LECTURE: EULER'S METHOD (I)

Today: How to solve ODE on a computer, because in practice it is impossible to solve them by hand.

The simplest way is by using **Euler's Method**:

1. Motivation

Example 1:

Find an approximate value of $y(2)$ where

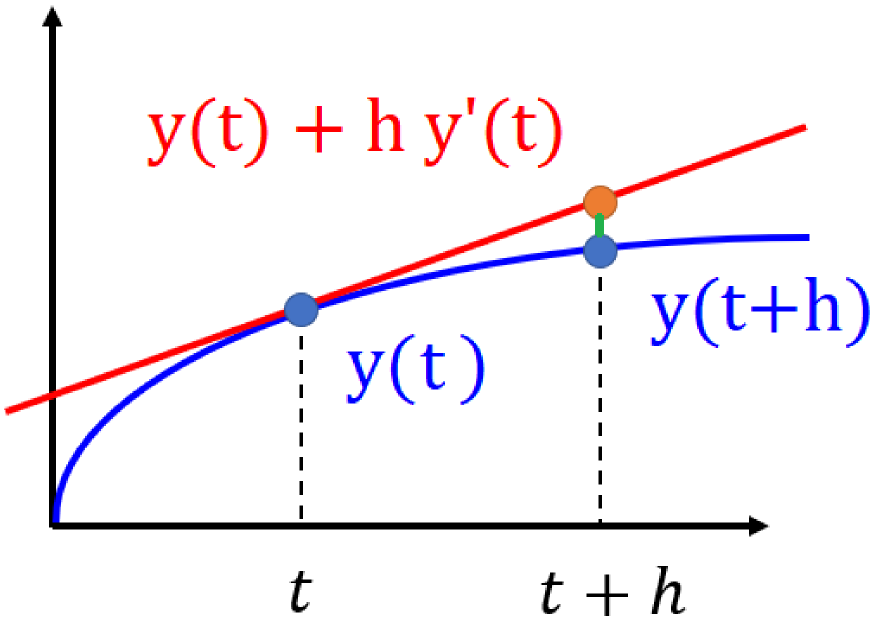
$$\begin{cases} y' = y^2 + 3t \\ y(0) = 2 \end{cases}$$

Main Idea: (Linear Approximation)

$$y'(t) = \lim_{h \to 0} \frac{y(t+h) - y(t)}{h}$$

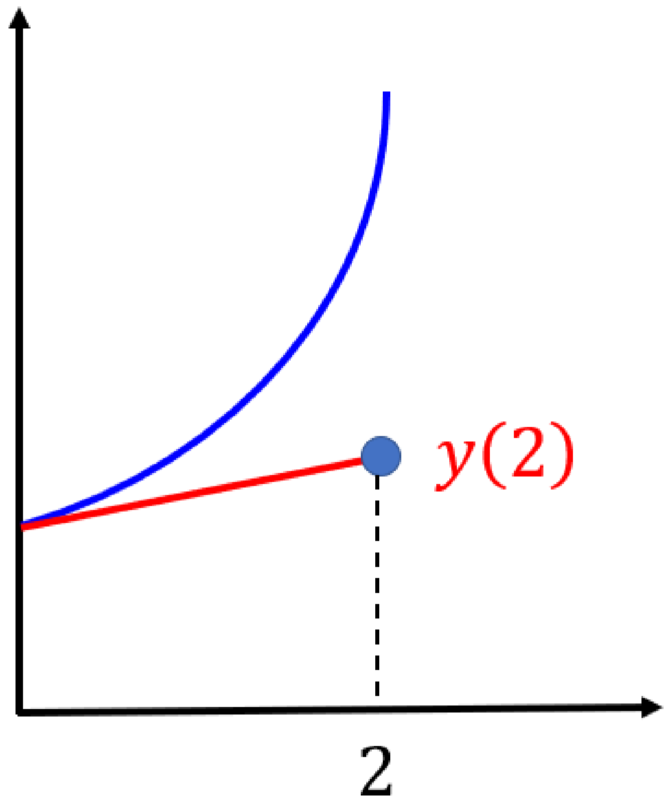
Therefore, for small h, we get

$$y'(t) \approx \frac{y(t+h) - y(t)}{h} \Rightarrow y(t+h) \approx y(t) + hy'(t)$$

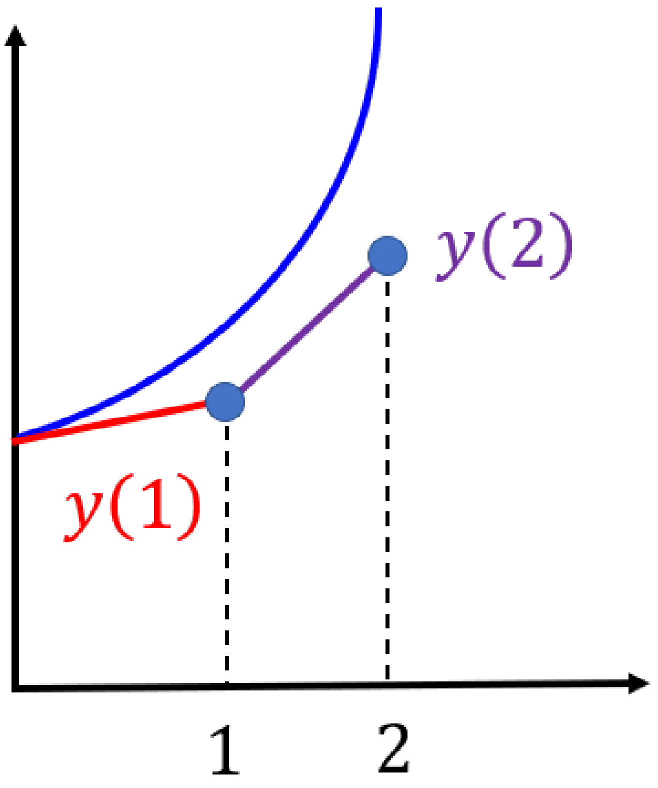


In this case with $t = 0$ and $h = 2$, we get

$$y(2) = y(0+2) \approx y(0)+2{\color{blue}y'(0)} \overset{\text{ODE}}{=} 2+2\times\left({\color{blue}(y(0))^2 + 3(0)}\right) = 2+2\times 2^2 = 10$$



Idea: Instead of doing one big step from 0 to 2, do two smaller steps, from 0 to 1 and from 1 to 2, like in the following picture:



Example 2:

Find an approximate value of $y(2)$ using $N = 2$ steps

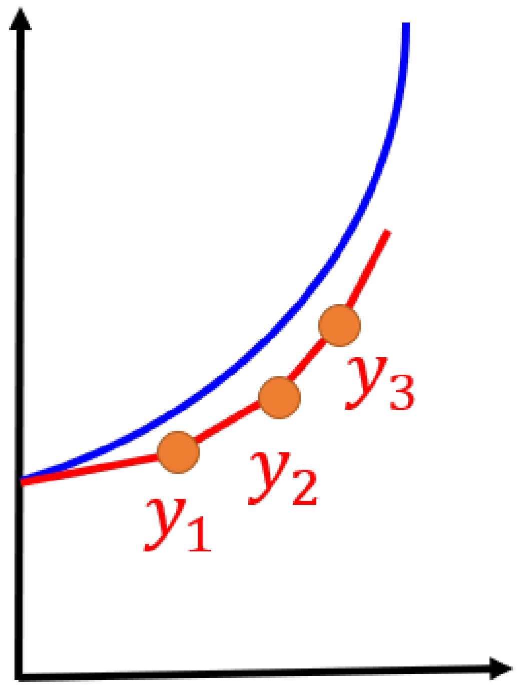
$$y(1) = y(0{+}1) \approx y(0){+}1y'(0) \overset{\text{ODE}}{=} 2{+}1{\times}\Big((y(0))^2 + 3(0)\Big) = 2{+}2^2 = 6$$

Use *that* value of $y(1)$ to calculate $y(2)$:

$$y(2) = y(1{+}1) = y(1){+}1y'(1) \overset{\text{ODE}}{=} 6{+}1{\times}\Big((y(1))^2 + 3(1)\Big) = 6{+}6^2{+}3 = 45$$

By using smaller steps, we get a much better approximation.

This is the essence of Euler's method: take lots of very tiny steps to get a better and better approximations.

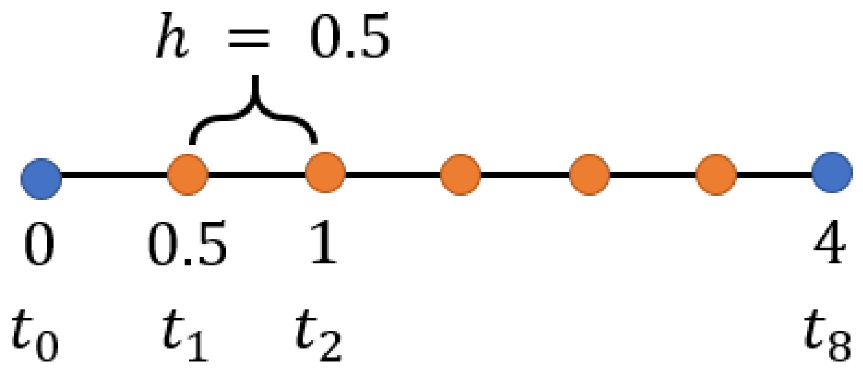


2. Forward Euler Method

Example 3:

Apply Euler with $N = 8$ steps to approximate the ODE on $[0, 4]$

$$\begin{cases} y' &= 3 + t - y \\ y(0) &= 1 \end{cases}$$



STEP 1: Find the step-size h

Definition:

If the interval is $[a, b]$ and the number of steps is N then

$$h = \frac{b-a}{N}$$

$$\text{Here } h = \frac{4-0}{8} = \frac{1}{2} = 0.5$$

This gives us our $t-$values

$$t_0 = 0 \qquad t_1 = 0.5 \qquad t_2 = 1 \qquad \cdots \qquad t_8 = 4$$

Definition:

$$t_n = a + nh$$

STEP 2:

Goal: Find the values

$$y_0 = y(t_0) \qquad y_1 = y(t_1) \qquad y_2 = y(t_2) \qquad \cdots \qquad y_8 = y(t_8)$$

$$y_0 = y(t_0) = y(0) = 1 \text{ Initial condition}$$

Euler's Method

$$y_{n+1} = y_n + hf(y_n, t_n)$$

Here f is given by the ODE, $y' = f(y,t) = 3 + t - y$

Why?

$$y_{n+1} = y(t_{n+1}) = y(t_n + h) \approx y(t_n) + hy'(t_n) \overset{\text{ODE}}{=} y_n + hf(y_n, t_n)\checkmark$$

$$\begin{aligned} y_1 =& y_0 + hf(y_0, t_0) \\ =& 1 + 0.5f(1, 0) \\ =& 1 + 0.5\,(3 + 0 - 1) \\ =& 1 + 0.5 \times 2 \\ =& 2 \end{aligned}$$

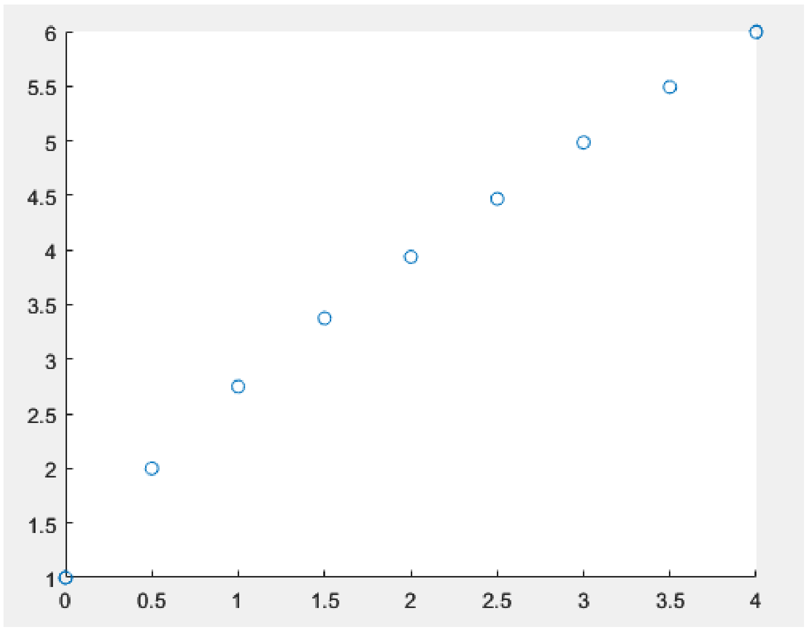
$$\begin{aligned} y_2 =& y_1 + hf(y_1, t_1) \\ =& 2 + 0.5f(2, 0.5) \\ =& 2 + 0.5\,(3 + 0.5 - 2) \\ =& 2 + 0.5 \times 1.5 \\ =& 2.75 \end{aligned}$$

$$\begin{aligned} y_3 =& y_2 + hf(y_2, t_2) \\ =& 2.75 + 0.5f(2.75, 1) \\ =& 2.75 + 0.5\,(3 + 1 - 2.75) \\ =& 2.75 + 0.5 \times 1.25 \\ =& 2.75 + 0.625 \\ =& 3.375 \end{aligned}$$

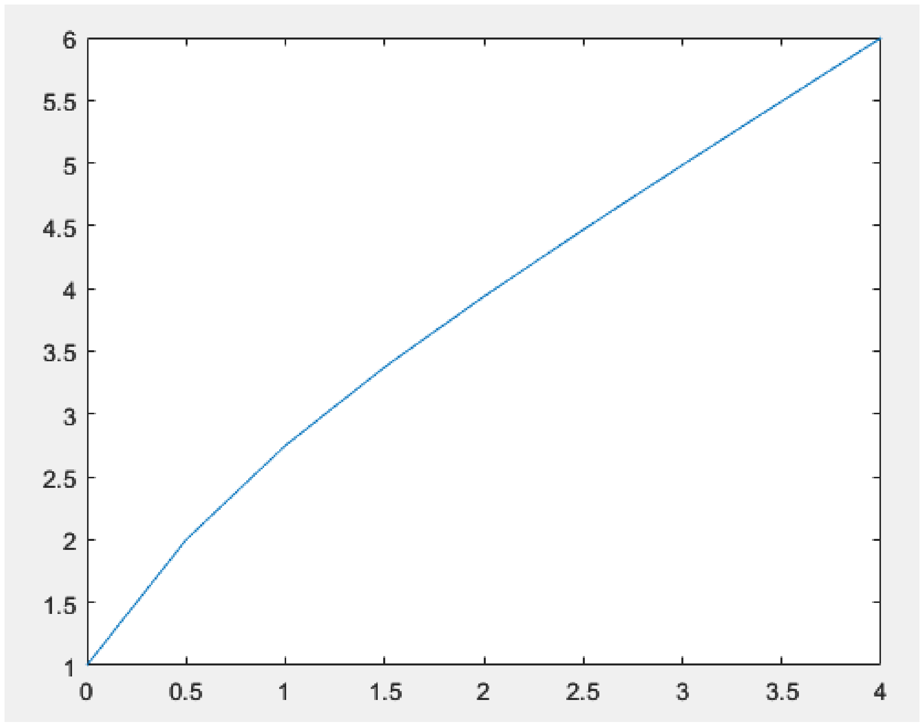
$$\begin{aligned} y_3 &\approx 3.375 \\ y_4 &\approx 3.94 \\ y_5 &\approx 4.47 \\ y_6 &\approx 4.98 \\ y_7 &\approx 5.49 \\ y_8 &\approx 5.99 \end{aligned}$$

3. Plotting

What's the point? Using this, you can then plot the (t_i, y_i) values on a chart, and then get an approximate graph of your solution.



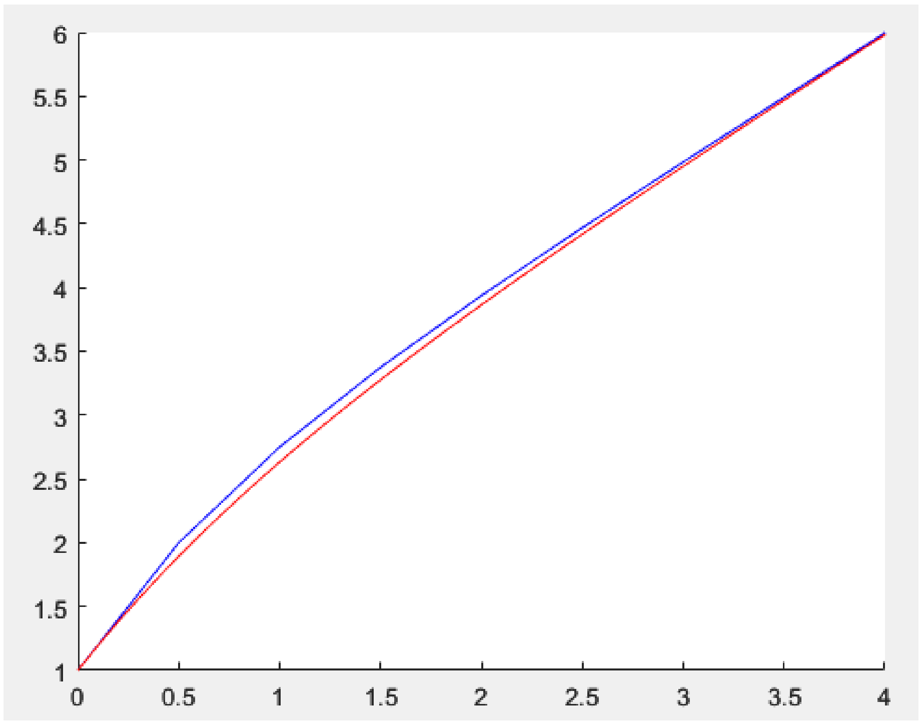
You can connect them if you want.



The real goal here is to figure out if we're as close to the true solution as possible. Luckily here we can solve the ODE using the integrating factor $e^{at} = e^t$ and you ultimately get

$$y = t + 2 - e^{-t}$$

The cool thing is that it works! Our approximation (blue graph) is close to the true solution (red graph)



And the smaller the h, the better the approximation.

4. More Practice

Example 4:

Use Euler's method with $N = 3$ to find y_0, y_1, y_2, y_3 on $[1, 7]$ where

$$\begin{cases} y' = 1 - t - 2y \\ y(1) = 3 \end{cases}$$

STEP 1: Find h

$$h = \frac{b-a}{N} = \frac{7-1}{3} = 2$$

STEP 2: Find t_0, t_1, t_2, t_3

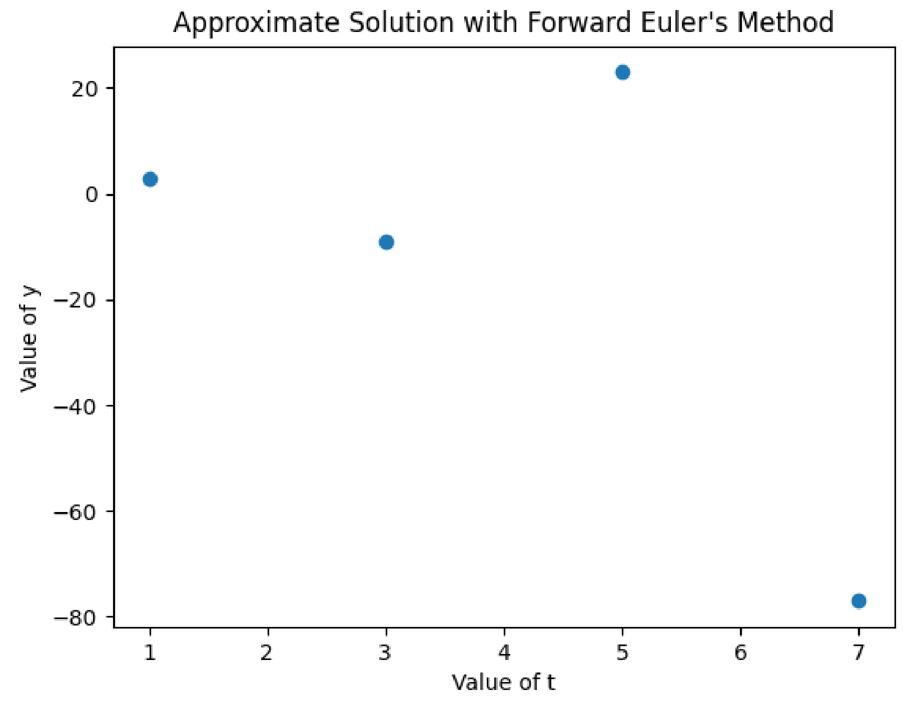
$t_0 = 1$ and $t_n = t_0 + nh = 1 + 2n$

$$t_0 = 1 \qquad t_1 = 3 \qquad t_3 = 5 \qquad t_4 = 7$$

STEP 3: Find y_0, y_1, y_2, y_3

$$\boxed{y_{n+1} = y_n + hf(y_n, t_n)}$$

$$\begin{aligned} y_0 =& y(1) = 3 \text{ Initial Condition} \\ y_1 =& y_0 + hf(y_0, t_0) = 3 + 2f(3,1) = 3 + 2\left(1 - 1 - 2(3)\right) = -9 \\ y_2 =& y_1 + hf(y_1, t_1) = -9 + 2f(-9,3) = -9 + 2\left(1 - 3 - 2(-9)\right) = -9 + 32 = 23 \\ y_3 =& y_2 + hf(y_2, t_2) = 23 + 2f(23,5) = 23 + 2\left(1 - 5 - 2(23)\right) = -77 \end{aligned}$$



Notice how the values here oscillate! We'll discuss this next time

5. Python Implementation

In practice, you don't do this by hand, but with the help of a computer.

Example 5:

Apply Euler's method with $N = 100$ steps to approximate the ODE on $[0, 10]$

$$\begin{cases} y' = & -y + \sin(t) \\ y(0) = & 1 \end{cases}$$

STEP 0: Start your Jupyter notebook by clicking on this website

STEP 1: **The basics:** Import the External Python libraries

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

The first package is used to perform numerical calculations, and the second one is used to plot graphs

STEP 2: **Basic data**

Here our interval is $[0, 10]$, so $a = 0$ and $b = 10$, and $y_0 = y(0) = 1$

```
a = 0
b = 10
y0 = 1
n = 101
h = (b-a)/(n-1)
```

Note: We chose $n = 101$ because we get 101 points in total. Here $N = n - 1 = 100$, as above.

STEP 3: $t-$values

Create your vector of $t-$values $t_0, t_1, \cdots, t_{100}$. Again, the vector has 101 entries, not 100

```
t = np.linspace(a,b,n)
```

$$t = \begin{bmatrix} 0 & 0.1 & 0.2 & \cdots & 10 \end{bmatrix}$$

linspace divides an interval evenly

STEP 4: Initialize the y-values

```
y = np.zeros([n])
```

This creates a vector y given by

$$y = \begin{bmatrix} 0 & 0 & \cdots & 0 \end{bmatrix}$$

And the idea is to update the vector y with the successive values

```
    y[0] = y0
    for i in range(1,n):
    y[i] = y[i-1] + h * (-y[i-1] + np.sin(t[i-1]))
```

(Don't forget about the indent)

This is the analog of

$$y_{n+1} = y_n + hf(y_n, t_n) = y_n + h\left(-y_n + \sin(t_n)\right)$$

It goes through each loop, and successively updates the vector y

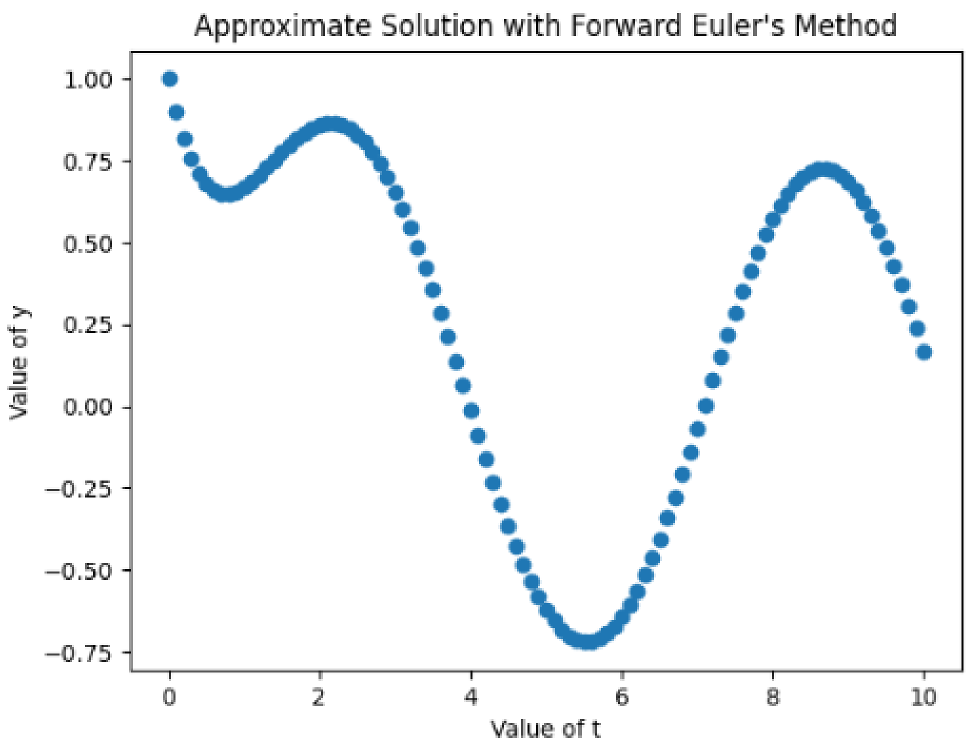
Note: If you want to print the t and y values, you would type

```
for i in range(n):
print(t[i],y[i])
```

STEP 5: Plot

```
plt.plot(t,y,'o')
plt.xlabel("Value of t")
plt.ylabel("Value of y")
plt.title("Approximate Solution with Forward Euler's Method")
plt.show()
```

We then obtain the following beautiful plot:

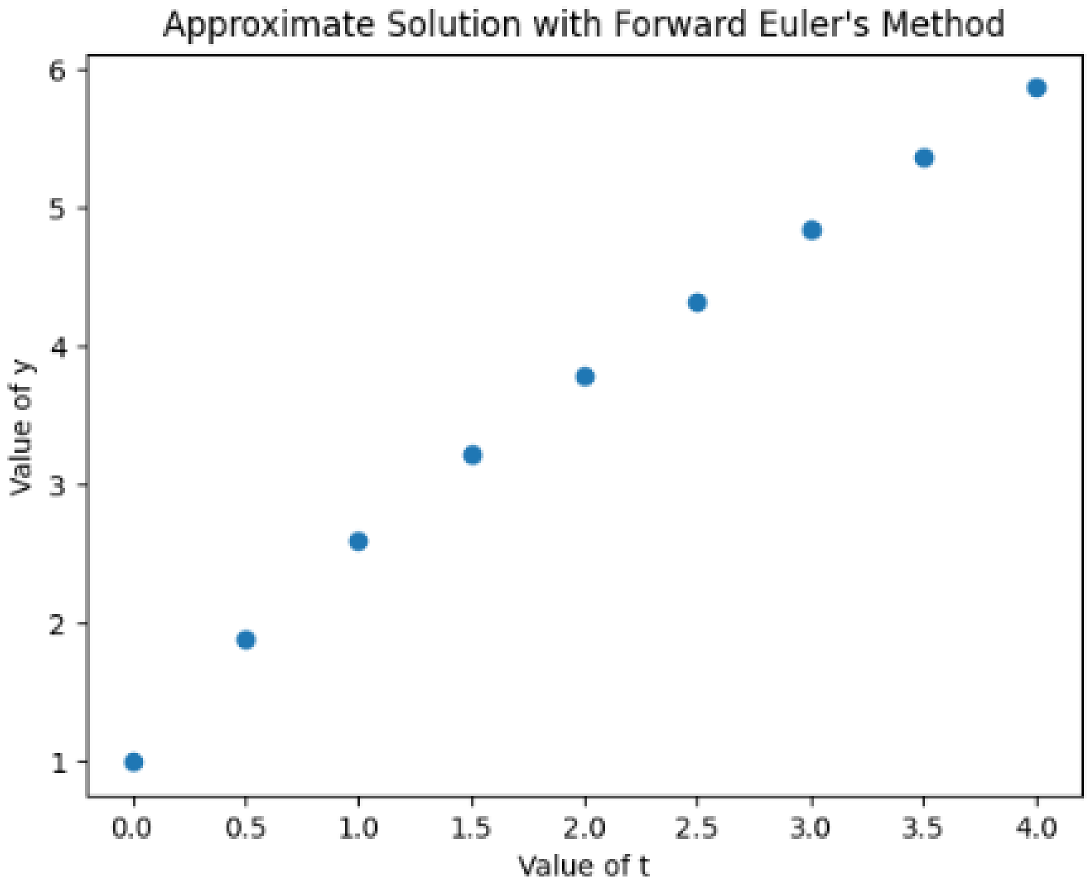


Summary of Code:

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

a = 0
b = 10
y0 = 1

n = 101
h = (b-a)/(n-1)

t = np.linspace(a,b,n)

y = np.zeros([n])

y[0] = y0
for i in range(1,n):
y[i] = y[i-1] + h * (-y[i-1] + np.sin(t[i-1]))

for i in range(n):
print(t[i],y[i])

plt.plot(t,y,'o')
plt.xlabel("Value of t")
plt.ylabel("Value of y")
plt.title("Approximate Solution with Forward Euler's Method")
plt.show()
```

Note: If you want to re-do the example with $y' = 3 + t - y$ above, the only changes you would have to make are

```
b = 4
n = 9
y[i] = y[i-1] + h * (3 + t[i-1]-y[i-1])
```


Approximate Solution with Forward Euler's Method
Value of y
Value of t