Justine John "JJ" A. Serdoncillo
October 10, 2022

Homework 1

Number 1 a)

Calculating the analytical solution we find:

$$\frac{dy}{dt} = -2y$$
$$\int \frac{dy}{y} = \int -2dt$$
$$[ln|y|]_{y_0}^{y} = -2t$$
$$ln|\frac{y}{y_0}| = -2t$$
$$\frac{y}{y_0} = -2t$$

$$\boxed{y = y_0 e^{-2t} = 4e^{-2t}}$$

Using the code used in the appendix, the following plots were made below.

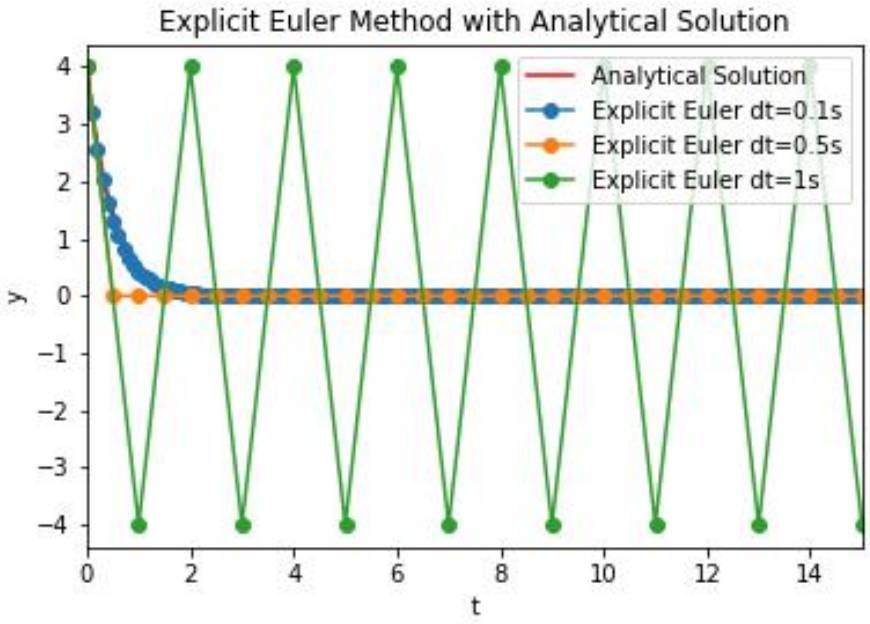

Figure 1: Explicit Euler at different time steps

From the figure above it can be seen that all of the different time steps were stable. However, the largest time step's output oscillates between 4 and -4 which doesn't help depict the actual solution. It seems to be the critical time step between stability and unstability. For the medium time step, it immediately jumped to 0 and became it's steady state value which is somewhat true but doesn't show the details of the exponential function. The smallest time step accurately depicts the analytical equation properly.

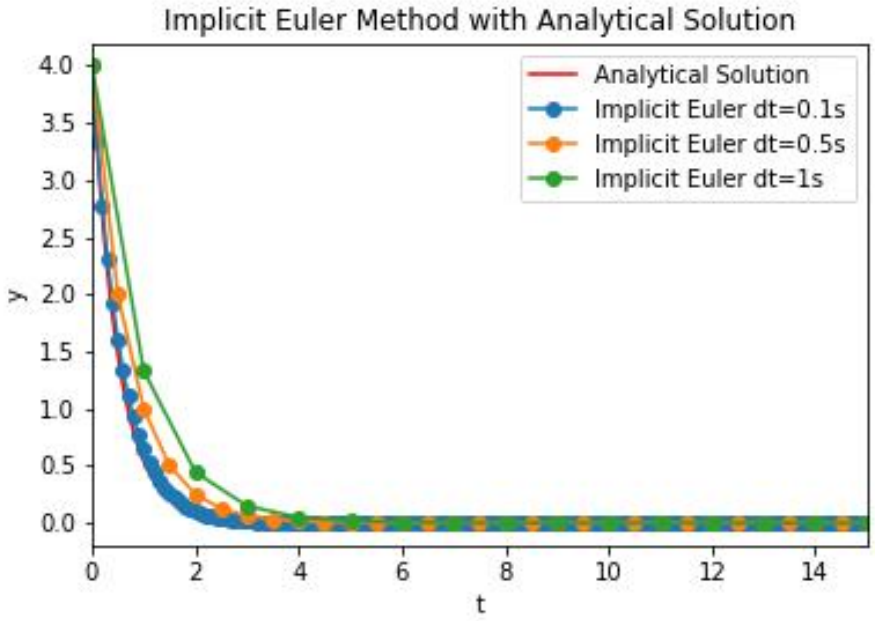

Figure 2: Implicit Euler at different time steps

For the implicit euler method, all of the time steps were all seemingly accurate and stable. It's accuracy increased as the time step decreased and was even almost on top of the exact solution for the smallest time step.

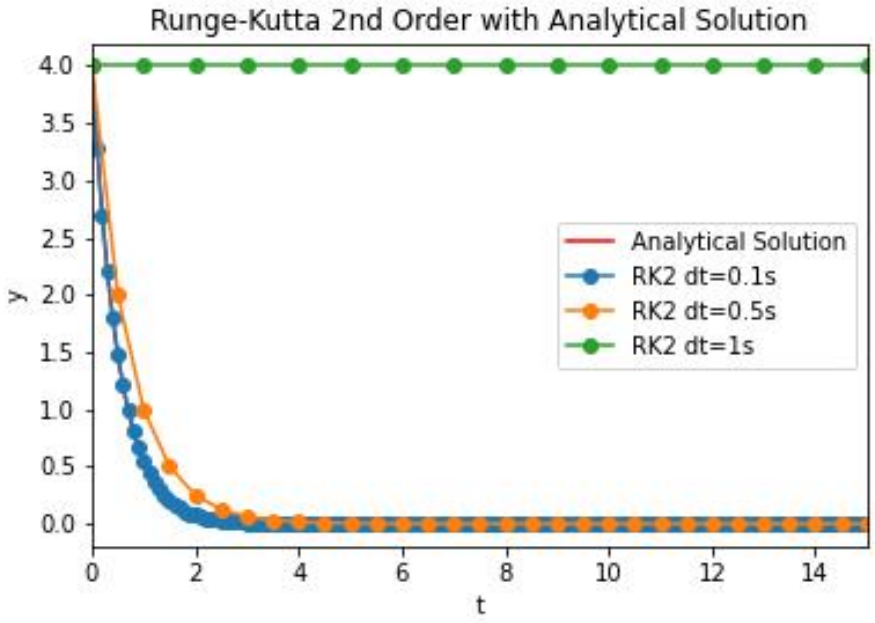

Figure 3: Runge-Kutta 2nd order at different time steps

For the 2nd order Runge-Kutta method, the plots are similar to the implicit euler except for the largest time step. It seemed like it is stable but only hovers at 4 which doesn't explain the analytical solution whatsoever.

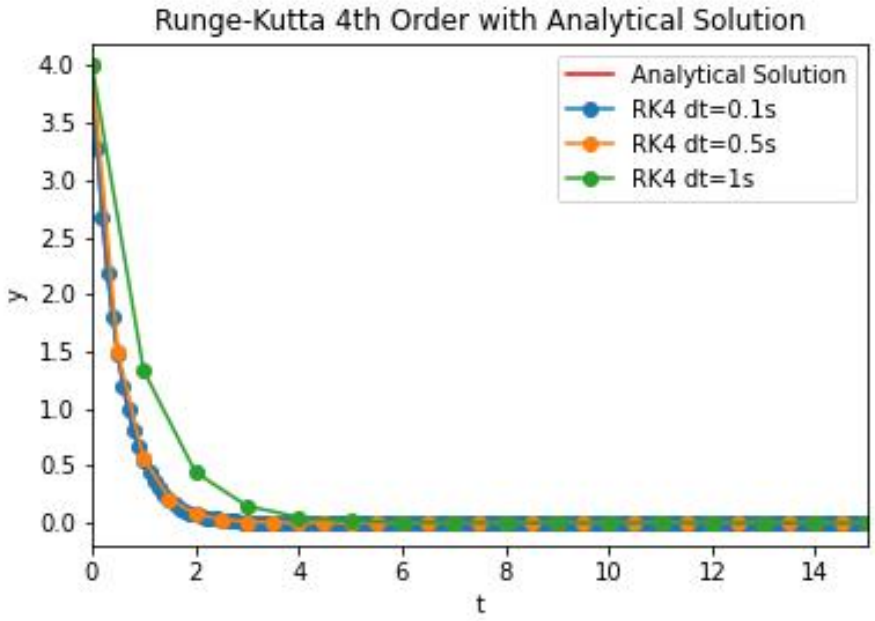

Figure 4: Runge-Kutta 4th order at different time steps

The 4th order Runge-Kutta method all have the plots stable and accurate for the different time steps. Comparing with the implicit euler method, it can be seen that it does not perform as well with the largest time step but does better in the medium time step.

Number 1 b)

Using the model equation $\frac{dy}{dt} = g(y,t) = cy$ to the given equation. It can be found that $c = -2$. The critical time step for stable but oscillating can be found by forcing $-1 \leq \sigma \leq 0$.

(a) Explicit Euler:

$$\frac{y_{n+1} - y_n}{\Delta t} = -2y_n$$
$$y_{n+1} = y_n - 2\Delta t y_n$$
$$y_{n+1} = y_n(1 - 2\Delta t)$$
$$\sigma = (1 - 2\Delta t)$$
$$-1 \leq 1 - 2\Delta t \leq 0$$
$$-2 \leq -2\Delta t \leq -1$$
$$1 > \Delta t > \frac{1}{2}$$
$$\frac{1}{2} < \Delta t < 1$$

critical time step at $\boxed{\Delta t = \frac{1}{2}}$

(b) Implicit Euler:

$$\frac{y_{n+1} - y_n}{\Delta t} = -2y_{n+1}$$
$$y_{n+1} = \frac{y_n}{1 + 2\Delta t}$$
$$\sigma = \frac{1}{1 + 2\Delta t}$$

$$-1 \leq \frac{1}{1+2\Delta t} \leq 0$$
$$-1 - 2\Delta t \leq 1$$
$$-2\Delta t \leq 2$$
$$-1 < \Delta t$$

no critical time step

(c) Runge-Kutta 2nd order:

$$y_{n+1} = y_n(1 + c\Delta t + \frac{1}{2}(c\Delta t)^2)$$
$$\sigma = 1 + c\Delta t + \frac{1}{2}(c\Delta t)^2$$
$$\sigma = 1 - 2\Delta t + \frac{1}{2}(-2\Delta t)^2$$
$$0 \leq 1 - 2\Delta t + 2\Delta t^2 \leq -1$$

from here it can be seen that there is no critical time step for oscillation yet stable

$$|1 - 2\Delta t + 2\Delta t^2| \leq 1$$
$$-1 \leq 1 - 2\Delta t + 2\Delta t^2 \leq 1$$

solving from both sides

$$0 < \Delta t < 1$$

critical time step at $\Delta t = 1$

(d) Runge-Kutta 4th order:

$$y_{n+1} = y_n(1 + c\Delta t + \frac{1}{2}(c\Delta t)^2 + \frac{1}{6}(c\Delta t)^3 + \frac{1}{24}(c\Delta t)^4)$$
$$\sigma = 1 + c\Delta t + \frac{1}{2}(c\Delta t)^2 + \frac{1}{6}(c\Delta t)^3 + \frac{1}{24}(c\Delta t)^4$$
$$\sigma = 1 - 2\Delta t + \frac{1}{2}(-2\Delta t)^2 + \frac{1}{6}(-2\Delta t)^3 + \frac{1}{24}(-2\Delta t)^4$$
$$-1 \leq 1 - 2\Delta t + 2\Delta t^2 - \frac{4}{3}\Delta t^3 + \frac{2}{3}\Delta t^4 \leq 0$$

from here it can be seen that there is no critical time step for oscillation yet stable

$$|1 - 2\Delta t + 2\Delta t^2 - \frac{4}{3}\Delta t^3 + \frac{2}{3}\Delta t^4| \leq 1$$

solving for Δt we get

$$0 < \Delta t < 1.393$$

critical time step at $\Delta t = 1.393$

Number 2 a)

Using the model equation $\frac{dy}{dt} = g(y,t) = cy$ to the given equation. It can be found that $c = -(2+0.01^2)$. The estimated maximum stable time step over the entire domain of interest can be found by forcing $\sigma \leq 1$ and $t \leq 15$.

(a) Explicit Euler:

$$\frac{y_{n+1} - y_n}{\Delta t} = -(2 + 0.01t^2)y_n$$
$$y_{n+1} = y_n - (2 + 0.01t^2)\Delta t y_n$$
$$y_{n+1} = y_n(1 - (2 + 0.01t^2)\Delta t)$$
$$\sigma = (1 - (2 + 0.01t^2)\Delta t)$$
$$|1 - (2 + 0.01t^2)\Delta t| \leq 1$$
$$-1 \leq 1 - (2 + 0.01t^2)\Delta t \leq 1$$
$$-2 \leq -(2 + 0.01t^2)\Delta t$$
$$-(2 + 0.01t^2)\Delta t < 2$$
$$\Delta t < \frac{1}{1 + 0.005t^2}$$

as $t \to \infty$, $\frac{1}{1+0.005t^2}$ decreases. To get the maximum stable time step, we minimize the value which occurs at $t = 0$
Estimated maximum stable time step at $\boxed{\Delta t = 1}$

(b) Implicit Euler:

$$\frac{y_{n+1} - y_n}{\Delta t} = -(2 + 0.01t^2)y_{n+1}$$
$$y_{n+1} = \frac{y_n}{1 + (2 + 0.01t^2)\Delta t}$$
$$\sigma = \frac{1}{1 + (2 + 0.01t^2)\Delta t}$$
$$\left|\frac{1}{1 + (2 + 0.01t^2)\Delta t}\right| \leq 1$$
$$\frac{1}{1 + (2 + 0.01t^2)\Delta t} \leq 1$$
$$1 \leq 1 + (2 + 0.01t^2)\Delta t$$
$$0 \leq (2 + 0.01t^2)\Delta t$$

$2 + 0.01t^2$ is always greater than 0, hence

$$0 \leq \Delta t$$

$\boxed{\text{no critical time step}}$

(c) Runge-Kutta 2nd order:

$$y_{n+1} = y_n(1 + c\Delta t + \frac{1}{2}(c\Delta t)^2)$$
$$\sigma = 1 + c\Delta t + \frac{1}{2}(c\Delta t)^2$$
$$\sigma = 1 - (2 + 0.01t^2)\Delta t + \frac{1}{2}(-(2 + 0.01t^2)\Delta t)^2$$
$$|1 - (2 + 0.01t^2)\Delta t + \frac{1}{2}(2 + 0.01t^2)^2\Delta t^2| \leq 1$$

By evaluating at different values of t it can be seen that the maximum Δt also changes. At $t = 15$ the maximum stable time step is $\Delta t = 0.471$, however at $t = 0$ the maximum stable time step is $\Delta t = 1$. Taking the higher value of the two, the estimated maximum stable time step at $\boxed{\Delta t = 1}$

(d) Runge-Kutta 4th order:

$$y_{n+1} = y_n(1 + c\Delta t + \frac{1}{2}(c\Delta t)^2 + \frac{1}{6}(c\Delta t)^3 + \frac{1}{24}(c\Delta t)^4)$$

$$\sigma = 1 + c\Delta t + \frac{1}{2}(c\Delta t)^2 + \frac{1}{6}(c\Delta t)^3 + \frac{1}{24}(c\Delta t)^4$$

$$\sigma = 1 - (2 + 0.01t^2)\Delta t + \frac{1}{2}(-(2 + 0.01t^2)\Delta t)^2 + \frac{1}{6}(-(2 + 0.01t^2)\Delta t)^3 + \frac{1}{24}(-(2 + 0.01t^2)\Delta t)^4$$

$$|1 - (2 + 0.01t^2)\Delta t + \frac{1}{2}(-(2 + 0.01t^2)\Delta t)^2 + \frac{1}{6}(-(2 + 0.01t^2)\Delta t)^3 + \frac{1}{24}(-(2 + 0.01t^2)\Delta t)^4| \leq 1$$

Using the same thought process as for the previous scheme, at $t = 15$ the maximum stable time step is $\Delta t = 0.348$, however at $t = 0$ the maximum stable time step is $\Delta t = 0.74$. Taking the higher value of the two, the estimated maximum stable time step at $\boxed{\Delta t = 0.74}$

Number 2 b)

Calculating the analytical solution we find:

$$\frac{dy}{dt} = -(2 + 0.01t^2)y$$
$$\int \frac{dy}{y} = \int -(2 + 0.01t^2)dt$$
$$[ln|y|]_{y_0}^{y} = -(2t + \frac{0.01}{3}t^3)$$
$$ln|\frac{y}{y_0}| = -(2t + \frac{0.01}{3}t^3)$$
$$\frac{y}{y_0} = -(2t + \frac{0.01}{3}t^3)$$

$$\boxed{\text{y} = \text{y}_0 e^{-(2t + \frac{0.01}{3}t^3)} = 4e^{-(2t + \frac{0.01}{3}t^3)}}$$

The implicit solution can be derived by taking two equations with two unknowns. More can be seen in the code below.
Using the code used in the appendix, the following plots were made below.

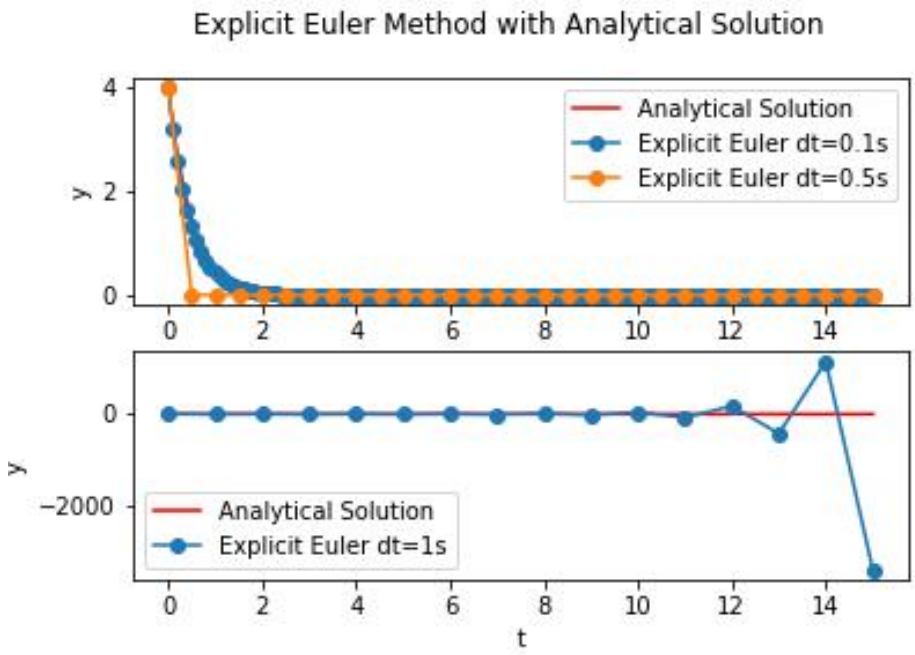

Figure 5: Explicit Euler at different time steps

Similar with the case with the previous problem, the largest time step was unstable and the medium time step, although stable quickly moved to near 0 which lost the details for the actual solution. The smallest time step is both stable and accurate.

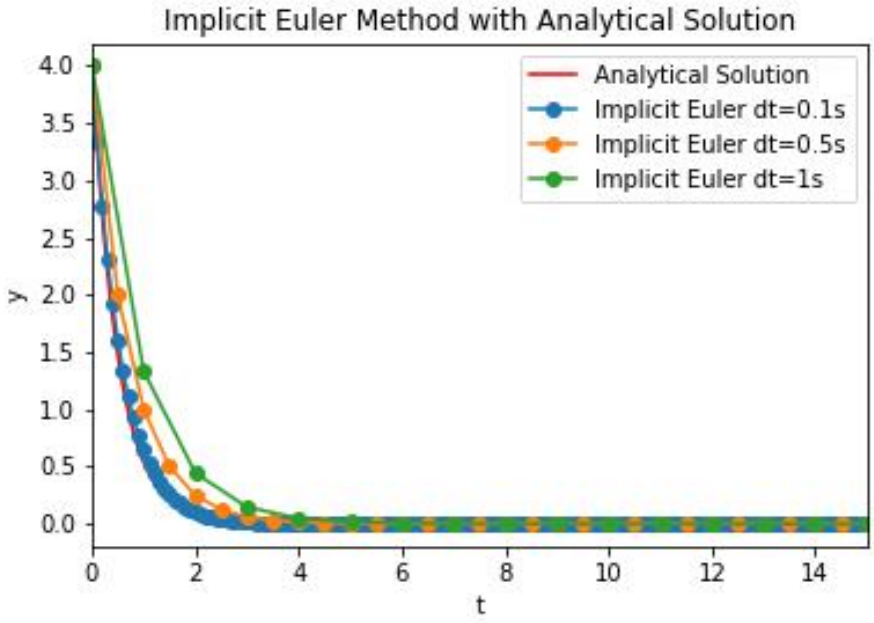

Figure 6: Implicit Euler at different time steps

For implicit euler, all of the plots were stable and getting more accurate as the time step decreased.

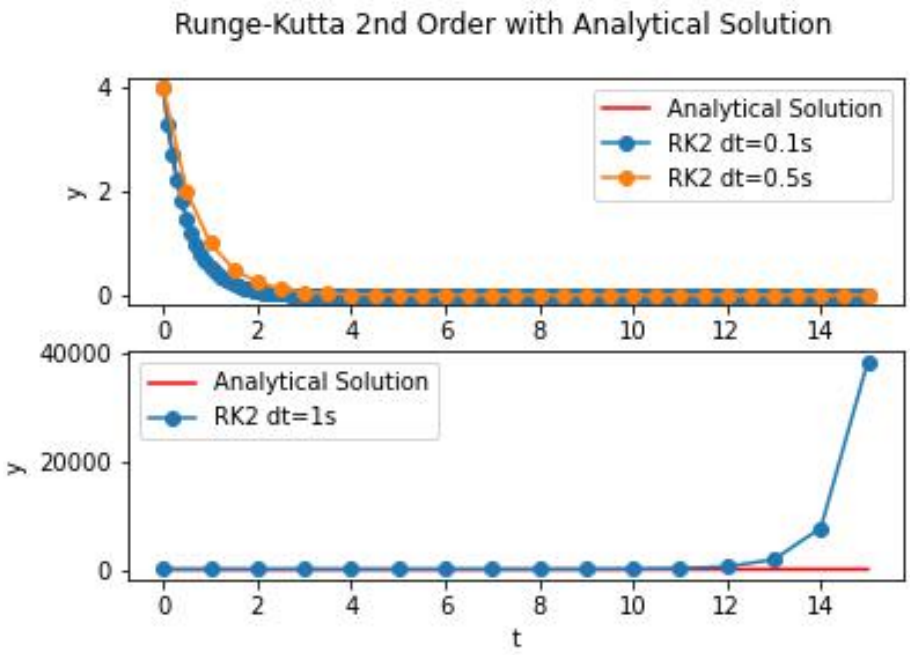

Figure 7: Runge-Kutta 2nd order at different time steps

The plots showed to be stable for the first two time steps but was unstable in the largest time step.

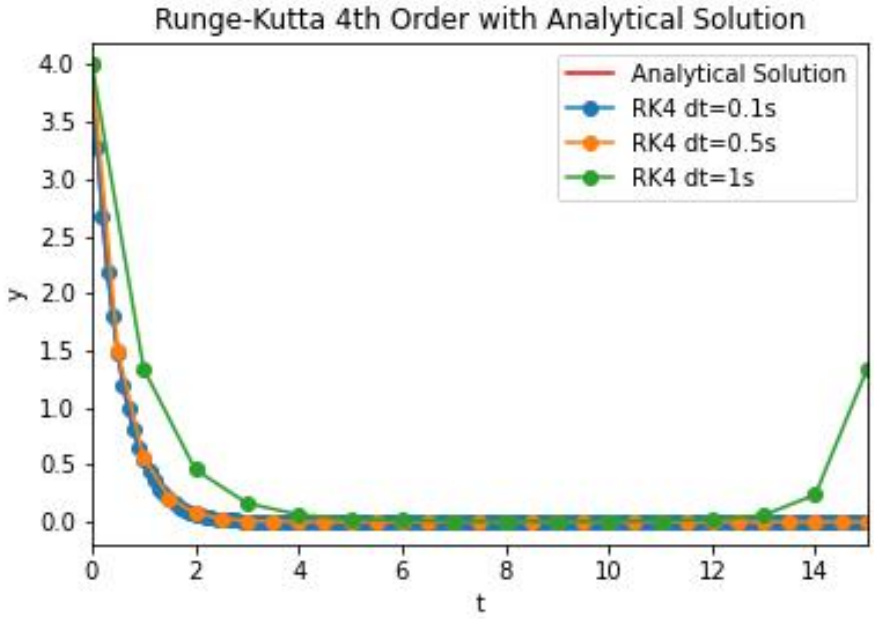

Figure 8: Runge-Kutta 4th order at different time steps

For the Runge-Kutta 4th order method, it seemed like all of the plots were somewhat accurate within the range of the domain but the largest time step showed signs of instability near the end.

Number 3 a)

Linearizing the second order equation, the following is derived below

$$\frac{d\theta}{dt} = \dot{\theta} \quad \text{and} \quad \frac{d\dot{\theta}}{dt} = -\frac{g}{l}\theta$$

or converted into matrices

$$\frac{d}{dt}\begin{bmatrix}\theta \\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}0 & 1 \\ -\frac{g}{l} & 0\end{bmatrix}\begin{bmatrix}\theta \\ \dot{\theta}\end{bmatrix}$$

Using the code used in the appendix, the following plots were made below.

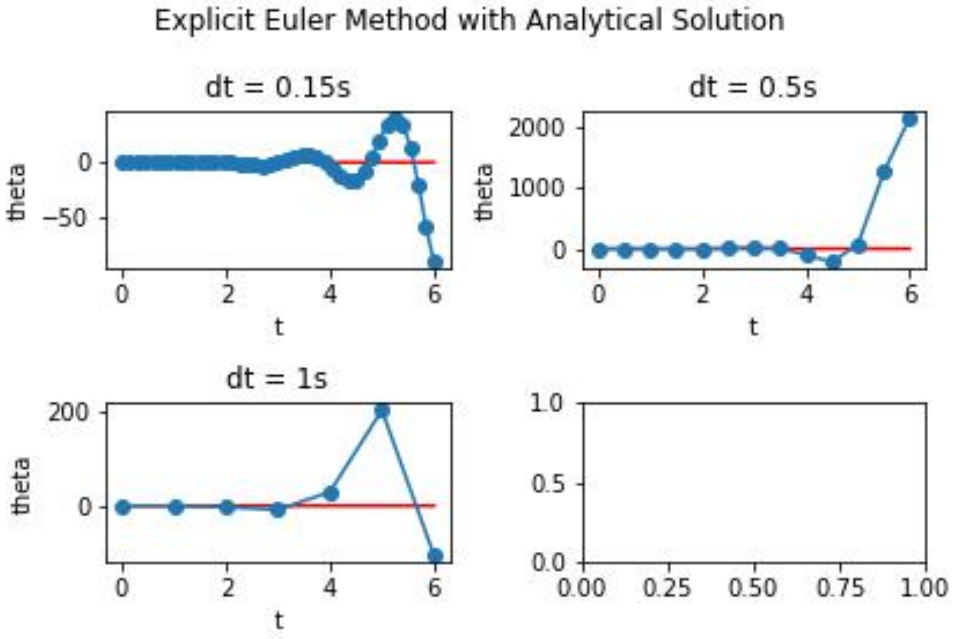

Figure 9: Explicit Euler at different time steps

From the figure above which uses the explicit Euler method, it can be seen that the plots of all of the different time steps are neither accurate nor stable. At the smallest time step, the first few values might be accurate enough but the oscillation grew larger and deviated from the exact solution.

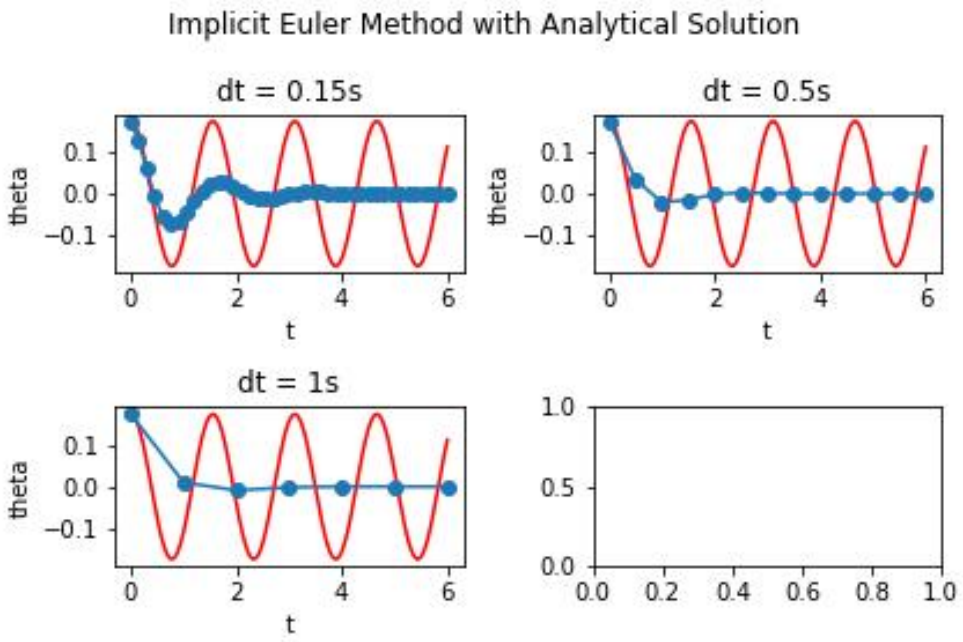

Figure 10: Implicit Euler at different time steps

For the implicit scheme, the case is different as all of the plots of the different timesteps were all stable. However, looking at the figure, it can be seen that neither of these looks even close to the exact solution.

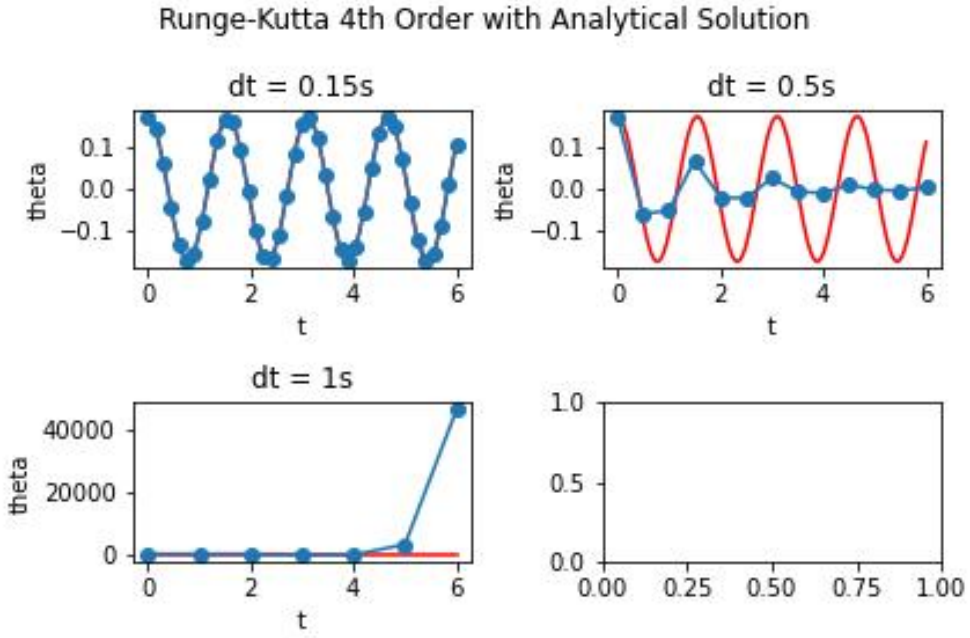

Figure 11: Runge-Kutta 4^{th} order at different time steps

Lastly for the Runge-Kutta 4^{th} order method, the figures were stable for the first two time steps. In the case of accuracy, the second time step was not accurate in representing the exact solution but showed an oscillatory though. The smallest time step looks to be both stable and

Number 3 b)

Using the code used in the appendix, the following plots were made below.

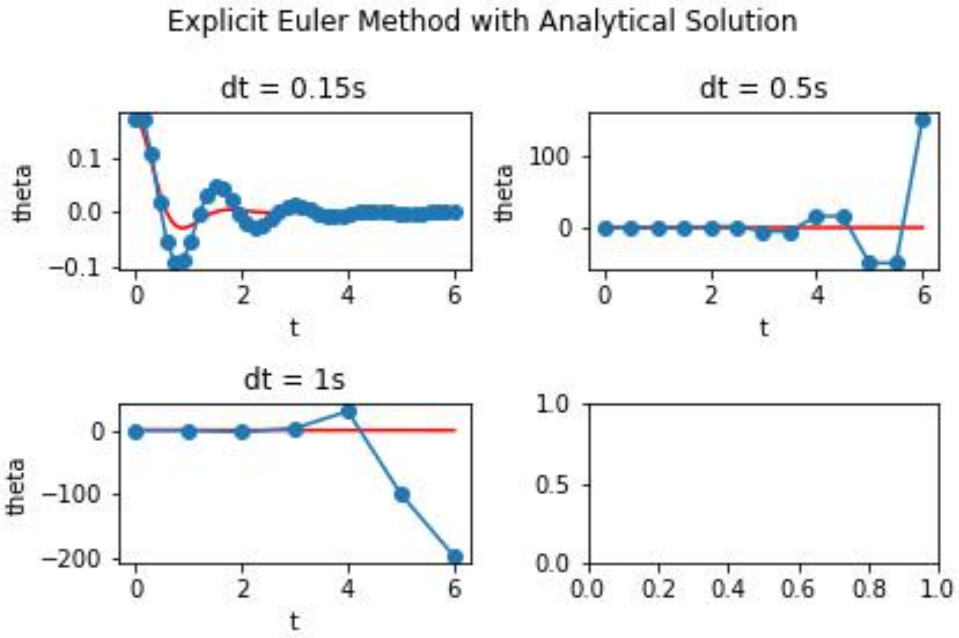

Figure 12: Explicit Euler at different time steps

From the figure above, it can be seen that the explicit euler method showed to be stable in the smallest timestep. Even though it didn't match the frequency of the oscillation nor the damping right, there was a damped motion that can be seen and that the value rose at almost the same time step as with the exact solution. For the other time steps, it can be seen that they are unstable. For the second time step, the oscillation was growing rather than decreasing while for the last time step, the oscillation is barely seen.

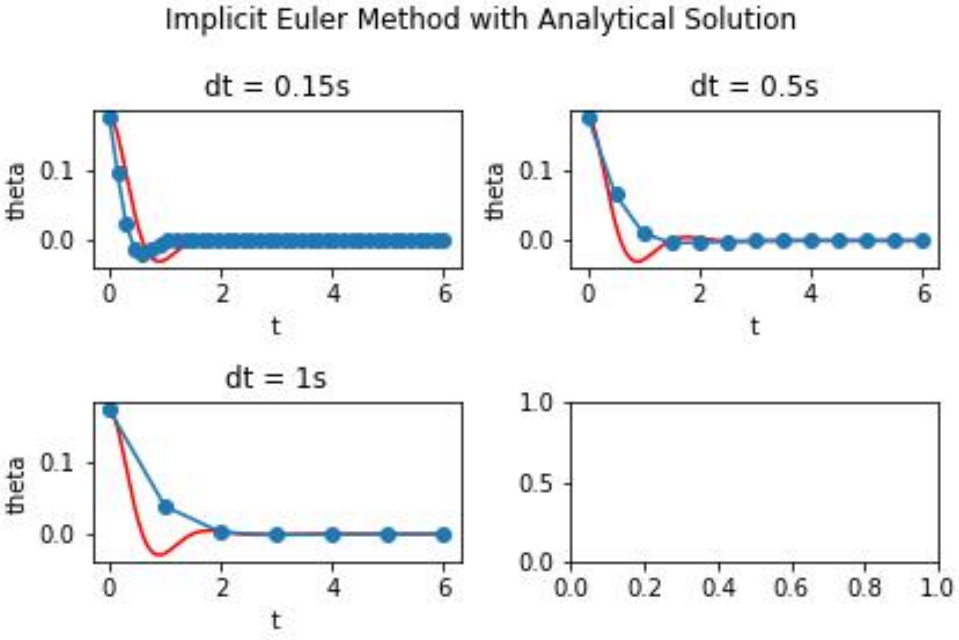

Figure 13: Implicit Euler at different time steps

For the implict euler method, all of the time steps showed to be stable and even approached the right steady state solution. However it's accuracy seems to be highly compromised because in the largest time step, there seems to be no damping seen and barely noticeable in the medium time step. In the smallest time step however, it can be seen to mimic the trend of the exact solution but not accurately enough.

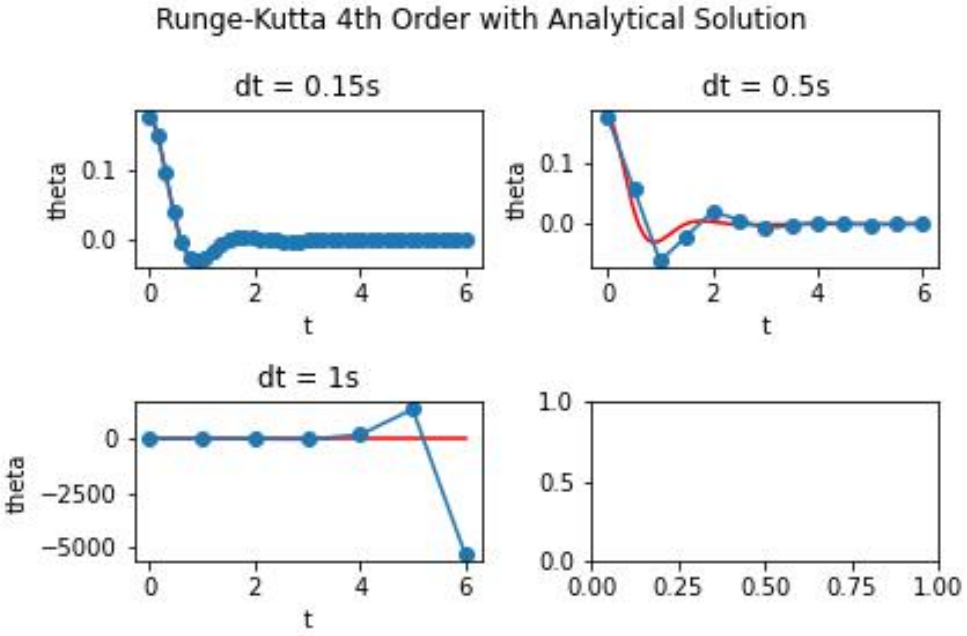

Figure 14: Runge-Kutta 4th order at different time steps

Lastly for the Runge-Kutta method, it can be seen that the largest time time is unstable and doesn't seem to be accurate as well. In the medium time step, it was stable and that some of the points were close to the actual value and it showed the damped response in the actual solution. For the smallest time step, it can be seen that it accurately depicted the actual solution and also stable as well.

Number 4 a)

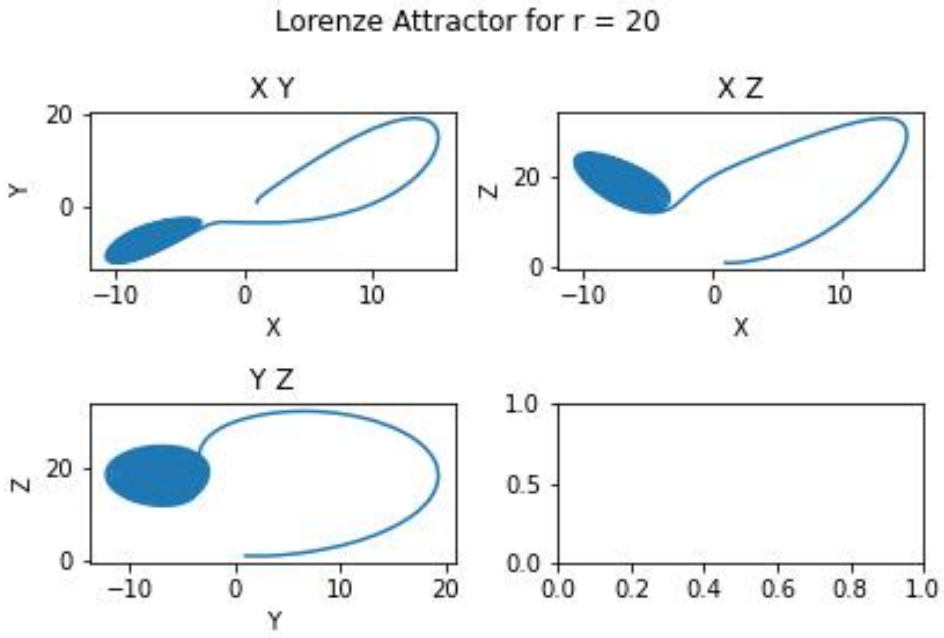

Figure 15: Lorenz attractor at r=20 starting at (1,1,1)

The figure above shows the plot of the attractor in all planes and it can be seen that the figures are stable and approach a specific domain in space.

Number 4 b)

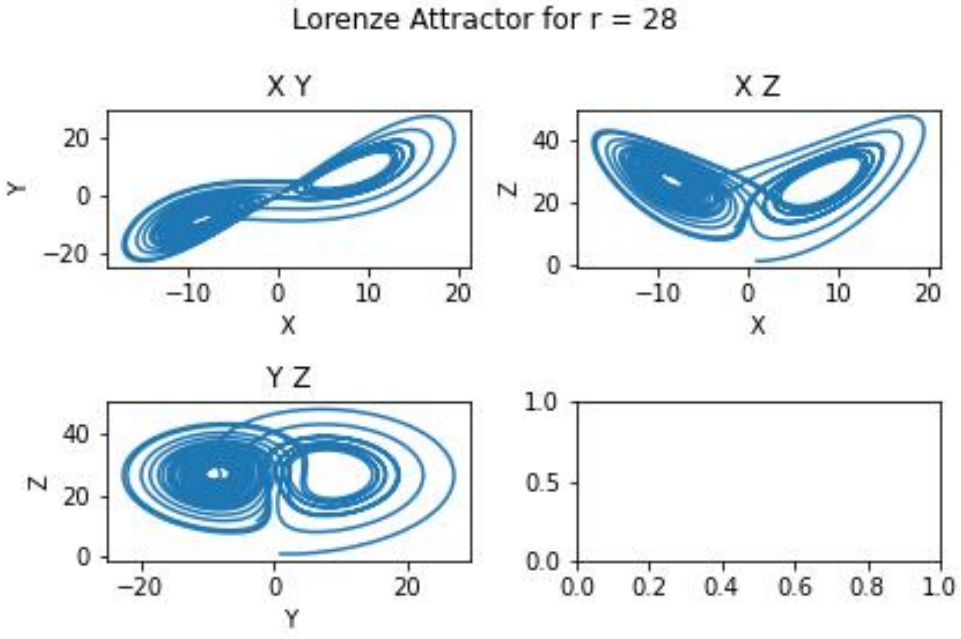

Figure 16: Lorenz attractor at r=28 starting at (1,1,1)

As the value of r was increased beyond its limit, it can be seen that the plot now goes around in space a lot more and seems to be orbiting between two different areas in space. It does seem to still be confined in a specific domain and does not blow up to much higher values.

Number 4 c)

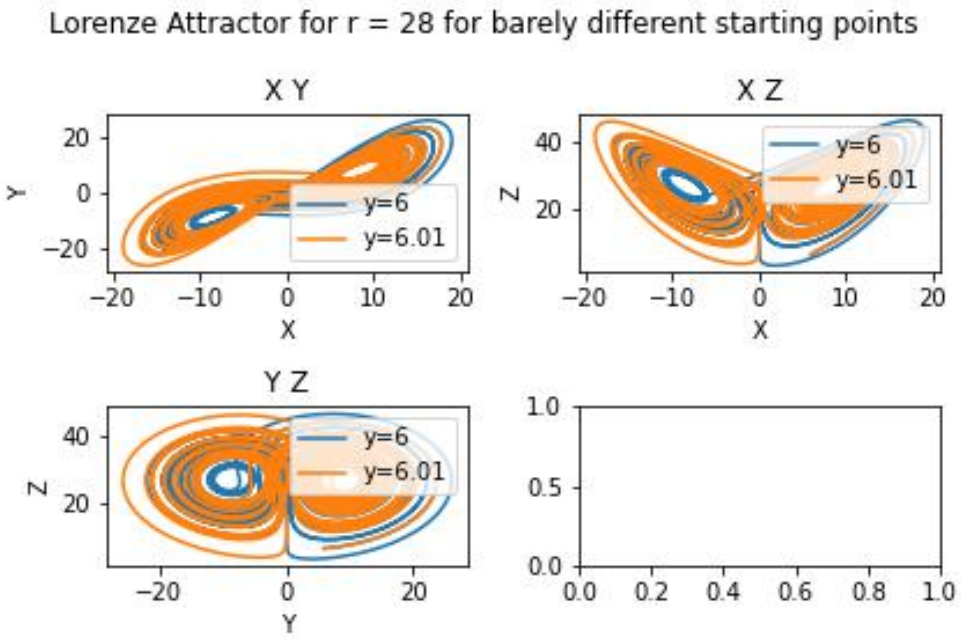

Figure 17: Lorenz attractor at r=28 with two different starting point at (6,6,6) and (6,6.01,6)

By only changing y by 0.01, it can be seen that from the figure above that the plot changed drastically. Both of the figures have a similar trend and movement that is also close to the attractor from the previous section but it is distinct enough from each other that the plots can be seen separately.

Number 5 a)

Using the code used in the appendix below. A following function was made to create the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ and calculate their eigenvalues as shown below. For simplicity only the first 4 eigenvalues by magnitude will be seen in the table below.

N	A	B
11	1.98j, -1.98j, 1.82j, -1.82j	0.08, 0.69, 1.72, 2.83
21	1.99j, -1.99j, 1.95j, -1.95j	4.00, 3.91, 3.91, 3.65
31	2.00j, -2.00j, 1.98j, -1.98j	0.01, 0.09, 0.25, 0.48
41	2.00j, -2.00j, 1.99j, -1.99j	0.01, 0.05, 0.14, 0.28
51	2.00j, -2.00j, 1.98j, -1.98j	0.00, 0.03, 0.09, 0.18

Table 1: Eigenvalues of A and B

It can be seen that from the trend above for the eigenvalues of $\boldsymbol{A}$ and $\boldsymbol{B}$ that as $N \to \infty$, the eigenvalues approach $2j$ and $-2j$ for matrix $\boldsymbol{A}$ and approaches 0 or 4 for matrix $\boldsymbol{B}$

Number 5 b)

Using this information from above and based on the plots of $C_R\Delta t$ vs $C_I\Delta t$ shown in class and as seen in the figure below, matrix $\boldsymbol{A}$ due to all eigenvalues in the $C_I\Delta t$ can take advantage best with the implicit and rk3 onwards. On the other hand matrix $\boldsymbol{B}$ has eigenvalues only on $C_R\Delta t$ and can take advantage of probably all the different schemes.

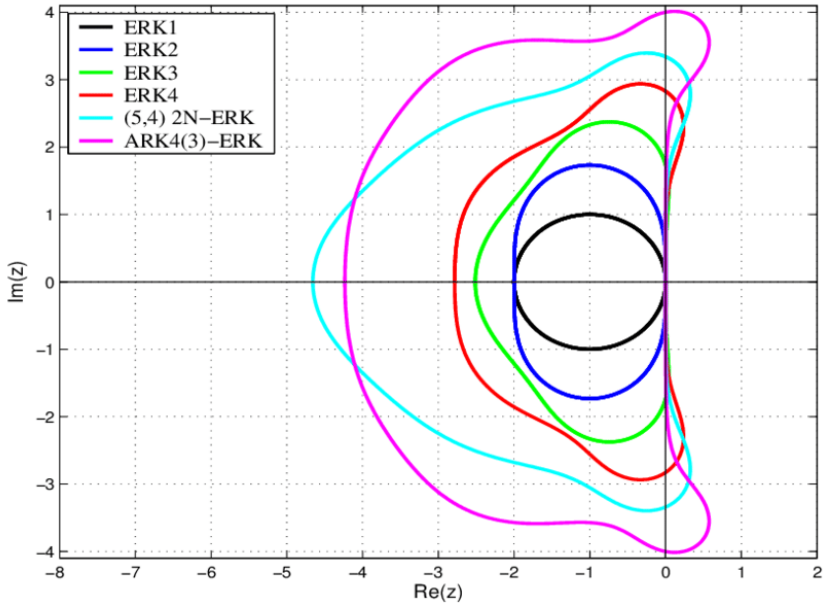

Figure 18: Stability Region [**article**]

Number 5 c)

To pick the time step, $|\sigma \leq 1|$ and the following can be calculated. From the previous problem, the wave equation can't use the explicit euler scheme. For the 1d heat equation.

$$\frac{4\nu\Delta t}{(\Delta x)^2} \leq 2$$

$$\boxed{\Delta t \leq \frac{(\Delta x)^2}{2}}$$

Appendix)

Python Code for 1 a)

```
"""
Fall 2022 AEM 5253
Justine John "JJ" A. Serdoncillo
Homework 1 Number 1
"""
import numpy as np
import matplotlib.pyplot as plt

dt = [0.1,0.5,1]

# g(y(t), t)
def g(y, t):
    return - 2 * y;

# solvers
def anl(ti,tf,yi):
    t = np.linspace(ti,tf,1000)
    y = np.zeros(len(t))
    y[0] = yi
    y = yi * np.exp(-2*t)
    return t, y
def exp(ti,tf,yi,dt):
    N = int((tf-ti)/dt)
    t = np.linspace(ti,tf,N+1)
    y = np.zeros(len(t))
    y[0] = yi
    for i in range(len(t)-1):
        y[i+1] = y[i] + dt * g(y[i],t[i])
    return t, y
def imp(ti,tf,yi,dt):
    N = int((tf-ti)/dt)
    t = np.linspace(ti,tf,N+1)
    y = np.zeros(len(t))
    y[0] = yi
    for i in range(len(t)-1):
        nume = y[i]
        denu = 1 + 2 * dt
        y[i+1] = nume / denu
    return t, y
def rk2(ti,tf,yi,dt):
    N = int((tf-ti)/dt)
    t = np.linspace(ti,tf,N+1)
    y = np.zeros(len(t))
    y[0] = yi
    for i in range(len(t)-1):
        k0 = g(y[i],t[i])
        k1 = g(y[i]+dt/2*k0,t[i]+dt/2)
        y[i+1] = y[i] + dt * k1
    return t, y
def rk4(ti,tf,yi,dt):
    N = int((tf-ti)/dt)
    t = np.linspace(ti,tf,N+1)
    y = np.zeros(len(t))
    y[0] = yi
    for i in range(len(t)-1):
        k0 = g(y[i],t[i])
        k1 = g(y[i]+dt/2*k0,t[i]+dt/2)
        k2 = g(y[i]+dt/2*k1,t[i]+dt/2)
        k3 = g(y[i]+dt  *k2,t[i]+dt  )
        y[i+1] = y[i] + dt/6 * (k0+2*k1+2*k2+k3)
    return t, y

# Analytical
t_anal, y_anal = anl(0,15,4)

# Explicit
fig1, ax1 = plt.subplots()
```

```
ax1.plot(t_anal,y_anal,'-r',label='Analytical Solution')
for i in dt:
    t, y = exp(0,15,4,i)
    label = 'Explicit Euler dt=' + str(i) + 's'
    ax1.plot(t,y,'-o',label=label)
ax1.set_title('Explicit Euler Method with Analytical Solution')
ax1.set_xlabel('t')
ax1.set_ylabel('y')
ax1.set_xlim([0,15])
ax1.legend()
plt.savefig('images/exp1a.jpg')
plt.show()

# Implicit
fig1, ax1 = plt.subplots()
ax1.plot(t_anal,y_anal,'-r',label='Analytical Solution')
for i in dt:
    t, y = imp(0,15,4,i)
    label = 'Implicit Euler dt=' + str(i) + 's'
    ax1.plot(t,y,'-o',label=label)
ax1.set_title('Implicit Euler Method with Analytical Solution')
ax1.set_xlabel('t')
ax1.set_ylabel('y')
ax1.set_xlim([0,15])
#ax1.set_ylim([0, 4])
ax1.legend()
plt.savefig('images/imp1a.jpg')
plt.show()

# RK2
fig1, ax1 = plt.subplots()
ax1.plot(t_anal,y_anal,'-r',label='Analytical Solution')
for i in dt:
    t, y = rk2(0,15,4,i)
    label = 'RK2 dt=' + str(i) + 's'
    ax1.plot(t,y,'-o',label=label)
ax1.set_title('Runge-Kutta 2nd Order with Analytical Solution')
ax1.set_xlabel('t')
ax1.set_ylabel('y')
ax1.set_xlim([0,15])
ax1.legend()
plt.savefig('images/rk21a.jpg')
plt.show()

# RK4
fig1, ax1 = plt.subplots()
ax1.plot(t_anal,y_anal,'-r',label='Analytical Solution')
for i in dt:
    t, y = rk4(0,15,4,i)
    label = 'RK4 dt=' + str(i) + 's'
    ax1.plot(t,y,'-o',label=label)
ax1.set_title('Runge-Kutta 4th Order with Analytical Solution')
ax1.set_xlabel('t')
ax1.set_ylabel('y')
ax1.set_xlim([0,15])
ax1.legend()
plt.savefig('images/rk41a.jpg')
plt.show()
```

Python Code for 2 a)

```
"""
Fall 2022 AEM 5253
Justine John "JJ" A. Serdoncillo
Homework 1 Number 2
"""

import numpy as np
import matplotlib.pyplot as plt

dt = [0.1,0.5,1]

```

```
# solvers
def g(y, t):
    return - (2+0.01*t**2) * y;
def anl(ti,tf,yi):
    t = np.linspace(ti,tf,1000)
    y = np.zeros(len(t))
    y[0] = yi
    y = yi * np.exp(-2*t - 0.01/3*t**3)
    return t, y
def imp(ti,tf,yi,dt):
    N = int((tf-ti)/dt)
    t = np.linspace(ti,tf,N+1)
    y = np.zeros(len(t))
    y[0] = yi
    for i in range(len(t)-1):
        nume = y[i]
        denu = 1 + dt * (2+0.01*t[i]**2)
        y[i+1] = nume / denu
    return t, y
######
def exp(ti,tf,yi,dt):
    N = int((tf-ti)/dt)
    t = np.linspace(ti,tf,N+1)
    y = np.zeros(len(t))
    y[0] = yi
    for i in range(len(t)-1):
        y[i+1] = y[i] + dt * g(y[i],t[i])
    return t, y
def rk2(ti,tf,yi,dt):
    N = int((tf-ti)/dt)
    t = np.linspace(ti,tf,N+1)
    y = np.zeros(len(t))
    y[0] = yi
    for i in range(len(t)-1):
        k0 = g(y[i],t[i])
        k1 = g(y[i]+dt/2*k0,t[i]+dt/2)
        y[i+1] = y[i] + dt * k1
    return t, y
def rk4(ti,tf,yi,dt):
    N = int((tf-ti)/dt)
    t = np.linspace(ti,tf,N+1)
    y = np.zeros(len(t))
    y[0] = yi
    for i in range(len(t)-1):
        k0 = g(y[i],t[i])
        k1 = g(y[i]+dt/2*k0,t[i]+dt/2)
        k2 = g(y[i]+dt/2*k1,t[i]+dt/2)
        k3 = g(y[i]+dt  *k2,t[i]+dt  )
        y[i+1] = y[i] + dt/6 * (k0+2*k1+2*k2+k3)
    return t, y

# Analytical
t_anal, y_anal = anl(0,15,4)

# Explicit
fig, ax = plt.subplots(2,1)
ax[0].plot(t_anal,y_anal,'-r',label='Analytical Solution')
ax[1].plot(t_anal,y_anal,'-r',label='Analytical Solution')
fig.suptitle('Explicit Euler Method with Analytical Solution')
i=dt[0]
t, y = exp(0,15,4,i)
label = 'Explicit Euler dt=' + str(i) + 's'
ax[0].plot(t,y,'-o',label=label)
i=dt[1]
t, y = exp(0,15,4,i)
label = 'Explicit Euler dt=' + str(i) + 's'
ax[0].plot(t,y,'-o',label=label)
i=dt[2]
t, y = exp(0,15,4,i)
label = 'Explicit Euler dt=' + str(i) + 's'
ax[1].plot(t,y,'-o',label=label)
ax[0].set_xlabel('t')
```

```
ax[0].set_ylabel('y')
ax[1].set_xlabel('t')
ax[1].set_ylabel('y')
ax[0].legend()
ax[1].legend()
plt.savefig('images/exp2a.jpg')
plt.show()

# Implicit
fig, ax = plt.subplots()
ax.plot(t_anal,y_anal,'-r',label='Analytical Solution')
for i in dt:
    t, y = imp(0,15,4,i)
    label = 'Implicit Euler dt=' + str(i) + 's'
    ax.plot(t,y,'-o',label=label)
ax.set_title('Implicit Euler Method with Analytical Solution')
ax.set_xlabel('t')
ax.set_ylabel('y')
ax.set_xlim([0,15])
#ax.set_ylim([0, 4])
ax.legend()
plt.savefig('images/imp2a.jpg')
plt.show()

# RK2
fig, ax = plt.subplots(2,1)
ax[0].plot(t_anal,y_anal,'-r',label='Analytical Solution')
ax[1].plot(t_anal,y_anal,'-r',label='Analytical Solution')
fig.suptitle('Runge-Kutta 2nd Order with Analytical Solution')
i=dt[0]
t, y = rk2(0,15,4,i)
label = 'RK2 dt=' + str(i) + 's'
ax[0].plot(t,y,'-o',label=label)
i=dt[1]
t, y = rk2(0,15,4,i)
label = 'RK2 dt=' + str(i) + 's'
ax[0].plot(t,y,'-o',label=label)
i=dt[2]
t, y = rk2(0,15,4,i)
label = 'RK2 dt=' + str(i) + 's'
ax[1].plot(t,y,'-o',label=label)
ax[0].set_xlabel('t')
ax[0].set_ylabel('y')
ax[1].set_xlabel('t')
ax[1].set_ylabel('y')
ax[0].legend()
ax[1].legend()
plt.savefig('images/rk22a.jpg')
plt.show()

# RK4
fig, ax = plt.subplots()
ax.plot(t_anal,y_anal,'-r',label='Analytical Solution')
for i in dt:
    t, y = rk4(0,15,4,i)
    label = 'RK4 dt=' + str(i) + 's'
    ax.plot(t,y,'-o',label=label)
ax.set_title('Runge-Kutta 4th Order with Analytical Solution')
ax.set_xlabel('t')
ax.set_ylabel('y')
ax.set_xlim([0,15])
ax.legend()
plt.savefig('images/rk42a.jpg')
plt.show()
```

Python Code for 3 a)

```
"""
Fall 2022 AEM 5253
Justine John "JJ" A. Serdoncillo
Homework 1 Number 3 Part B
"""
```

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

dt = [0.15, 0.5, 1]
grav = 9.81
Leng = 0.6

# g(y(t), t)
def g1(theta, dtheta, t):
    return dtheta;
def g2(theta, dtheta, t):
    return -grav/Leng * theta;

# solvers
def anl(ti,tf,yi,vi):
    t = np.linspace(ti,tf,1000)
    y = np.zeros(len(t))
    y[0] = yi
    for i in range(len(t)):
        y[i] = yi * math.cos(t[i] * math.sqrt(grav/Leng))
    return t, y
def exp(ti,tf,yi,vi,dt):
    N = int((tf-ti)/dt)
    t = np.linspace(ti,tf,N+1)
    y = np.zeros(len(t))
    v = np.zeros(len(t))
    y[0] = yi
    v[0] = vi
    for i in range(len(t)-1):
        y[i+1] = y[i] + dt * g1(y[i], v[i], t[i]);
        v[i+1] = v[i] + dt * g2(y[i], v[i], t[i]);
    return t, y
def imp(ti,tf,yi,vi,dt):
    N = int((tf-ti)/dt)
    t = np.linspace(ti,tf,N+1)
    y = np.zeros(len(t))
    v = np.zeros(len(t))
    y[0] = yi
    v[0] = vi
    for i in range(len(t)-1):
        # version 1; based on coupled equations
        nume = y[i] + v[i] * dt
        denu = 1 + grav/Leng * dt**2
        y[i+1] = nume / denu
        v[i+1] = (y[i+1] - y[i]) / dt;
    return t, y
def rk4(ti,tf,yi,vi,dt):
    N = int((tf-ti)/dt)
    t = np.linspace(ti,tf,N+1)
    y = np.zeros(len(t))
    v = np.zeros(len(t))
    y[0] = yi
    v[0] = vi
    for i in range(len(t)-1):
        k01 = g1(y[i], v[i], t[i])
        k02 = g2(y[i], v[i], t[i])
        k11 = g1(y[i]+dt/2*k01, v[i]+dt/2*k02, t[i]+dt/2)
        k12 = g2(y[i]+dt/2*k01, v[i]+dt/2*k02, t[i]+dt/2)
        k21 = g1(y[i]+dt/2*k11, v[i]+dt/2*k12, t[i]+dt/2)
        k22 = g2(y[i]+dt/2*k11, v[i]+dt/2*k12, t[i]+dt/2)
        k31 = g1(y[i]+dt*k21, v[i]+dt*k22, t[i]+dt)
        k32 = g2(y[i]+dt*k21, v[i]+dt*k22, t[i]+dt)
        y[i+1] = y[i] + dt/6 * (k01+2*k11+2*k21+k31)
        v[i+1] = v[i] + dt/6 * (k02+2*k12+2*k22+k32)
    return t, y

# Analytical
t_anal, y_anal = anl(0,6,math.radians(10),0)

# Explicit
fig, ax = plt.subplots(2,2)
```

```
fig.suptitle('Explicit Euler Method with Analytical Solution')
ax[0,0].plot(t_anal,y_anal,'-r')
ax[0,1].plot(t_anal,y_anal,'-r')
ax[1,0].plot(t_anal,y_anal,'-r')
ax[0,0].set_title("dt = 0.15s")
ax[0,1].set_title("dt = 0.5s")
ax[1,0].set_title("dt = 1s")
t, y = exp(0,6,math.radians(10),0,0.15)
ax[0,0].plot(t,y,'-o')
t, y = exp(0,6,math.radians(10),0,0.5)
ax[0,1].plot(t,y,'-o')
t, y = exp(0,6,math.radians(10),0,1)
ax[1,0].plot(t,y,'-o')

ax[0,0].set_xlabel('t')
ax[0,0].set_ylabel('theta')
ax[0,1].set_xlabel('t')
ax[0,1].set_ylabel('theta')
ax[1,0].set_xlabel('t')
ax[1,0].set_ylabel('theta')

fig.tight_layout()
plt.savefig('images/exp3a.jpg')
plt.show()

# Implicit
fig, ax = plt.subplots(2,2)
fig.suptitle('Implicit Euler Method with Analytical Solution')
ax[0,0].plot(t_anal,y_anal,'-r')
ax[0,1].plot(t_anal,y_anal,'-r')
ax[1,0].plot(t_anal,y_anal,'-r')
ax[0,0].set_title("dt = 0.15s")
ax[0,1].set_title("dt = 0.5s")
ax[1,0].set_title("dt = 1s")
t, y = imp(0,6,math.radians(10),0,0.15)
ax[0,0].plot(t,y,'-o')
t, y = imp(0,6,math.radians(10),0,0.5)
ax[0,1].plot(t,y,'-o')
t, y = imp(0,6,math.radians(10),0,1)
ax[1,0].plot(t,y,'-o')

ax[0,0].set_xlabel('t')
ax[0,0].set_ylabel('theta')
ax[0,1].set_xlabel('t')
ax[0,1].set_ylabel('theta')
ax[1,0].set_xlabel('t')
ax[1,0].set_ylabel('theta')

fig.tight_layout()
plt.savefig('images/imp3a.jpg')
plt.show()

# Runge-Kutta 4
fig, ax = plt.subplots(2,2)
fig.suptitle('Runge-Kutta 4th Order with Analytical Solution')
ax[0,0].plot(t_anal,y_anal,'-r')
ax[0,1].plot(t_anal,y_anal,'-r')
ax[1,0].plot(t_anal,y_anal,'-r')
ax[0,0].set_title("dt = 0.15s")
ax[0,1].set_title("dt = 0.5s")
ax[1,0].set_title("dt = 1s")
t, y = rk4(0,6,math.radians(10),0,0.15)
ax[0,0].plot(t,y,'-o')
t, y = rk4(0,6,math.radians(10),0,0.5)
ax[0,1].plot(t,y,'-o')
t, y = rk4(0,6,math.radians(10),0,1)
ax[1,0].plot(t,y,'-o')

ax[0,0].set_xlabel('t')
ax[0,0].set_ylabel('theta')
ax[0,1].set_xlabel('t')
ax[0,1].set_ylabel('theta')
```

```
ax[1,0].set_xlabel('t')
ax[1,0].set_ylabel('theta')

fig.tight_layout()
plt.savefig('images/rk43a.jpg')
plt.show()
```

Python Code for 3 b)

```
"""
Fall 2022 AEM 5253
Justine John "JJ" A. Serdoncillo
Homework 1 Number 3 Part B
"""
import numpy as np
import matplotlib.pyplot as plt
import math

dt = [0.15, 0.5, 1]
grav = 9.81
Leng = 0.6
c = 4

# g(y(t), t)
def g1(theta, dtheta, t):
    return dtheta;
def g2(theta, dtheta, t):
    return -grav/Leng * theta - c * dtheta;

# solvers
def anl(ti,tf,yi,vi):
    t = np.linspace(ti,tf,1000)
    y = np.zeros(len(t))
    y[0] = yi
    alph = math.sqrt(4*grav/Leng - c**2)/2
    beta = c/math.sqrt(4*grav/Leng - c**2)
    for i in range(len(t)):
        y[i] = yi * math.exp(-c*t[i]/2) * (math.cos(alph*t[i]) + beta * math.sin(alph*t[i]))
    return t, y
def exp(ti,tf,yi,vi,dt):
    N = int((tf-ti)/dt)
    t = np.linspace(ti,tf,N+1)
    y = np.zeros(len(t))
    v = np.zeros(len(t))
    y[0] = yi
    v[0] = vi
    for i in range(len(t)-1):
        y[i+1] = y[i] + dt * g1(y[i], v[i], t[i]);
        v[i+1] = v[i] + dt * g2(y[i], v[i], t[i]);
    return t, y
def imp(ti,tf,yi,vi,dt):
    N = int((tf-ti)/dt)
    t = np.linspace(ti,tf,N+1)
    y = np.zeros(len(t))
    v = np.zeros(len(t))
    y[0] = yi
    v[0] = vi
    for i in range(len(t)-1):
        nume = y[i] * (1 + c * dt) + v[i] * dt
        denu = 1 + (1 + c * dt) + grav/Leng * dt**2
        y[i+1] = nume / denu
        v[i+1] = (y[i+1] - y[i]) / dt;
    return t, y
def rk4(ti,tf,yi,vi,dt):
    N = int((tf-ti)/dt)
    t = np.linspace(ti,tf,N+1)
    y = np.zeros(len(t))
    v = np.zeros(len(t))
    y[0] = yi
    v[0] = vi
    for i in range(len(t)-1):
        k01 = g1(y[i], v[i], t[i])
```

```
        k02 = g2(y[i], v[i], t[i])
        k11 = g1(y[i]+dt/2*k01, v[i]+dt/2*k02, t[i]+dt/2)
        k12 = g2(y[i]+dt/2*k01, v[i]+dt/2*k02, t[i]+dt/2)
        k21 = g1(y[i]+dt/2*k11, v[i]+dt/2*k12, t[i]+dt/2)
        k22 = g2(y[i]+dt/2*k11, v[i]+dt/2*k12, t[i]+dt/2)
        k31 = g1(y[i]+dt*k21, v[i]+dt*k22, t[i]+dt)
        k32 = g2(y[i]+dt*k21, v[i]+dt*k22, t[i]+dt)
        y[i+1] = y[i] + dt/6 * (k01+2*k11+2*k21+k31)
        v[i+1] = v[i] + dt/6 * (k02+2*k12+2*k22+k32)
    return t, y

# Analytical
t_anal, y_anal = anl(0,6,math.radians(10),0)

# Explicit
fig, ax = plt.subplots(2,2)
fig.suptitle('Explicit Euler Method with Analytical Solution')
ax[0,0].plot(t_anal,y_anal,'-r')
ax[0,1].plot(t_anal,y_anal,'-r')
ax[1,0].plot(t_anal,y_anal,'-r')
ax[0,0].set_title("dt = 0.15s")
ax[0,1].set_title("dt = 0.5s")
ax[1,0].set_title("dt = 1s")
t, y = exp(0,6,math.radians(10),0,0.15)
ax[0,0].plot(t,y,'-o')
t, y = exp(0,6,math.radians(10),0,0.5)
ax[0,1].plot(t,y,'-o')
t, y = exp(0,6,math.radians(10),0,1)
ax[1,0].plot(t,y,'-o')

ax[0,0].set_xlabel('t')
ax[0,0].set_ylabel('theta')
ax[0,1].set_xlabel('t')
ax[0,1].set_ylabel('theta')
ax[1,0].set_xlabel('t')
ax[1,0].set_ylabel('theta')

fig.tight_layout()
plt.savefig('images/exp3b.jpg')
plt.show()

# Implicit
fig, ax = plt.subplots(2,2)
fig.suptitle('Implicit Euler Method with Analytical Solution')
ax[0,0].plot(t_anal,y_anal,'-r')
ax[0,1].plot(t_anal,y_anal,'-r')
ax[1,0].plot(t_anal,y_anal,'-r')
ax[0,0].set_title("dt = 0.15s")
ax[0,1].set_title("dt = 0.5s")
ax[1,0].set_title("dt = 1s")
t, y = imp(0,6,math.radians(10),0,0.15)
ax[0,0].plot(t,y,'-o')
t, y = imp(0,6,math.radians(10),0,0.5)
ax[0,1].plot(t,y,'-o')
t, y = imp(0,6,math.radians(10),0,1)
ax[1,0].plot(t,y,'-o')

ax[0,0].set_xlabel('t')
ax[0,0].set_ylabel('theta')
ax[0,1].set_xlabel('t')
ax[0,1].set_ylabel('theta')
ax[1,0].set_xlabel('t')
ax[1,0].set_ylabel('theta')

fig.tight_layout()
plt.savefig('images/imp3b.jpg')
plt.show()

# Runge-Kutta 4
fig, ax = plt.subplots(2,2)
fig.suptitle('Runge-Kutta 4th Order with Analytical Solution')
ax[0,0].plot(t_anal,y_anal,'-r')
```

```
ax[0,1].plot(t_anal,y_anal,'-r')
ax[1,0].plot(t_anal,y_anal,'-r')
ax[0,0].set_title("dt = 0.15s")
ax[0,1].set_title("dt = 0.5s")
ax[1,0].set_title("dt = 1s")
t, y = rk4(0,6,math.radians(10),0,0.15)
ax[0,0].plot(t,y,'-o')
t, y = rk4(0,6,math.radians(10),0,0.5)
ax[0,1].plot(t,y,'-o')
t, y = rk4(0,6,math.radians(10),0,1)
ax[1,0].plot(t,y,'-o')

ax[0,0].set_xlabel('t')
ax[0,0].set_ylabel('theta')
ax[0,1].set_xlabel('t')
ax[0,1].set_ylabel('theta')
ax[1,0].set_xlabel('t')
ax[1,0].set_ylabel('theta')

fig.tight_layout()
plt.savefig('images/rk43b.jpg')
plt.show()
```

Python Code for 4

```
"""
Fall 2022 AEM 5253
Justine John "JJ" A. Serdoncillo
Homework 1 Number 4
"""
import numpy as np
import matplotlib.pyplot as plt
import math

### Number 1 ###
sigm = 10
b = 8/3.

def gx(x, y, z, r):
    return sigm*(y-x);
def gy(x, y, z, r):
    return r*x - y  - x*z;
def gz(x, y, z, r):
    return x*y - b*z;

# solvers
def rk4(ti,tf,xi,yi,zi,r,dt):
    N = int((tf-ti)/dt)
    t = np.linspace(ti,tf,N+1)
    x = np.zeros(len(t))
    y = np.zeros(len(t))
    z = np.zeros(len(t))
    x[0] = xi
    y[0] = yi
    z[0] = zi
    for i in range(len(t)-1):
         k0x = gx(x[i], y[i], z[i], r)
         k0y = gy(x[i], y[i], z[i], r)
         k0z = gz(x[i], y[i], z[i], r)
         k1x = gx(x[i]+dt/2*k0x, y[i]+dt/2*k0y, z[i]+dt/2*k0z, r)
         k1y = gy(x[i]+dt/2*k0x, y[i]+dt/2*k0y, z[i]+dt/2*k0z, r)
         k1z = gz(x[i]+dt/2*k0x, y[i]+dt/2*k0y, z[i]+dt/2*k0z, r)
         k2x = gx(x[i]+dt/2*k1x, y[i]+dt/2*k1y, z[i]+dt/2*k1z, r)
         k2y = gy(x[i]+dt/2*k1x, y[i]+dt/2*k1y, z[i]+dt/2*k1z, r)
         k2z = gz(x[i]+dt/2*k1x, y[i]+dt/2*k1y, z[i]+dt/2*k1z, r)
         k3x = gx(x[i]+dt*k2x, y[i]+dt*k2y, z[i]+dt*k2z, r)
         k3y = gy(x[i]+dt*k2x, y[i]+dt*k2y, z[i]+dt*k2z, r)
         k3z = gz(x[i]+dt*k2x, y[i]+dt*k2y, z[i]+dt*k2z, r)
         x[i+1] = x[i] + dt/6 * (k0x+2*k1x+2*k2x+k3x)
         y[i+1] = y[i] + dt/6 * (k0y+2*k1y+2*k2y+k3y)
         z[i+1] = z[i] + dt/6 * (k0z+2*k1z+2*k2z+k3z)
    return t, x, y, z
```

```
48
49 # 4 a
50 fig, ax = plt.subplots(2,2)
51 fig.suptitle('Lorenze Attractor for r = 20')
52 ax[0,0].set_title("X Y")
53 ax[0,1].set_title("X Z")
54 ax[1,0].set_title("Y Z")
55 t, x, y, z = rk4(0,25,1,1,1,20,0.005)
56 ax[0,0].plot(x,y,'-')
57 ax[0,1].plot(x,z,'-')
58 ax[1,0].plot(y,z,'-')
59
60 ax[0,0].set_xlabel('X')
61 ax[0,0].set_ylabel('Y')
62 ax[0,1].set_xlabel('X')
63 ax[0,1].set_ylabel('Z')
64 ax[1,0].set_xlabel('Y')
65 ax[1,0].set_ylabel('Z')
66
67 fig.tight_layout()
68 plt.savefig('images/rk44a.jpg')
69 plt.show()
70
71 # 4 b
72 fig, ax = plt.subplots(2,2)
73 fig.suptitle('Lorenze Attractor for r = 28')
74 ax[0,0].set_title("X Y")
75 ax[0,1].set_title("X Z")
76 ax[1,0].set_title("Y Z")
77 t, x, y, z = rk4(0,25,1,1,1,28,0.005)
78 ax[0,0].plot(x,y,'-')
79 ax[0,1].plot(x,z,'-')
80 ax[1,0].plot(y,z,'-')
81
82 ax[0,0].set_xlabel('X')
83 ax[0,0].set_ylabel('Y')
84 ax[0,1].set_xlabel('X')
85 ax[0,1].set_ylabel('Z')
86 ax[1,0].set_xlabel('Y')
87 ax[1,0].set_ylabel('Z')
88
89 fig.tight_layout()
90 plt.savefig('images/rk44b.jpg')
91 plt.show()
92
93 # 4 c
94 fig, ax = plt.subplots(2,2)
95 fig.suptitle('Lorenze Attractor for r = 28 for barely different starting points')
96 ax[0,0].set_title("X Y")
97 ax[0,1].set_title("X Z")
98 ax[1,0].set_title("Y Z")
99 t, x, y, z = rk4(0,25,6,6,6,28,0.005)
100 ax[0,0].plot(x,y,'-', label='y=6')
101 ax[0,1].plot(x,z,'-', label='y=6')
102 ax[1,0].plot(y,z,'-', label='y=6')
103 t, x, y, z = rk4(0,25,6,6.01,6,28,0.005)
104 ax[0,0].plot(x,y,'-', label='y=6.01')
105 ax[0,1].plot(x,z,'-', label='y=6.01')
106 ax[1,0].plot(y,z,'-', label='y=6.01')
107
108 ax[0,0].set_xlabel('X')
109 ax[0,0].set_ylabel('Y')
110 ax[0,1].set_xlabel('X')
111 ax[0,1].set_ylabel('Z')
112 ax[1,0].set_xlabel('Y')
113 ax[1,0].set_ylabel('Z')
114
115 ax[0,0].legend()
116 ax[0,1].legend()
117 ax[1,0].legend()
118 fig.tight_layout()
119 plt.savefig('images/rk44c.jpg')
```

```
plt.show()
```

Python Code for 5

```
"""
Fall 2022 AEM 5253
Justine John "JJ" A. Serdoncillo
Homework 1 Number 5
"""
import numpy as np
from numpy import linalg as LA

Num = np.linspace(11,51,5)

def Agen(N):
    A = np.zeros((N,N))
    A[0, -1] = -1
    A[-1, 0] =  1
    for i in range(N):
        A[i-1,i  ] =  1
        A[i  ,i-1] = -1
    return A
def Bgen(N):
    B = np.zeros((N,N))
    B[0, -1] =  1
    B[-1, 0] =  1
    for i in range(N):
        B[i-1,i  ] =  1
        B[i  ,i-1] =  1
        B[i-1,i-1] =  2
    return B

for i in range(len(Num)):
    print('for N = ', str(Num[i]))
    Aeig = LA.eigvals(Agen(int(Num[i])))
    for j in range(4):
        AeigYeah = round(Aeig[j].real,2) + round(Aeig[j].imag,2) * 1j
        print(AeigYeah)
for i in range(len(Num)):
    print('for N = ', str(Num[i]))
    Beig = LA.eigvals(Bgen(int(Num[i])))
    Beig.argsort()[::-1]
    for j in range(4):
        BeigYeah = round(Beig[j].real,2) + round(Beig[j].imag,2) * 1j
        print(BeigYeah)
```