Demo1: Simple Harmonic Oscillator

In this jupyter notebook, we will learn how to use the Euler's method to solve for the motions of a simple harmonic oscillaotr.

Governing equations

The governing equations are

$$a^{t^n} = -\omega_0^2 x^{t^n},$$

$$x^{t^{n+1}} = x^{t^n} + v^{t^n} \times \Delta t,$$

and

$$v^{t^{n+1}} = v^{t^n} + a^{t^n} \times \Delta t.$$

Initial Conditions

At time $t=0$, position $x=1$ and velocity $v=0$.\ Set $A=k=m=1$, and $\omega_0 = \sqrt{k/m} = 1$ as well.

Exercie 1:

Use a small time step $\Delta t = 0.01$ and solve for the solution at $t=20$.

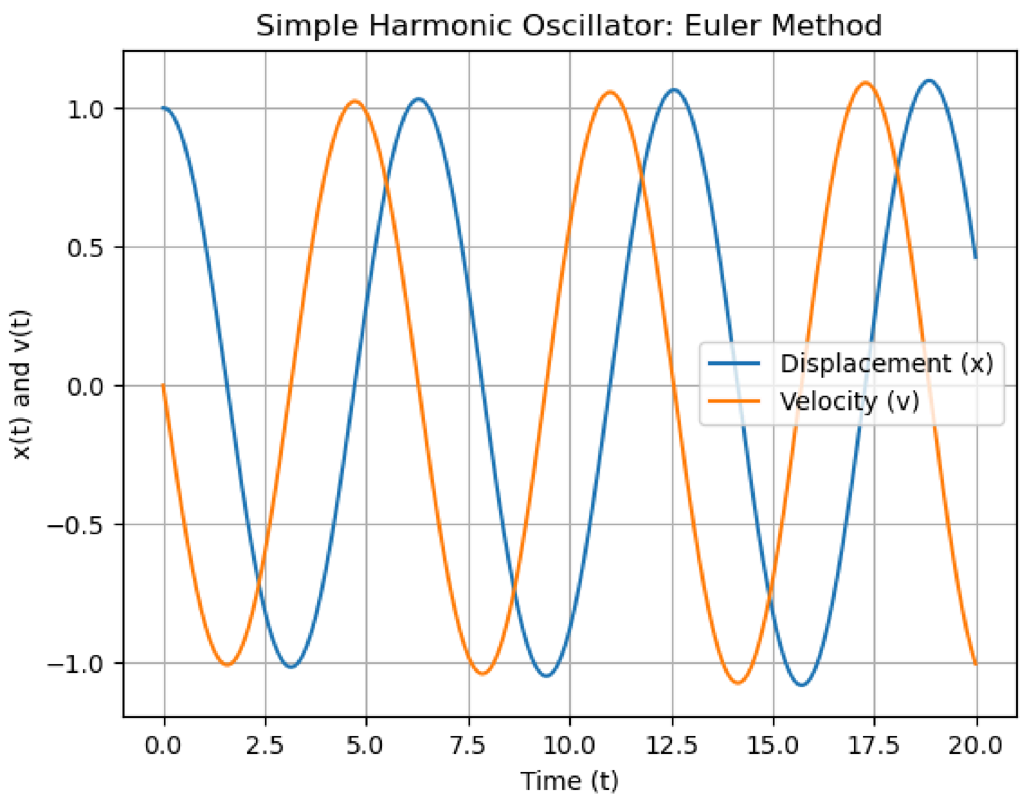
In [2]:
```
# import required libraries
import numpy as np
import matplotlib.pyplot as plt
```

In [9]:
```
#
# This is a simple example of how to solve a simple harmonic oscillator using the Euler method
#

# Step 1: set up the parameters of the problem
A=1
k=1
m=1
omega0 = np.sqrt(k/m)
dt = 0.01
# t range
t_max = 20

def sho_euler(A, k, m, dt, t_max):
# Step 2: set up the time and solution arrays
    t = np.arange(0, t_max, dt)  # Time array
    x = np.zeros_like(t)         # Displacement array
    v = np.zeros_like(t)         # Velocity array

    # Step 3: set up the initial conditions
    x[0] = A       # Initial displacement
    v[0] = 0       # Initial velocity

    # Step 4: solve the difference equation using the Euler method
    for i in range(1, len(t)):
        # Update position and velocity using Euler's method
        v[i] = v[i-1] - (k/m) * x[i-1] * dt  # Velocity update
        x[i] = x[i-1] + v[i-1] * dt          # Position update

    return t, x, v
```

```python
t, x, v = sho_euler(A, k, m, dt, t_max)
```

In [10]:

```python
# Step 5: plot the solution
#TODO
plt.plot(t, x, label="Displacement (x)")
plt.plot(t, v, label="Velocity (v)")
plt.xlabel("Time (t)")
plt.ylabel("x(t) and v(t)")
plt.title("Simple Harmonic Oscillator: Euler Method")
plt.legend()
plt.grid(True)
plt.show()
```



We could verify our numerical solution be comparing it with the analytical solutions. The analytical solutions are:

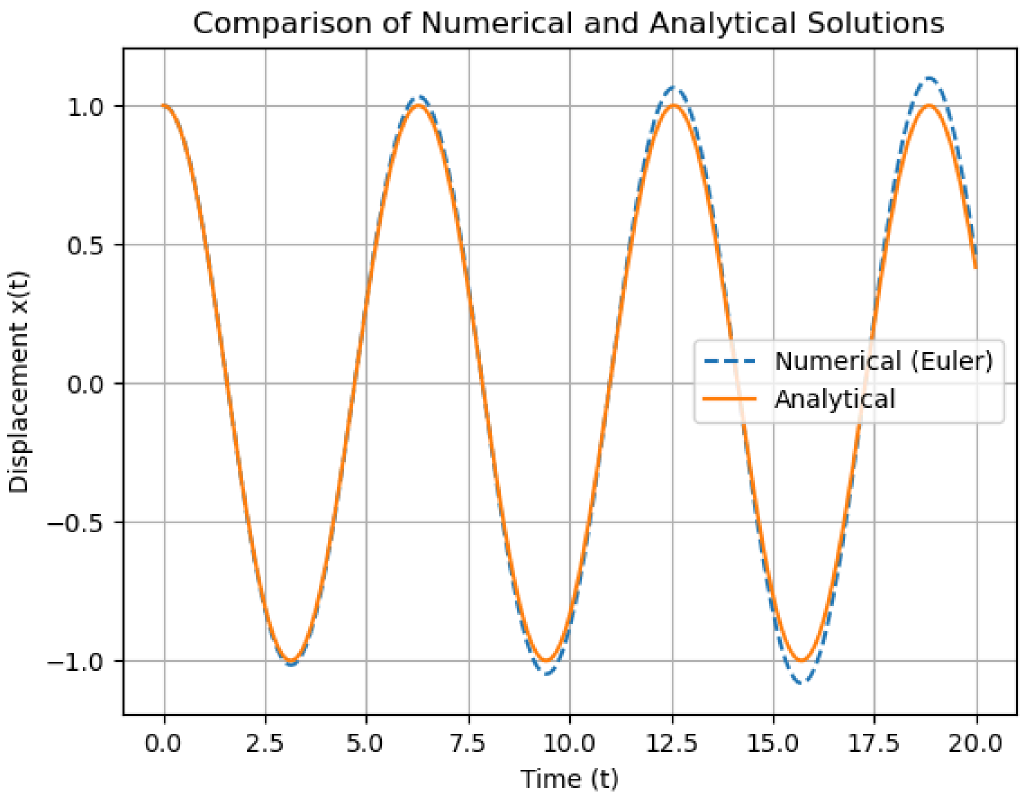
$$x = A\cos(\omega_0 t + \phi),$$

and

$$ v = -A \omega_0 \sin(\omega_0 t + \delta).

In [11]:

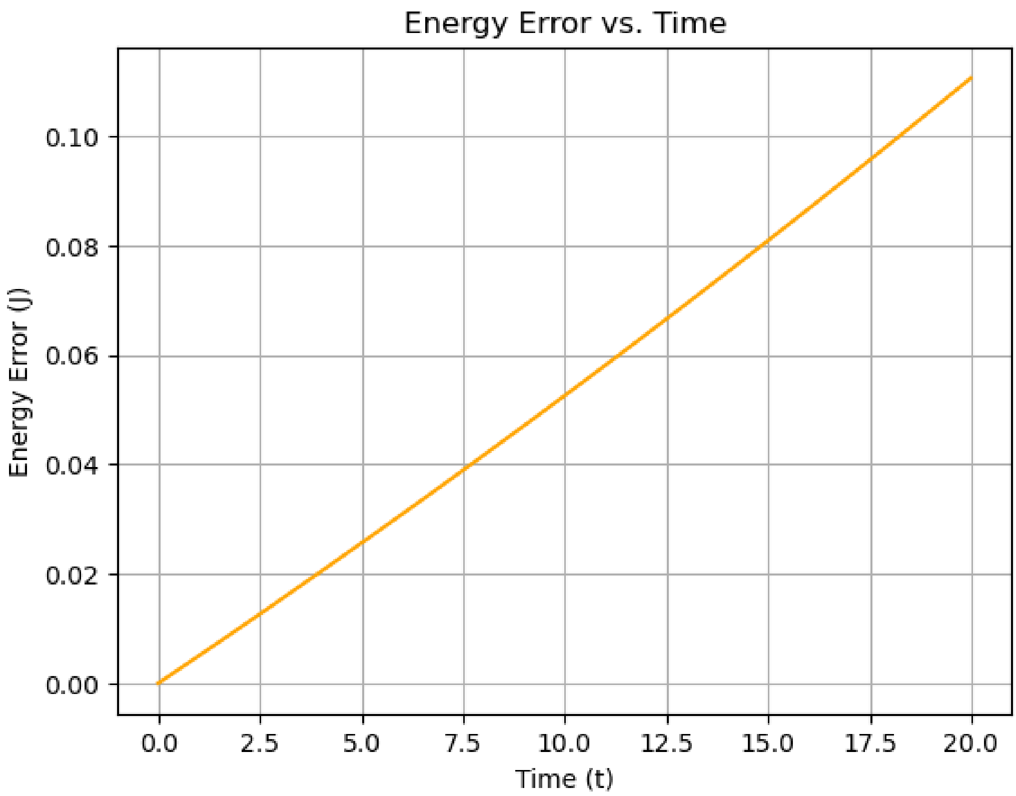
```python
# Step 6: evaluate the analytical solution and plot it

# TODO
# Analytical solution for displacement
x_analytical = A * np.cos(omega0 * t)

# Plot numerical and analytical solutions
plt.plot(t, x, label="Numerical (Euler)", linestyle='--')
plt.plot(t, x_analytical, label="Analytical", linestyle='-')
plt.xlabel("Time (t)")
plt.ylabel("Displacement x(t)")
plt.title("Comparison of Numerical and Analytical Solutions")
plt.legend()
plt.grid(True)

plt.show()
```



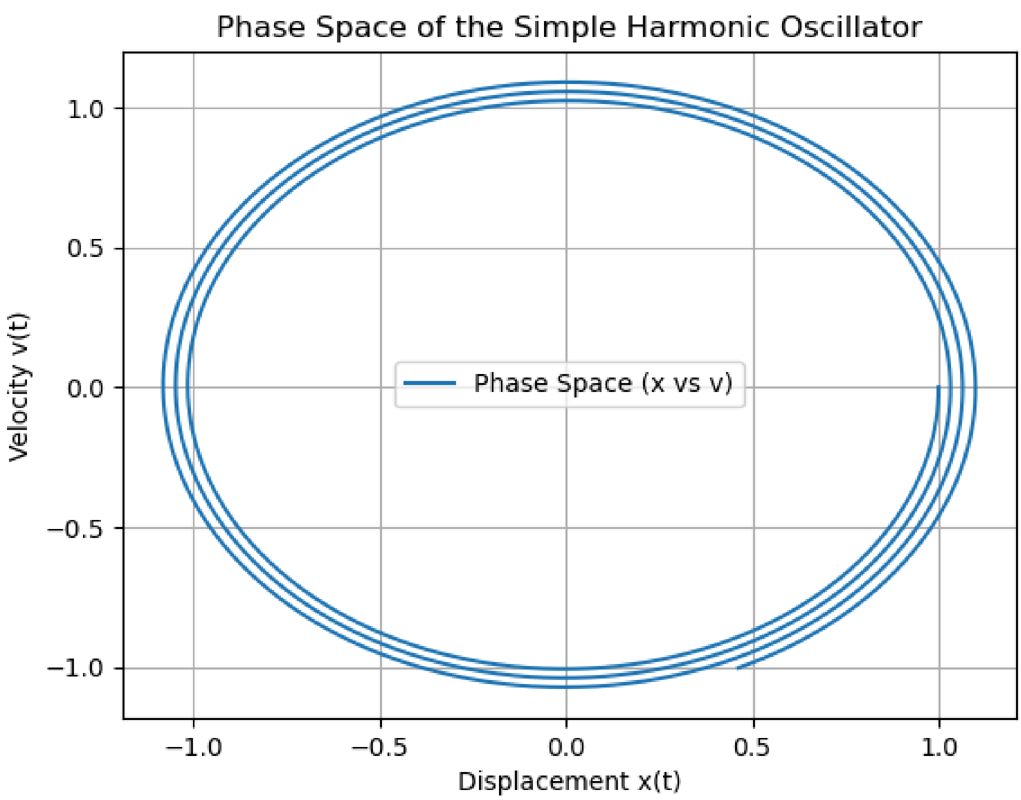
Another way to chekc the accuray of our numerical solution is to check the energy conservation and the phase-sapce diagram.

In [12]:

```
# Step 7: evaluate the energy (error) of the system

# TODO
# Calculate Kinetic Energy (KE) and Potential Energy (PE)
KE = 0.5 * m * v**2      # Kinetic energy
PE = 0.5 * k * x**2      # Potential energy

# Total energy
E_total = KE + PE

# Theoretical total energy
E_theoretical = 0.5 * k * A**2  # Initial energy (when x = A and v = 0)

# Evaluate the error (difference between total energy and theoretical energy)
energy_error = E_total - E_theoretical

# Plot the energy error over time
plt.plot(t, energy_error, label="Energy Error", color='orange')
plt.xlabel("Time (t)")
plt.ylabel("Energy Error (J)")
plt.title("Energy Error vs. Time")
plt.grid(True)

plt.show()
```



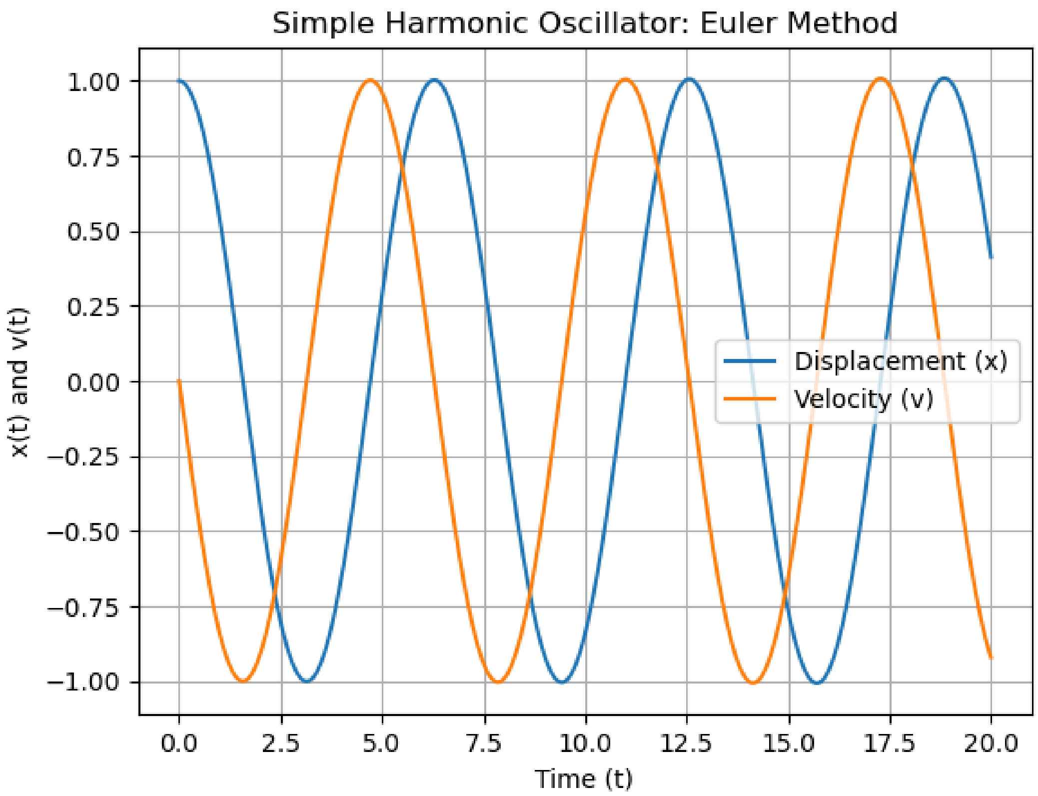
In [13]:
```
# Step 8: evaluate the phase space

# TODO
# Plot the phase space (x vs v)
plt.plot(x, v, label="Phase Space (x vs v)")
plt.xlabel("Displacement x(t)")
plt.ylabel("Velocity v(t)")
plt.title("Phase Space of the Simple Harmonic Oscillator")
plt.grid(True)
plt.legend()
plt.show()
```

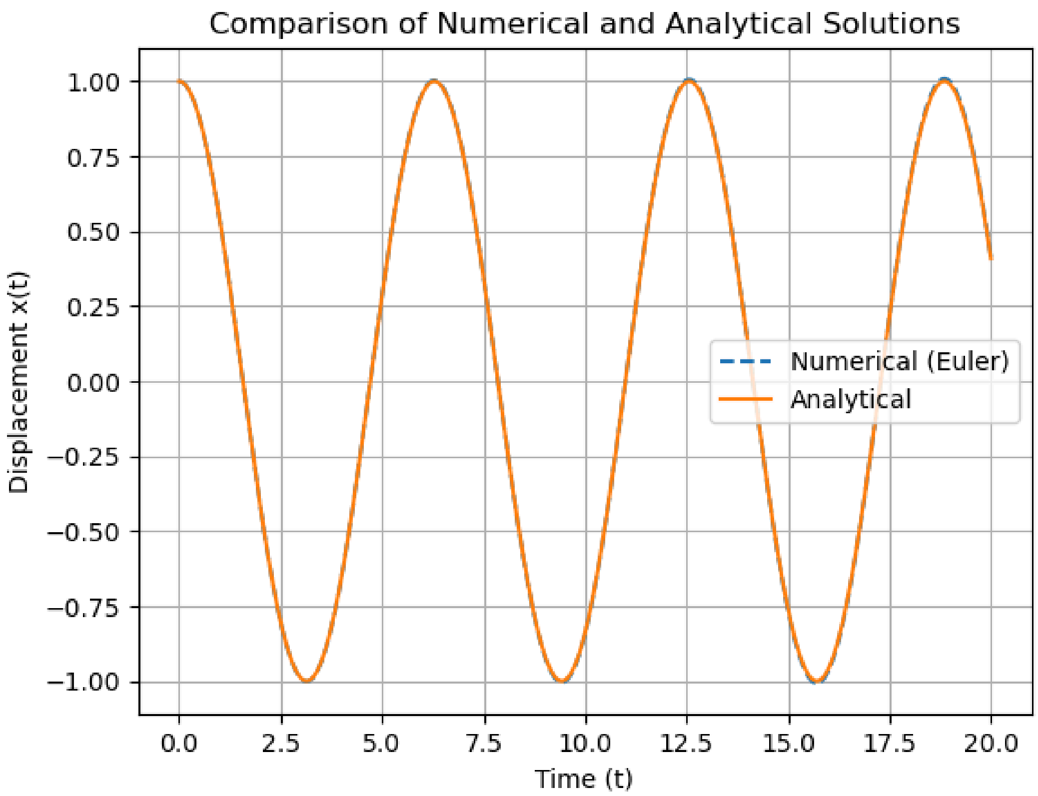


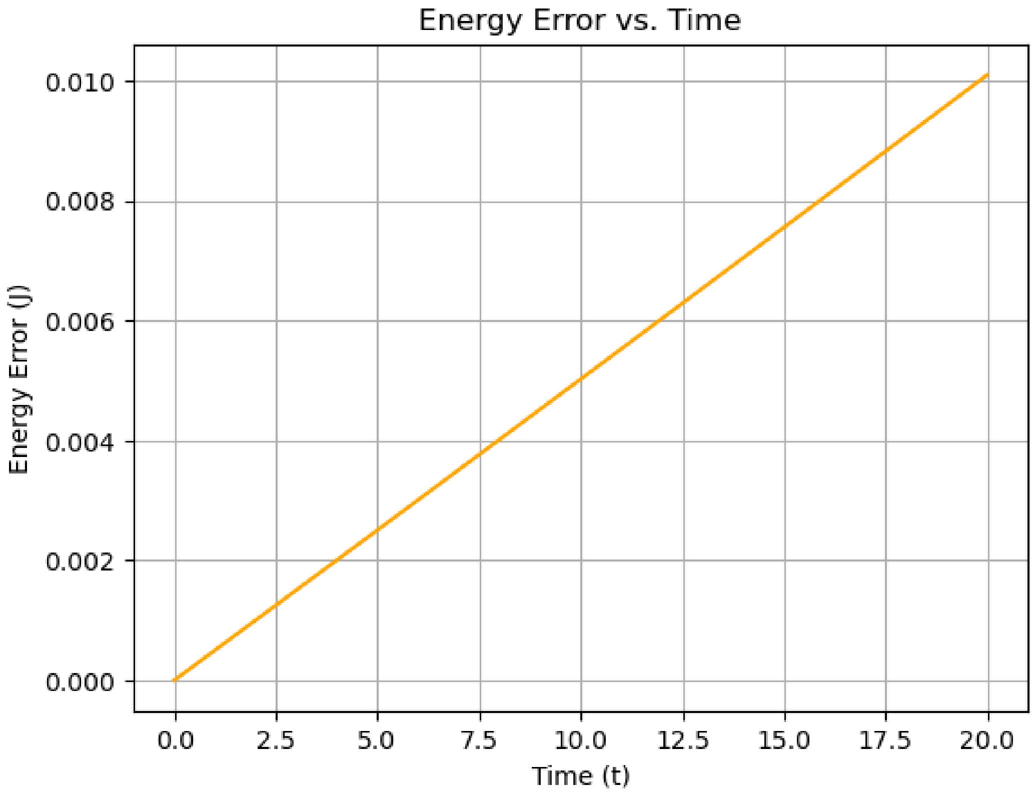
Exercise 2:

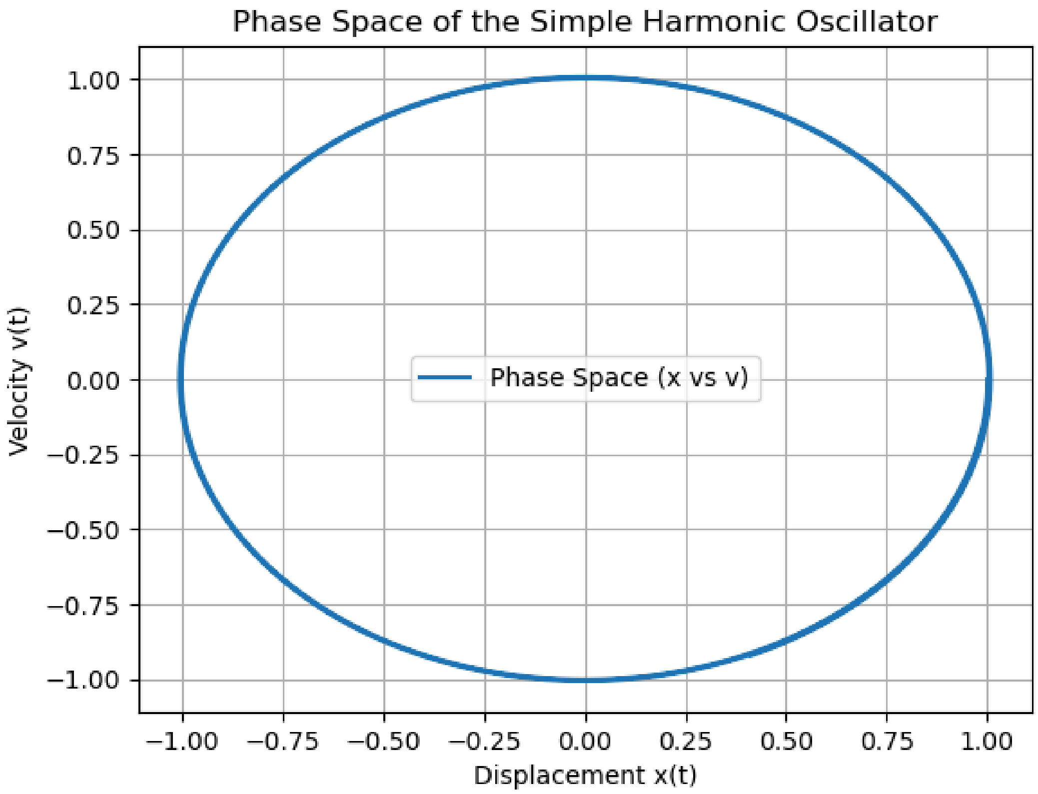
Check if the accuracy can be improved by reducing the time step to $\Delta t = 0.001$.

In [14]:
```
# Step 1: set up the parameters of the problem
A=1
k=1
m=1
omega0 = np.sqrt(k/m)
dt = 0.001
# t range
t_max = 20
t, x, v = sho_euler(A, k, m, dt, t_max)

# Step 2: plot the solution
plt.plot(t, x, label="Displacement (x)")
plt.plot(t, v, label="Velocity (v)")
plt.xlabel("Time (t)")
plt.ylabel("x(t) and v(t)")
plt.title("Simple Harmonic Oscillator: Euler Method")
plt.legend()
plt.grid(True)
plt.show()

# Step 3: evaluate the analytical solution and plot it
# Analytical solution for displacement
x_analytical = A * np.cos(omega0 * t)

# Plot numerical and analytical solutions
plt.plot(t, x, label="Numerical (Euler)", linestyle='--')
plt.plot(t, x_analytical, label="Analytical", linestyle='-')
plt.xlabel("Time (t)")
plt.ylabel("Displacement x(t)")
plt.title("Comparison of Numerical and Analytical Solutions")
plt.legend()
plt.grid(True)
plt.show()

# Step 3: evaluate the energy (error) of the system
# Calculate Kinetic Energy (KE) and Potential Energy (PE)
KE = 0.5 * m * v**2      # Kinetic energy
PE = 0.5 * k * x**2      # Potential energy
# Total energy
E_total = KE + PE
# Theoretical total energy
E_theoretical = 0.5 * k * A**2  # Initial energy (when x = A and v = 0)
# Evaluate the error (difference between total energy and theoretical energy)
energy_error = E_total - E_theoretical
# Plot the energy error over time
plt.plot(t, energy_error, label="Energy Error", color='orange')
plt.xlabel("Time (t)")
plt.ylabel("Energy Error (J)")
plt.title("Energy Error vs. Time")
plt.grid(True)
plt.show()

# Step 4: evaluate the phase space
# Plot the phase space (x vs v)
plt.plot(x, v, label="Phase Space (x vs v)")
plt.xlabel("Displacement x(t)")
plt.ylabel("Velocity v(t)")
plt.title("Phase Space of the Simple Harmonic Oscillator")
plt.grid(True)
plt.legend()
plt.show()
```


Simple Harmonic Oscillator: Euler Method
x(t) and v(t)
Time (t)
Displacement (x)
Velocity (v)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Comparison of Numerical and Analytical Solutions
Displacement x(t)
Time (t)
Numerical (Euler)
Analytical
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0





Note

Reducing the time step is not the best solution. The better solution is to use higher-order schemes. Do NOT use Eulter's method in any production runs.