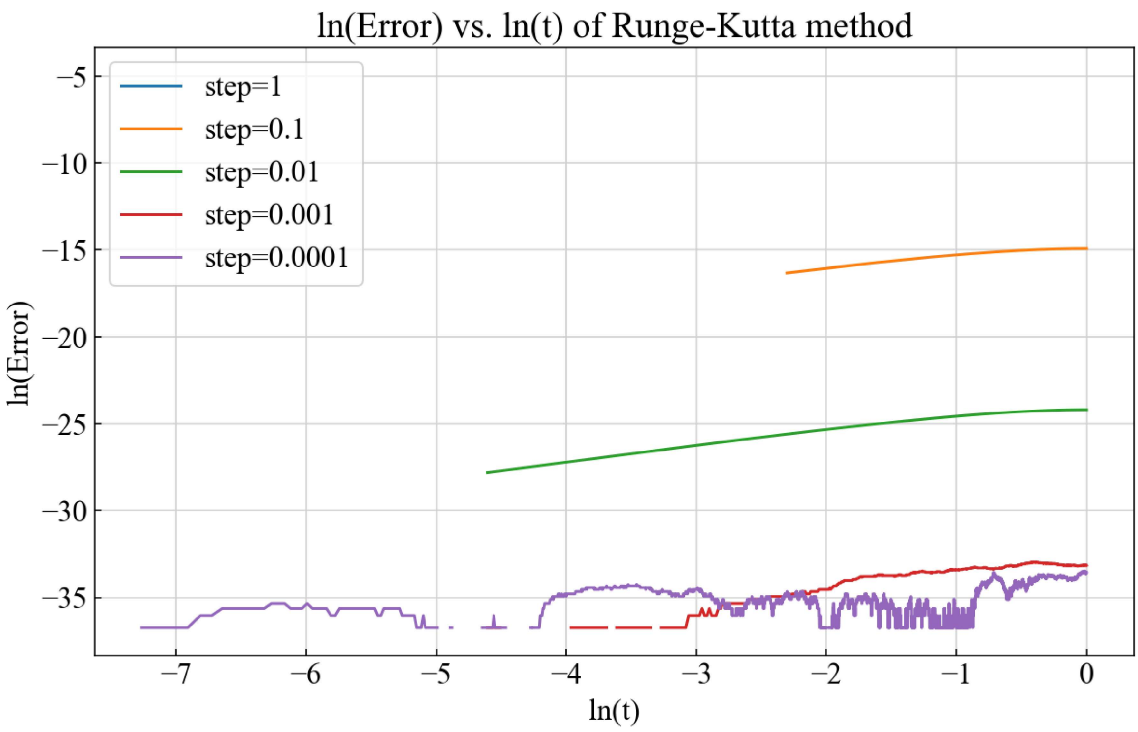
In [ ]:
```
def runge_kutta(x: float, step: float):
    k1 = -x
    k2 = -(x + step / 2 * k1)
    k3 = -(x + step / 2 * k2)
    k4 = -(x + step * k3)
    return x + step / 6 * (k1 + 2 * k2 + 2 * k3 + k4)

plt.rcParams.update({'font.size': 16})
plt.figure(figsize=(10, 6))
plt.title("Runge-Kutta method")

for step_power in [0, 1, 2, 3, 4]:
    step = pow(10, -step_power)
    t, x = t_x(step, runge_kutta)
    plt.plot(t, x, label=f"step={step}")
```

```
t, x = t_x_exact(step, exact_function)
plt.plot(t, x, label="Exact")

plt.legend()
plt.grid(color="#D1D1D1")
```

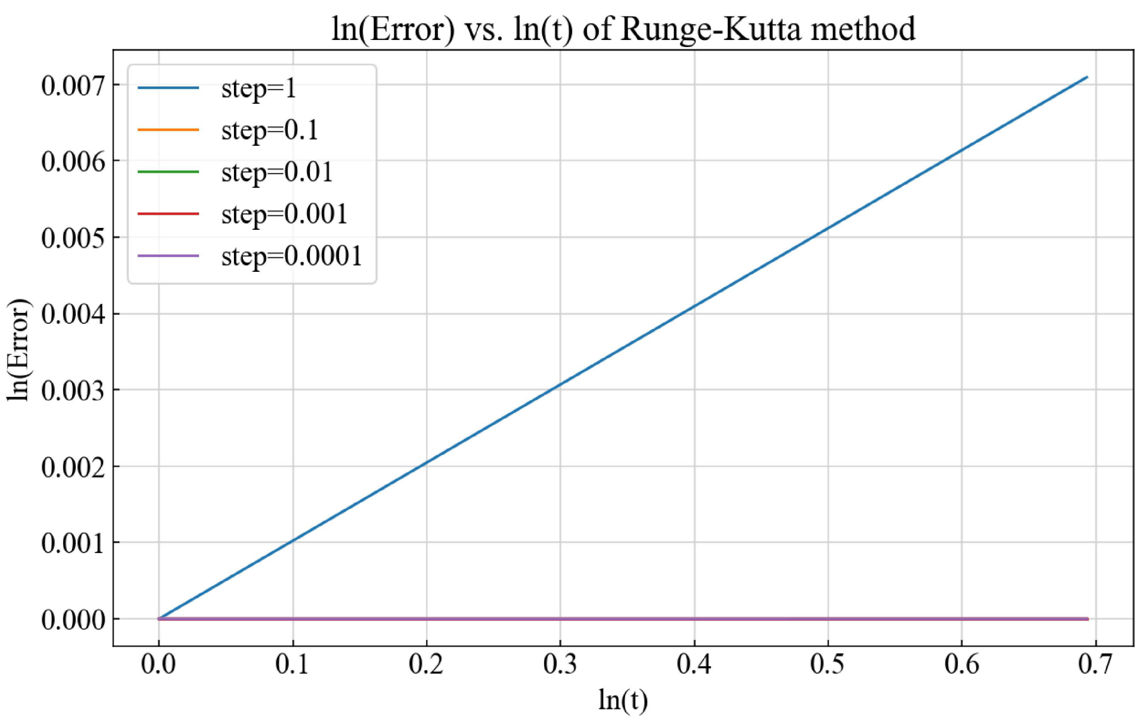


In [ ]:

```
plt.rcParams.update({'font.size': 16})
plt.figure(figsize=(10, 6))
plt.title("ln(Error) vs. ln(t) of Runge-Kutta method")

for step_power in [0, 1, 2, 3, 4]:
    step = pow(10, -step_power)
    t, error = t_error(step, runge_kutta, exact_function)
    plt.plot(np.log(t), np.log(error), label=f"step={step}")

plt.xlabel("ln(t)")
plt.ylabel("ln(Error)")
plt.legend()
plt.grid(color="#D1D1D1")
plt.show()
```



In [ ]:
```python
plt.rcParams.update({'font.size': 16})
plt.figure(figsize=(10, 6))
plt.title("ln(Error) vs. ln(t) of Runge-Kutta method")

for step_power in [0, 1, 2, 3, 4]:
    step = pow(10, -step_power)
    t, error = t_error(step, runge_kutta, exact_function)
    plt.plot(np.log1p(t), np.log1p(error), label=f"step={step}")

plt.xlabel("ln(t)")
plt.ylabel("ln(Error)")
plt.legend()
plt.grid(color="#D1D1D1")
plt.show()
```



In [ ]:
```
(np.diff(np.log1p(error)) < 0).sum()
```

Out[ ]:
```
1340
```

In [ ]:
```
(np.diff(np.log1p(error)) > 0).sum()
```

Out[ ]:
```
1352
```