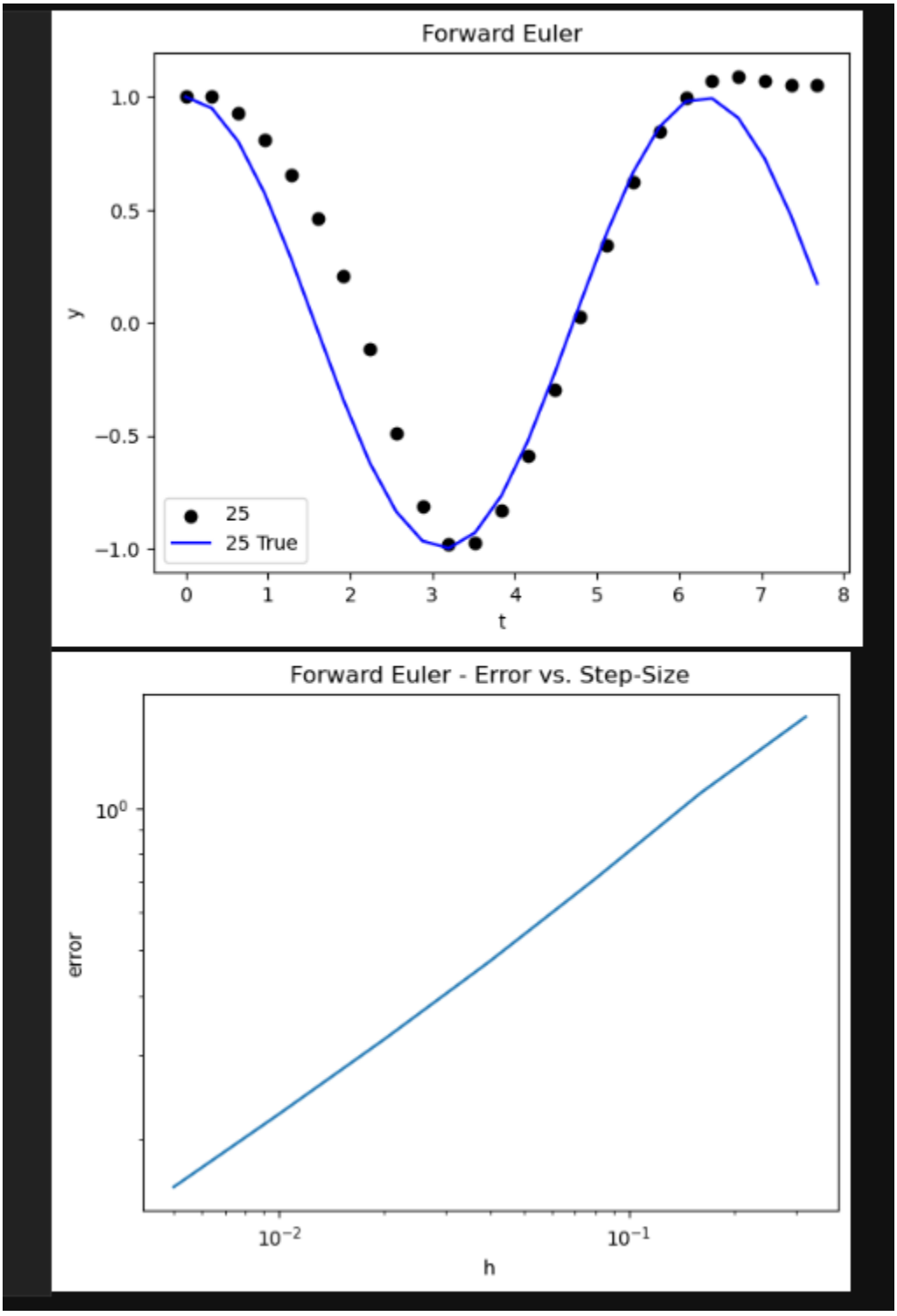
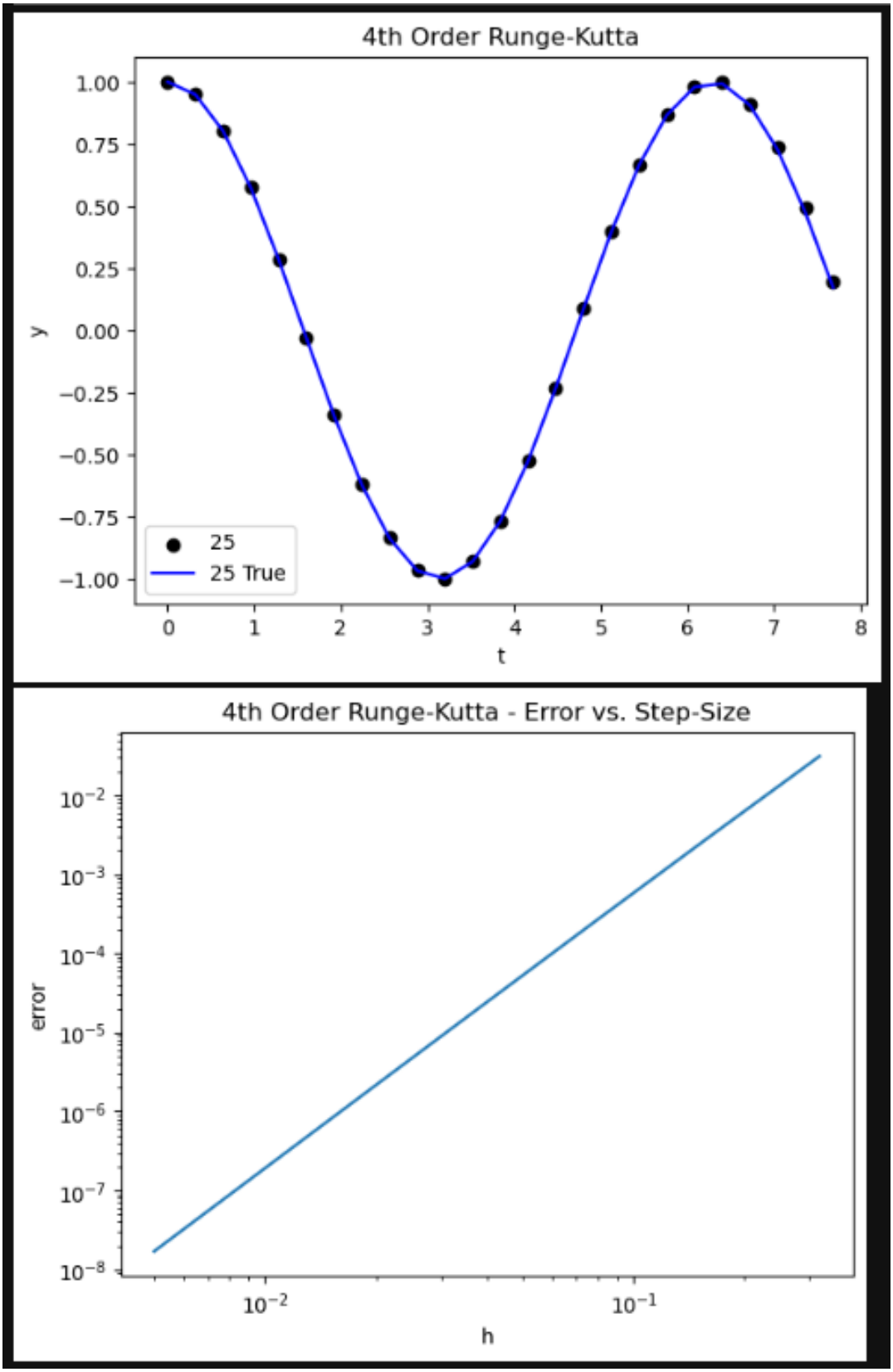
Problem 5 (Note: Used the L2 Norm for the Errors)

```python
# 5 (a).

# Parameters.
N = np.array([25, 50, 100, 200, 400, 800, 1600])
T = 8

# Make plot.
fig, ax = plt.subplots()
plt.xlabel("t")
plt.ylabel("y")
plt.title("Forward Euler")

# For each value of N, compute the time steps. There should
# be N total steps. So for N = 25, t should be length 25.
h_vals = []
errors = []
for n in N:
    h = T / n
    h_vals.append(h)
    t = np.arange(n) * h

    # Now run the iteration.
    y = np.zeros(shape=n)
    y[0] = 1
    for i in range(1, n):
        y[i] = y[i-1] + h * ((y[i-1]) ** 2 - np.sin(t[i-1]) - (np.cos(t[i-1]) ** 2))

    # Get true values.
    y_true = np.cos(t)

    # Norm of difference.
    errors.append(np.linalg.norm(y_true - y, 2))

    # Add to the plot.
    if n == 25:
        plt.scatter(t, y, marker='o', color='black', label=str(n))
        plt.plot(t, y_true, color='blue', label=str(n) + " True")

# Finish plot.
plt.legend()
plt.show()

# Plot errors versesu step-size.
fig, ax = plt.subplots()
plt.xlabel("h")
plt.ylabel("error")
plt.title("Forward Euler - Error vs. Step-Size")
plt.loglog(h_vals, errors)
plt.show()
```

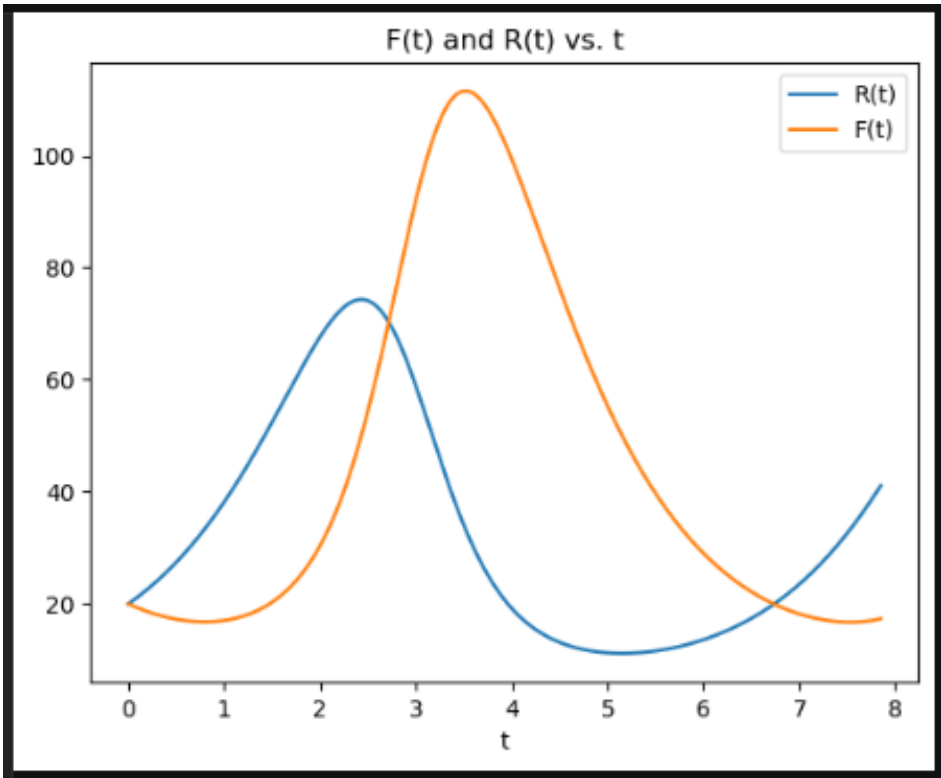

Forward Euler
1.0
0.5
0.0
−0.5
−1.0
y
0
1
2
3
4
5
6
7
8
t
25
25 True
Forward Euler - Error vs. Step-Size
10^0
error
10^{-2}
10^{-1}
h

```
# 5 (b).

# Parameters.
N = np.array([25, 50, 100, 200, 400, 800, 1600])
T = 8

# Make plot.
fig, ax = plt.subplots()
plt.xlabel("t")
plt.ylabel("y")
plt.title("4th Order Runge-Kutta")

# For each value of N, compute the time steps. There should
# be N total steps. So for N = 25, t should be length 25.
h_vals = []
errors = []
for n in N:
    h = T / n
    h_vals.append(h)
    t = np.arange(n) * h

    # Now run the iteration.
    y = np.zeros(shape=n)
    y[0] = 1
    for i in range(1, n):
        q1 = f(t[i-1], y[i-1])
        q2 = f(t[i-1] + h/2, y[i-1] + h*q1/2)
        q3 = f(t[i-1] + h/2, y[i-1] + h*q2/2)
        q4 = f(t[i-1] + h, y[i-1] + h*q3)
        y[i] = y[i-1] + (h / 6) * (q1 + 2 * q2 + 2 * q3 + q4)

    # Get true values.
    y_true = np.cos(t)

    # Norm of difference.
    errors.append(np.linalg.norm(y_true - y, 2))

    # Add to the plot.
    if n == 25:
        plt.scatter(t, y, marker='o', color='black', label=str(n))
        plt.plot(t, y_true, color='blue', label=str(n) + " True")

# Finish plot.
plt.legend()
plt.show()

# Plot errors versesu step-size.
fig, ax = plt.subplots()
plt.xlabel("h")
plt.ylabel("error")
plt.title("4th Order Runge-Kutta - Error vs. Step-Size")
plt.loglog(h_vals, errors)
plt.show()
```


4th Order Runge-Kutta
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y
0
1
2
3
4
5
6
7
8
t
25
25 True
4th Order Runge-Kutta - Error vs. Step-Size
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
error
10^{-2}
10^{-1}
h

Problem 6 (Note: Used Forward Euler)

```python
# 6.

# Parameters.
n = 10000
T = 5 * np.pi / 2

# Make plot.
fig, ax = plt.subplots()
plt.xlabel("t")
plt.title("F(t) and R(t) vs. t")

# Make space for R anf F / set initial conditions.
R = np.zeros(shape=n)
F = np.zeros(shape=n)

# Spacing.
h = T / n
h_vals.append(h)
t = np.arange(n) * h

# Now run the iteration.
R[0] = 20
F[0] = 20
for i in range(1, n):
    # For R.
    fR_i = fR(F[i-1], R[i-1])

    # For F.
    fF_i = fF(F[i-1], R[i-1])

    # Update.
    R[i] = R[i-1] + h * fR_i
    F[i] = F[i-1] + h * fF_i


# Finish plot.
plt.plot(t, R, label="R(t)")
plt.plot(t, F, label="F(t)")
plt.legend()
plt.show()

# Make plot.
fig, ax = plt.subplots()
plt.xlabel("R")
plt.ylabel("F")
plt.title("F(t) vs. R(t)")
plt.plot(R, F)
```




```
[<matplotlib.lines.Line2D at 0x1979e69a890>]
```