1) All we need to do is show that RHS is $\left|\frac{\partial f}{\partial y}\right|$ is bounded with $f(t, y(t)) = y'(t)$. So for $t \geq 1$ we have

$$f(t, y(t)) = \frac{1}{t^4 + 2y^2}$$

So,

note this derivative exists as $t \geq 1$, never $(t=0, y=0)$

$$\left|\frac{\partial f}{\partial y}\right| = \left|\frac{-4y}{(t^4+2y^2)^2}\right| = \frac{4|y|}{(t^4+2y^2)^2} \leq \frac{4y}{(t^4+2y^2)^2}$$

To show this function is bounded, we compute it's derivative, set it equal to 0, and find the maximum. As $t \geq 1$, the above is bounded above by $\frac{4y}{(1+2y^2)^2}$, so we can actually find the maximum of this instead.

$$\xrightarrow{\frac{d}{dy}} \frac{-2(4y)\cdot 4y}{(1+2y^2)^3} + \frac{4}{(1+2y^2)^2} = 0$$

$$\rightarrow \quad -32y^2 + 4(1+2y^2) = 0$$

$$\rightarrow \quad -24y^2 + 4 = 0$$

$$y^2 = 1/6 \rightarrow y = \pm\sqrt{1/6}$$

y': $-$ $\smile$ $+$ $\frown$ $-$, at $-1/\sqrt{6}$ and $1/\sqrt{6}$

checking to see which is the maximum

So $\left|\frac{\partial f}{\partial y}\right|$ is bounded above by

$$C = \frac{4\sqrt{1/6}}{\left(1 + 2\sqrt{\tfrac{1}{6}}^{\,2}\right)^2}$$

$$= \frac{4}{\sqrt{6}} \frac{1}{\left(1+\frac{1}{3}\right)^2} = \frac{\cancel{4}}{\sqrt{6}} \frac{3^2}{4^{\cancel{2}}}$$

$$= \frac{9}{4\sqrt{6}}$$

. Hence f is Lipschitz, hence the system

$$y'(t) = f(t, y(t))$$

has a unique solution.

2) We have

$$\bullet \quad u'' = -Ku$$
$$u(t_0) = u_0$$
$$u'(t_0) = v_0$$

If we put $v = u'$, then the above becomes

$$\begin{aligned} u' &= v \\ v' &= -Ku \end{aligned} = \underbrace{\begin{pmatrix} 0 & 1 \\ -K & 0 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} u \\ v \end{pmatrix}}_{\vec{x}}$$

$$= A\vec{x} = f(\vec{x})$$

Now take $\vec{x}, \vec{y} \in \mathbb{R}^2$, then ☆

$$\| f(\vec{x}) - f(\vec{y}) \| = \| A\vec{x} - A\vec{y} \|$$

$$= \|A\| \, \|\vec{x} - \vec{y}\|$$

So to satisfy the Lipschitz condition, we need

$$\|A\| \, \|\vec{x} - \vec{y}\| \le C \|\vec{x} - \vec{y}\| \rightarrow C \ge \|A\|$$

Hence $C = \|A\|$ is the smallest Lipschitz constant of f, for a given norm.
The smallest constants for various norms are computed next. We have

(next page)

$C_1 = \|A\|_1 =$ "max absolute column sum"

$= \max(1, K)$

$C_2 = \|A\|_2 = \Big\{$ By noticing that A is Hermitian ($A = A^T$), the singular values are the absolute values of the eigenvalues of A. So

$$\det(A - \lambda I) = \det \begin{bmatrix} -\lambda & 1 \\ -K & -\lambda \end{bmatrix}$$

$$= \lambda^2 + K := 0$$

$$\rightarrow \lambda_{\pm} = \pm i\sqrt{K}, \text{ so that } \overset{\sigma_{\pm}}{\overset{\shortparallel}{|\lambda_{\pm}|}} = \sqrt{K}$$

So as $\|A\|_2$ is the largest singular value of A,

$$C_2 = \|A\|_2 = \sqrt{K}$$

$C_\infty = \|A\|_\infty =$ "max absolute row sum"

$= \max(1, K)$

3) Firstly, by Taylor's theorem

$$f_{n+1} = f(t_{n+1}, y_{n+1}) = f(t_n + h, y_{n+1}) = f_n + h \overbrace{\frac{df}{dt}\Big|_{(t_n, y_{n+1})}}^{g_n} + O(h^2)$$

So our step is (numerical solution)

$$y_{n+1} = y_n + h\left[\theta f_n + (1-\theta) f_{n+1}\right]$$

$$= y_n + h\left[\theta f_n + (1-\theta)\left[f_n + h g_n + O(h^2)\right]\right]$$

$$= y_n + h\left[f_n + (1-\theta) h g_n + O(h^2)\right]$$

The actual solution is

$$y(t_{n+1}) = y(t_n + h) = y(t_n) + h y'(t_n) + \frac{h^2}{2} y''(t_n) + O(h^3)$$

$$= y(t_n) + h f_n + \frac{h^2}{2} g_n + O(h^3)$$

So the local truncation error is (replacing $y(t_j)$ w/ y_j)

$$\frac{y_{n+1} - y(t_{n+1})}{h} = \frac{1}{h}\left(\cancel{y_n} + \cancel{h f_n} + h^2(1-\theta) g_n - \cancel{y_n} - \cancel{h f_n} - \frac{h^2}{2} g_n + O(h^3)\right]$$

$$= \frac{1}{h}\left[\frac{(1/2 - \theta)}{2} h^2 g_n + O(h^3)\right]$$

$$= \overbrace{\frac{1}{2} h g_n (1/2 - \theta)}^{(*)} + O(h^2)$$

So when $\theta = 1/2$ it is $O(h^2)$ as $(*) = 0$, and when $\theta \neq 1/2$ it is $O(h)$ as $(*) \neq 0$.

4) The numerical solution is

$$y_{n+1} = y_n + \frac{h}{2}\left[f(t_{n+1}, \tilde{y}_{n+1}) + f_n \right]$$

$$= y_n + \frac{h}{2}\left[\underbrace{f(t_n+h, y_n + hf_n)}_{A} + f_n \right] \quad (*)$$

Using Taylor's thm, centered at (t_n, y_n),
A becomes

$$= f_n + h\,[\overbrace{f_t + f f_y}^{a_n}] + \frac{h^2}{2}[\overbrace{f_{tt} + 2ff_{ty} + f_n^2 f_{yy}}^{b_n}] + O(h^3)$$

$$\quad\; \underbrace{\quad}_{y'(t_n,y_n)} \qquad\qquad \underbrace{\quad}_{y''(t_n,y_n)}$$

So

$$(*) = y_n + hf_n + \frac{h^2}{2}a_n + \frac{h^3}{4}b_n + O(h^4)$$

The exact solution is

$$y(t_{n+1}) = y(t_n + h)$$

$$= y(t_n) + hf_n + \frac{h^2}{2}a_n + \frac{h^3}{6}b_n + O(h^4)$$

So the local truncation error is (putting $y_n = y(t_n)$)

$$\frac{y_{n+1} - y(t_{n+1})}{h} = \frac{\left[\cancel{y_n} + \cancel{hf_n} + \cancel{\frac{h^2}{2}a_n} + \frac{h^3}{4}b_n - \cancel{y_n} - \cancel{hf_n} - \cancel{\frac{h^2}{2}a_n} - \frac{h^3}{6}b_n + O(h^4)\right]}{h}$$

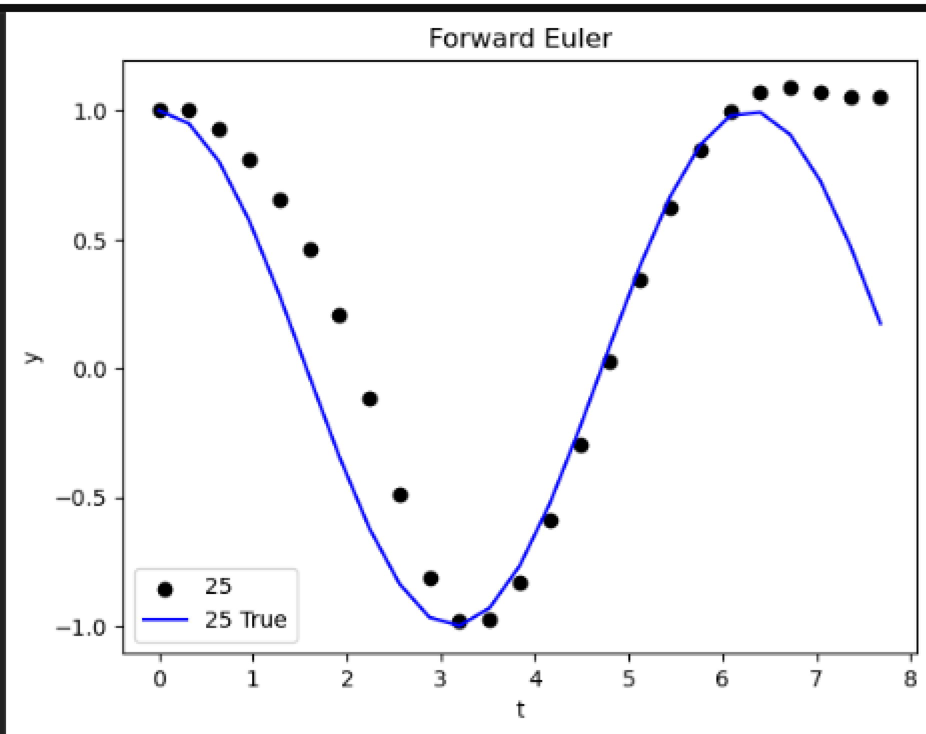
$$= \left(\frac{1}{4} - \frac{1}{6}\right)h^2 b_n + O(h^3)$$

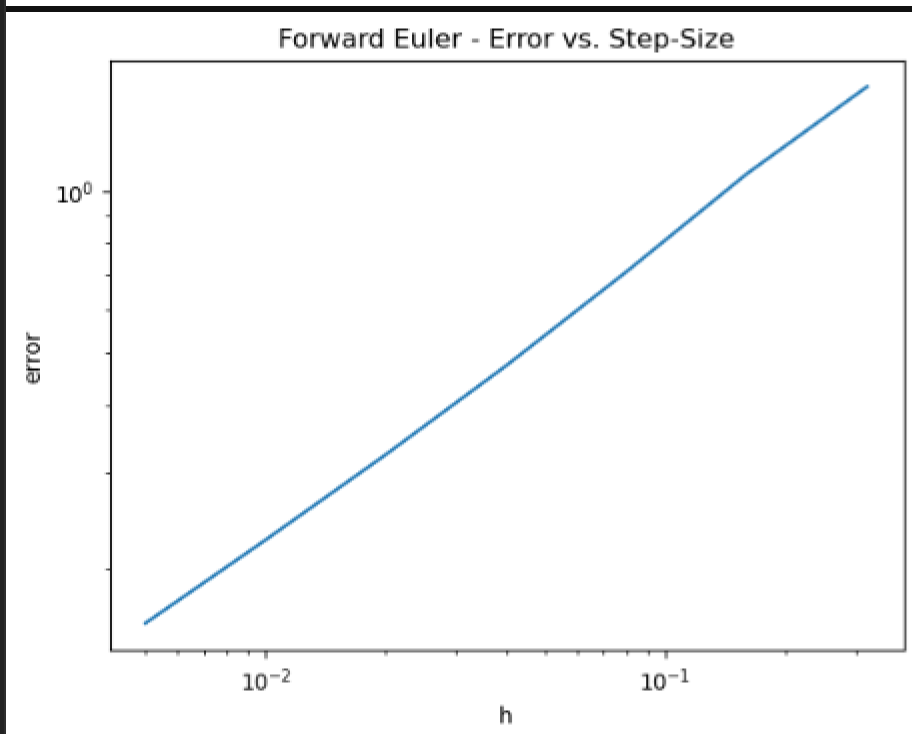
So the error is $O(h^2)$ with leading term

$$\frac{1}{12}b_n(t_n, y_n) = \frac{1}{12}y'''(t_n)$$

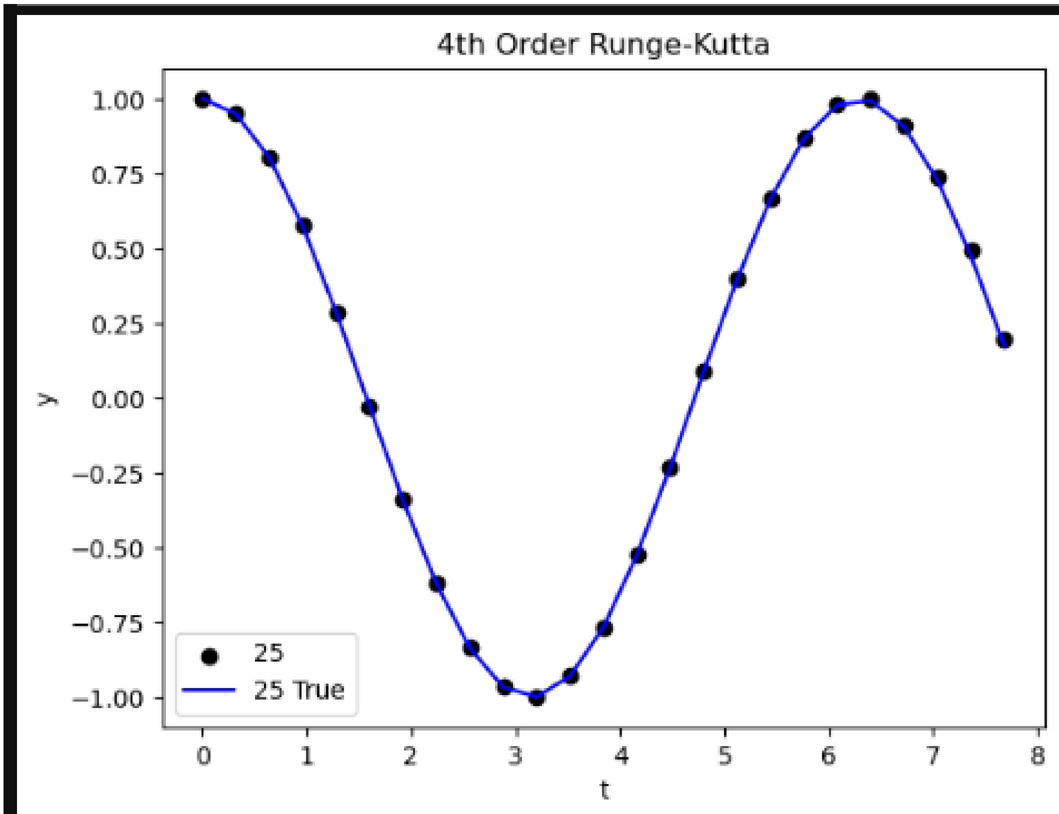
Problem 5 (Note: Used the L2 Norm for the Errors)

```
# 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

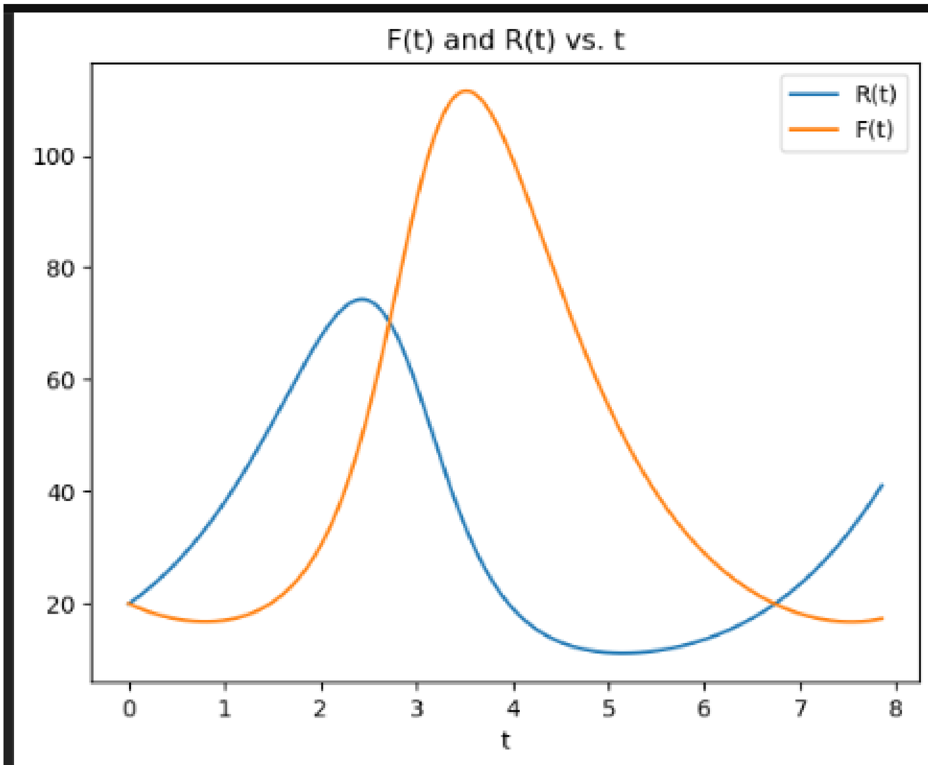
```python
# 5 (b).

# 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>]
```