1 Problem

Approximate the angle of the pendulum as a function of time where the governing differential equation is:

$$\frac{d^2\theta}{dt^2} + \frac{g}{l}sin(\theta) = 0 \tag{1}$$

$$\theta(0) = \theta_0 \tag{2}$$

$$\dot{\theta}(0) = \dot{\theta}_0 \tag{3}$$

We will derive solutions for Equation (1), and also the small-angle approximation of that equation:

$$\frac{d^2\theta}{dt^2} + \frac{g}{l}\theta = 0 \tag{4}$$

We give the constants in the equations with values in standard MKS units:

$$g = 9.81\text{m/s}^2 \tag{5}$$

$$l = 0.6\text{m} \tag{6}$$

Since we will use Equation (4) as our analytical solution, we will use a small starting θ so that our small-angle approximation is valid:

$$\theta(0) = 0.1\text{rad} \tag{7}$$

$$\dot{\theta}(0) = 0\text{rad/s} \tag{8}$$

Therefore, our analytical solution that we use to calculate residuals of the later numerical methods is:

$$\theta(t) = 0.1cos(4.044t) \tag{9}$$

2 Euler implicit method (backward differencing)

2.1 Numerical solution of Equation (1)

Our backward differencing scheme is as follows:

$$\theta_n = \theta_{n-1} + \dot{\theta}_n \cdot \Delta t \tag{10}$$

$$\dot{\theta}_n = \dot{\theta}_{n-1} + \frac{g}{l} sin(\theta_n) \cdot \Delta t \tag{11}$$

Since this is a non-linear system of equations to solve for θ_n and $\dot{\theta}_n$, we will use the gradient descent algorithm to iterate for these values, with initial guesses of $\theta_n = \theta_{n-1}$ and $\dot{\theta}_n = \dot{\theta}_{n-1}$ (details appendix code).

This gives use the following solutions for various values for Δt:

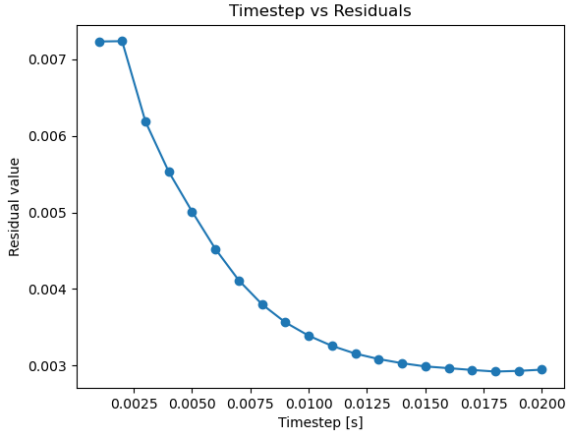


Figure 1: Residuals

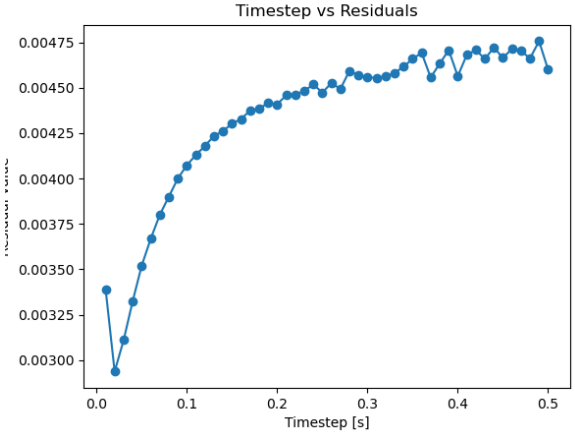


Figure 2: More Residuals

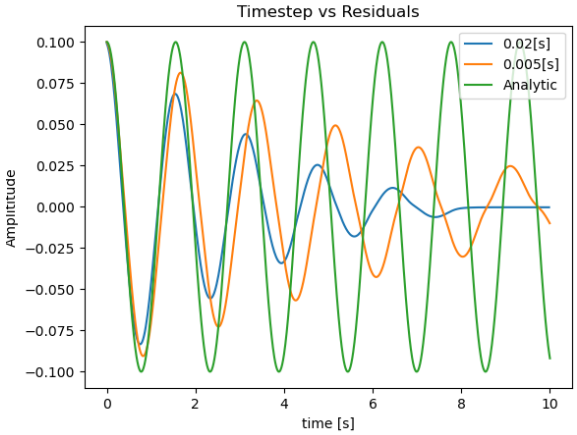


Figure 3: Plots of selection solutions

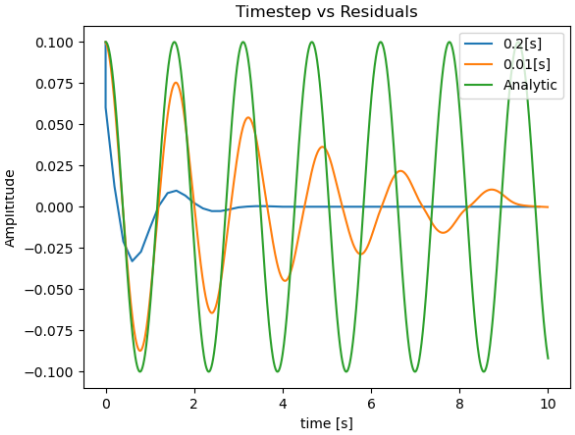


Figure 4: More plots of selection solutions

The timestep at which the residual is minimized is about $\Delta t = 0.02$s.

2.2 Numerical solution of Equation (4)

Our backward differencing scheme is as follows:

$$\theta_n = \theta_{n-1} + \dot{\theta}_n \cdot \Delta t \tag{12}$$

$$\dot{\theta}_n = \dot{\theta}_{n-1} + \frac{g}{l}\theta_n \cdot \Delta t \tag{13}$$

This gives use the following solutions for various values for Δt:

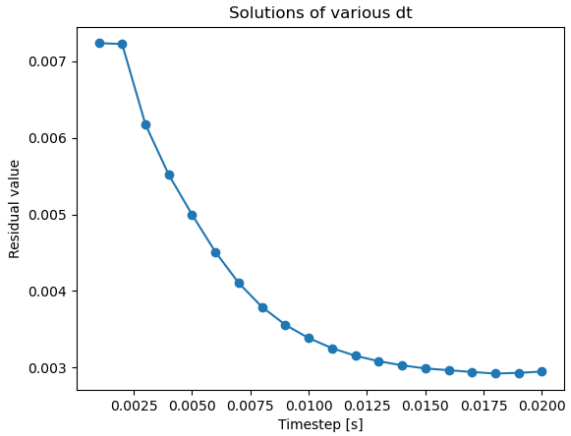


Figure 5: Residuals

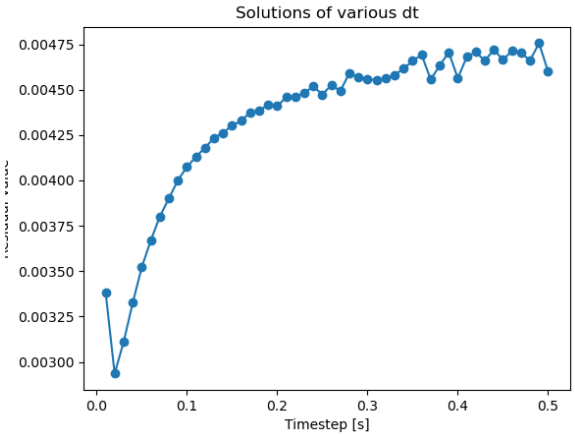


Figure 6: More Residuals

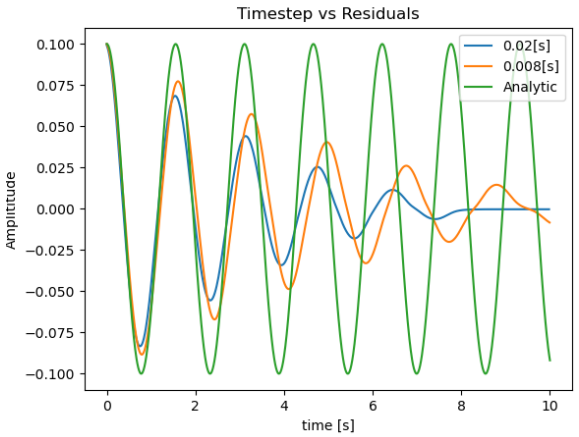


Figure 7: Plots of selection solutions

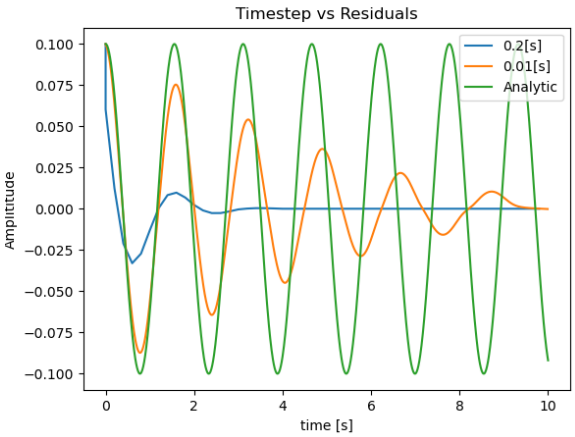


Figure 8: More plots of selection solutions

These results are nearly identical to that of section 2.1, which is expected as the small-angle approximation holds for the chosen starting angle. The timestep at which the residual is

minimized is about $\Delta t = 0.02$s. Since this is close to the result of section 2.1, we will use this timestep for section 2.3 where we find the θ value at which Equations (1) and (4) diverge.

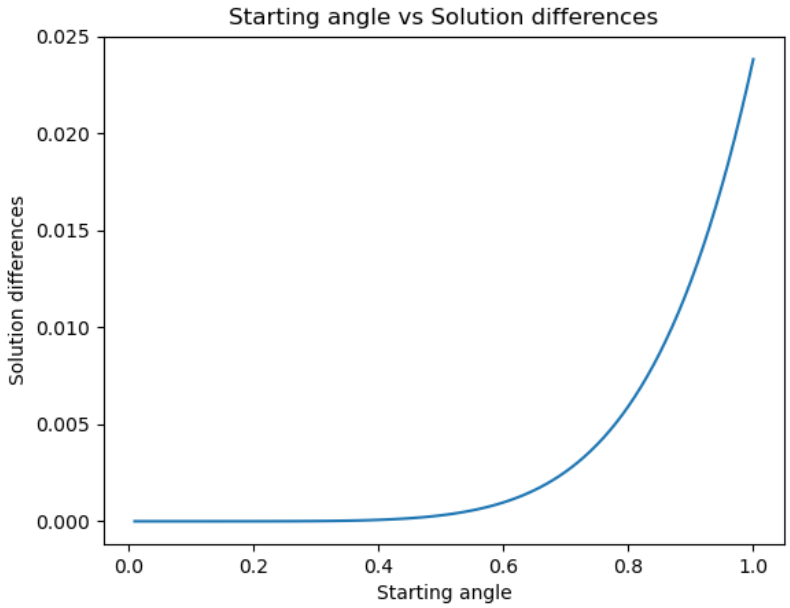
2.3 Finding θ_0 at which Equations (1) and (4) diverge

We will use $\Delta t = 0.02$s, vary the starting angle of the system.

We define the difference as:

$$\Delta_i = \frac{1}{n}(\theta_1(t_i) - \theta_4(t_i))^2 \tag{14}$$

Where n is the number of timesteps in the time domain. The divergence is gradual, and only starting to grow rapidly at about $\theta_0 = 0.6$rad. We will say that this is the point where the solutions of Equations (1) and (4) diverge.



3 Euler explicit method (forward differencing)

3.1 Numerical solution of Equation (4)

Our forward differencing scheme is as follows:

$$\theta_{n+1} = \theta_n + \dot{\theta}_n \cdot \Delta t \tag{15}$$

$$\dot{\theta}_{n+1} = \dot{\theta}_n + \frac{g}{l} sin(\theta_n) \cdot \Delta t \tag{16}$$

This gives use the following solutions for various values for Δt:

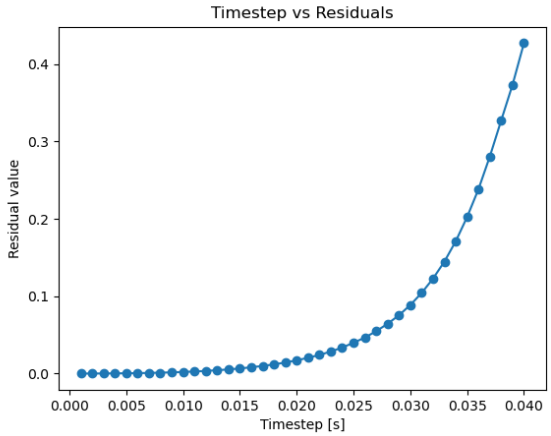


Figure 9: Residuals

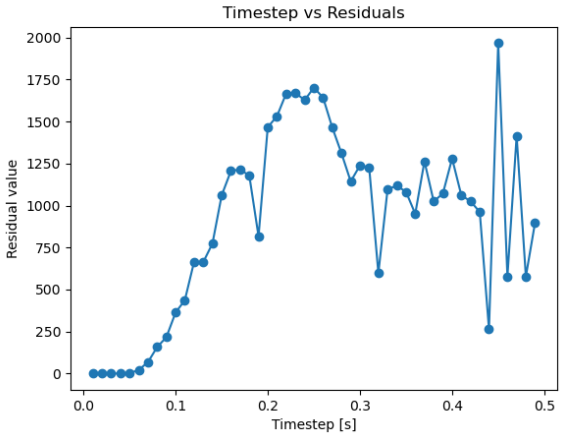


Figure 10: More Residuals

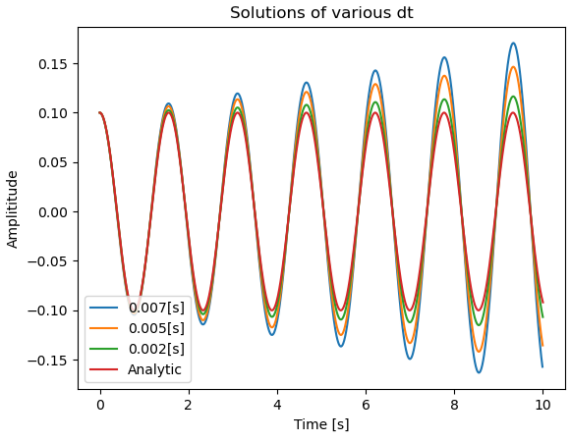


Figure 11: Plots of selection solutions

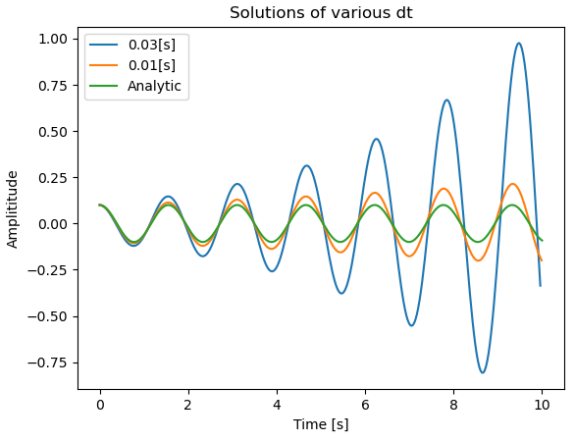


Figure 12: More plots of selection solutions

The residuals approaches zero as the timesteps are decreased, as shown for the timesteps chosen above. With these solutions, we would choose the timestep of $\Delta t = 0.02$s, as this is has a small enough residual, and is the same timestep chosen for the implicit methods in section 2.

3.2 Numerical solution of Equation (4)

Our forward differencing scheme is as follows:

$$\theta_{n+1} = \theta_n + \dot{\theta}_n \cdot \Delta t \tag{17}$$

$$\dot{\theta}_{n+1} = \dot{\theta}_n + \frac{g}{l}\theta_n \cdot \Delta t \tag{18}$$

This gives use the following solutions for various values for Δt:

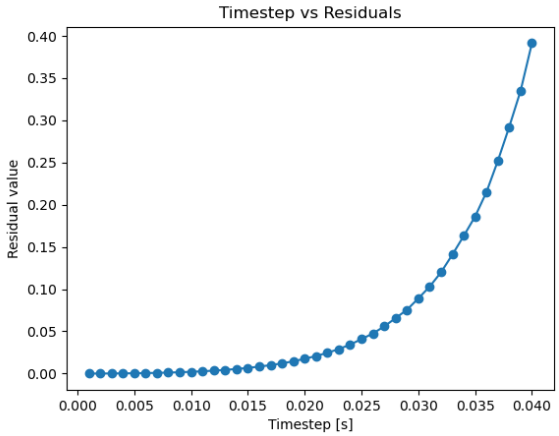


Figure 13: Residuals

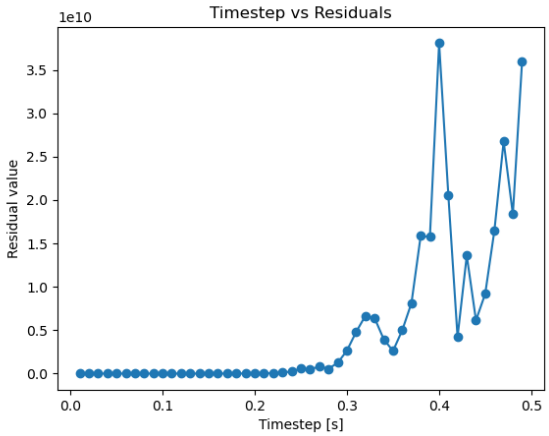


Figure 14: More Residuals

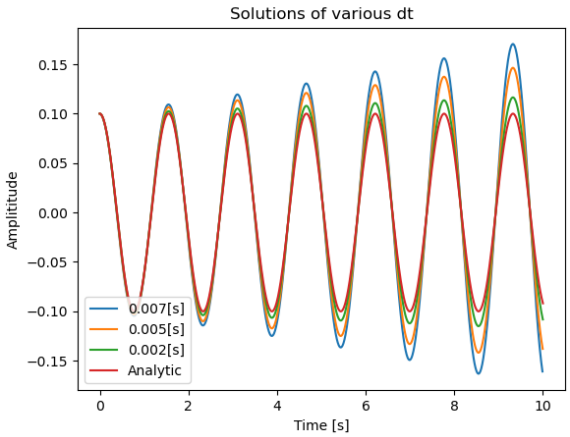


Figure 15: Plots of selection solutions

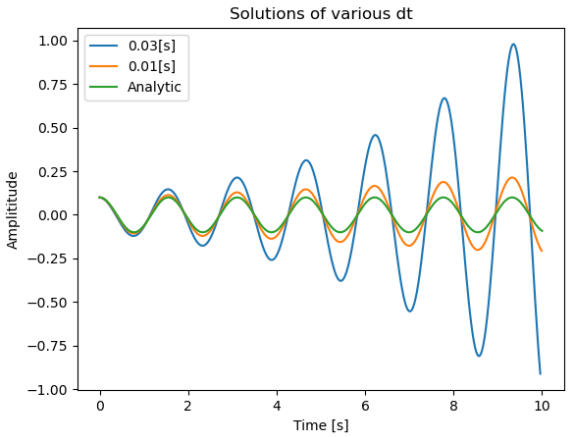


Figure 16: More plots of selection solutions

These results are nearly identical to that of section 3.1, which is expected as the small-angle approximation holds for the chosen starting angle. The notable exception is the larger

timesteps, which give large and seemingly chaotic residuals. Since this is close to the result of section 3.1, we will use $\Delta t = 0.02$s for section 3.3, where we find the θ value at which Equations (1) and (4) diverge.

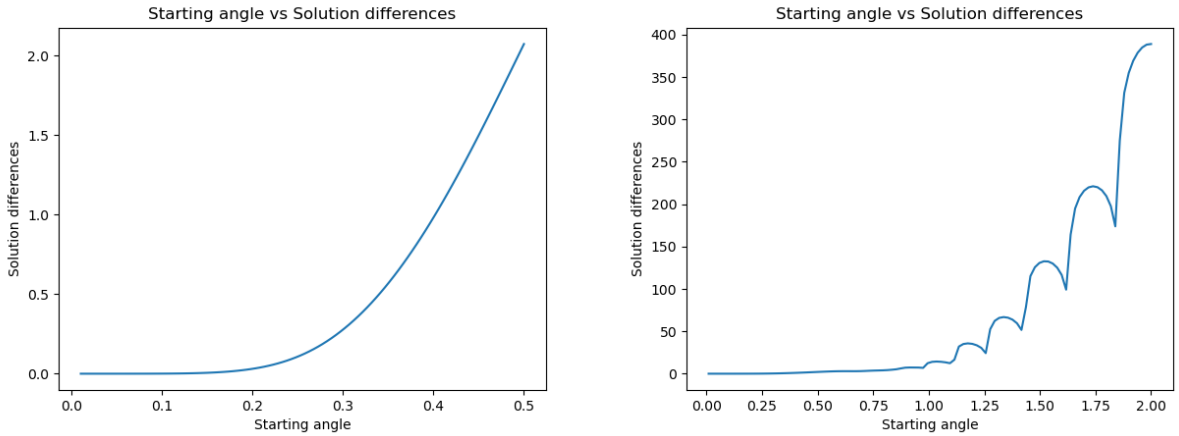
3.3 Finding θ_0 at which Equations (1) and (4) diverge

We will use $\Delta t = 0.02$s, vary the starting angle of the system.

We define the difference as:

$$\Delta_i = \frac{1}{n}(\theta_1(t_i) - \theta_4(t_i))^2 \tag{19}$$

Where n is the number of timesteps in the time domain. The divergence of the solutions grow quickly after about $\theta_0 = 0.2$rad. We will say that this is the point where the solutions of Equations (1) and (4) diverge.



Curiously, the rapid divergence of the exhibit a strange behavior after about $\theta_0 = 1$rad, which is shown in the right plot above.

4 4th Order Runge-Kutta

We will define a few equations:

$$\theta_{i+1} = \theta_i + \Delta\theta \tag{20}$$

$$\Delta\theta = c_1 \cdot \Delta\theta_1 + c_2 \cdot \Delta\theta_2 + c_3 \cdot \Delta\theta_3 + c_4 \cdot \Delta\theta_4 \tag{21}$$

Since we are approximating the fourth order, we will assign the coefficients as: $c_1 = 1/6, c_2 = 1/3, c_3 = 1/3$ and $c_4 = 1/6$.

The individual $\Delta\theta$ are defined as:

$$\Delta\theta_1 = f(t_i, \theta_i) \cdot \Delta t \tag{22}$$

$$\Delta\theta_2 = f(t_i + \Delta t/2, \theta_i + \Delta\theta_1/2) \cdot \Delta t \tag{23}$$

$$\Delta\theta_3 = f(t_i + \Delta t/2, \theta_i + \Delta\theta_2/2) \cdot \Delta t \tag{24}$$

$$\Delta\theta_4 = f(t_i + \Delta t, \theta_i + \Delta\theta_3) \cdot \Delta t \tag{25}$$

Where:

$$f(t, \theta) = \frac{d\theta}{dt} = \omega \tag{26}$$

The above equations are for θ, and the equations for $\theta^{'}$ (also written as ω) are:

$$\Delta\omega_1 = f(t_i, \omega_i) \cdot \Delta t \tag{27}$$

$$\Delta\omega_2 = f(t_i + \Delta t/2, \omega_i + \Delta\omega_1/2) \cdot \Delta t \tag{28}$$

$$\Delta\omega_3 = f(t_i + \Delta t/2, \omega_i + \Delta\omega_2/2) \cdot \Delta t \tag{29}$$

$$\Delta\omega_4 = f(t_i + \Delta t, \omega_i + \Delta\omega_3) \cdot \Delta t \tag{30}$$

Where:

$$f(t, \omega) = \frac{d\omega}{dt} = -\frac{g}{l} sin(\theta) \tag{31}$$

Equations (26) and (31) show that the derivative approximation for θ is in terms of ω, and the derivative approximation for ω is in terms of θ. Therefore, the RK4 approximation for these two quantities are coupled.

4.1 Numerical solution of Equation (1)

An iteration of RK4 for Equation (1) is:

$$\Delta\theta_1 = \omega_0 \cdot \Delta t \tag{32}$$

$$\Delta\omega_1 = -\frac{g}{l} sin(\theta_0) \cdot \Delta t \tag{33}$$

$$\Delta\theta_2 = (\omega_0 + 0.5 \cdot \Delta\omega_1) \cdot \Delta t \tag{34}$$

$$\Delta\omega_2 = -\frac{g}{l} sin(\theta_0 + 0.5 * \Delta\theta_1) \cdot \Delta t \tag{35}$$

$$\Delta\theta_3 = (\omega_0 + 0.5 \cdot \Delta\omega_2) \cdot \Delta t \tag{36}$$

$$\Delta\omega_3 = -\frac{g}{l} sin(\theta_0 + \Delta\theta_2) \cdot \Delta t \tag{37}$$

$$\Delta\theta_4 = (\omega_0 + \Delta\omega_3) \cdot \Delta t \tag{38}$$

$$\Delta\omega_4 = -\frac{g}{l} sin(\theta_0 + \Delta\theta_3) \cdot \Delta t \tag{39}$$

This gives use the following solutions for various values for Δt:

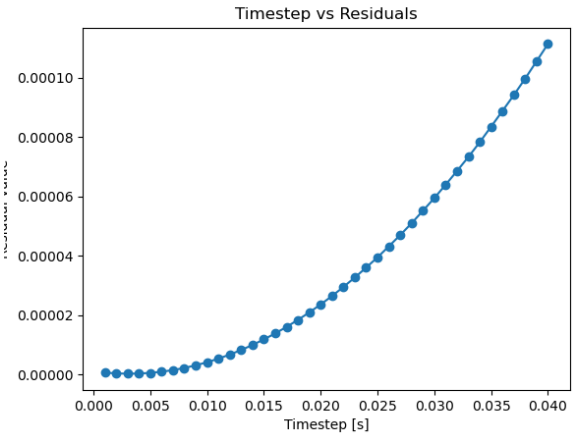


Figure 17: Residuals

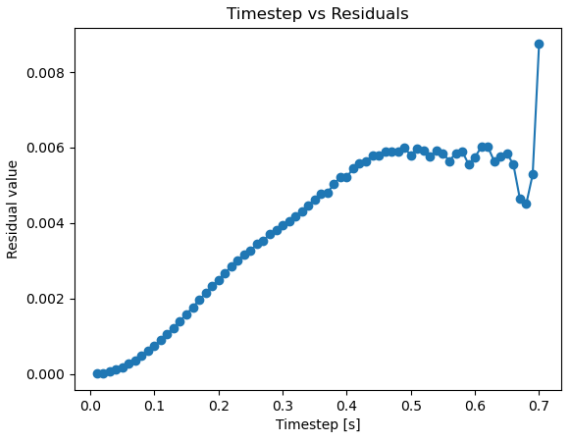


Figure 18: More Residuals

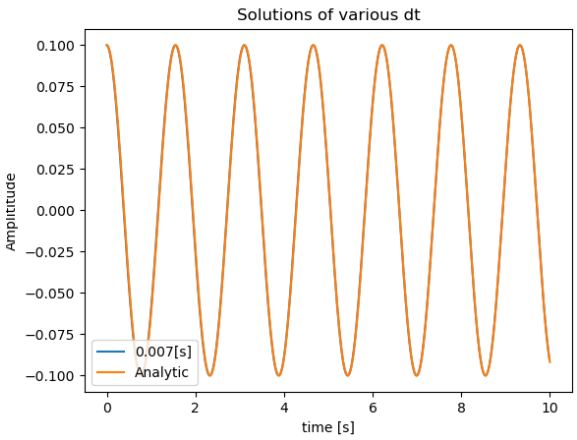


Figure 19: Plots of selection solutions

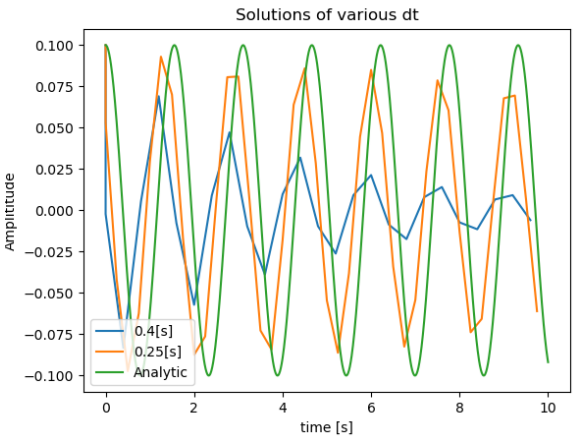


Figure 20: More plots of selection solutions

The residuals approaches zero as the timesteps are decreased, as shown for the timesteps chosen above. The residuals are much smaller than the previous Euler methods, and does not blow up for a wider range of timesteps. With these solutions, we would choose the timestep of $\Delta t = 0.01$s for section 4.3, as the residual does not become much smaller with further decrease in timestep size.

4.2 Numerical solution of Equation (4)

An iteration of RK4 for Equation (4) is:

$$\Delta\theta_1 = \omega_0 \cdot \Delta t \tag{40}$$

$$\Delta\omega_1 = -\frac{g}{l}\theta_0 \cdot \Delta t \tag{41}$$

$$\Delta\theta_2 = (\omega_0 + 0.5 \cdot \Delta\omega_1) \cdot \Delta t \tag{42}$$

$$\Delta\omega_2 = -\frac{g}{l}\theta_0 + 0.5 * \Delta\theta_1 \cdot \Delta t \tag{43}$$

$$\Delta\theta_3 = (\omega_0 + 0.5 \cdot \Delta\omega_2) \cdot \Delta t \tag{44}$$

$$\Delta\omega_3 = -\frac{g}{l}\theta_0 + \Delta\theta_2 \cdot \Delta t \tag{45}$$

$$\Delta\theta_4 = (\omega_0 + \Delta\omega_3) \cdot \Delta t \tag{46}$$

$$\Delta\omega_4 = -\frac{g}{l}\theta_0 + \Delta\theta_3 \cdot \Delta t \tag{47}$$

This gives use the following solutions for various values for Δt:

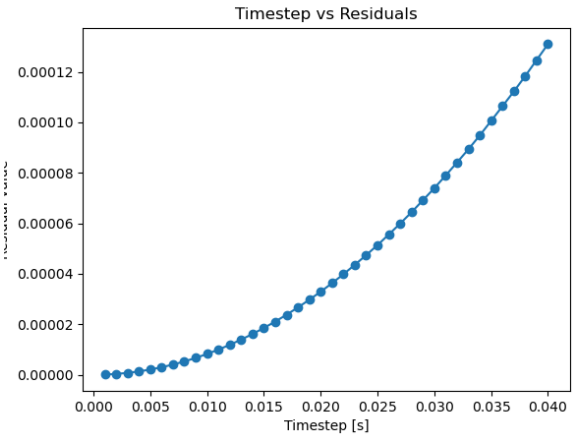


Figure 21: Residuals

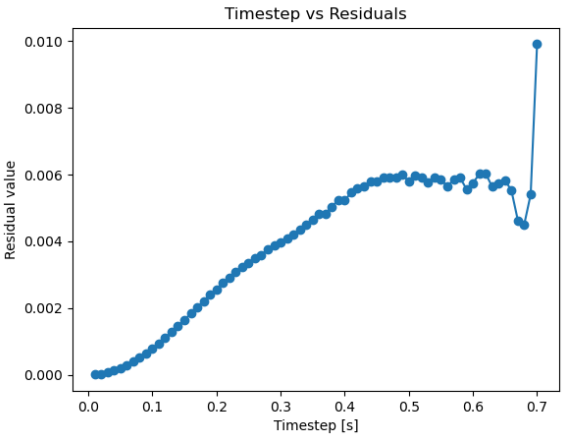


Figure 22: More Residuals

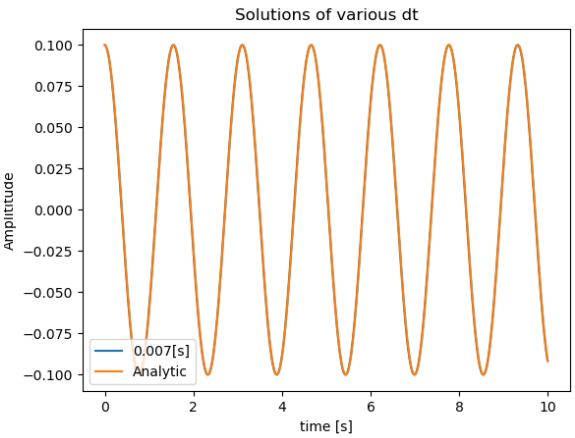


Figure 23: Plots of selection solutions

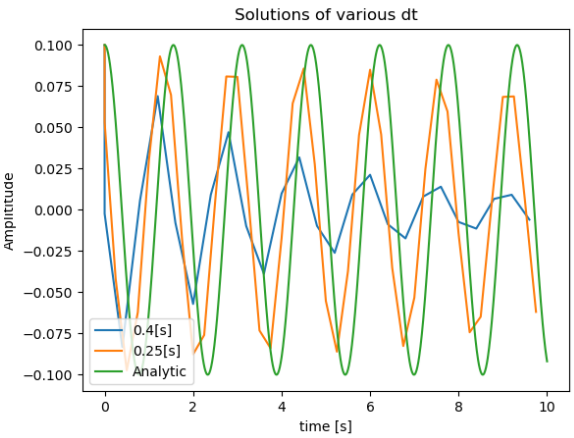


Figure 24: More plots of selection solutions

The residuals approaches zero as the timesteps are decreased, as shown for the timesteps chosen above. The residuals are much smaller than the previous Euler methods, and does not blow up for a wider range of timesteps. With these solutions, we would choose the timestep of $\Delta t = 0.01$s for section 4.3, as the residual does not become much smaller with further decrease in timestep size.

These results are nearly identical to that of section 4.1, which is expected as the small-angle approximation holds for the chosen starting angle. With these solutions, we would choose the timestep of $\Delta t = 0.01$s for section 4.3, where we find the θ value at which Equations (1) and (4) diverge. The residual does not become much smaller with further decrease in timestep size.

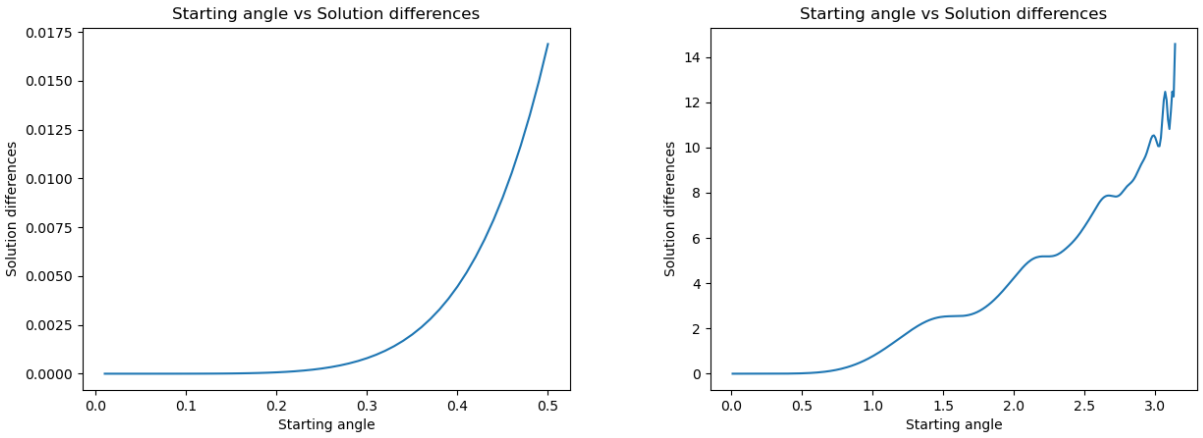
4.3 Finding θ_0 at which Equations (1) and (4) diverge

We will use $\Delta t = 0.02$s, vary the starting angle of the system.

We define the difference as:

$$\Delta_i = \frac{1}{n}(\theta_1(t_i) - \theta_4(t_i))^2 \tag{48}$$

Where n is the number of timesteps in the time domain. The divergence of the solutions grow quickly after about $\theta_0 = 0.3$rad. We will say that this is the point where the solutions of Equations (1) and (4) diverge. The solution different is much lower overall compared to the Euler methods.



Curiously, the rapid divergence of the exhibit a strange behavior after about $\theta_0 = 1.5$rad, which is shown in the right plot above. It is similar to that of the implicit differences. However, the differences do not blow up as much as the previous methods, even as the starting angle approaches π[rads], which is when the pendulum starts at an unstable equilibrium.

5 Appendix: Code

Section 2.1

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

# constants
l = 0.6
g = -9.81
theta_start = 0.1

#########################################################################
# explicit residuals
dts = np.linspace(0.49, 0.01, 49)
# dts = np.linspace(0.04, 0.001, 40)
res = []
sums = []

fig1 = plt.figure()
for dt in dts:
    dt = round(dt,3)
    theta = [theta_start]
    theta_dot = [0]
    x_dt = [0]
    for x in range(int(10. / dt)):
        x_dt.append(dt * x)

        next_theta = theta[-1] + theta_dot[-1] * dt
        next_theta_dot = theta_dot[-1] + (g / l) * np.sin(theta[-1]) * dt
        theta.append(next_theta)
        theta_dot.append(next_theta_dot)

    print(dt)
    sum = 0
    analytic = hp.analyticalSol(x_dt,theta_start)
    for i in range(len(theta)):
        sum = sum + (theta[i] - analytic[i]) ** 2
    sums.append((1 / len(theta)) * sum)
```

```
    # plotting
    if dt == 0.01 or dt == 0.03:
    # if dt in [0.007,0.005, 0.002]:
        plt.plot(x_dt, theta)

plt.title('Solutions␣of␣various␣dt')
plt.plot(np.linspace(0,10,1001), hp.analyticalSol(np.linspace(0,10,1001),
    ↪ theta_start))
# plt.legend(['0.007[s]','0.005[s]','0.002[s]','Analytic'])
plt.legend(['0.03[s]','0.01[s]','Analytic'])
plt.ylabel('Amplititude')
plt.xlabel('Time␣[s]')

plt.figure()
plt.title('Timestep␣vs␣Residuals')
plt.plot(dts, sums, marker='o')
plt.xlabel('Timestep␣[s]')
plt.ylabel('Residual␣value')
plt.show()
```

Section 2.2

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

# constants
l = 0.6
g = -9.81
theta_start = 0.1

##########################################################################
# explicit residuals
# dts = np.linspace(0.49, 0.01, 49)
dts = np.linspace(0.04, 0.001, 40)
res = []
sums = []

fig1 = plt.figure()
for dt in dts:
    dt = round(dt,3)
    theta = [theta_start]
```

```
    theta_dot = [0]
    x_dt = [0]
    for x in range(int(10. / dt)):
        x_dt.append(dt * x)

        next_theta = theta[-1] + theta_dot[-1] * dt
        next_theta_dot = theta_dot[-1] + (g / l) * theta[-1] * dt
        theta.append(next_theta)
        theta_dot.append(next_theta_dot)

    print(dt)
    sum = 0
    analytic = hp.analyticalSol(x_dt,theta_start)
    for i in range(len(theta)):
        sum = sum + (theta[i] - analytic[i]) ** 2
    sums.append((1 / len(theta)) * sum)

    # plotting
    # if dt == 0.01 or dt == 0.03:
    if dt in [0.007,0.005, 0.002]:
        plt.plot(x_dt, theta)

plt.title('Solutions␣of␣various␣dt')
plt.plot(np.linspace(0,10,1001), hp.analyticalSol(np.linspace(0,10,1001),
    ↪ theta_start))
plt.legend(['0.007[s]','0.005[s]','0.002[s]','Analytic'])
# plt.legend(['0.03[s]','0.01[s]','Analytic'])
plt.ylabel('Amplititude')
plt.xlabel('Time␣[s]')

plt.figure()
plt.title('Timestep␣vs␣Residuals')
plt.plot(dts, sums, marker='o')
plt.xlabel('Timestep␣[s]')
plt.ylabel('Residual␣value')
plt.show()
```

Section 2.3

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

```
# constants
l = 0.6
g = -9.81
dt = 0.02
theta_starts = np.linspace(0.01, 0.5, 100)

######################################################################
# explicit


difference = []

for theta_start in theta_starts:
    dt = round(dt,3)
    theta = [theta_start]
    theta_dot = [0]
    x_dt = [0]
    for x in range(int(10. / dt)):
        x_dt.append(dt * x)

        next_theta = theta[-1] + theta_dot[-1] * dt
        next_theta_dot = theta_dot[-1] + (g / l) * np.sin(theta[-1]) * dt
        theta.append(next_theta)
        theta_dot.append(next_theta_dot)

    equation1theta = theta.copy()

    theta = [theta_start]
    theta_dot = [0]
    x_dt = [0]
    for x in range(int(10. / dt)):
        x_dt.append(dt * x)

        next_theta = theta[-1] + theta_dot[-1] * dt
        next_theta_dot = theta_dot[-1] + (g / l) * theta[-1] * dt
        theta.append(next_theta)
        theta_dot.append(next_theta_dot)

    equation4theta = theta.copy()

    print(theta_start)
```

```
    diff = 0
    for eq1, eq4 in zip(equation1theta, equation4theta):
        diff += (eq1 - eq4)**2

    difference.append(diff/len(equation1theta))

plt.plot(theta_starts, difference)
plt.title('Starting angle vs Solution differences')
# plt.legend(['0.01[rad]','0.1[rad]','0.5[rad]','1[rad]'])
plt.ylabel('Solution differences')
plt.xlabel('Starting angle')
plt.show()
```

Section 3.1

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

# constants
l = 0.6
g = -9.81
theta_start = 0.1
##########################################################################
# implicit residuals
dts = np.linspace(0.50, 0.01, 50)
# dts = np.linspace(0.02, 0.001, 20)

sums = []

for dt in dts:
    theta = [theta_start]
    theta_dot = [0]
    x_dt = [0]

    for x in range(int(10. / dt)):
        x_dt.append(dt * x)

        # The system of equations to solve
        # next_theta = theta + next_theta_dot * dt
        # next_theta_dot = theta_dot + (g / l) * np.sin(next_theta) * dt
```

```
        theta_func = lambda x: theta[-1] + x[1] * dt
        theta_dot_func = lambda x: theta_dot[-1] + (g/l) * np.sin(x[0]) * dt
        soe = [theta_func, theta_dot_func]

        dtheta_func = lambda x: 2*(x[0] - theta_func(x)) + 2*(x[1] -
            ↪ theta_dot_func(x))*(-1*(g/l) * np.cos(x[0]) * dt)
        dtheta_func_dot = lambda x: 2*(x[0] - theta_func(x))*(-1*dt) + 2*(x
            ↪ [1] - theta_dot_func(x))
        partials = [dtheta_func, dtheta_func_dot]
        [next_theta, next_theta_dot] = hp.gradientDescent(soe, [theta[-1],
            ↪ theta_dot[-1]], hp.errorDef1, partials, 0.01, 0.0000001)

        theta_dot.append(next_theta_dot)
        theta.append(next_theta)

    print(dt)
    sum = 0
    analytic = hp.analyticalSol(x_dt, theta_start)
    for i in range(len(theta)):
        ratio = dt / 0.01
        sum = sum + (theta[i] - analytic[i]) ** 2
    sums.append((1 / len(theta)) * sum)

    # plotting
    # if dt == 0.001 or dt == 0.008:
    if dt == 0.02 or dt == 0.005:
        plt.plot(x_dt, theta)

plt.title('Solutions␣of␣various␣dt')
plt.plot(np.linspace(0,10,1001), hp.analyticalSol(np.linspace(0,10,1001),
    ↪ theta_start))
plt.legend(['0.02[s]','0.005[s]','Analytic'])
plt.ylabel('Amplititude')
plt.xlabel('time␣[s]')

plt.figure()
plt.title('Timestep␣vs␣Residuals')
plt.plot(dts, sums, marker='o')
plt.xlabel('Timestep␣[s]')
plt.ylabel('Residual␣value')
plt.show()
```

Section 3.2

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

# constants
l = 0.6
g = -9.81
theta_start = 0.1
#########################################################################
# implicit residuals
# dts = np.linspace(0.50, 0.01, 50)
dts = np.linspace(0.02, 0.001, 20)

sums = []

for dt in dts:
    theta = [theta_start]
    theta_dot = [0]
    x_dt = [0]

    for x in range(int(10. / dt)):
        x_dt.append(dt * x)

        # The system of equations to solve
        # next_theta = theta + next_theta_dot * dt
        # next_theta_dot = theta_dot + (g / l) * np.sin(next_theta) * dt

        theta_func = lambda x: theta[-1] + x[1] * dt
        theta_dot_func = lambda x: theta_dot[-1] + (g/l) * x[0] * dt
        soe = [theta_func, theta_dot_func]

        dtheta_func = lambda x: 2*(x[0] - theta_func(x)) + 2*(x[1] -
            ↪ theta_dot_func(x))*(-1*(g/l) * dt)
        dtheta_func_dot = lambda x: 2*(x[0] - theta_func(x))*(-1*dt) + 2*(x
            ↪ [1] - theta_dot_func(x))
        partials = [dtheta_func, dtheta_func_dot]
        [next_theta, next_theta_dot] = hp.gradientDescent(soe, [theta[-1],
            ↪ theta_dot[-1]], hp.errorDef1, partials, 0.01, 0.0000001)
```

```
        theta_dot.append(next_theta_dot)
        theta.append(next_theta)

    print(dt)
    sum = 0
    analytic = hp.analyticalSol(x_dt, theta_start)
    for i in range(len(theta)):
        ratio = dt / 0.01
        sum = sum + (theta[i] - analytic[i]) ** 2
    sums.append((1 / len(theta)) * sum)

    # plotting
    if dt == 0.020 or dt == 0.008:
    # if dt == 0.2 or dt == 0.01:
        plt.plot(x_dt, theta)

plt.title('Timestep vs Residuals')
plt.plot(np.linspace(0,10,1001), hp.analyticalSol(np.linspace(0,10,1001),
    ↪ theta_start))
plt.legend(['0.02[s]','0.008[s]','Analytic'])
plt.ylabel('Amplititude')
plt.xlabel('time [s]')

plt.figure()
plt.title('Solutions of various dt')
plt.plot(dts, sums, marker='o')
plt.xlabel('Timestep [s]')
plt.ylabel('Residual value')
plt.show()
```

Section 3.3

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

# constants
l = 0.6
g = -9.81
dt = 0.02
theta_starts = np.linspace(0.01, 1, 100)
```

```
########################################################################
# implicit


difference = []

for theta_start in theta_starts:
    theta = [theta_start]
    theta_dot = [0]
    x_dt = [0]

    for x in range(int(10. / dt)):
        x_dt.append(dt * x)

        # The system of equations to solve
        # next_theta = theta + next_theta_dot * dt
        # next_theta_dot = theta_dot + (g / l) * np.sin(next_theta) * dt

        theta_func = lambda x: theta[-1] + x[1] * dt
        theta_dot_func = lambda x: theta_dot[-1] + (g/l) * x[0] * dt
        soe = [theta_func, theta_dot_func]

        dtheta_func = lambda x: 2*(x[0] - theta_func(x)) + 2*(x[1] -
            ↪ theta_dot_func(x))*(-1*(g/l) * dt)
        dtheta_func_dot = lambda x: 2*(x[0] - theta_func(x))*(-1*dt) + 2*(x
            ↪ [1] - theta_dot_func(x))
        partials = [dtheta_func, dtheta_func_dot]
        [next_theta, next_theta_dot] = hp.gradientDescent(soe, [theta[-1],
            ↪ theta_dot[-1]], hp.errorDef1, partials, 0.01, 0.0000001)

        theta_dot.append(next_theta_dot)
        theta.append(next_theta)

    equation1theta = theta.copy()

    theta = [theta_start]
    theta_dot = [0]
    x_dt = [0]

    for x in range(int(10. / dt)):
        x_dt.append(dt * x)
```

```
        # The system of equations to solve
        # next_theta = theta + next_theta_dot * dt
        # next_theta_dot = theta_dot + (g / l) * np.sin(next_theta) * dt

        theta_func = lambda x: theta[-1] + x[1] * dt
        theta_dot_func = lambda x: theta_dot[-1] + (g/l) * np.sin(x[0]) * dt
        soe = [theta_func, theta_dot_func]

        dtheta_func = lambda x: 2*(x[0] - theta_func(x)) + 2*(x[1] -
            ↪ theta_dot_func(x))*(-1*(g/l) * np.cos(x[0]) * dt)
        dtheta_func_dot = lambda x: 2*(x[0] - theta_func(x))*(-1*dt) + 2*(x
            ↪ [1] - theta_dot_func(x))
        partials = [dtheta_func, dtheta_func_dot]
        [next_theta, next_theta_dot] = hp.gradientDescent(soe, [theta[-1],
            ↪ theta_dot[-1]], hp.errorDef1, partials, 0.01, 0.0000001)

        theta_dot.append(next_theta_dot)
        theta.append(next_theta)

    equation4theta = theta.copy()

    print(theta_start)

    diff = 0
    for eq1, eq4 in zip(equation1theta, equation4theta):
        diff += (eq1 - eq4)**2

    difference.append(diff/len(equation1theta))

plt.plot(theta_starts, difference,marker='o')
plt.title('Starting␣angle␣vs␣Solution␣differences')
# plt.legend(['0.01[rad]','0.1[rad]','0.5[rad]','1[rad]'])
plt.ylabel('Solution␣differences')
plt.xlabel('Starting␣angle')
plt.show()
```

Section 4.1

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

```
#################################################################
# RK4
# constants
l = 0.6
g = -9.81
theta_start = 0.1

def rk4(theta, theta_dot):
    current_theta = theta[-1]
    current_omega = theta_dot[-1]

    theta1 = current_omega * dt
    omega1 = dOmega(current_theta) * dt

    theta2 = (current_omega + omega1*0.5) * dt
    omega2 = dOmega(current_theta + theta1*0.5) * dt

    theta3 = (current_omega + omega2*0.5) * dt
    omega3 = dOmega(current_theta + theta2*0.5) * dt

    theta4 = (current_omega + omega3) * dt
    omega4 = dOmega(current_theta + theta3) * dt

    theta.append(current_theta + (1/6)*(theta1 + 2*theta2 + 2*theta3 +
        ↪ theta4))
    theta_dot.append(current_omega + (1/6)*(omega1 + 2*omega2 + 2*omega3 +
        ↪ omega4))

def dOmega(theta_dd):
    return (g/l)*np.sin(theta_dd)

# dts = np.linspace(0.7, 0.01, 70)
dts = np.linspace(0.04, 0.001, 40)
sums = []

fig1 = plt.figure()
for dt in dts:
    dt = round(dt,3)
    theta = [theta_start]
    theta_dot = [0]
    x_dt = [0]
```

```
    for x in range(int(10. / dt)):
        x_dt.append(dt * x)
        rk4(theta, theta_dot)

    print(dt)
    sum = 0
    analytic = hp.analyticalSol(x_dt,theta_start)
    for i in range(len(theta)):
        sum = sum + (theta[i] - analytic[i]) ** 2
    sums.append((1 / len(theta)) * sum)

    # plotting
    # if dt in [0.4, 0.25]:
    if dt in [0.007]:
        plt.plot(x_dt, theta)

plt.title('Solutions␣of␣various␣dt')
plt.plot(np.linspace(0,10,1001), hp.analyticalSol(np.linspace(0,10,1001),
    ↪ theta_start))
plt.legend(['0.007[s]','Analytic'],loc='lower␣left')
# plt.legend(['0.4[s]','0.25[s]','Analytic'],loc='lower left')
plt.ylabel('Amplititude')
plt.xlabel('time␣[s]')


plt.figure()
plt.title('Timestep␣vs␣Residuals')
plt.plot(dts, sums, marker='o')
plt.xlabel('Timestep␣[s]')
plt.ylabel('Residual␣value')
plt.show()
```

Section 4.2

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

#################################################################
# RK4
# constants
```

```
l = 0.6
g = -9.81
theta_start = 0.1

def rk4(theta, theta_dot):
    current_theta = theta[-1]
    current_omega = theta_dot[-1]

    theta1 = current_omega * dt
    omega1 = dOmega(current_theta) * dt

    theta2 = (current_omega + omega1*0.5) * dt
    omega2 = dOmega(current_theta + theta1*0.5) * dt

    theta3 = (current_omega + omega2*0.5) * dt
    omega3 = dOmega(current_theta + theta2*0.5) * dt

    theta4 = (current_omega + omega3) * dt
    omega4 = dOmega(current_theta + theta3) * dt

    theta.append(current_theta + (1/6)*(theta1 + 2*theta2 + 2*theta3 +
       ↪ theta4))
    theta_dot.append(current_omega + (1/6)*(omega1 + 2*omega2 + 2*omega3 +
       ↪ omega4))

def dOmega(theta_dd):
    return (g/l)*theta_dd

dts = np.linspace(0.7, 0.01, 70)
# dts = np.linspace(0.04, 0.001, 40)
sums = []

fig1 = plt.figure()
for dt in dts:
    dt = round(dt,3)
    theta = [theta_start]
    theta_dot = [0]
    x_dt = [0]

    for x in range(int(10. / dt)):
        x_dt.append(dt * x)
        rk4(theta, theta_dot)
```

```
    print(dt)
    sum = 0
    analytic = hp.analyticalSol(x_dt,theta_start)
    for i in range(len(theta)):
        sum = sum + (theta[i] - analytic[i]) ** 2
    sums.append((1 / len(theta)) * sum)

    # plotting
    if dt in [0.4, 0.25]:
    # if dt in [0.007]:
        plt.plot(x_dt, theta)

plt.title('Solutions␣of␣various␣dt')
plt.plot(np.linspace(0,10,1001), hp.analyticalSol(np.linspace(0,10,1001),
    ↪ theta_start))
# plt.legend(['0.007[s]','Analytic'],loc='lower left')
plt.legend(['0.4[s]','0.25[s]','Analytic'],loc='lower␣left')
plt.ylabel('Amplititude')
plt.xlabel('time␣[s]')


plt.figure()
plt.title('Timestep␣vs␣Residuals')
plt.plot(dts, sums, marker='o')
plt.xlabel('Timestep␣[s]')
plt.ylabel('Residual␣value')
plt.show()
```

Section 4.3

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

# constants
l = 0.6
g = -9.81
dt = 0.02
theta_starts = np.linspace(0.01, np.pi, 314)

def rk4(theta, theta_dot):
```

```
    current_theta = theta[-1]
    current_omega = theta_dot[-1]

    theta1 = current_omega * dt
    omega1 = dOmega(current_theta) * dt

    theta2 = (current_omega + omega1*0.5) * dt
    omega2 = dOmega(current_theta + theta1*0.5) * dt

    theta3 = (current_omega + omega2*0.5) * dt
    omega3 = dOmega(current_theta + theta2*0.5) * dt

    theta4 = (current_omega + omega3) * dt
    omega4 = dOmega(current_theta + theta3) * dt

    theta.append(current_theta + (1/6)*(theta1 + 2*theta2 + 2*theta3 +
       ↪ theta4))
    theta_dot.append(current_omega + (1/6)*(omega1 + 2*omega2 + 2*omega3 +
       ↪ omega4))

def dOmega(theta_dd):
    if isSin:
      return (g/l)*np.sin(theta_dd)
    else:
        return (g / l) * theta_dd

#########################################################################
# rk4


difference = []
isSin = True

for theta_start in theta_starts:
    dt = round(dt,3)
    theta = [theta_start]
    theta_dot = [0]
    x_dt = [0]
    isSin = True

    for x in range(int(10. / dt)):
        x_dt.append(dt * x)
```

```
        rk4(theta, theta_dot)

    equation1theta = theta.copy()

    theta = [theta_start]
    theta_dot = [0]
    x_dt = [0]
    isSin = False
    for x in range(int(10. / dt)):
        x_dt.append(dt * x)
        rk4(theta, theta_dot)

    equation4theta = theta.copy()

    print(theta_start)

    diff = 0
    for eq1, eq4 in zip(equation1theta, equation4theta):
        diff += (eq1 - eq4)**2

    difference.append(diff/len(equation1theta))

plt.plot(theta_starts, difference)
plt.title('Starting angle vs Solution differences')
# plt.legend(['0.01[rad]','0.1[rad]','0.5[rad]','1[rad]'])
plt.ylabel('Solution differences')
plt.xlabel('Starting angle')
plt.show()
```

Helper functions

```
import numpy as np

def analyticalSol(x, theta_start):
    y = []
    for i, num in enumerate(x):
        y.append(theta_start*np.cos(4.04351*x[i]))
    return y

def gradientDescent(soe, initialGuess, errorDef, partials, alpha,
    ↪ errorThreshold):
```

```
    #make initial newIter
    newResult = []
    for equation in soe:
        newResult.append(equation(initialGuess))

    # initialError
    error = errorDef(initialGuess, newResult)

    #first try!
    if(error < errorThreshold):
        return initialGuess

    #first new guess
    newIter = []
    for i, partial in enumerate(partials):
        newIter.append(initialGuess[i] - alpha * partial(initialGuess))

    newResult = []
    for equation in soe:
        newResult.append(equation(newIter))
    error = errorDef(newIter, newResult)

    count = 1

    #iterations of gradient descent
    while(error > errorThreshold):
        oldIter = newIter.copy()
        newIter = []

        for i, partial in enumerate(partials):
            newIter.append(oldIter[i] - alpha*partial(oldIter))

        newResult = []
        for equation in soe:
            newResult.append(equation(newIter))

        error = errorDef(newIter, newResult)

    return newIter

def errorDef1(initials, newIter):
    errorSum = 0
```

```
    for i, initial in enumerate(initials):
        errorSum = errorSum + (initials[i] - newIter[i])**2
    return errorSum
```