Numerically Solving Ordinary Differential Equations (ODEs): Euler's Method

Differential Equations

A **differential equation** is any equation that contains one or more derivatives:

Constant velocity motion: $$\frac{dx}{dt} = v_0$$

Simple harmonic oscillator: $$m\frac{d^2x}{dt^2} + kx = 0$$

Motion of a charge in E and B fields: $$m\frac{d^2\vec{r}}{dt^2} = q\vec{E} + q\vec{v} \times \vec{B}$$

Poisson's equation: $$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = -\frac{\rho}{\epsilon_0}$$

Differential Equations

Ordinary differential equations (ODEs) - single independent variable.

$$\frac{dx}{dt} = v_0 \qquad\qquad m\frac{d^2x}{dt^2} + kx = 0$$

Partial differential equations (PDEs) - multiple independent variables.

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = -\frac{\rho}{\epsilon_0}$$

Ordinary Differential Equations (ODEs)

The order of an ODE is given by the highest derivative present

1st order ODEs:

$$\frac{dx}{dt} = v_0 \qquad \frac{dx}{dt} = -kx^2$$

2nd order ODEs:

$$m\frac{d^2x}{dt^2} + kx = 0 \qquad m\frac{d^2\vec{r}}{dt^2} = \frac{kq_1q_2}{|\vec{r}|^2}$$

$$m\frac{d^2\vec{r}}{dt^2} = q\vec{E} + q\vec{v} \times \vec{B}$$

2nd order ODEs often result from Newton's second law: $F_{net} = m\frac{d^2x}{dt^2}$

Solve an ODE analytically if you can

$$m\frac{d^2x}{dt^2} + kx = 0 \longrightarrow x(t) = A\sin(\omega t) + B\cos(\omega t) \qquad \omega = \sqrt{\frac{k}{m}}$$

If you can't, use numerical methods

$$\frac{d^2x}{dt^2} - \mu(1 - x^2)\frac{dx}{dt} + x = 0 \longrightarrow x(t) = ??$$

Numerical Integration of a First-Order ODE

Numerical integration is like analyzing frames of a movie, where the time between frames is $\Delta t = t_{n+1} - t_n$.

Δt

t_{n-2} t_{n-1} t_n t_{n+1} t_{n+2}

In general, we want to solve: $\dfrac{dx}{dt} = f(x, t) = v(x, t)$

Euler Method

Goal: predict the position x_{n+1} of an object on the next frame of the movie, given its current position x_n

Approximate the derivative:

$$\frac{dx}{dt} = v \quad \longrightarrow \quad \frac{x_{n+1} - x_n}{\Delta t} = v_n$$

Solve for the position x_{n+1}:

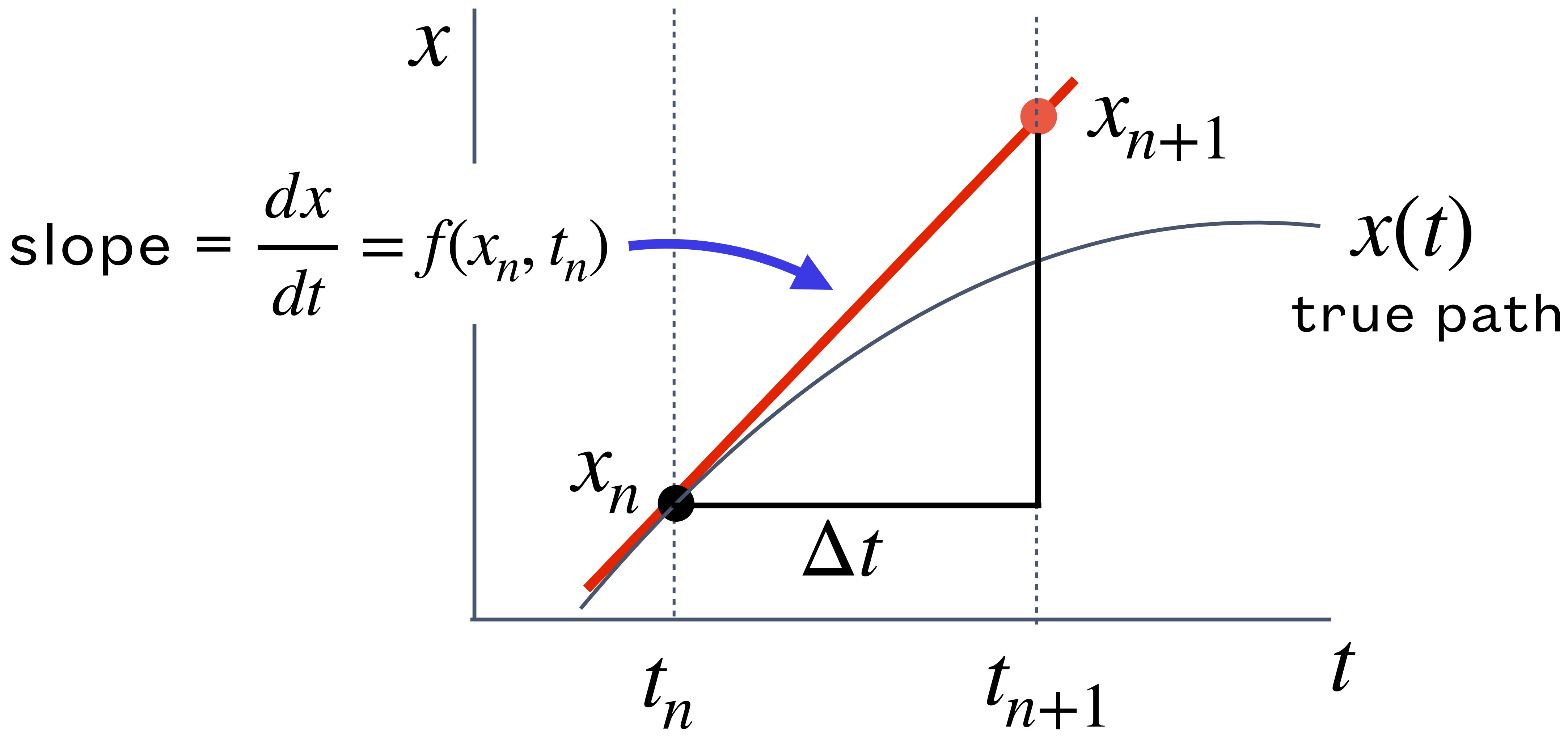
$$x_{n+1} = x_n + v_n \Delta t$$

(Euler **update rule**)

Euler Method

$$\frac{dx}{dt} = f(x, t)$$

$$x_{n+1} = x_n + v_n \Delta t$$



Approximate $x(t)$ as a Taylor series:

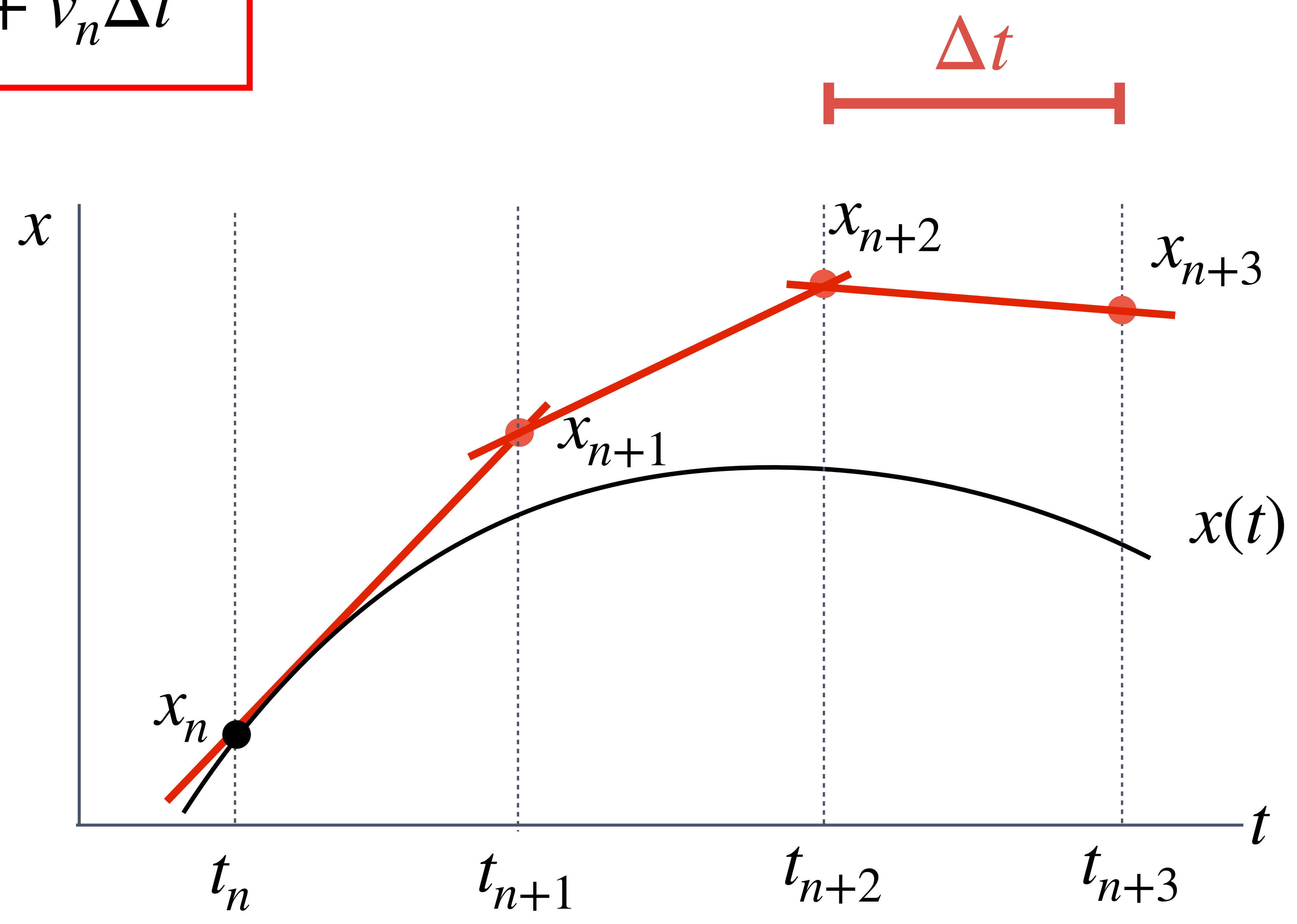
$$x(t_n + \Delta t) \approx x_n + \left(\frac{dx}{dt}\right)_{t_n} \Delta t + \left(\frac{d^2x}{dt^2}\right)_{t_n} \Delta t^2 + \ldots$$

The Euler method is equivalent to including the constant term and the linear term

The Euler method is said to be **first-order accurate**

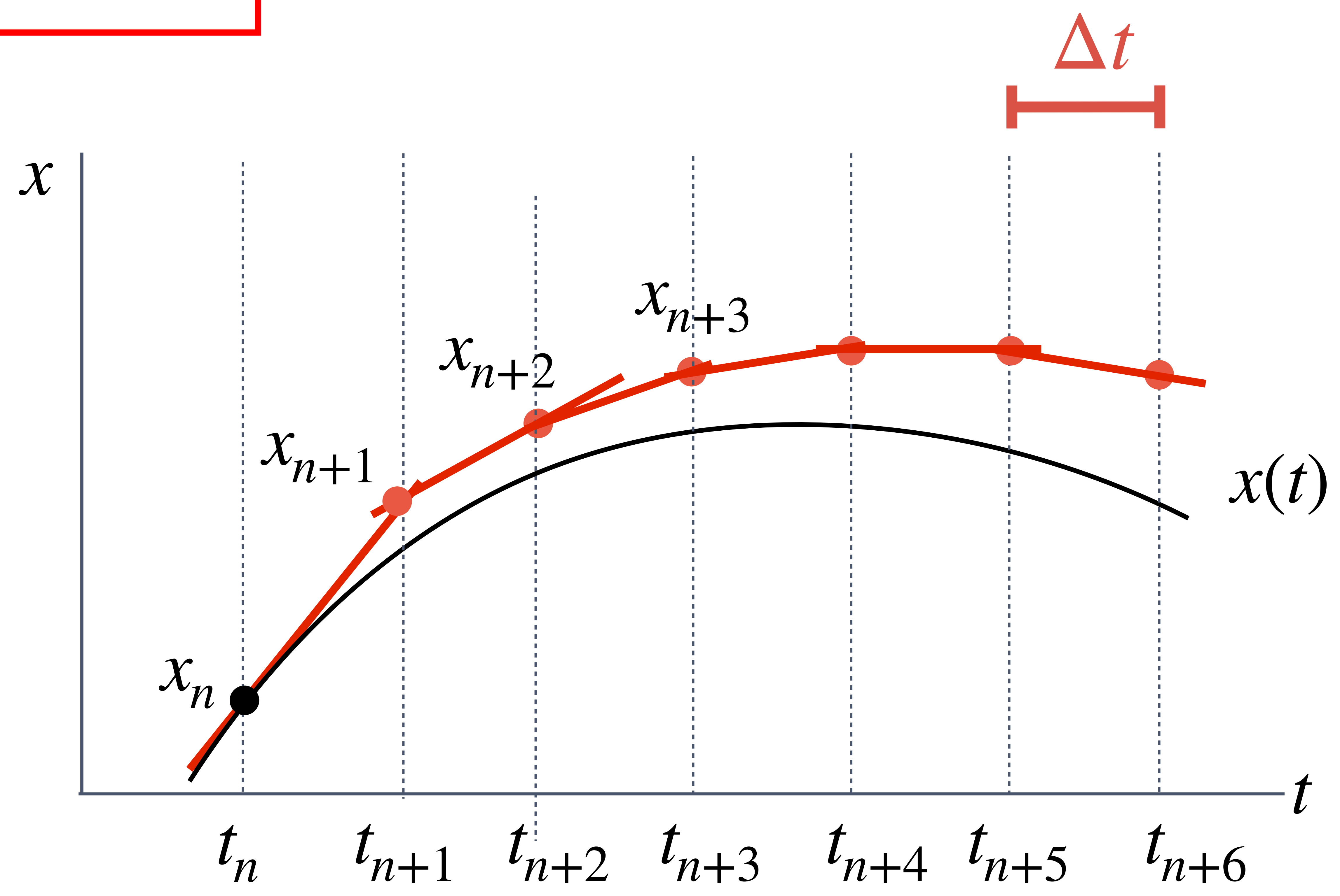
Euler Method

$$x_{n+1} = x_n + v_n \Delta t$$



Reducing time step helps reduce the error, but increases the computation time.

$$x_{n+1} = x_n + v_n \Delta t$$



Euler Method: Constant Velocity Motion

Differential Equation: $\frac{dx}{dt} = v_0$

Initial conditions: $x_0 = 0$ m

Parameters: $v = 10\ m/s$

$\Delta t = 2$ **s**

$t_{max} = 10\ s$

Euler update rule:

$$x_{n+1} = x_n + v\Delta t$$

Index	time	position
0	0	0
1	2	20
2	4	40
3	6	60
4	8	80
5	10	100

Pseudocode

Initialization

1.Define: object velocity v
2.Define: time step and final time
3.Calculate number of points N
4.Preallocate arrays to store t and x values
5.Store initial conditions in x[0] and t[0]

Iteration

6. Loop to calculate t[0] and x[0] for n = 1 to N

Present Results

7. Plot x vs. t

loop to fill in x and t values:

t[0]	t[1]	t[2]	t[3]	t[4]	t[5]
0	0	0	0	0	0

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]
0	0	0	0	0	0

loop to fill in x and t values:

t[0]	t[1]	t[2]	t[3]	t[4]	t[5]
0	2	4	6	8	10

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]
0	20	40	60	80	100

Euler Method: Constant Velocity Motion

```
import numpy as np
import matplotlib.pyplot as plt

#########  Parameters  #########

v    = 10               # velocity
tmax = 10               # maximum time
dt   = 2                # time step
x0   = 10               # initial value of x


#########  Create Arrays  #########

N = int(tmax/dt)+1      # number of steps in simulation

x = np.zeros(N)         # array to store positions
t = np.zeros(N)         # array to store times

x[0] = x0               # assign initial value

#########  Loop to implement the Euler update rule  #########

for n in range(N-1):
    x[n+1] = x[n] + v*dt   # Euler update rule for position
    t[n+1] = t[n] + dt     # update time
```

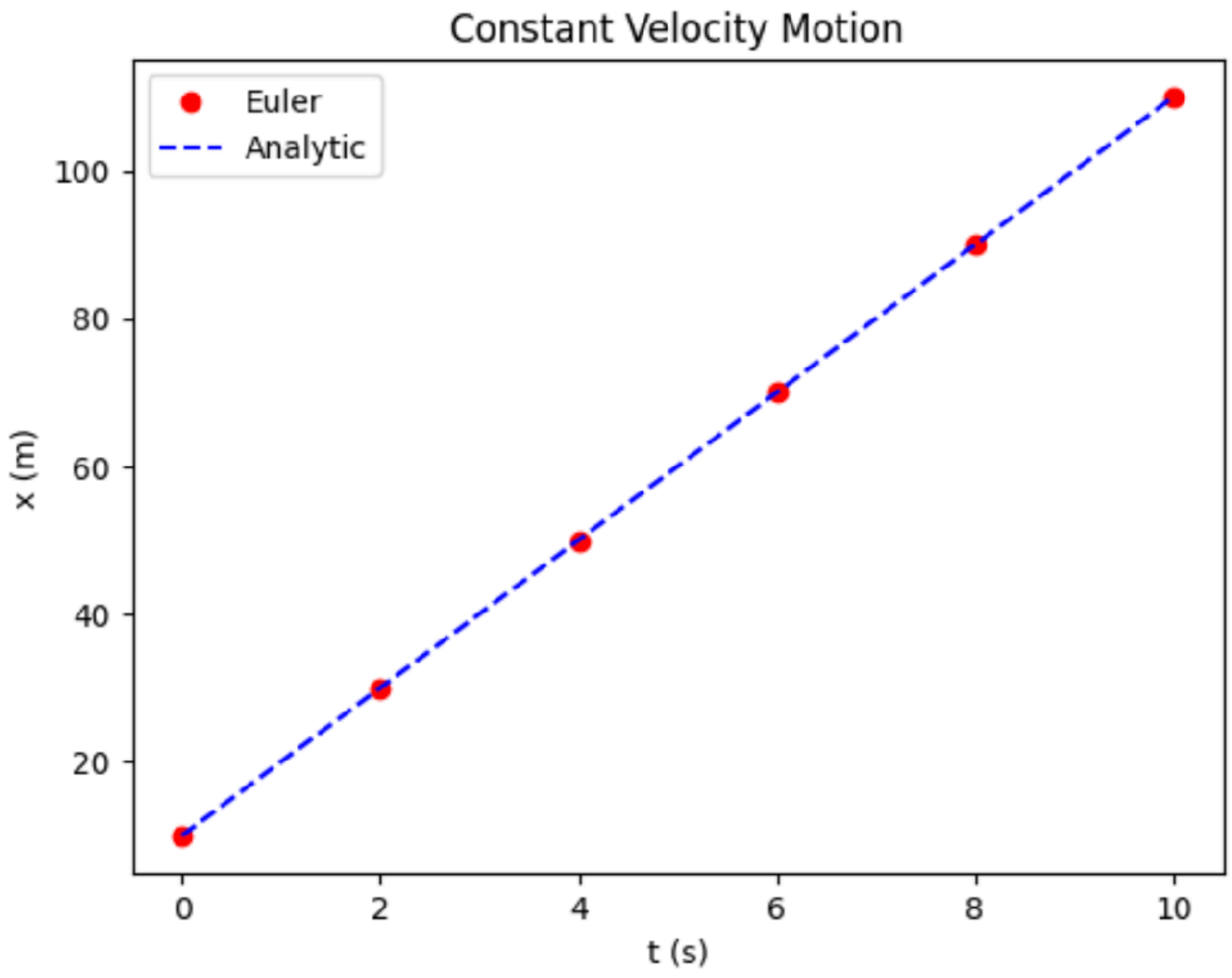
```
#########  Analytic Solution  #########

x_true = x0 + v*t


#########  Plot Solution  #########

plt.plot(t, x, 'ro', label='Euler')
plt.plot(t, x_true, 'b--', label='Analytic')

plt.xlabel('t (s)')
plt.ylabel('x (m)')
plt.title("Constant Velocity Motion")
plt.legend()
plt.show()
```


Constant Velocity Motion
Euler
Analytic
100
80
60
40
20
x (m)
0
2
4
6
8
10
t (s)

Euler Method: Exponential Growth

Differential Equation: $\frac{dy}{dt} = ay$

Initial conditions: $y_0 = 1$ m

Parameters: $a = 0.2$

$\Delta t = 1$ **s**

$t_{max} = 10\ s$

Euler update rule:

$$y_{n+1} = y_n + ay_n\Delta t$$

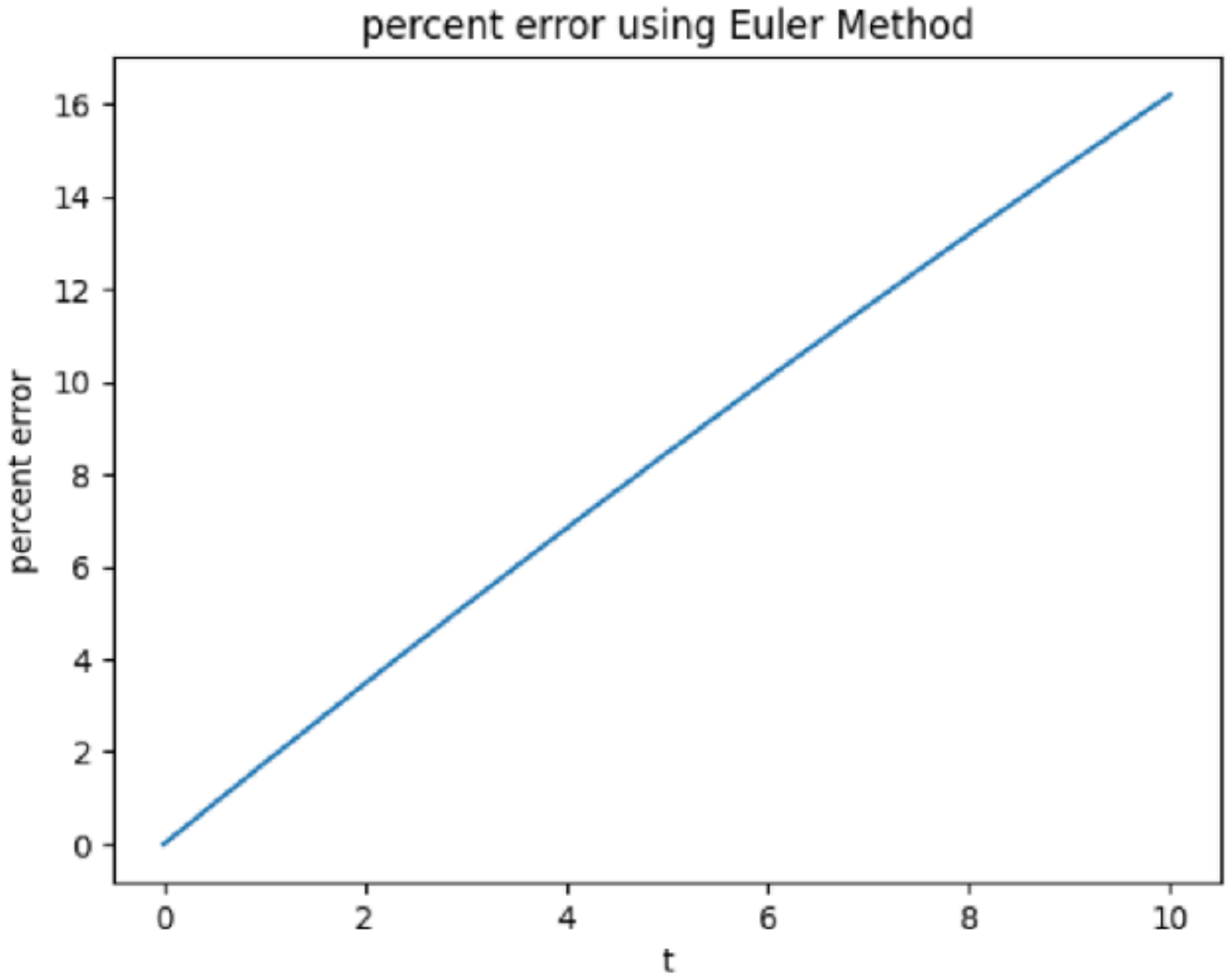
Index	time	position
0	0	0
1	1	1
2	2	1.20
3	3	1.44
4	4	1.73
5	5	2.07

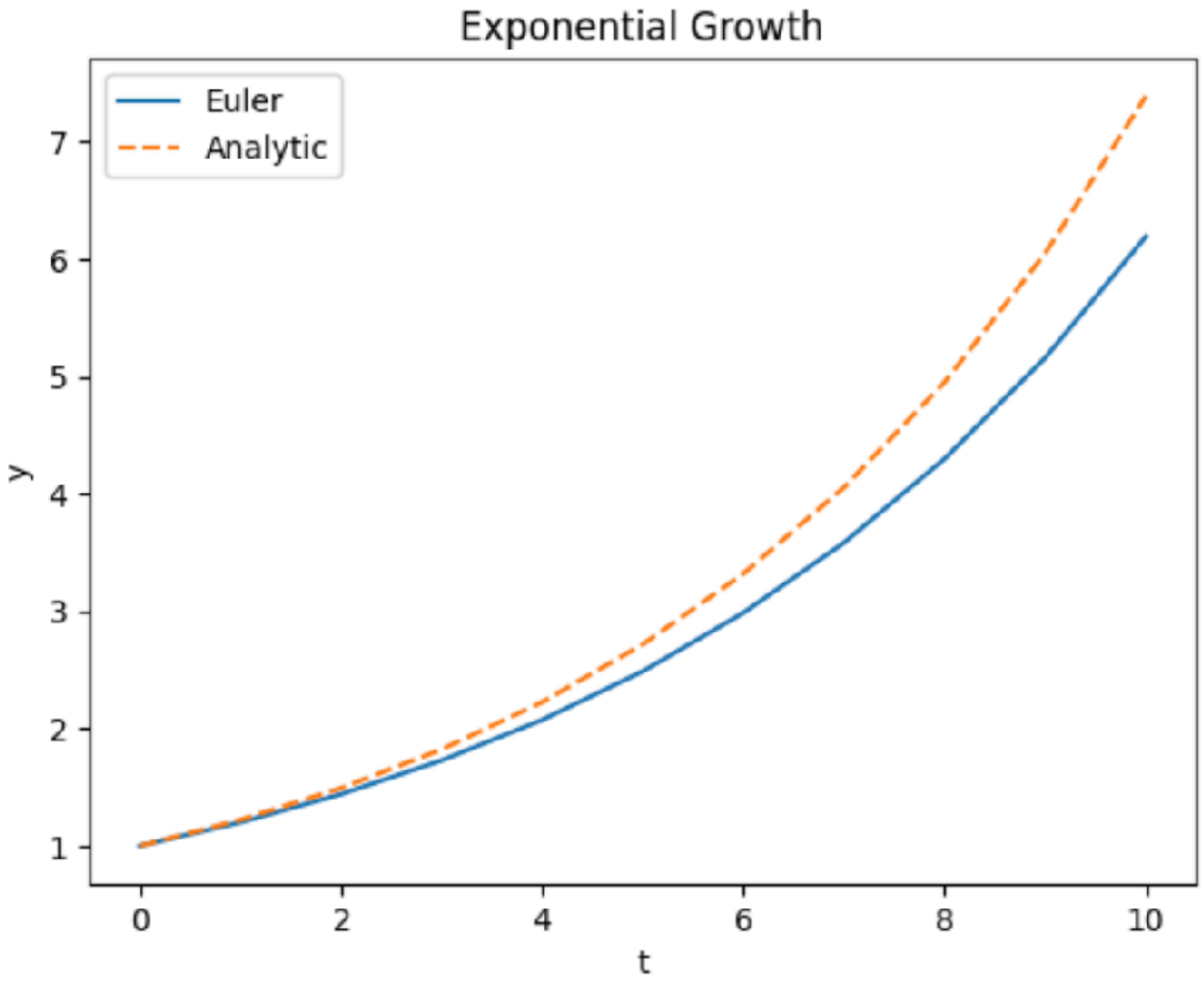
$\cdots$

Euler Method: Exponential Growth

Loop with Euler update rule

```
for n in range(N-1):
    y[n+1] = y[n] + a*y[n]*dt
    t[n+1] = t[n] + dt
```

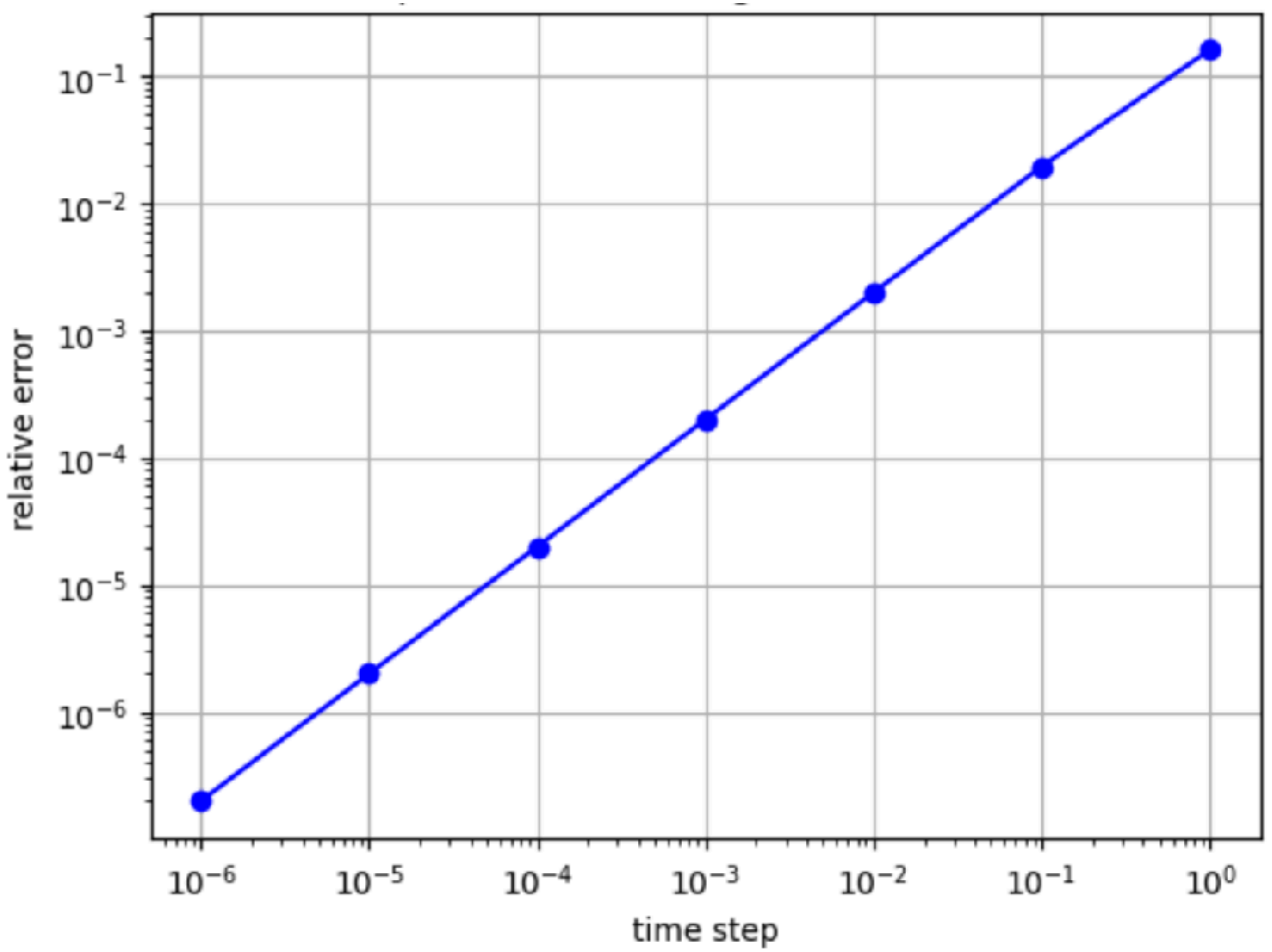




Euler Method: Decreasing step size, decreases error

Because the Euler method is **first-order** accurate, the error decreases in proportion ot the step size:

$$error \propto \Delta t$$



Stability

Stability determines whether numerical errors will grow uncontrollably causing the solution "blow up."

In many cases, stability can be achieved if the time step is small enough. The following example shows how to find the stability threshold for the time step.

Example: Stability of Euler's method for the Exponential Growth ODE

1) Start with the Euler update rule: $x_{n+1} = x_n + a x_n \Delta t$ → $x_{n+1} = x_n(1 + a\Delta t)$

2) Successive updates give:

$$x_1 = x_0(1 + a\Delta t)$$

$$x_2 = x_1(1 + a\Delta t) = x_0(1 + a\Delta t)^2$$

$$x_3 = x_2(1 + a\Delta t) = x_0(1 + a\Delta t)^3$$

$$x_n = x_0(1 + a\Delta t)^n$$

Stability

$$x_n = x_0(1 + a\Delta t)^n$$

3) Stability condition to prevent solution from blowing up: $x_n = |1 + a\Delta t| < 1$

4) Condition on time step:

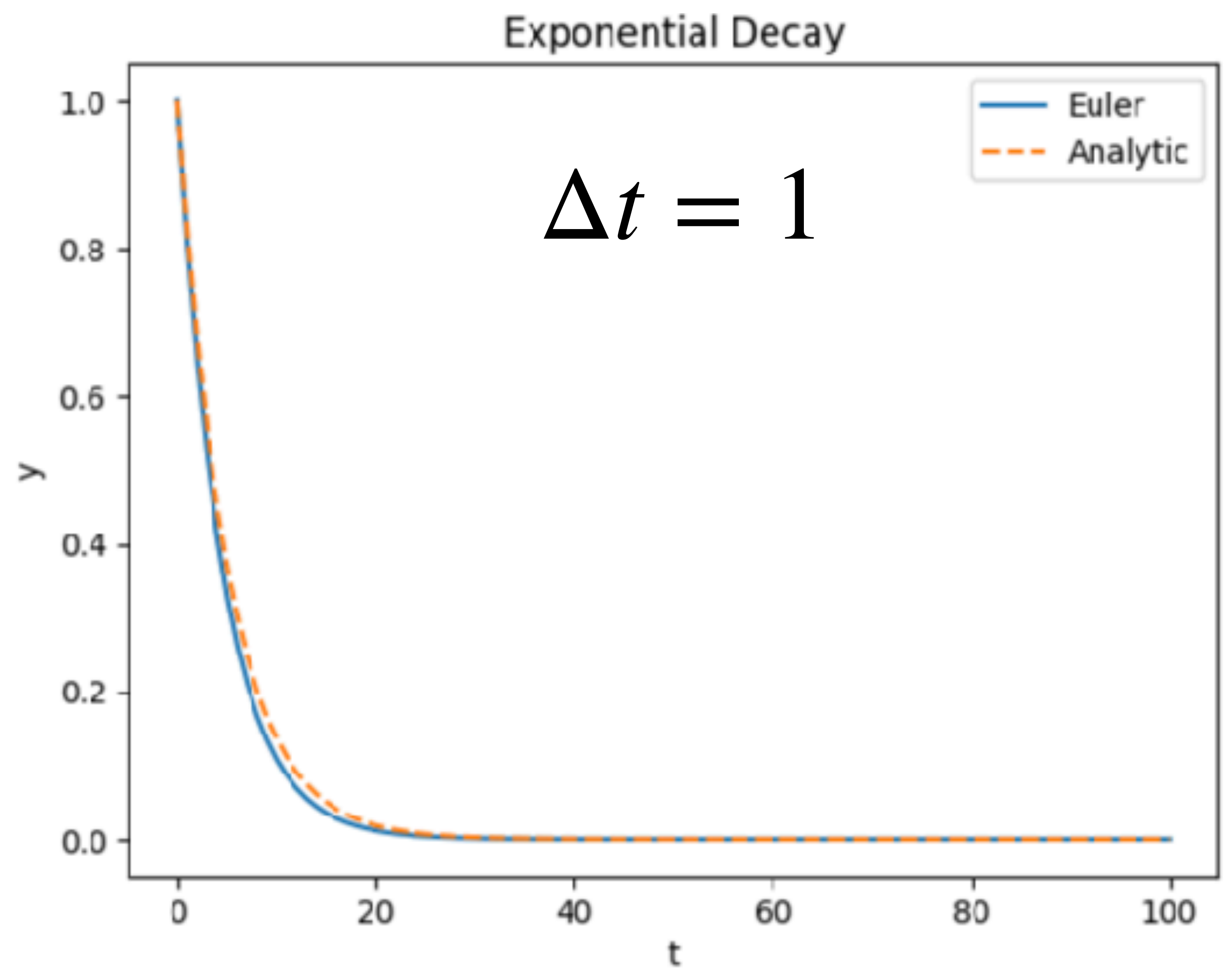
(a) If a > 0, $1 + a\Delta t > 1$ and the Euler method is unstable for all Δt values

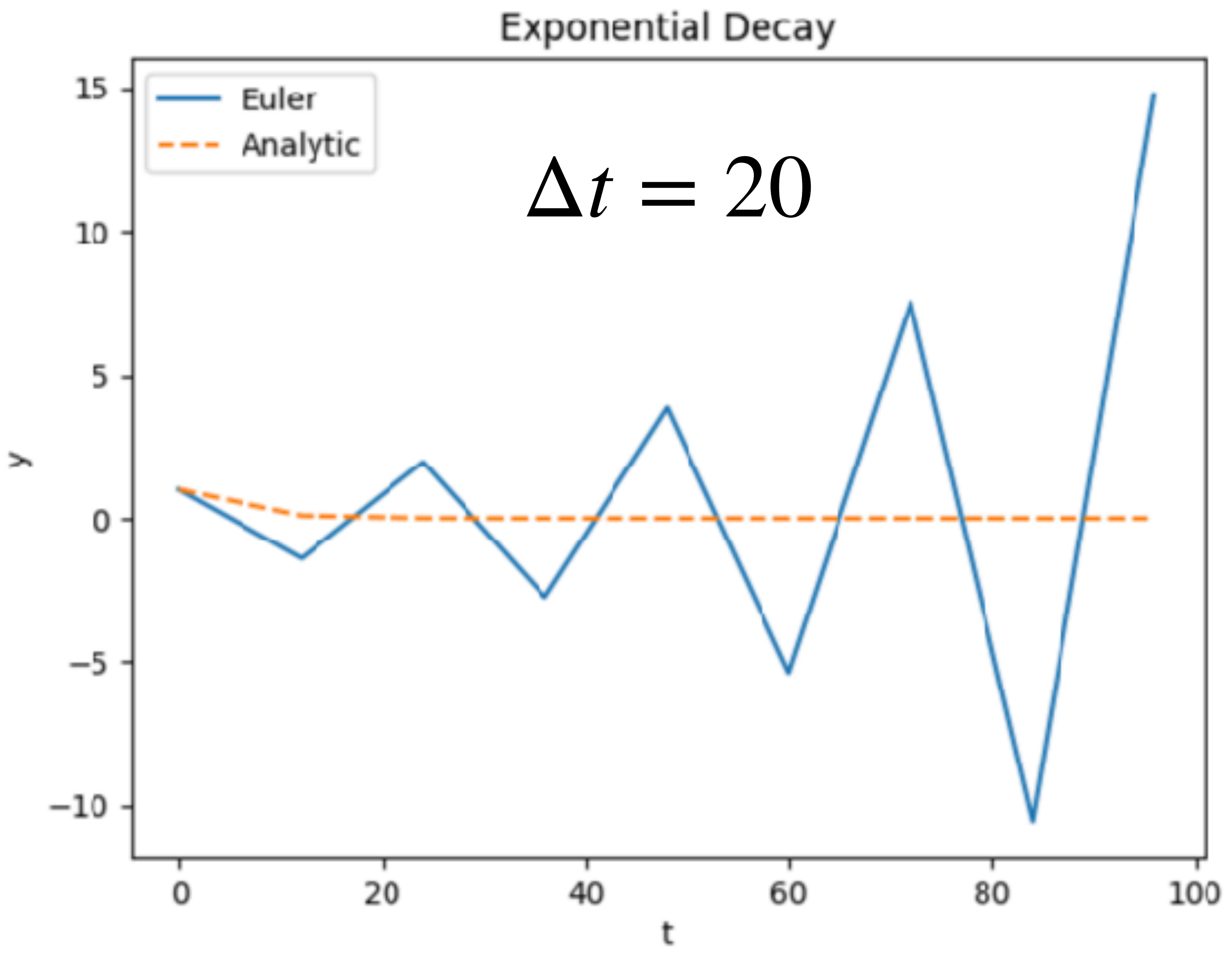
(b) If a < 0, stability condition on Δt is $\Delta t < \frac{2}{|a|}$

Example: if a = -0.2, then:

$\Delta t = 1$ is stable

$\Delta t = 20$ is unstable





Solving a 2nd Order ODE using the Euler Method

Break 2nd Order ODE into Two 1st Order ODEs

Newton's 2nd law produces an ODE of the form:

$$\frac{d^2x}{dt^2} = \frac{F}{m}$$

We can write this 2nd order equation as a system of two 1st order ODEs:

$$\frac{dx}{dt} = v$$

$$\frac{dv}{dt} = a \qquad \text{where } \; a = \frac{F}{m}$$

Solve a 2nd Order ODE

We can now solve each first-order equation using Euler's method:

Position | **Velocity**

$$\frac{d^2x}{dt^2} = \frac{F}{m} \quad \Rightarrow$$

Position

$$\frac{dx}{dt} = v$$

$$\boxed{x_{n+1} = x_n + v_n \Delta t}$$

Velocity

$$\frac{dv}{dt} = a \qquad \text{where } a = \frac{F}{m}$$

$$\boxed{v_{n+1} = v_n + a_n \Delta t}$$

The Euler method is first-order accurate and numerically unstable for many types of problems 🙁

Example: Simple Harmonic Oscillator

We can now solve each first-order equation using Euler's method:

$$m\frac{d^2x}{dt^2} = -kx$$

Position

$$\frac{dx}{dt} = v$$

$$x_{n+1} = x_n + v_n \Delta t$$

Velocity

$$\frac{dv}{dt} = a \qquad \text{where} \quad a = -\frac{kx}{m}$$

$$v_{n+1} = v_n + a_n \Delta t$$

$$a_n = -\frac{kx_n}{m}$$

Example: Simple Harmonic Oscillator

Parameters:

$$m = 1$$
$$k = 1$$
$$\Delta t = 0.05$$

Initial Conditions:

$$x_0 = 1$$
$$v_0 = 0$$

Update Equations

$$a_n = -\frac{kx_n}{m}$$
$$x_{n+1} = x_n + v_n \Delta t$$
$$v_{n+1} = v_n + a_n \Delta t$$

n	t	v	x
1	0	0	1.000
2	0.05	-0.050	0.997
3	0.01	-0.100	0.992
…	…	…	…

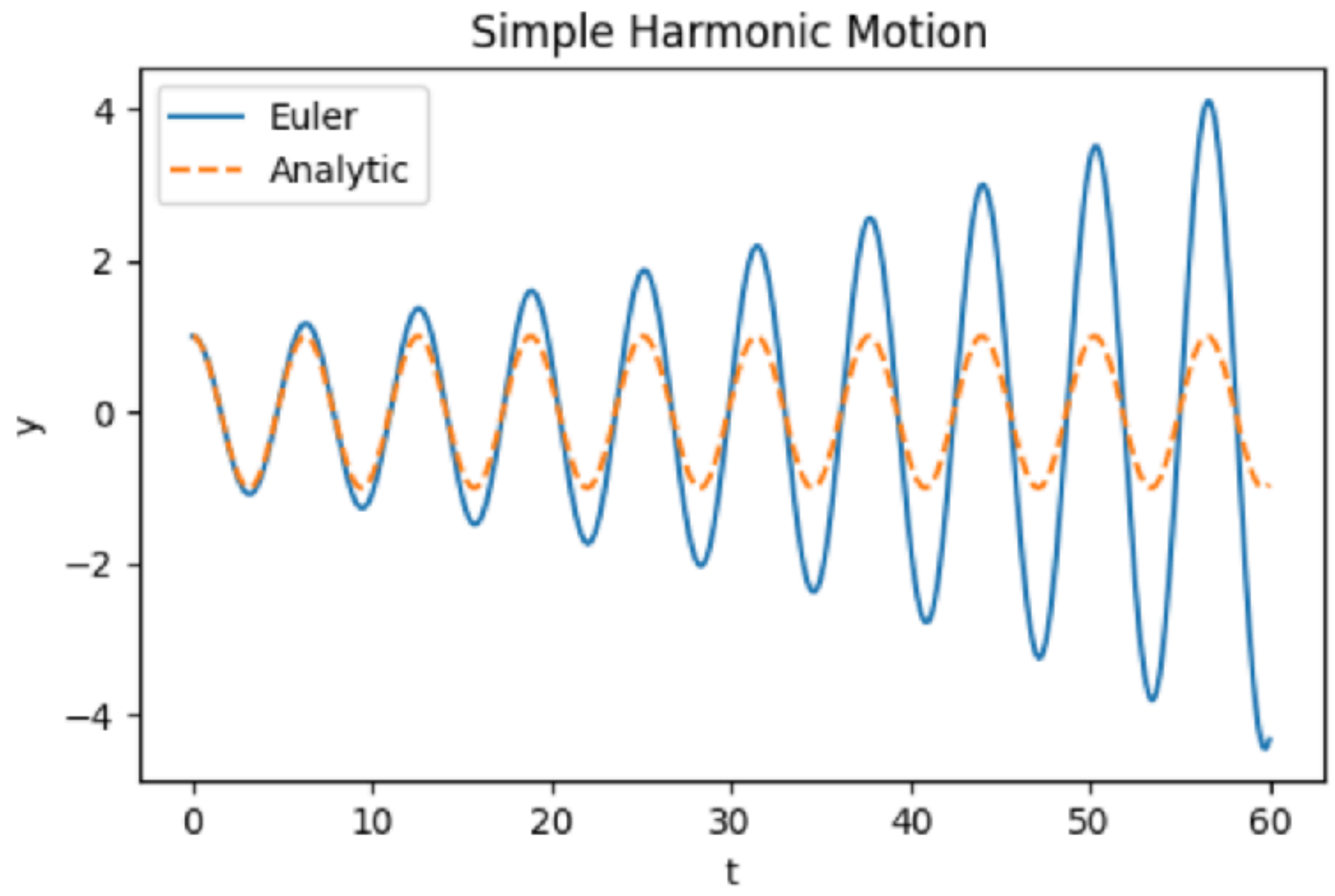
Euler's method is inherently unstable for many common 2nd-order systems

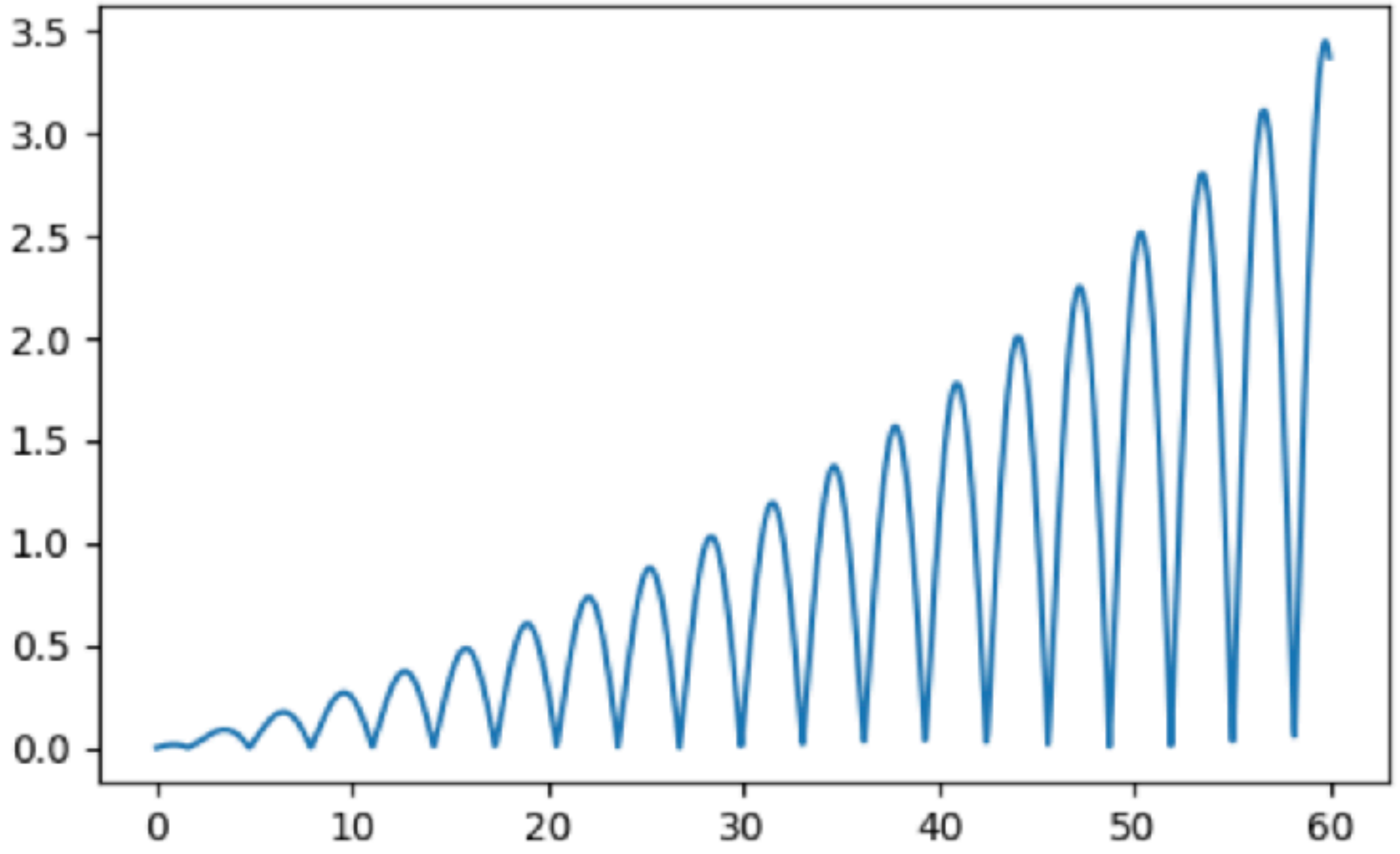
$\Delta t = 0.05$

$$x_{theory} = x_0 \cos(\omega t)$$

$$\omega = \sqrt{k/m}$$

Absolute error = $\left| x - x_{theory} \right|$



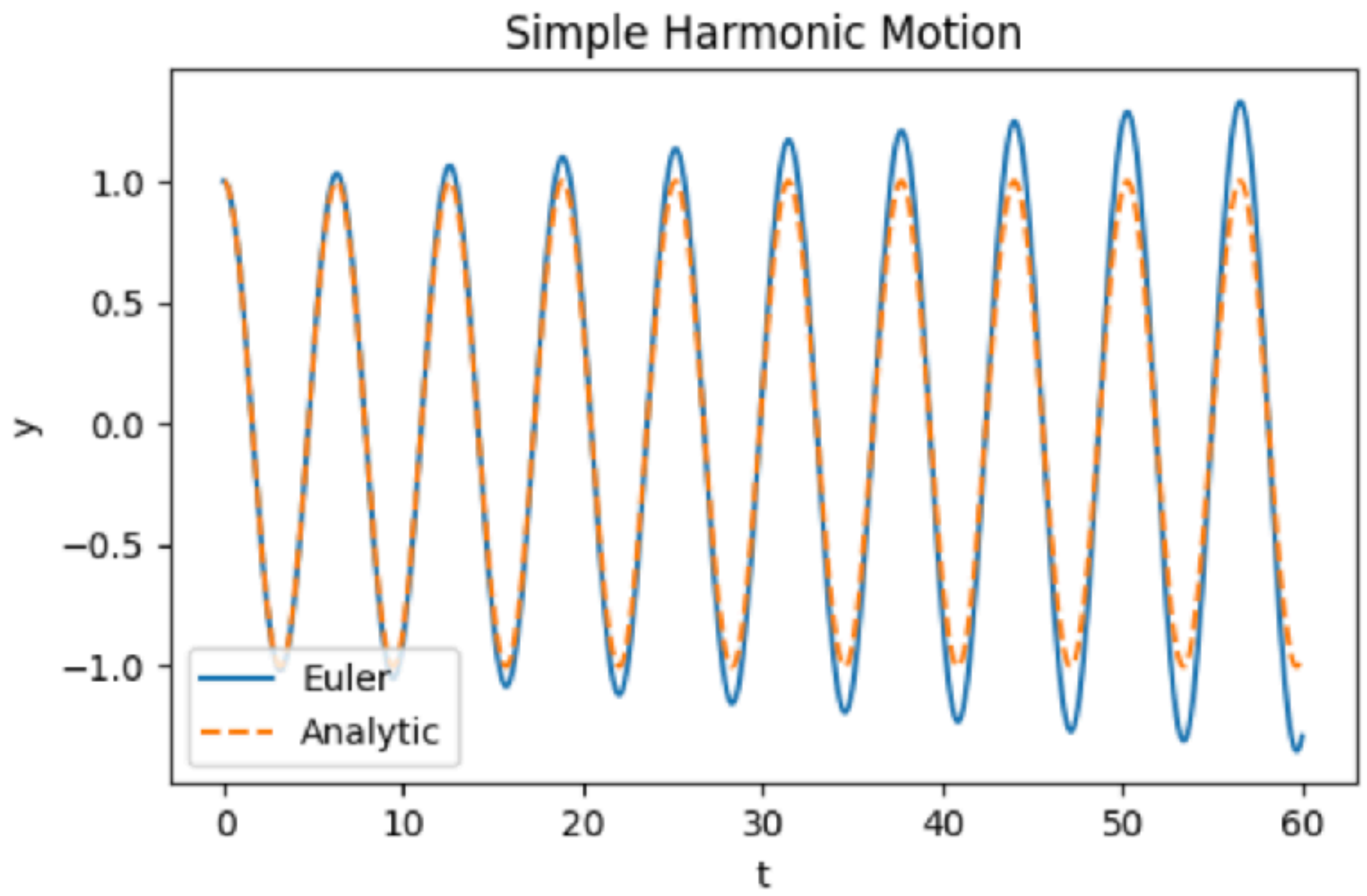


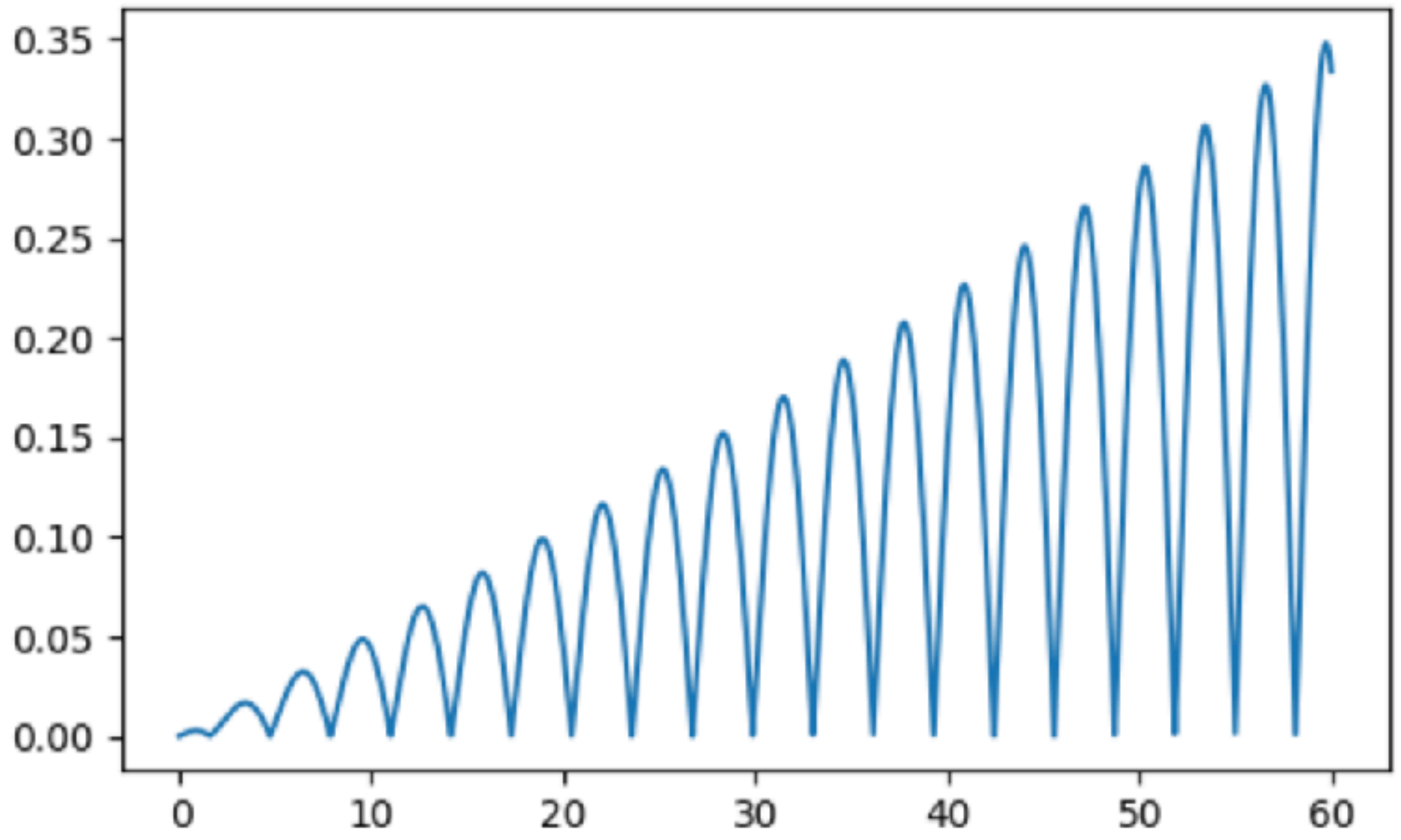
Reducing the time step improves accuracy - But errors grow over time.

$\Delta t = 0.01$

$$x_{theory} = x_0 \cos(\omega t)$$
$$\omega = \sqrt{k/m}$$

Absolute error = $\left| x - x_{theory} \right|$





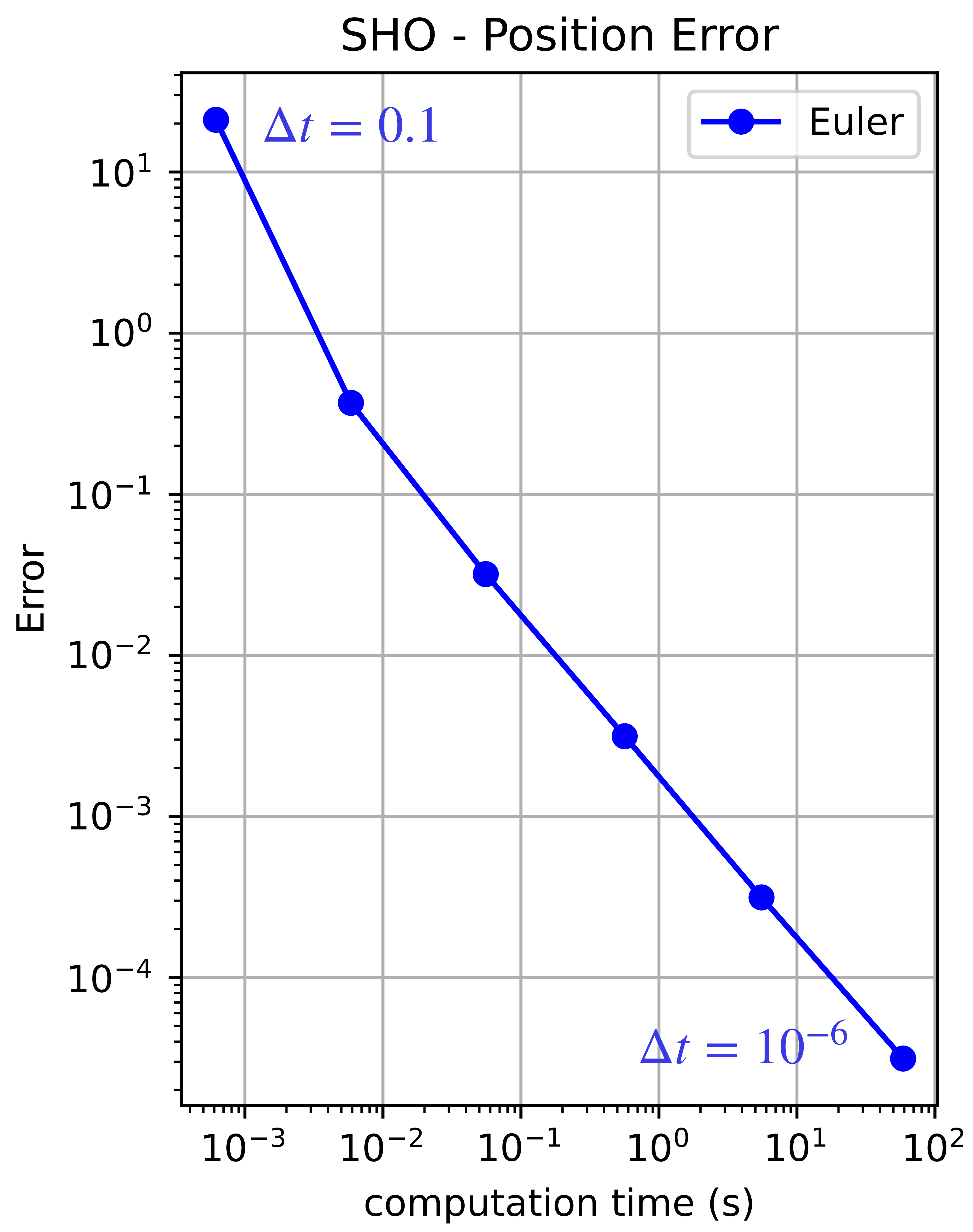
Effect of the time step on accuracy and computation time

t_max = 10 oscillation periods

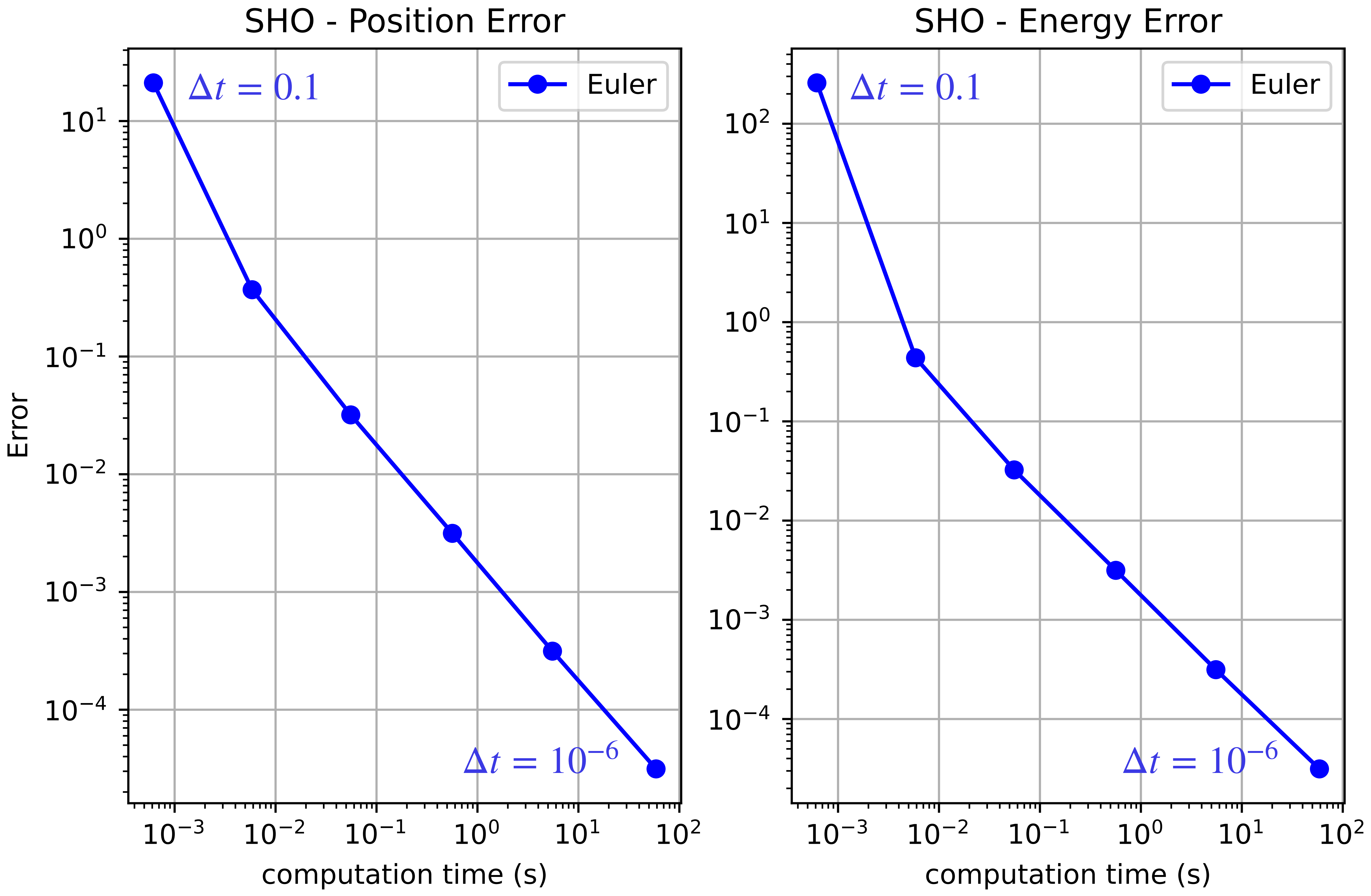
Positional error:

$$Error = |x(t_{end}) - x_{theory}(t_{end})|$$

Computation time = time (in seconds) to perform the numerical integration



Effect of time step on accuracy



Energy calculated from numerical solutions x and v:

$$E = \frac{1}{2}mv^2 + \frac{1}{2}kx^2$$

Energy calculated from theoretical x_{theory} and v_{theory}:

$$E = \frac{1}{2}mv_{theory}^2 + \frac{1}{2}kx_{theory}^2$$

$$x_{theory}(t) = x_0 \cos \omega t$$

$$v_{theory}(t) = -x_0 \omega \sin \omega t$$

$$\omega = \sqrt{k/m}$$

Phase Space

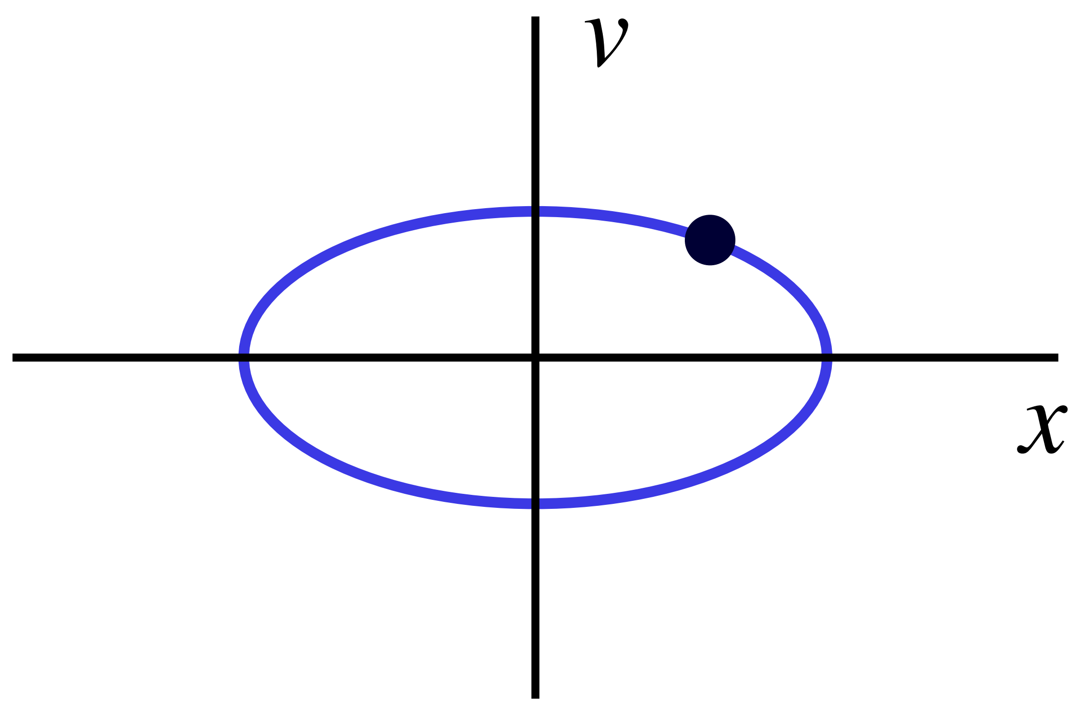
Phase space is an abstract space in which the state of a dynamical system is represented as a point that evolves in time, with each axis corresponding to one of the system's degrees of freedom.

Since the harmonic oscillator has two degrees of freedom, it will have a two-dimensional phase space (with axes often chosen to be the position and velocity of the particle).

Theoretical solution for the S.H.O:

$$x_{theory}(t) = x_0 \cos(\omega t)$$

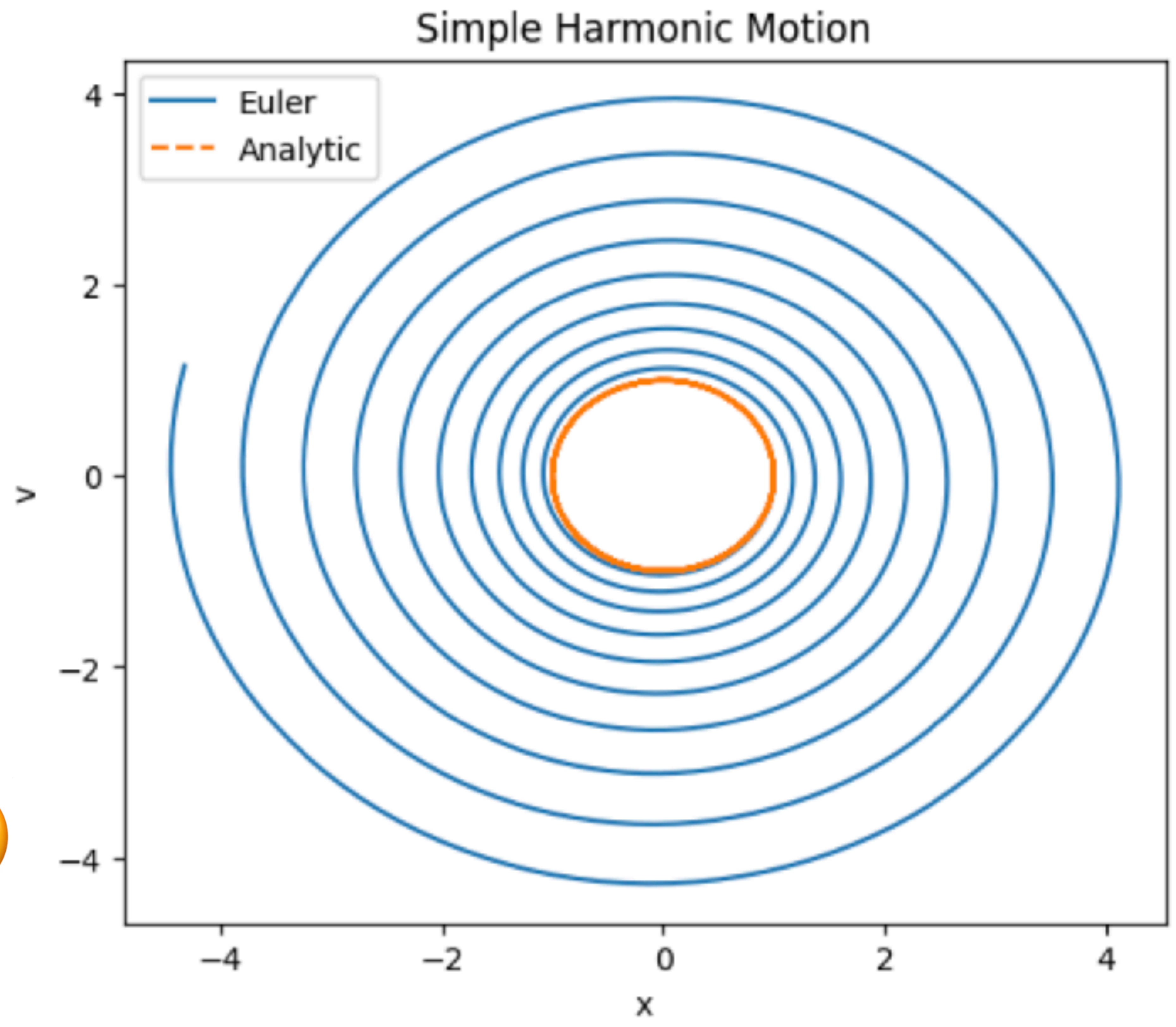
$$v_{theory}(t) = -x_0 \omega \sin(\omega t)$$



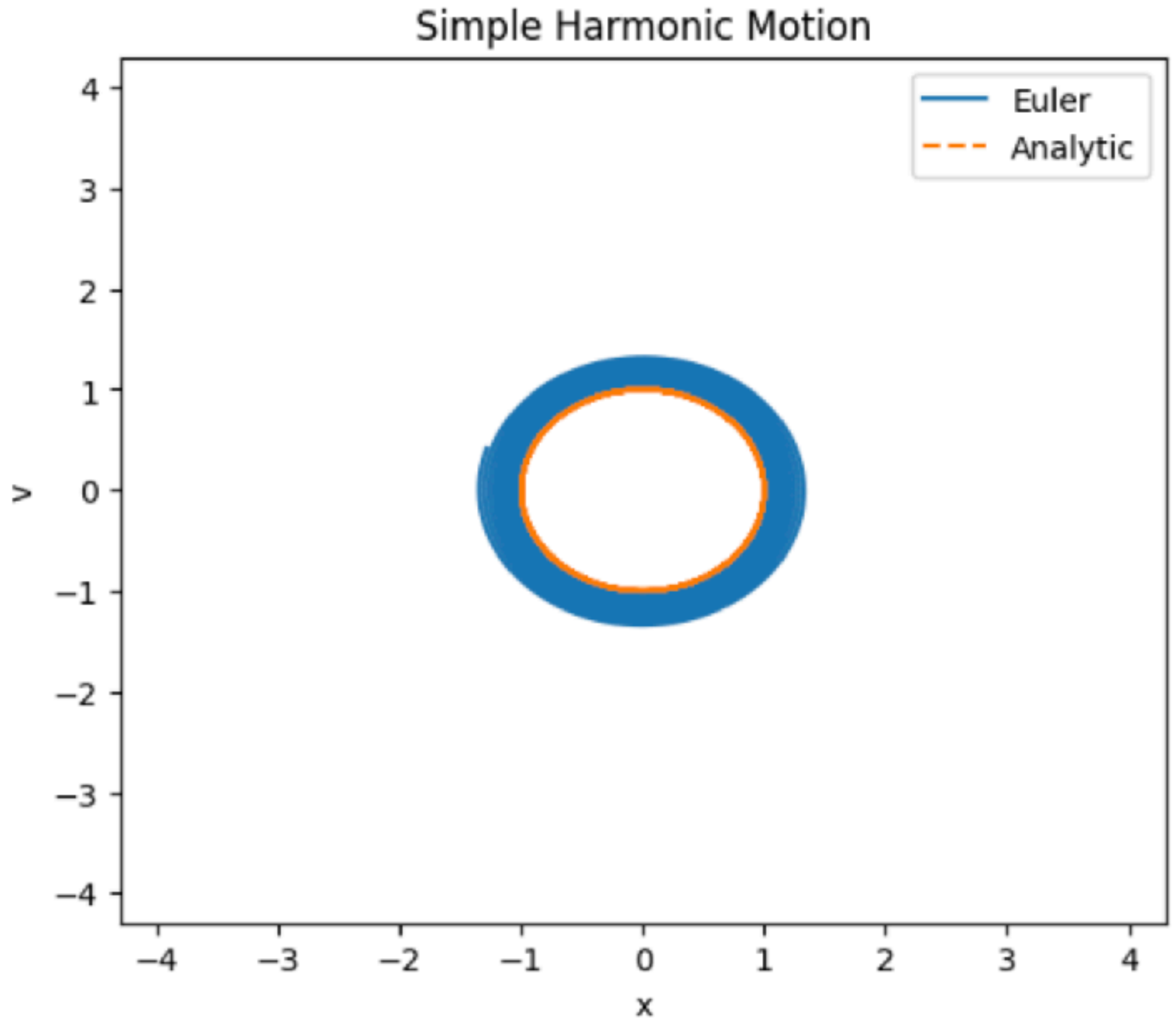
parametric equation of an ellipse

Phase Diagram: Plot v(t) vs. x(t)

$\Delta t = 0.05$



$\Delta t = 0.01$



The Euler-Cromer-Aspel Method

The Euler-Cromer-Aspel Method (or Aspel Method)

In 1980, Alan Cromer published a paper citing a high school student, Abby Aspel, for discovering a numerical integration method that was stable and more accurate than the Euler method, especially for solving oscillatory, 2nd-order ODE's.

While Aspel was credited in the paper, she was not listed as a co-author and the method is often called the "Euler-Cromer" or "Symplectic Euler" method.

Abby Aspel

Aspel's contributions have recently been rediscovered and her name is starting to be rightfully associated with the method.

Euler Method

$$x_{n+1} = x_n + v_n \Delta t$$

$$v_{n+1} = v_n + a_n \Delta t$$

Euler Method: The position update rule uses the OLD velocity v_n .

Euler Method is **not symplectic**, meaning it does not conserve energy.

Euler-Cromer-Aspel Method

$$v_{n+1} = v_n + a_n \Delta t$$

$$x_{n+1} = x_n + v_{n+1} \Delta t$$

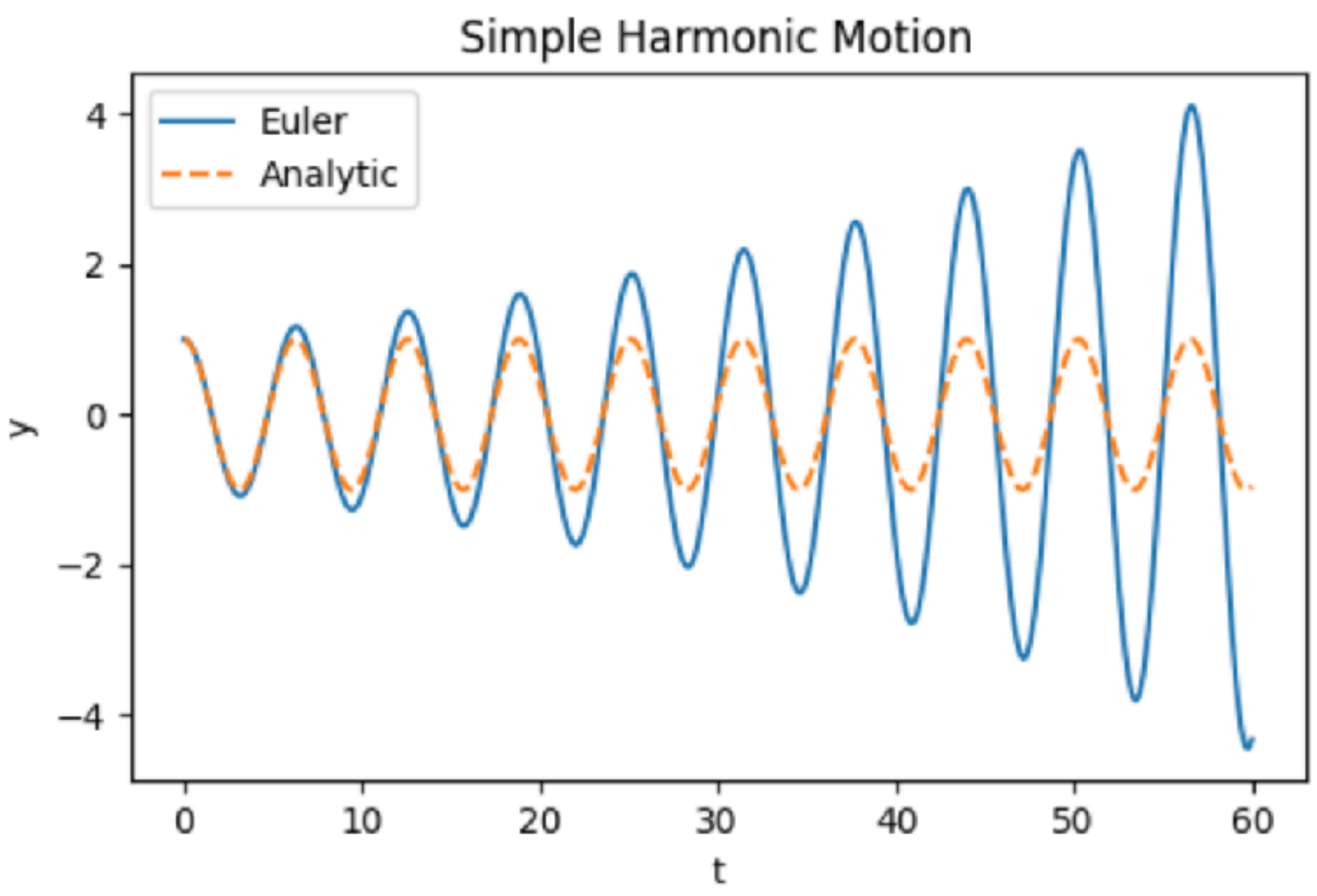
Aspel Method: The position update rule uses the NEW velocity v_{n+1} .

Aspel Method is **symplectic**, meaning it is energy conserving.

Simple Harmonic Oscillator

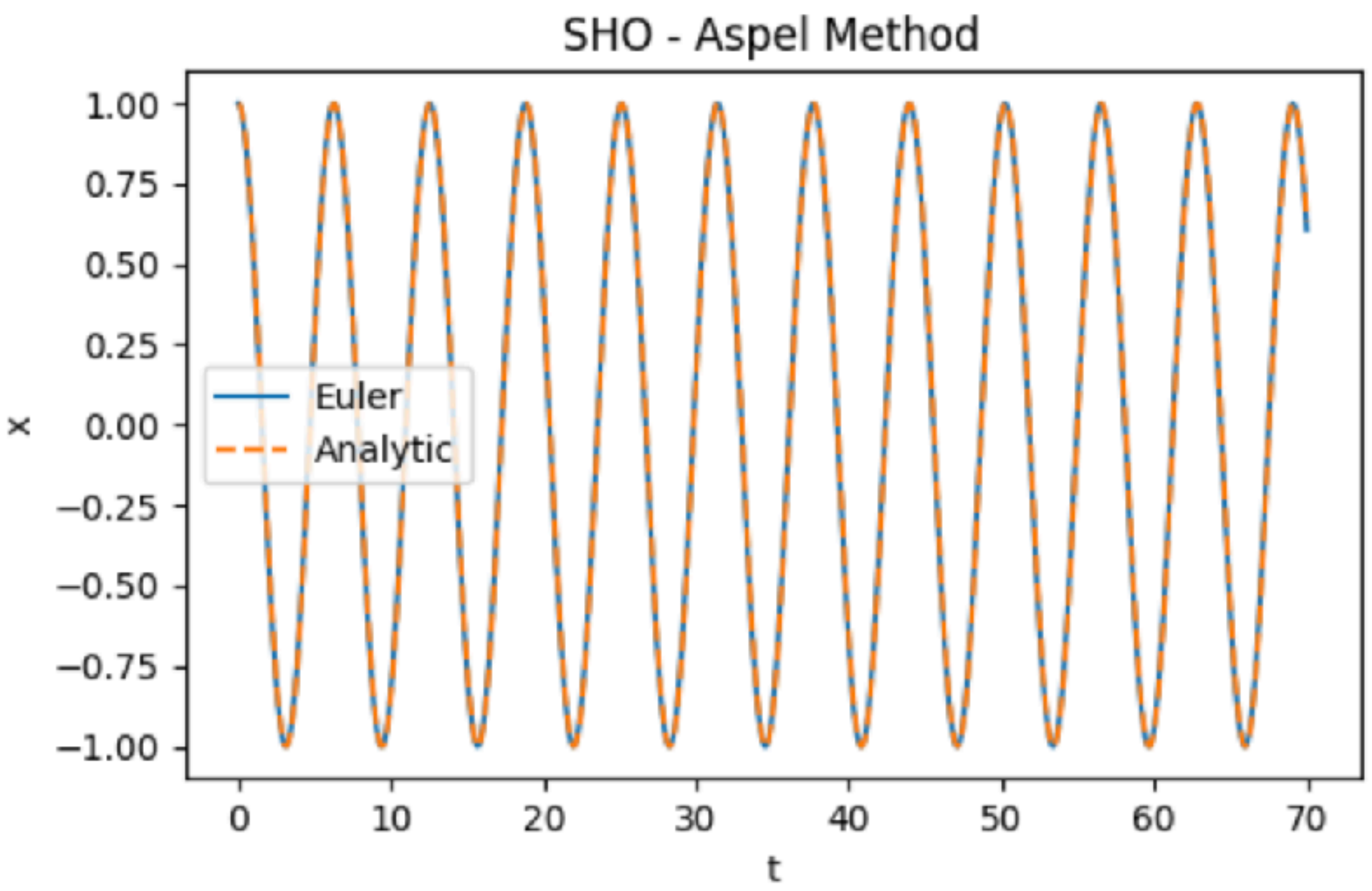
Euler Method

$\Delta t = 0.05$



Aspel Method

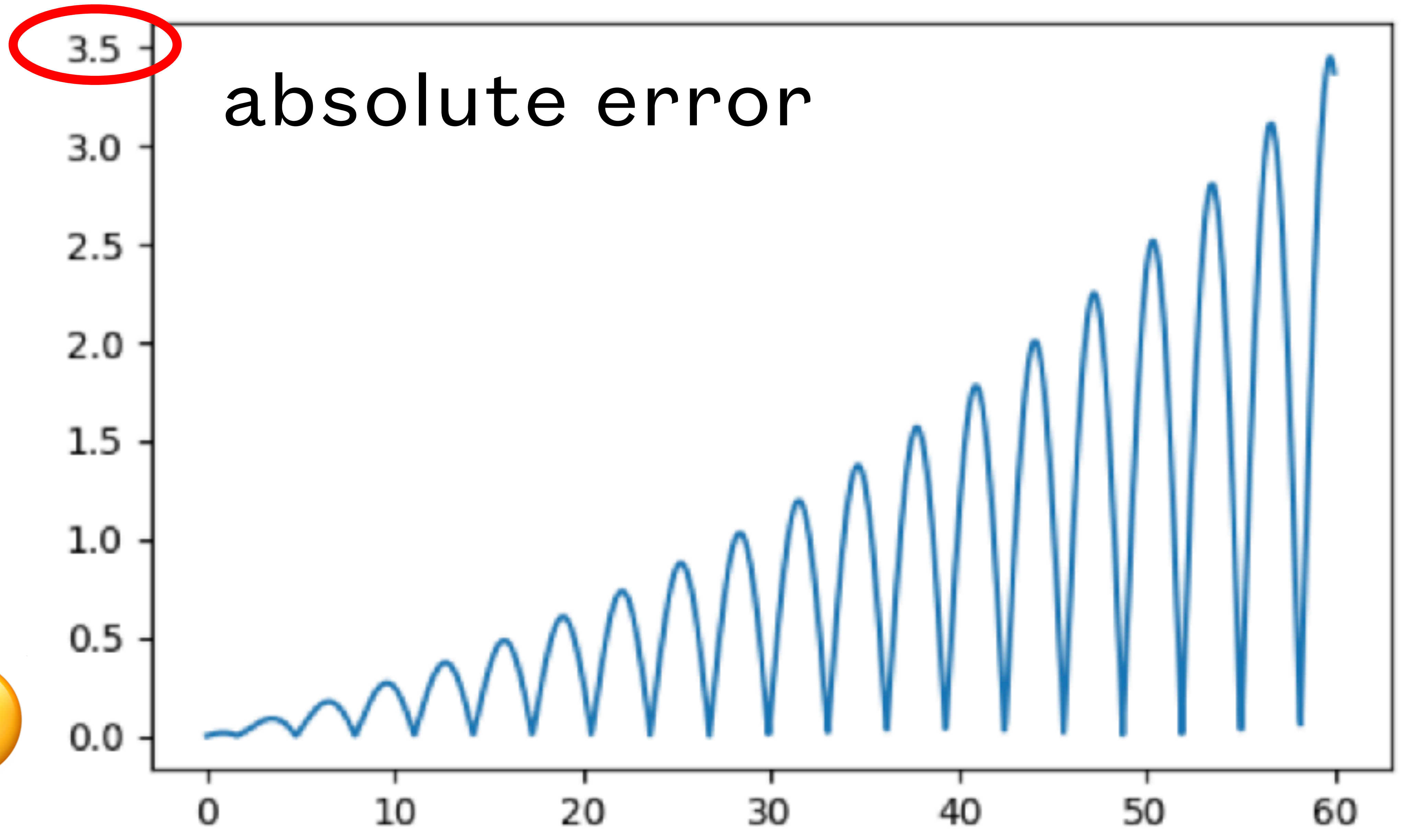
$\Delta t = 0.05$



Simple Harmonic Oscillator

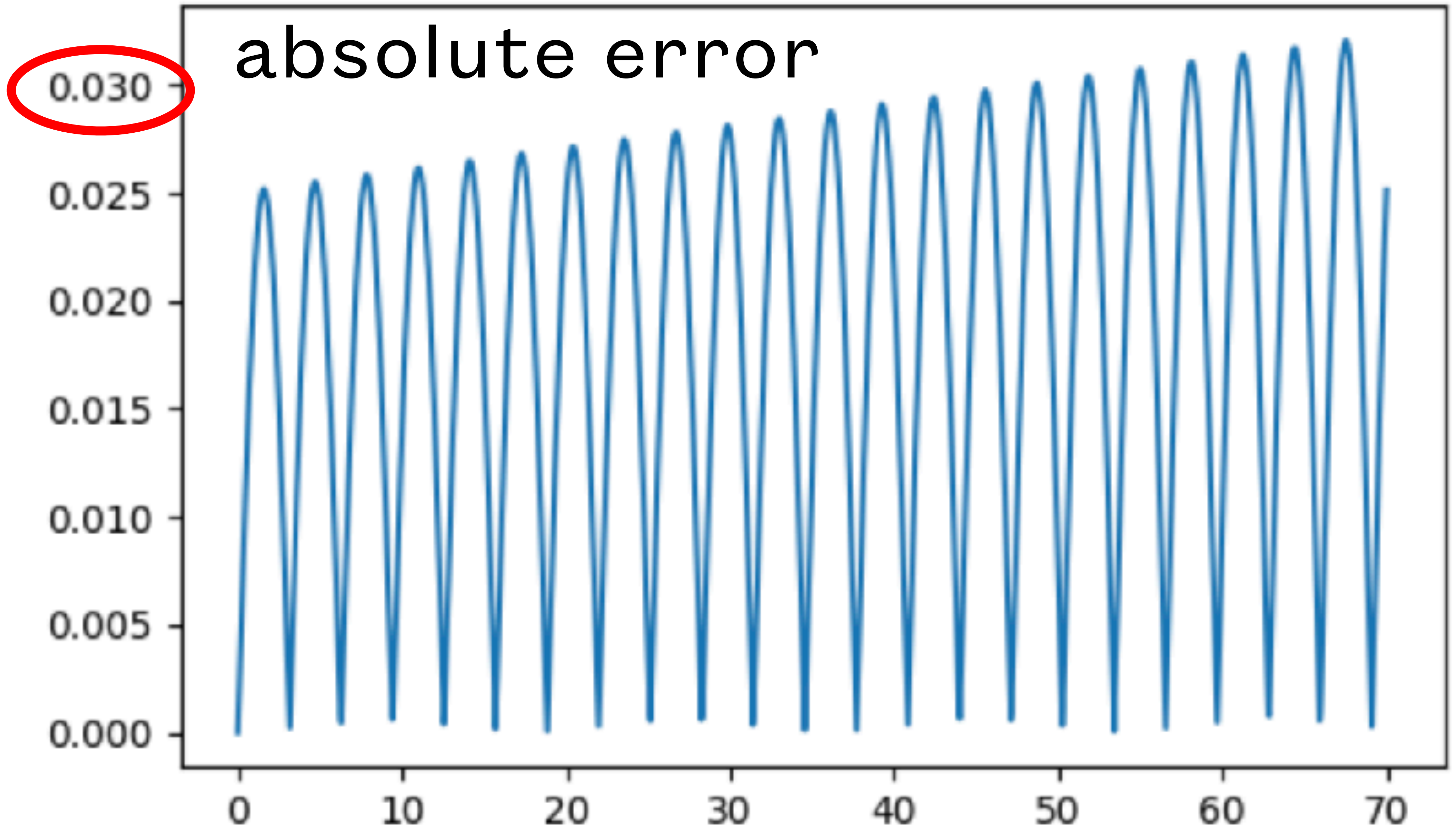
Euler Method

$\Delta t = 0.05$



Aspel Method

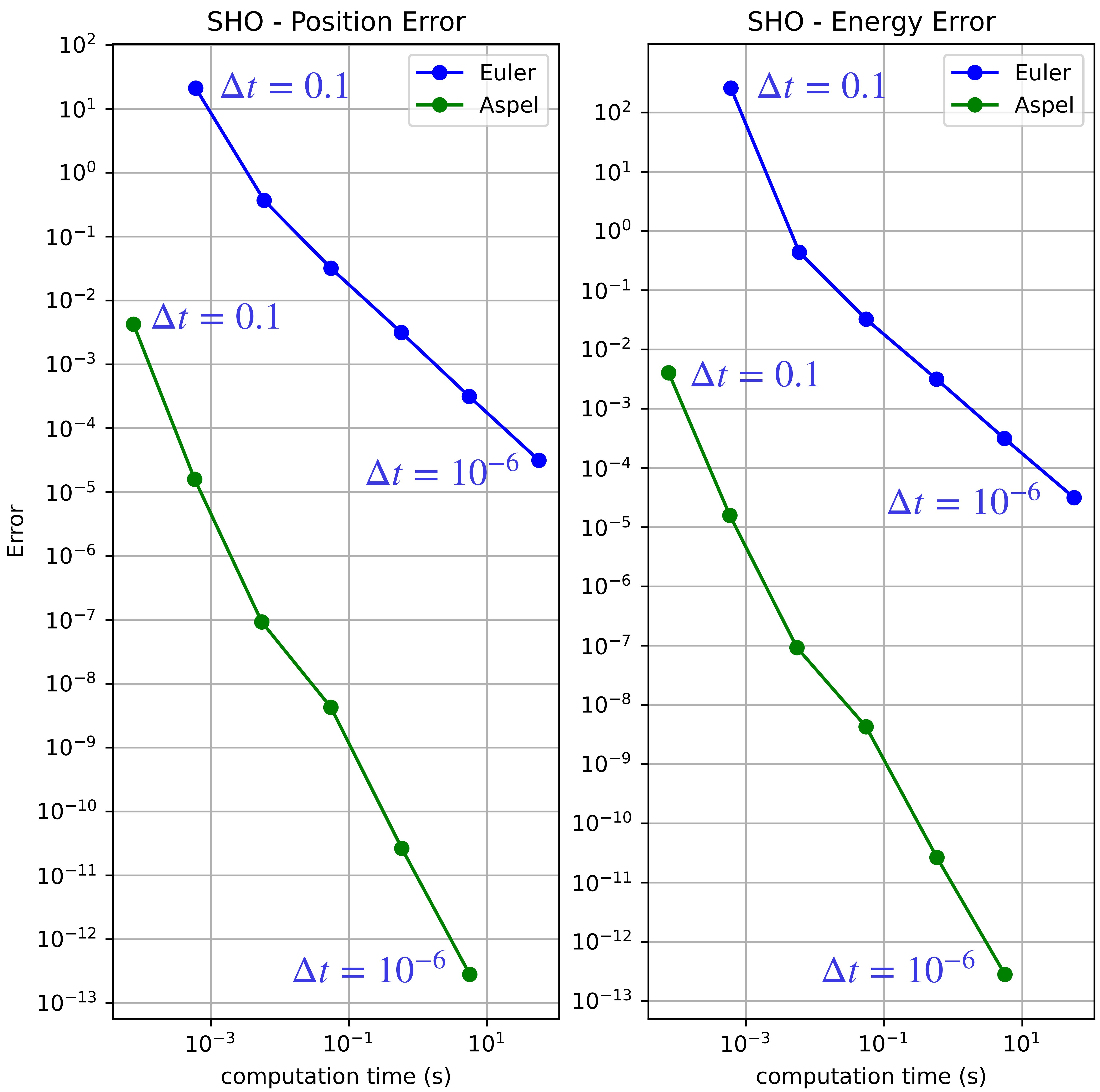
$\Delta t = 0.05$



In this example the Aspel method is >100x more accurate then Euler!

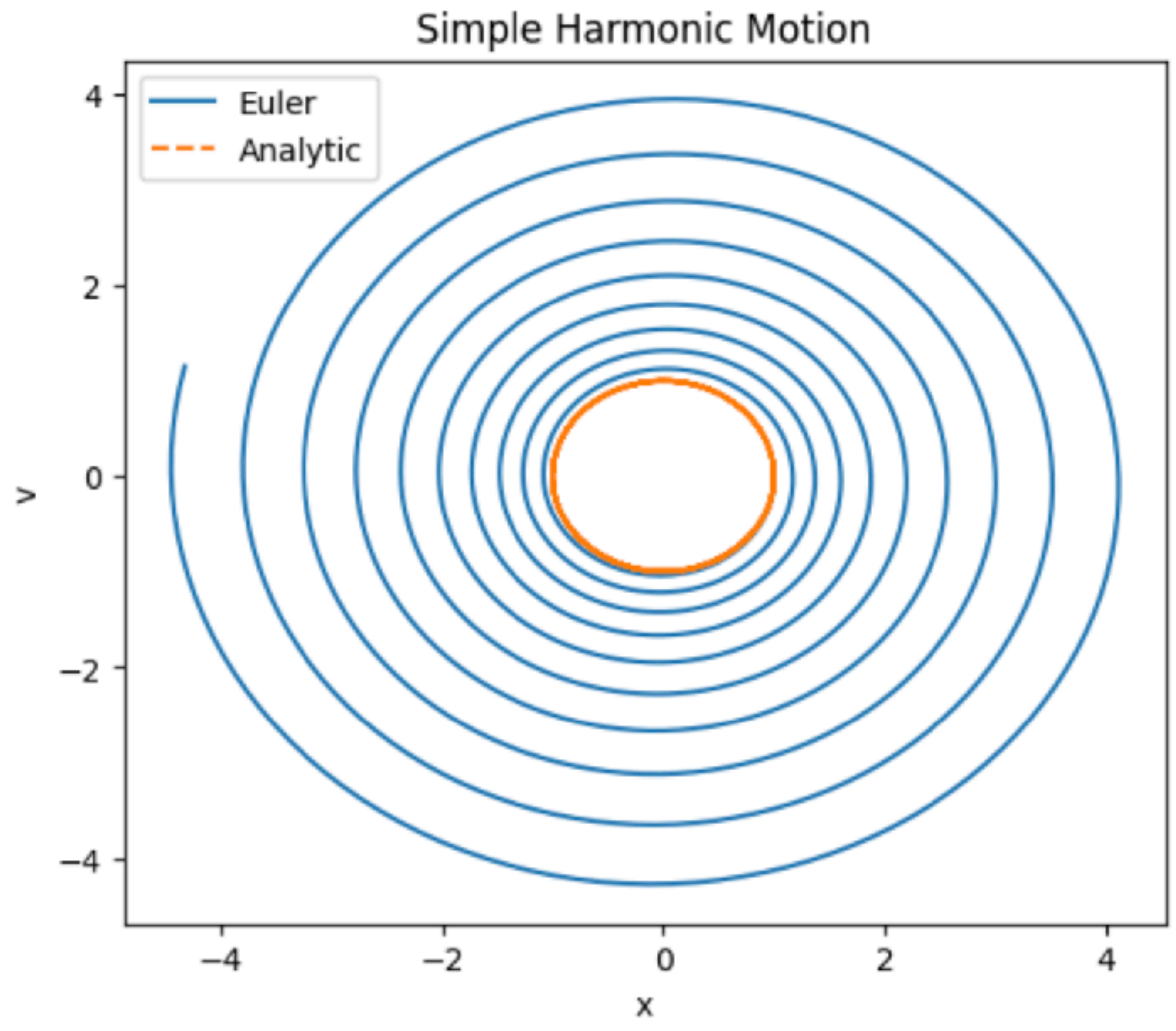
Effect of the time step on accuracy and computation time

Comparison between Euler and Euler-Cromer-Aspel Methods

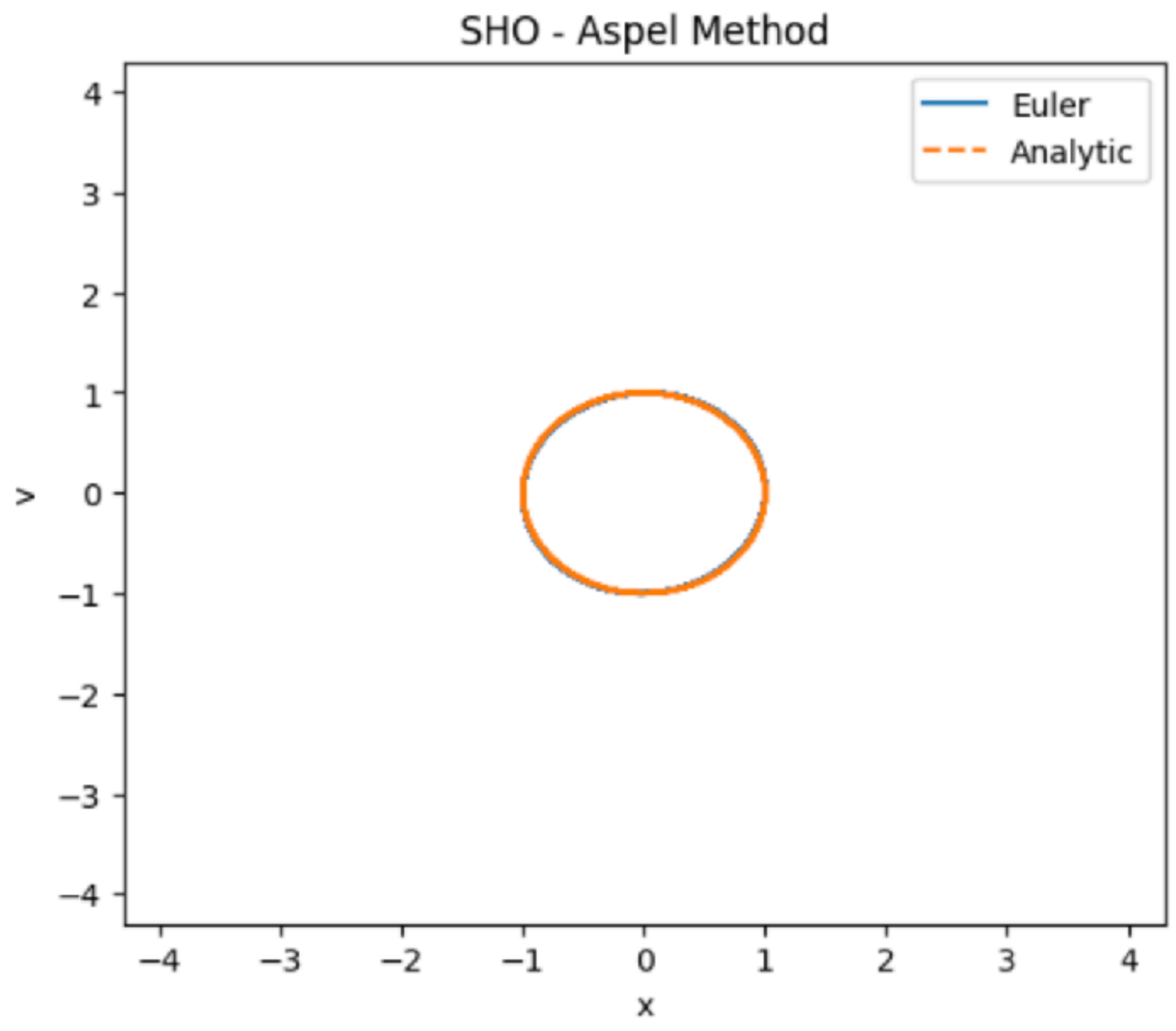


Simple Harmonic Oscillator: Phase Plot

Euler Method



Aspel Method



The Aspel method produces a trajectory in phase space that is much closer to being closed.

Next Lecture: second-order methods