Lab-10 - Forward Euler and Backward Euler

Kalp Shah - 202201457

Rakshit Pandhi - 202201426

Forward Euler

Part 1

Constructed a class for calculating the solution of the differential equation using Forward Euler Method.

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

class EulerMethod:
    def __init__(self, f, x0, y0, x_end, actual_solution=None):
        self.f = f
        self.x0 = x0
        self.y0 = y0
        self.x_end = x_end
        self.actual_solution = actual_solution

    def solve(self, h):
        x_values = np.arange(self.x0, self.x_end + h, h)
        y_values = np.zeros(len(x_values))
        y_values[0] = self.y0

        for i in range(1, len(x_values)):
            y_values[i] = y_values[i - 1] + h * self.f(x_values[i -
1], y_values[i - 1])

        return x_values, y_values

    def plot_solutions(self, step_sizes):
        plt.figure(figsize=(10, 6))

        # Plot Euler's solutions for each step size
        for h in step_sizes:
            x_values, y_values = self.solve(h)
            plt.plot(x_values, y_values, label=f"Euler's Method
(h={h})", linestyle='--', marker='*')

        # Plot the actual solution if provided
        if self.actual_solution:
```

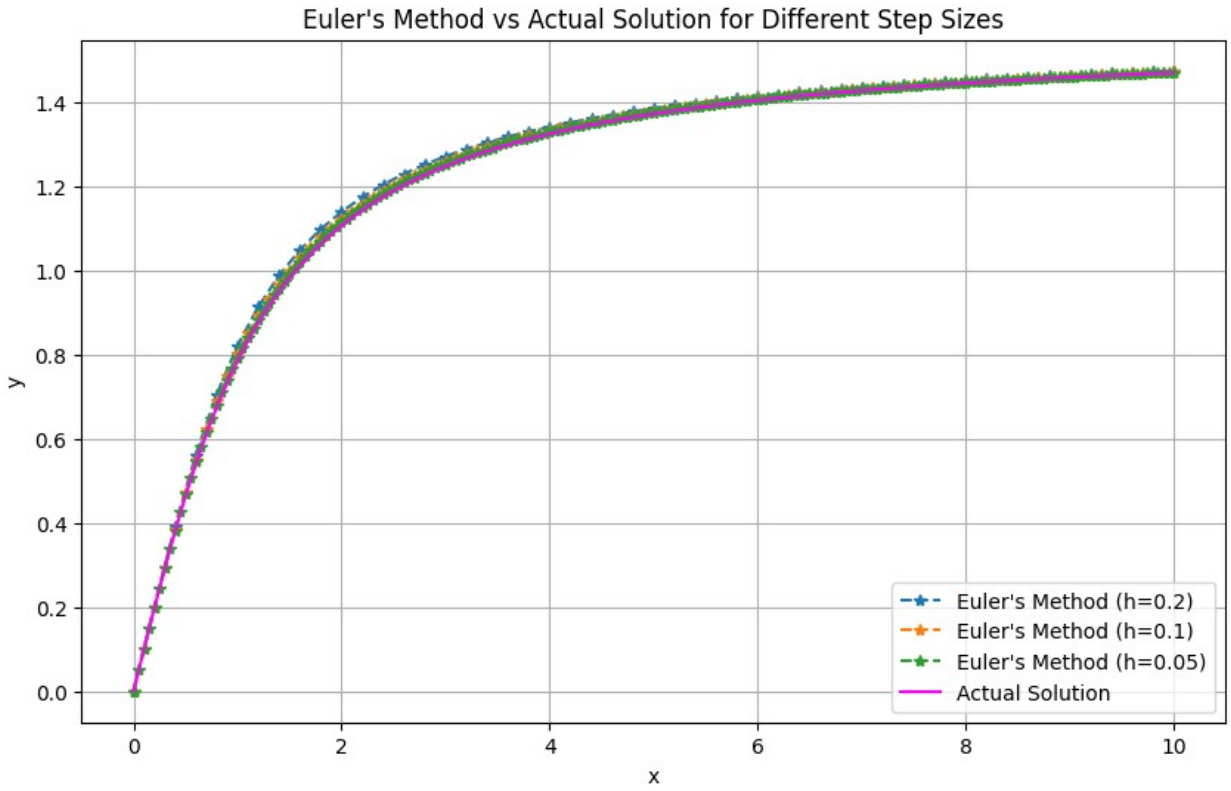
```
            x_fine = np.linspace(self.x0, self.x_end, 1000)
            actual_values = self.actual_solution(x_fine)
            plt.plot(x_fine, actual_values, label="Actual Solution",
color='magenta')

        plt.xlabel('x')
        plt.ylabel('y')
        plt.title("Euler's Method vs Actual Solution for Different
Step Sizes")
        plt.grid(True)
        plt.legend()
        plt.show()

        # Error plot for each step size
        plt.figure(figsize=(10, 6))
        for h in step_sizes:
            x_values, y_values = self.solve(h)
            actual_values = self.actual_solution(x_values)
            plt.plot(x_values, abs(actual_values - y_values),
label=f"Error (h={h})")

        plt.xlabel('x')
        plt.ylabel('Error')
        plt.title("Error between Euler's Method and Actual Solution
for Different Step Sizes")
        plt.grid(True)
        plt.legend()
        plt.show()
```

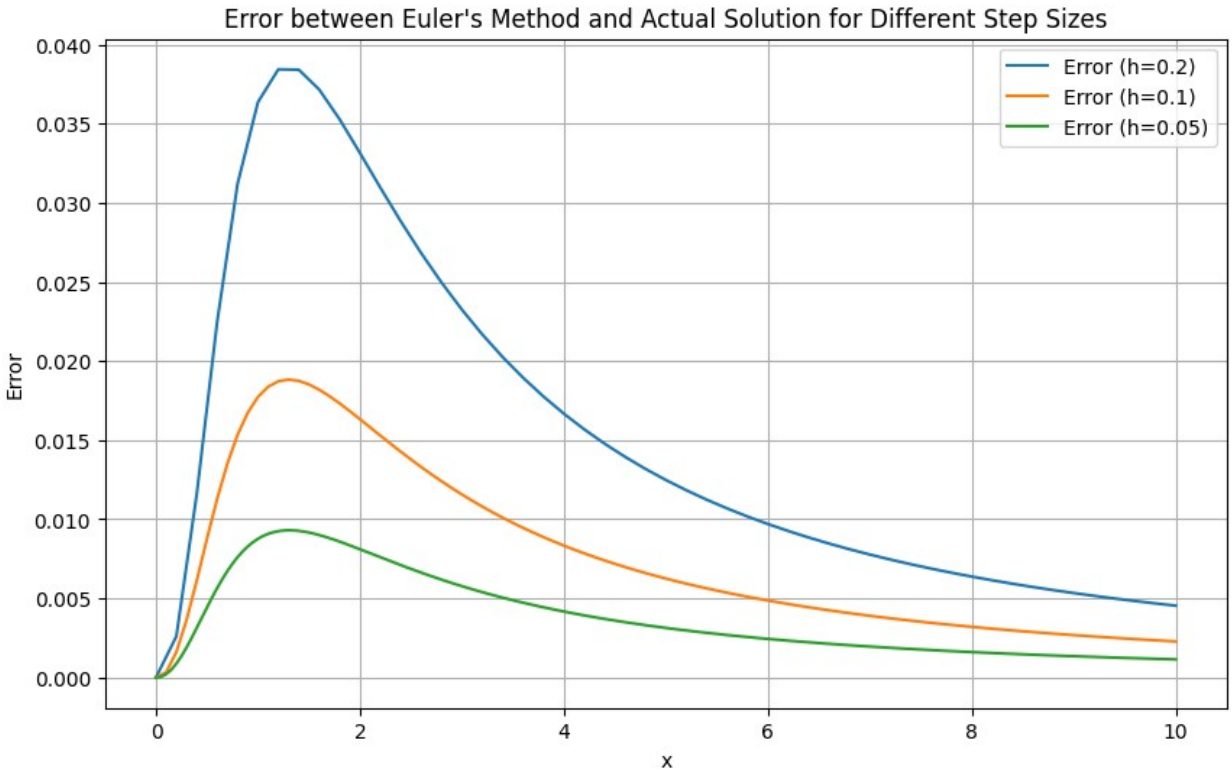
Question 1

From this snippet onwards only the class has been called for different plots required for different functions.

```
if __name__ == "__main__":
    def f(x, y):
        return (np.cos(y))**2

    def actual_solution(x):
        return np.arctan(x)

    x0 = 0
    y0 = 0
    x_end = 10
    step_sizes = [0.2, 0.1, 0.05]

    euler = EulerMethod(f, x0, y0, x_end,
actual_solution=actual_solution)
```

```
euler.plot_solutions(step_sizes)
```



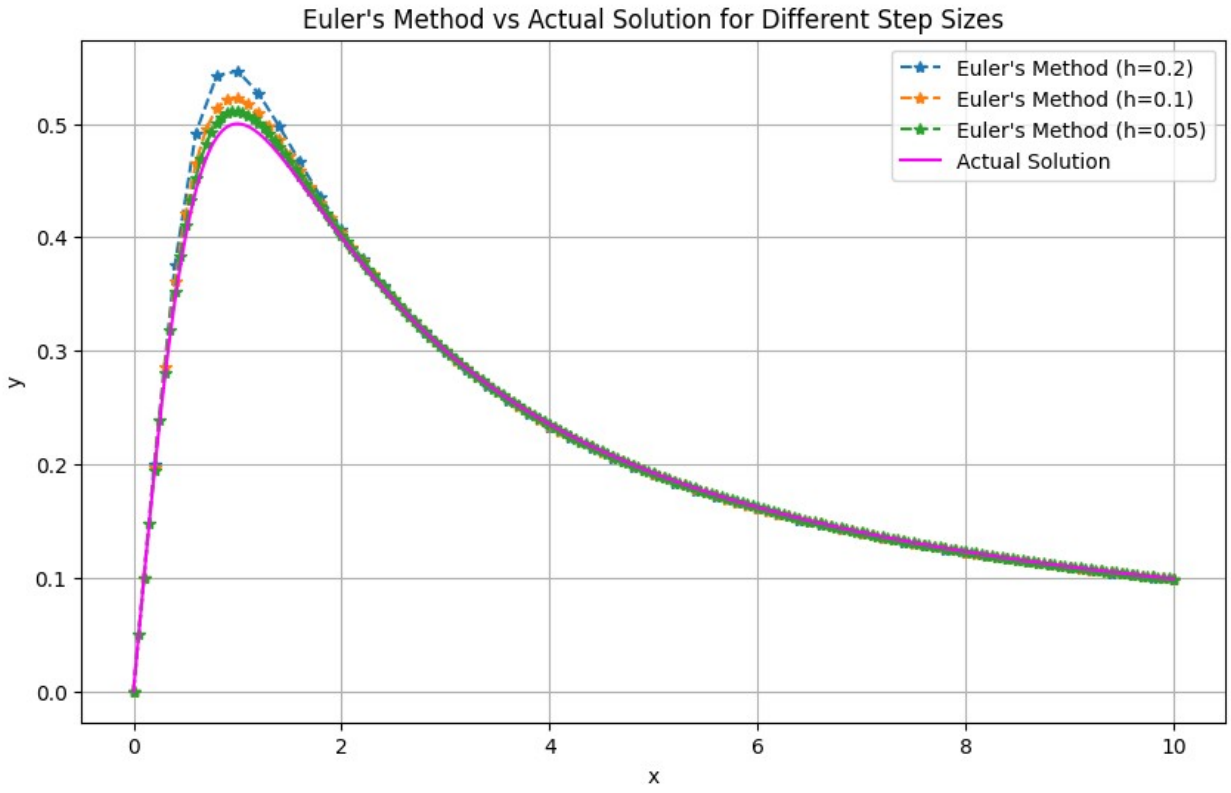


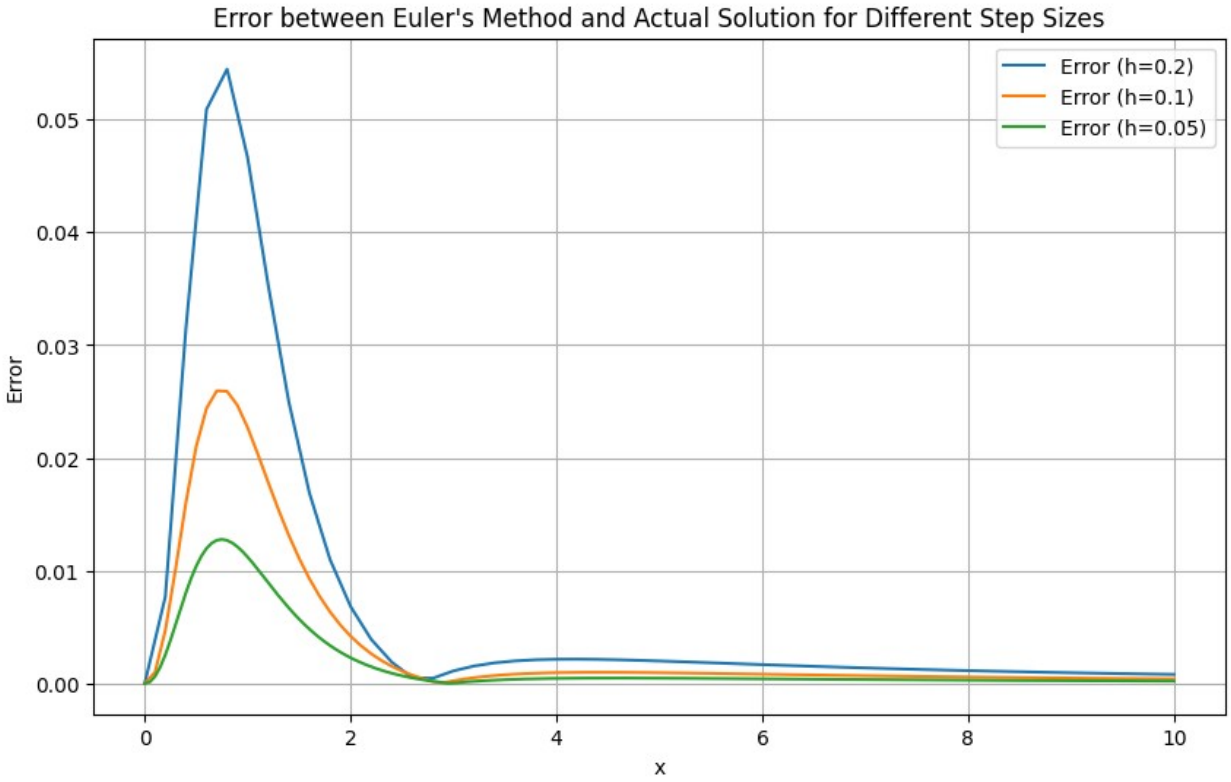
According to the theory as we studied that when the value of h gets halved the error also gets halved which can be seen in this plot as well.

Also, the euler plot is quite close to the actual solution with some error or perturbration for different h values.

Question 2

```
if __name__ == "__main__":
    def f(x, y):
        return (1/(1+x**2)) - 2*(y**2)

    def actual_solution(x):
        return x/(1+x**2)

    x0 = 0
    y0 = 0
    x_end = 10
    step_sizes = [0.2, 0.1, 0.05]

    euler = EulerMethod(f, x0, y0, x_end,
actual_solution=actual_solution)

    euler.plot_solutions(step_sizes)
```

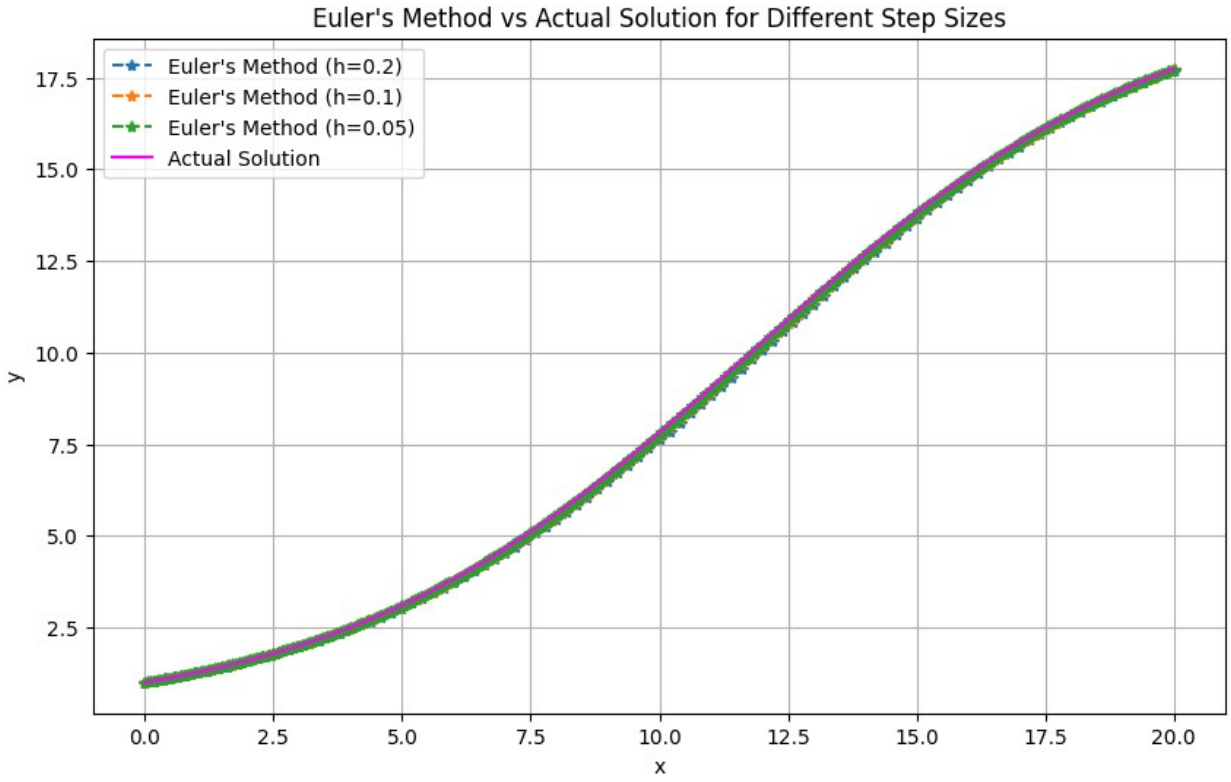


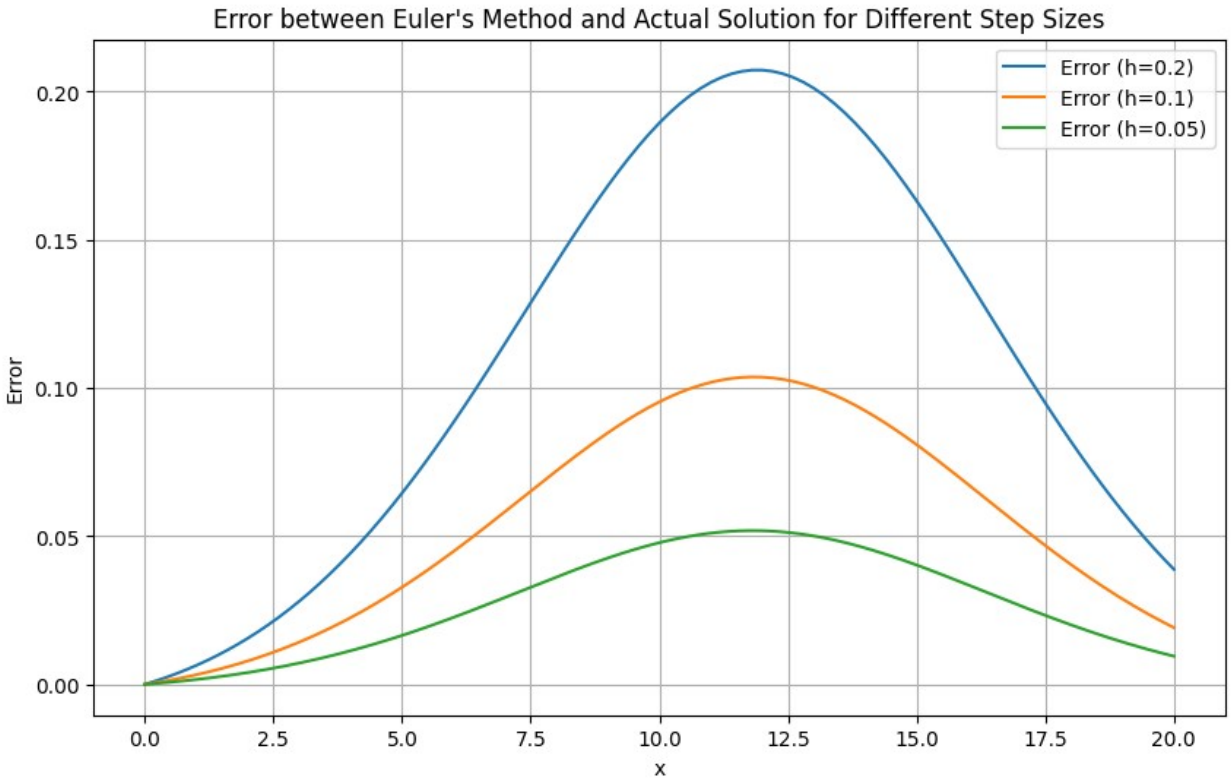


Here also we can see that error is getting halved whenever h is getting halved.

Question-3

```
if __name__ == "__main__":
    def f(x, y):
        return (y/4)*(1-(y/20))
    def actual_solution(x):
        return 20/(1+19*np.exp(-x/4))
    x0 = 0
    y0 = 1
    x_end = 20
    step_sizes = [0.2, 0.1, 0.05]

    euler = EulerMethod(f, x0, y0, x_end,
actual_solution=actual_solution)

    euler.plot_solutions(step_sizes)
```



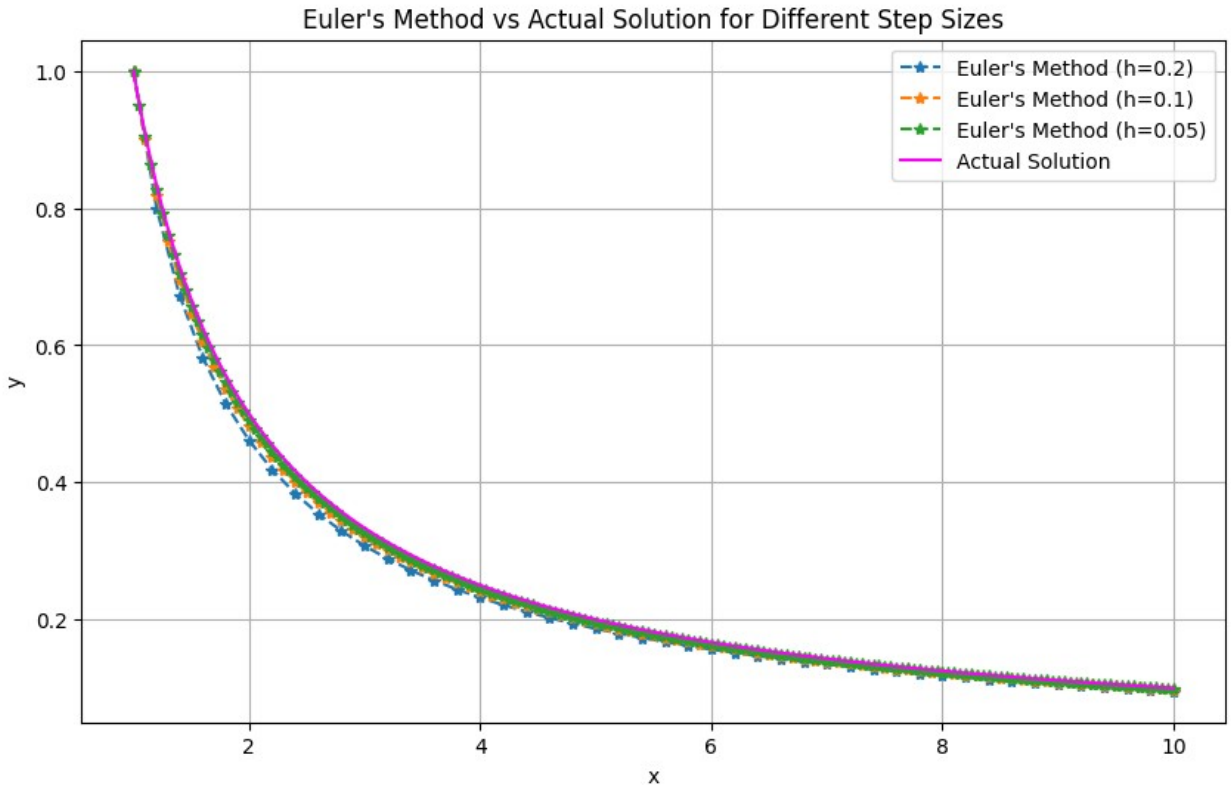


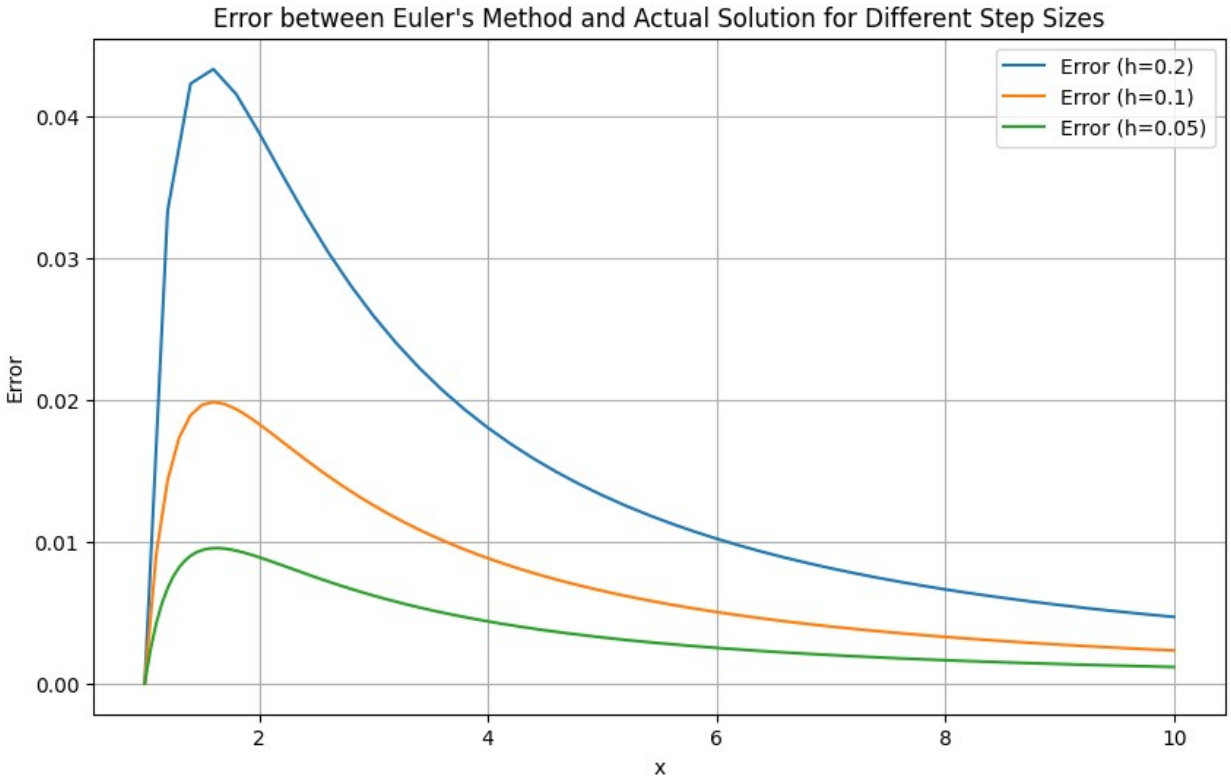
Similarly here also error gets halved whenever h getting halved.

Also, the euler plot is quite close to the actual solution with some error or perturbration for different h values.

Question-4

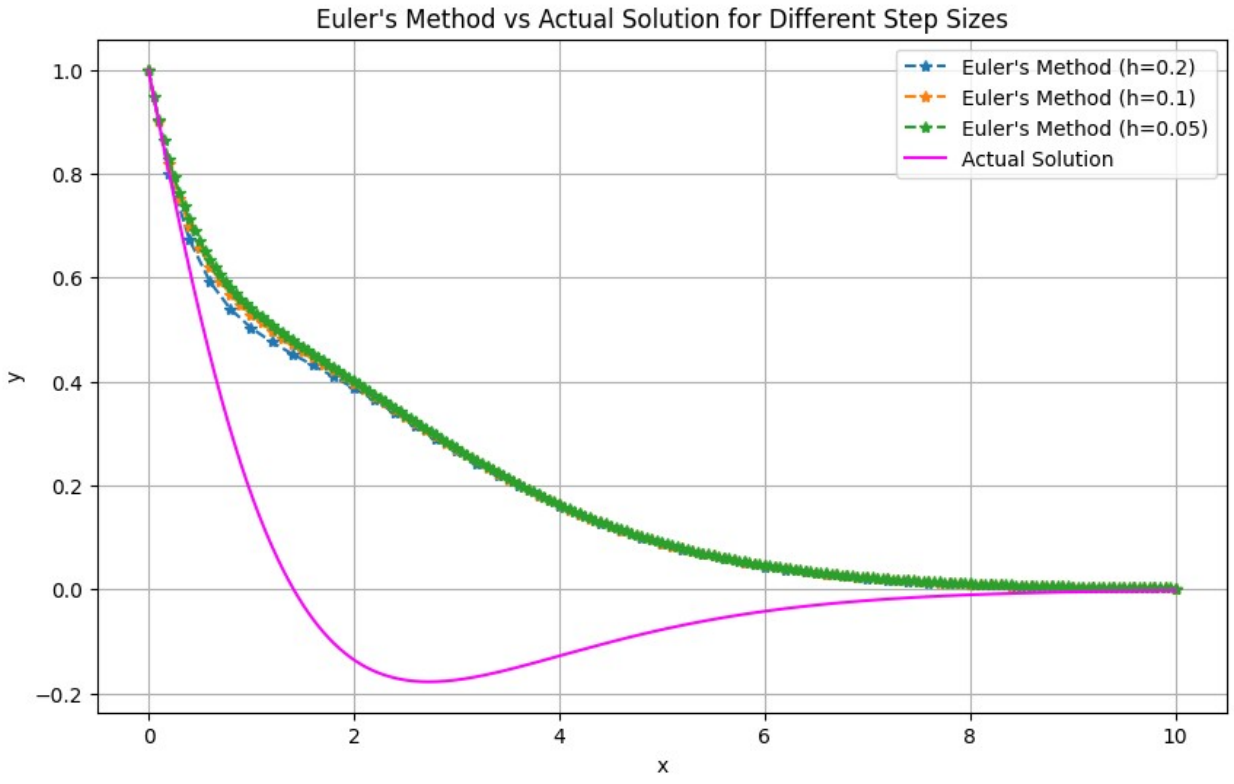
```
if __name__ == "__main__":
    def f(x, y):
        return -(y*y)
    def actual_solution(x):
        return 1/x

    x0 = 1
    y0 = 1
    x_end = 10
    step_sizes = [0.2, 0.1, 0.05]

    euler = EulerMethod(f, x0, y0, x_end,
actual_solution=actual_solution)

    euler.plot_solutions(step_sizes)
```

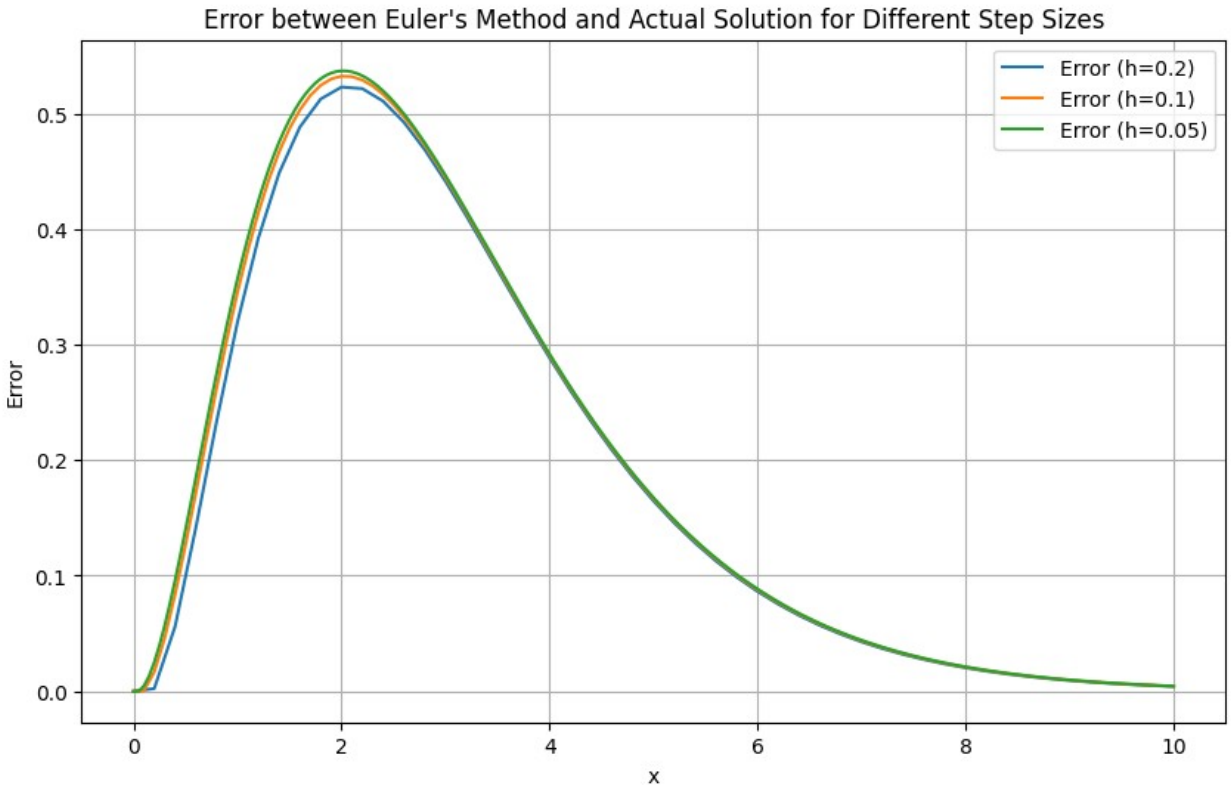

Euler's Method vs Actual Solution for Different Step Sizes
Euler's Method (h=0.2)
Euler's Method (h=0.1)
Euler's Method (h=0.05)
Actual Solution
y
x
1.0
0.8
0.6
0.4
0.2
2
4
6
8
10


Error between Euler's Method and Actual Solution for Different Step Sizes
Error (h=0.2)
Error (h=0.1)
Error (h=0.05)
Error
x
0.04
0.03
0.02
0.01
0.00
2
4
6
8
10

Question-5

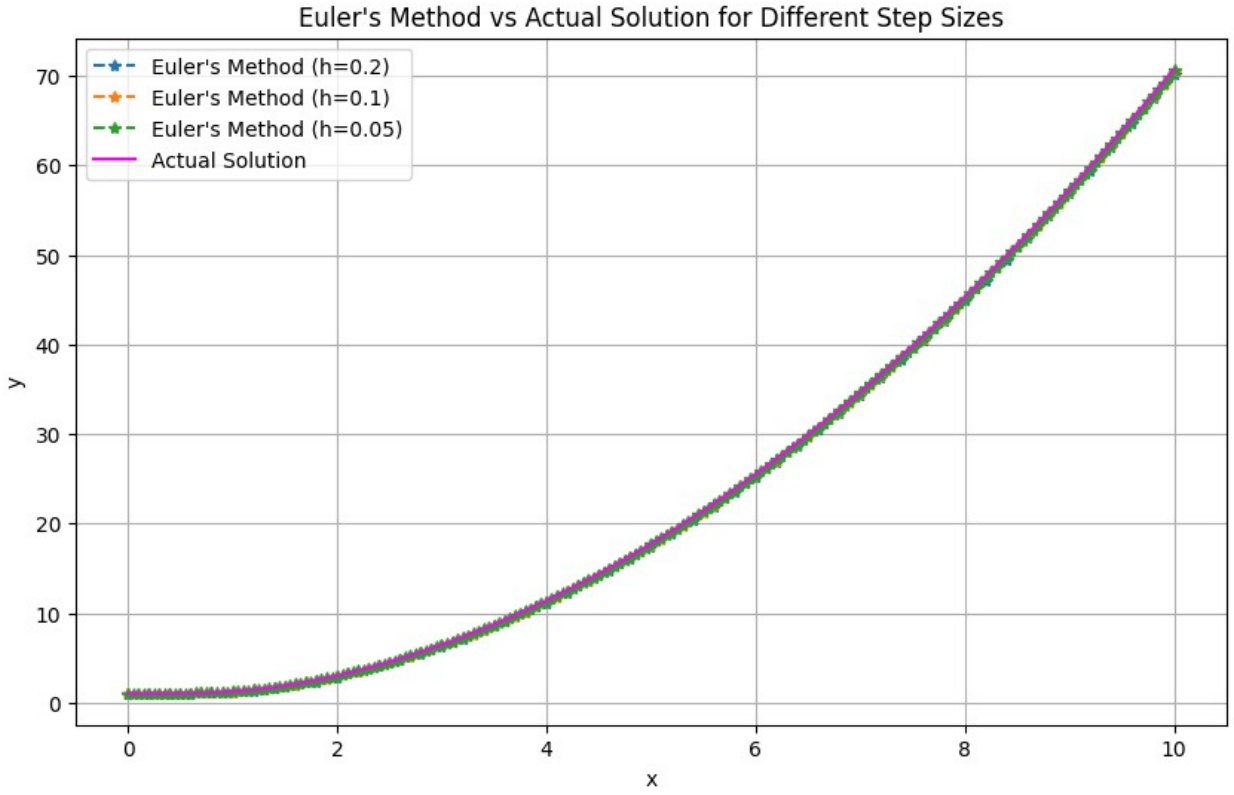
```
if __name__ == "__main__":
    def f(x, y):
        return x*np.exp(-x) - y

    def actual_solution(x):
        return (1-0.5*(x**2))*np.exp(-x)

    x0 = 0
    y0 = 1
    x_end = 10
    step_sizes = [0.2, 0.1, 0.05]

    euler = EulerMethod(f, x0, y0, x_end,
actual_solution=actual_solution)

    euler.plot_solutions(step_sizes)
```

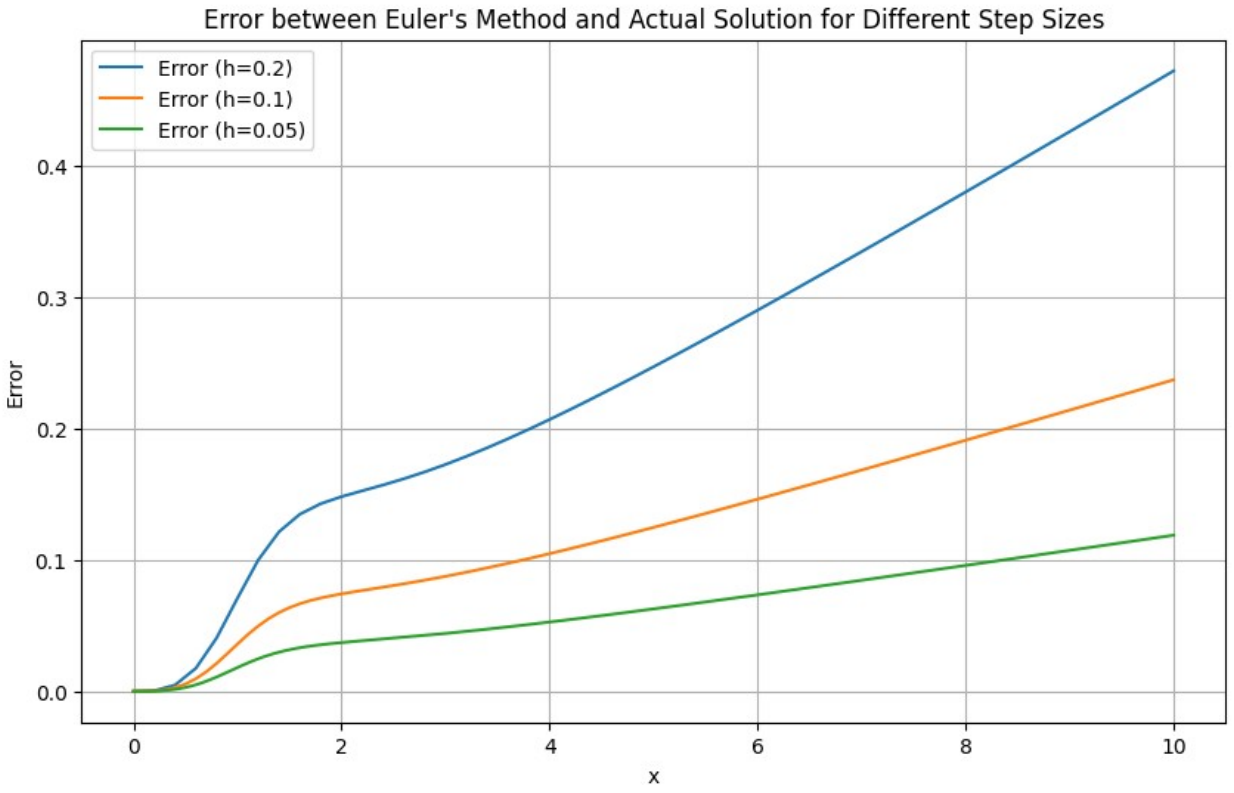




Question-6

```
if __name__ == "__main__":
    def f(x, y):
        return x**3/y

    def actual_solution(x):
        return (0.5*x**4+1)**(0.5)

    x0 = 0
    y0 = 1
    x_end = 10
    step_sizes = [0.2, 0.1, 0.05]

    euler = EulerMethod(f, x0, y0, x_end,
actual_solution=actual_solution)

    euler.plot_solutions(step_sizes)
```


Euler's Method vs Actual Solution for Different Step Sizes
Euler's Method (h=0.2)
Euler's Method (h=0.1)
Euler's Method (h=0.05)
Actual Solution
y
x
0
10
20
30
40
50
60
70
0
2
4
6
8
10


Error between Euler's Method and Actual Solution for Different Step Sizes
Error (h=0.2)
Error (h=0.1)
Error (h=0.05)
Error
x
0.0
0.1
0.2
0.3
0.4
0
2
4
6
8
10

In a nut shell the take away form this was that whenever h values decreases and becomes close to zero i.e. infinitisemal small as well the error also reduces as the step interval (h) is very small.

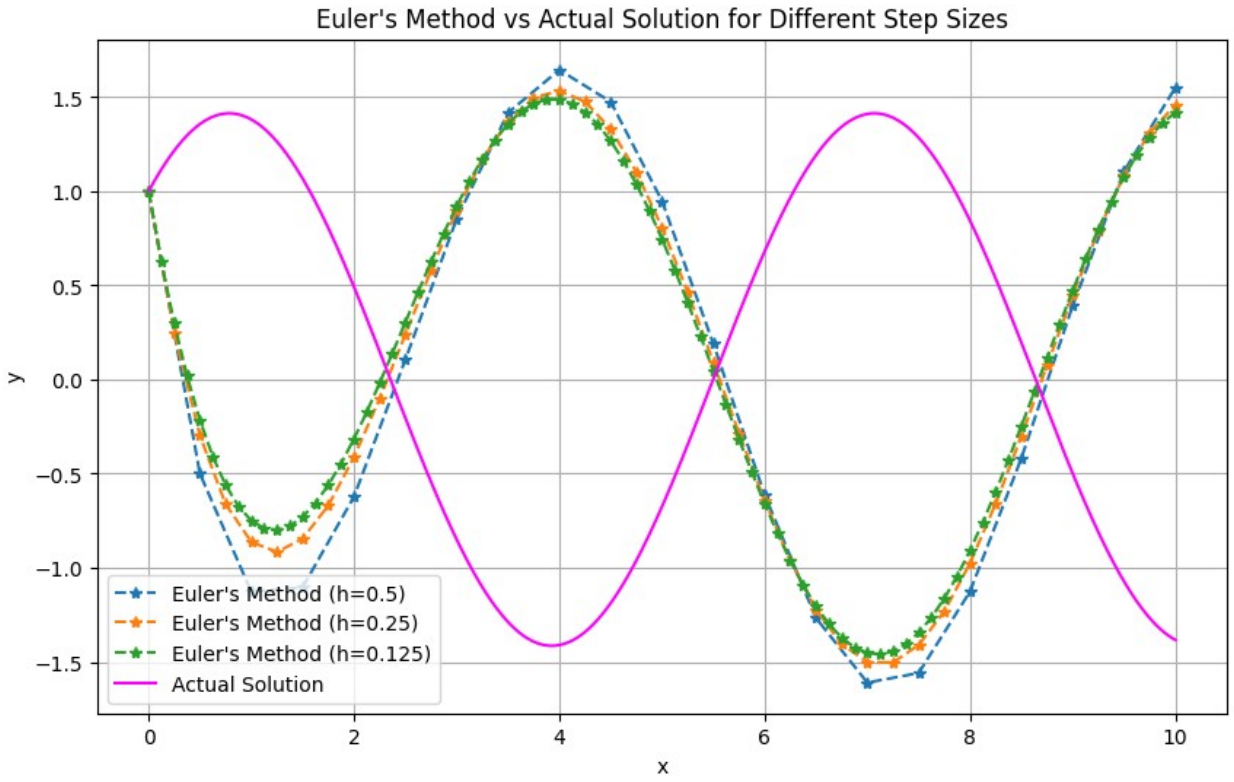
Also, whenever h gets halved the error also gets halved.

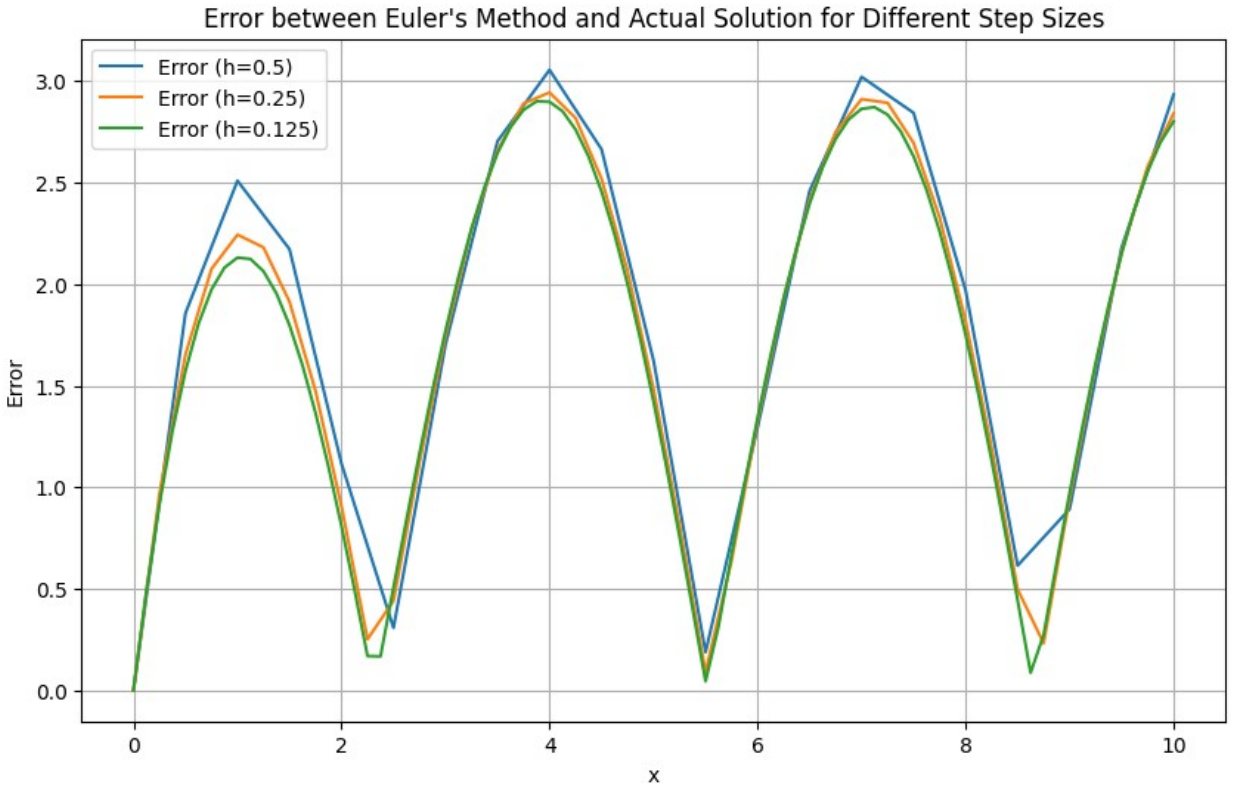
Part 2

Question-1

```
l=int(input('Enter value for lambda'))
if __name__ == "__main__":
    def f(x, y):
        return l*y - (1-l)*np.cos(x) - (1+l)*np.sin(x)

    def actual_solution(x):
        return np.sin(x) +np.cos(x)

    x0 = 0
    y0 = 1
    x_end = 10
    step_sizes = [0.5, 0.25, 0.125]

    euler = EulerMethod(f, x0, y0, x_end,
actual_solution=actual_solution)

    euler.plot_solutions(step_sizes)
```

```
Enter value for lambda-1
```





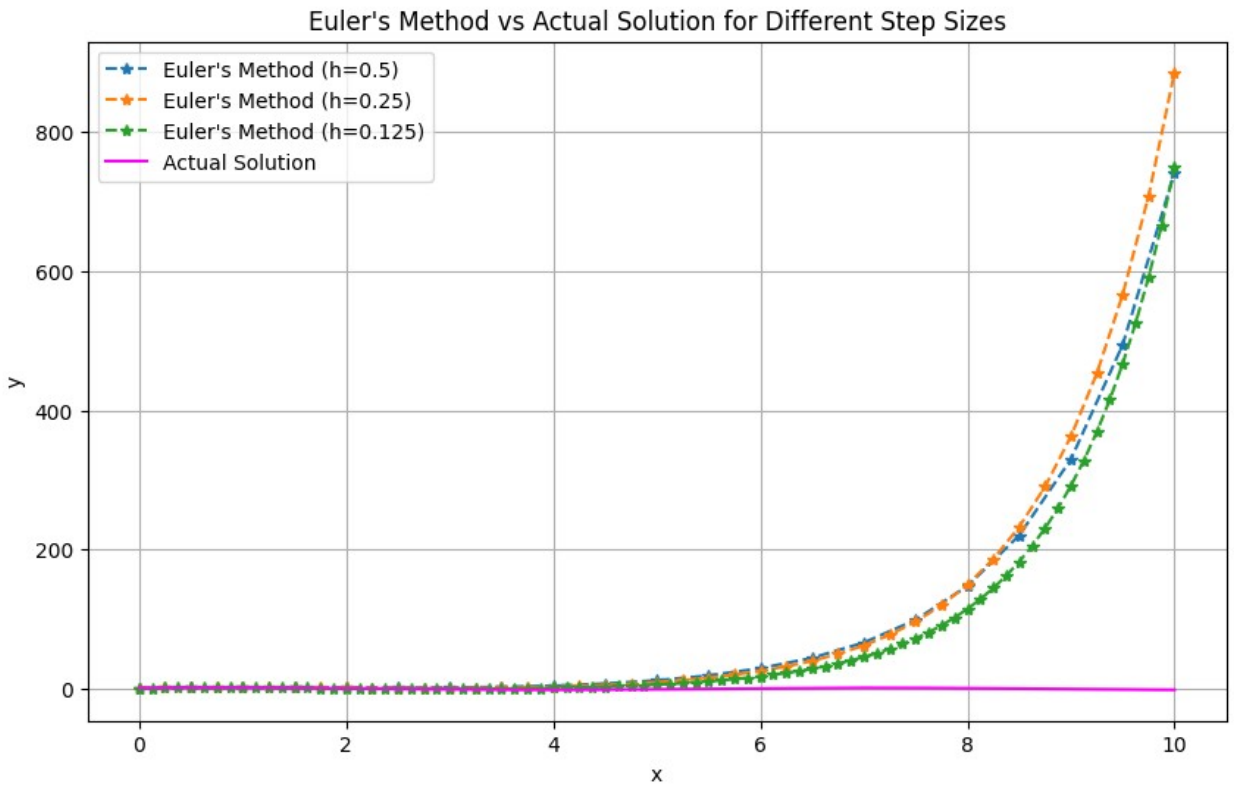
In this section as lambda was asked to be set as -1 we are getting inverted graph as of the original graph and hence the error is also greater because of the same.

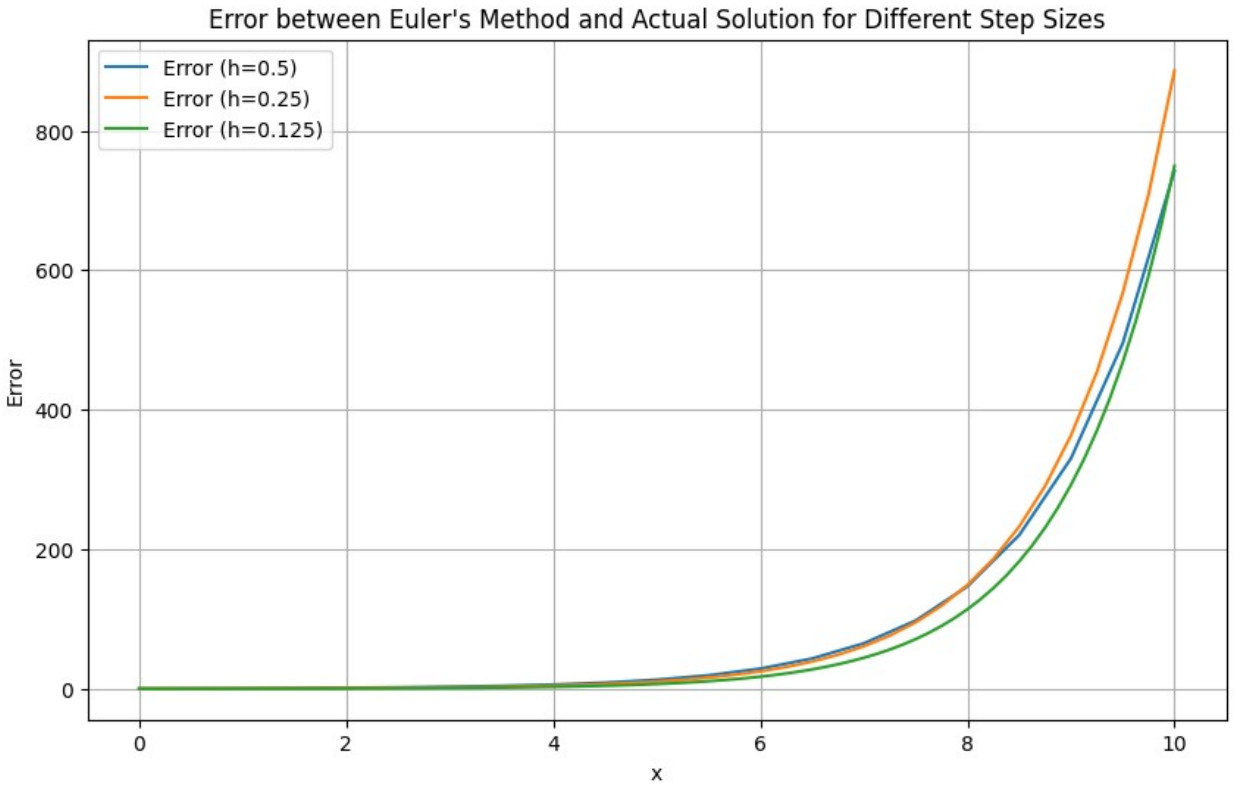
Here also the pattern of error becoming nearly half can be seen.

Question-2

```
l=int(input('Enter value for lambda'))
if __name__ == "__main__":
    def f(x, y):
        return l*y - (1-l)*np.cos(x) - (1+l)*np.sin(x)

    def actual_solution(x):
        return np.sin(x) +np.cos(x)

    x0 = 0
    y0 = 1
    x_end = 10
    step_sizes = [0.5, 0.25, 0.125]

    euler = EulerMethod(f, x0, y0, x_end,
actual_solution=actual_solution)

    euler.plot_solutions(step_sizes)
```

```
Enter value for lambda1
```





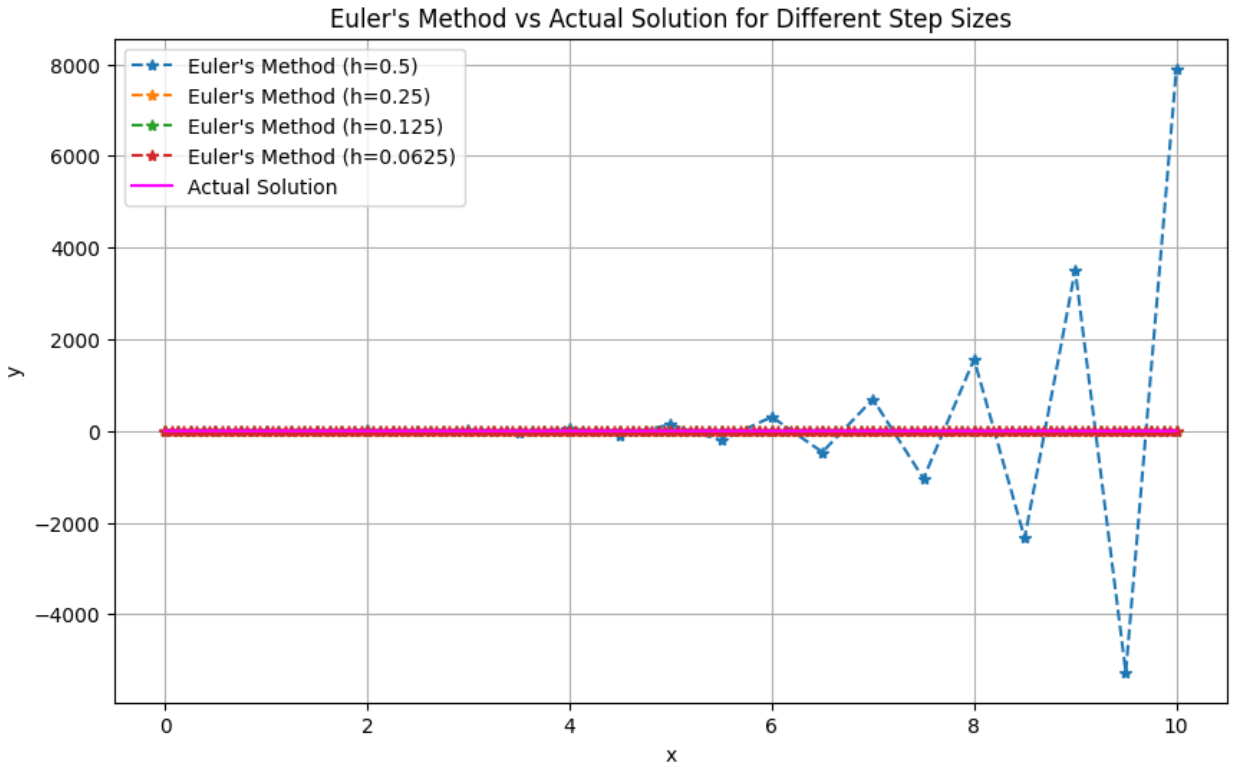
Here as lambda is told to set as one we can see that the euler plot is behaving like exponential and the actual plot is very negligible infront of it as the actual plot toggles between -1 to 1.

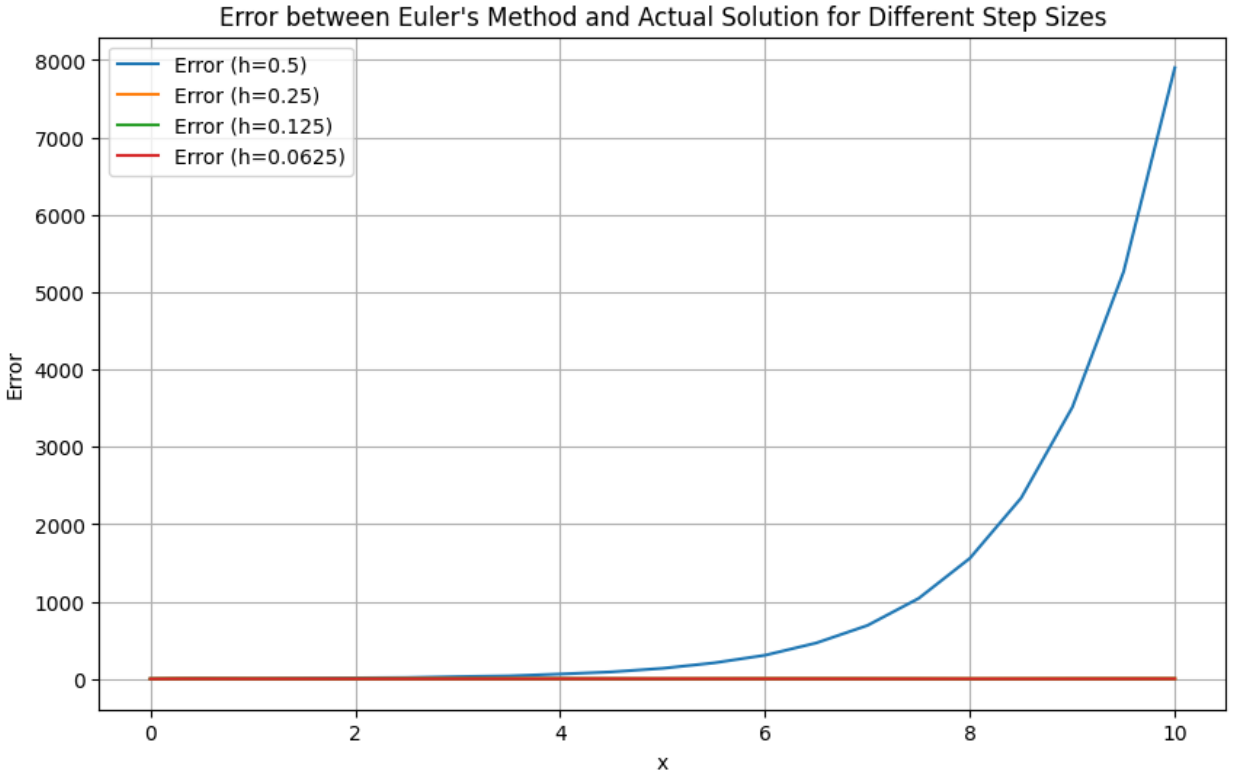
Hence we are seeing the exponential like nature of the error.

Question-3

```
l=int(input('Enter value for lambda'))
if __name__ == "__main__":
    def f(x, y):
        return l*y - (1-l)*np.cos(x) - (1+l)*np.sin(x)

    def actual_solution(x):
        return np.sin(x) +np.cos(x)

    x0 = 0
    y0 = 1
    x_end = 10
    step_sizes = [0.5, 0.25, 0.125,0.0625]

    euler = EulerMethod(f, x0, y0, x_end,
actual_solution=actual_solution)

    euler.plot_solutions(step_sizes)
```

```
Enter value for lambda-5
```



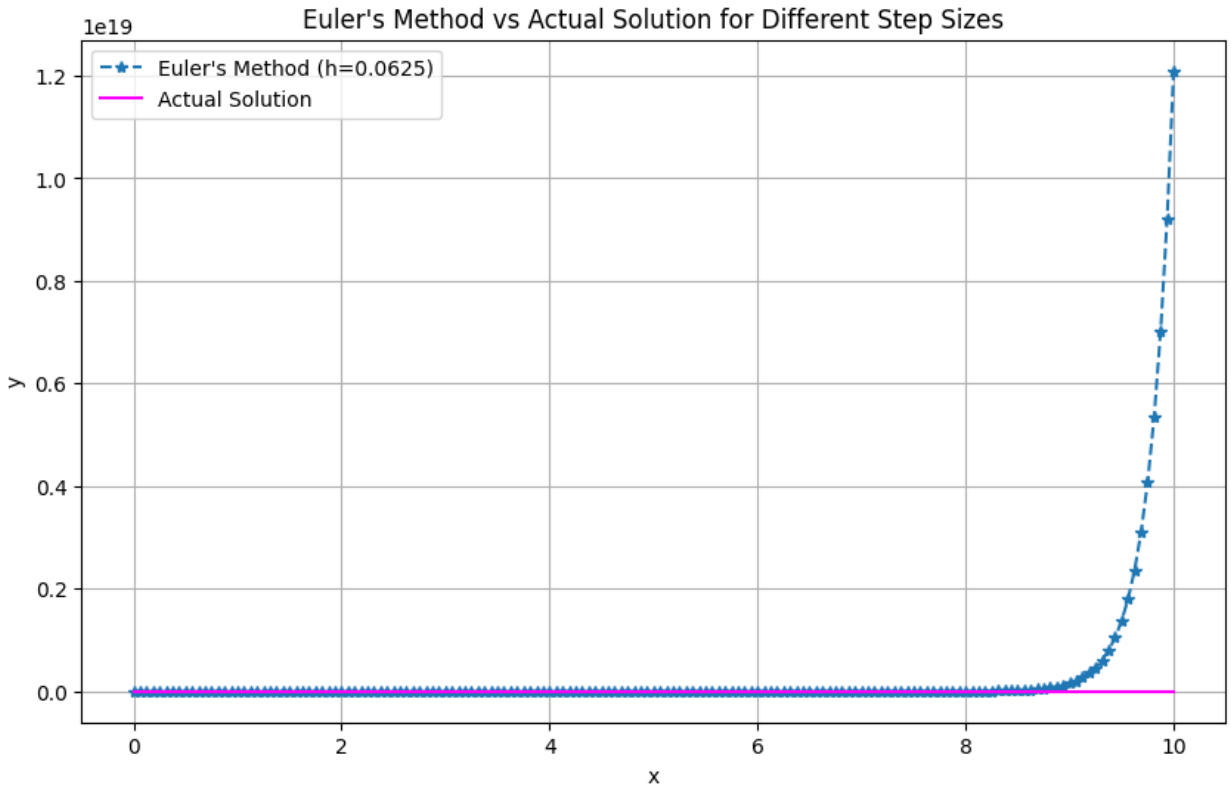


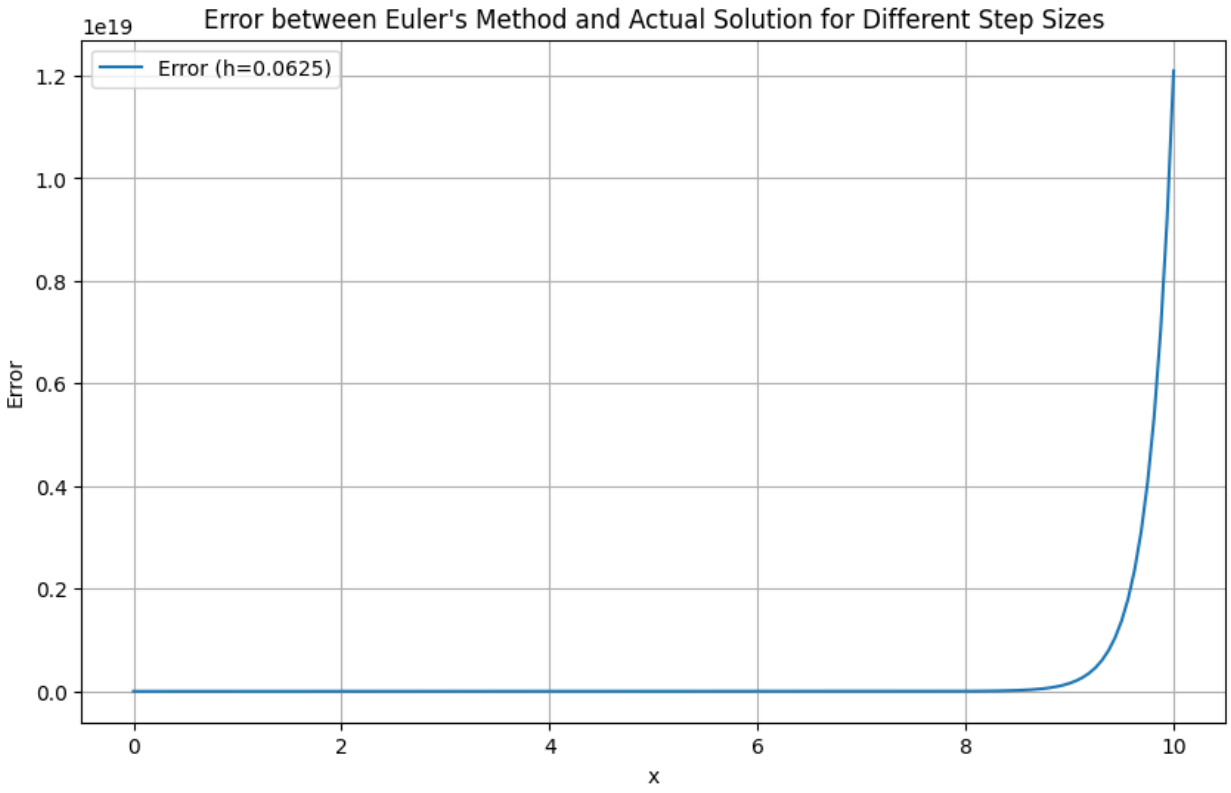
This plot also behaves like the one before.

Question-4

```
l=int(input('Enter value for lambda'))
if __name__ == "__main__":
    def f(x, y):
        return l*y - (1-l)*np.cos(x) - (1+l)*np.sin(x)

    def actual_solution(x):
        return np.sin(x) +np.cos(x)

    x0 = 0
    y0 = 1
    x_end = 10
    step_sizes = [0.0625]

    euler = EulerMethod(f, x0, y0, x_end,
actual_solution=actual_solution)

    euler.plot_solutions(step_sizes)
```

```
Enter value for lambda5
```





Here the error is so huge that the euler plot diverges completely and reaches upto the order 1e19.

Backward Euler