

## Homework 1

### Number 1 a)

Calculating the analytical solution we find:

$$\begin{aligned}\frac{dy}{dt} &= -2y \\ \int \frac{dy}{y} &= \int -2dt \\ [\ln|y|]_{y_0}^y &= -2t \\ \ln\left|\frac{y}{y_0}\right| &= -2t \\ \frac{y}{y_0} &= e^{-2t}\end{aligned}$$

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

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

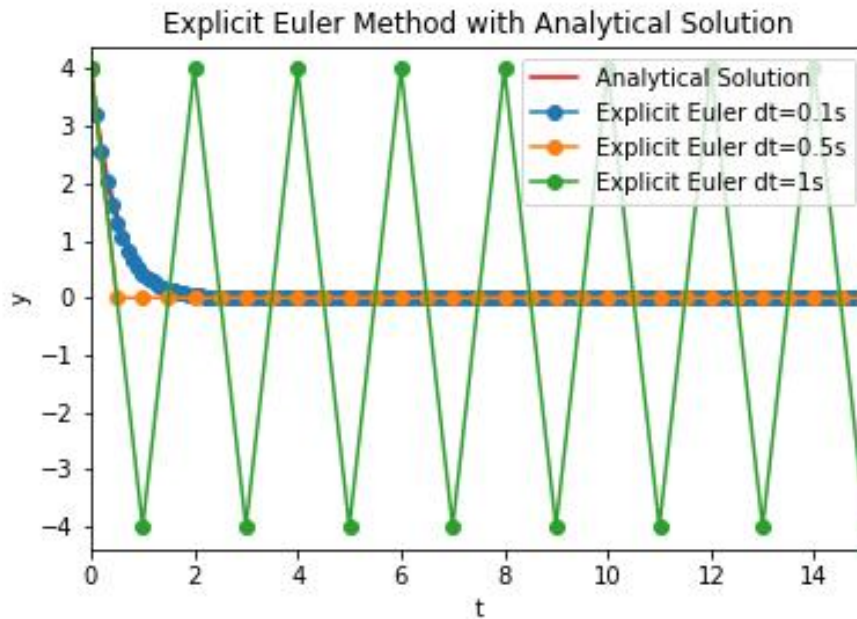


Figure 1: Explicit Euler at different time steps

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 instability. For the medium time step, it immediately jumped to 0 and became its 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.

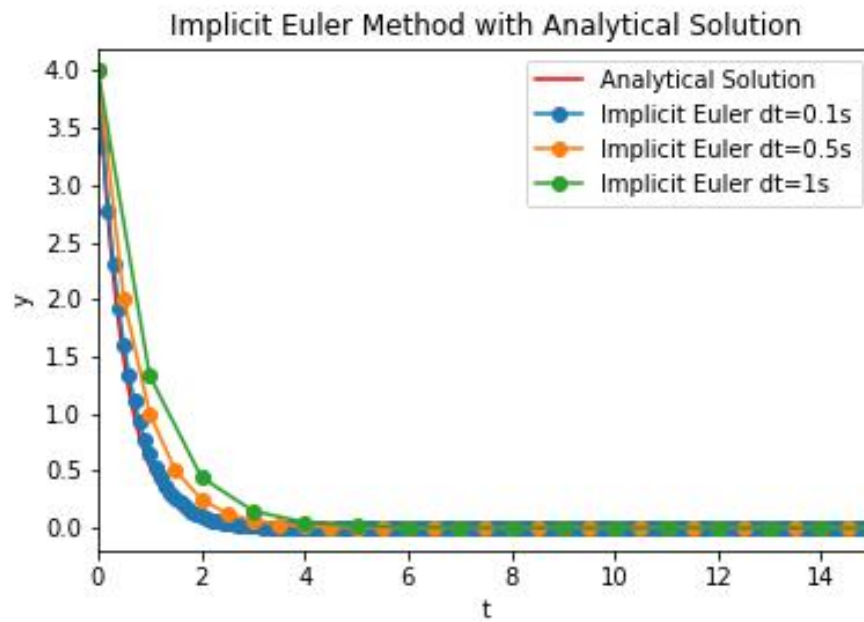


Figure 2: Implicit Euler at different time steps

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.

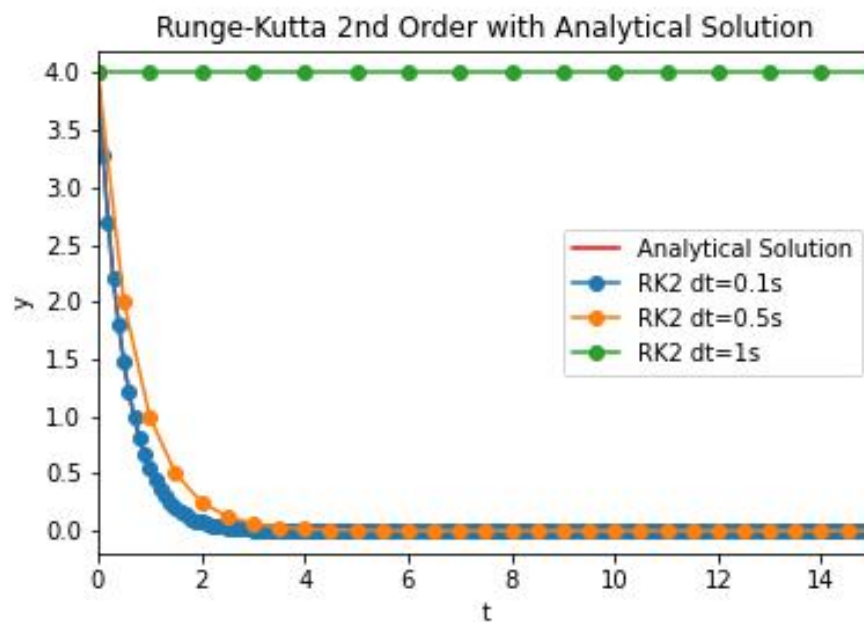


Figure 3: Runge-Kutta 2<sup>nd</sup> order at different time steps

For the 2<sup>nd</sup> 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.

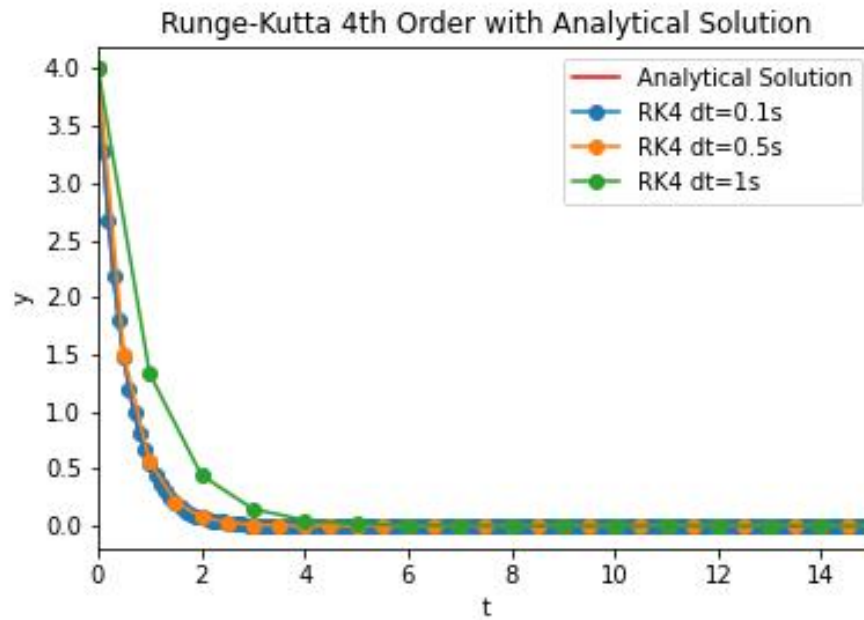


Figure 4: Runge-Kutta 4<sup>th</sup> order at different time steps

The 4<sup>th</sup> 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  $\Delta t = \frac{1}{2}$

(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 2<sup>nd</sup> 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 4<sup>th</sup> 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  $\Delta t$  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  $c = -(2 + 0.01t^2)$ . 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  $t \rightarrow \infty$ ,  $\frac{1}{1 + 0.005t^2}$  decreases. To get the maximum stable time step, we minimize the value which occurs at  $t = 0$

Estimated maximum stable time step at  $\Delta t = 1$

(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

$$0 \leq \Delta t$$

no critical time step

(c) Runge-Kutta 2<sup>nd</sup> 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  $\Delta t$  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  $\Delta t = 1$

(d) Runge-Kutta 4<sup>th</sup> 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  $\Delta t = 0.74$

## Number 2 b)

Calculating the analytical solution we find:

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

$$y = y_0 e^{-(2t + \frac{0.01}{3}t^3)} = 4e^{-(2t + \frac{0.01}{3}t^3)}$$

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.

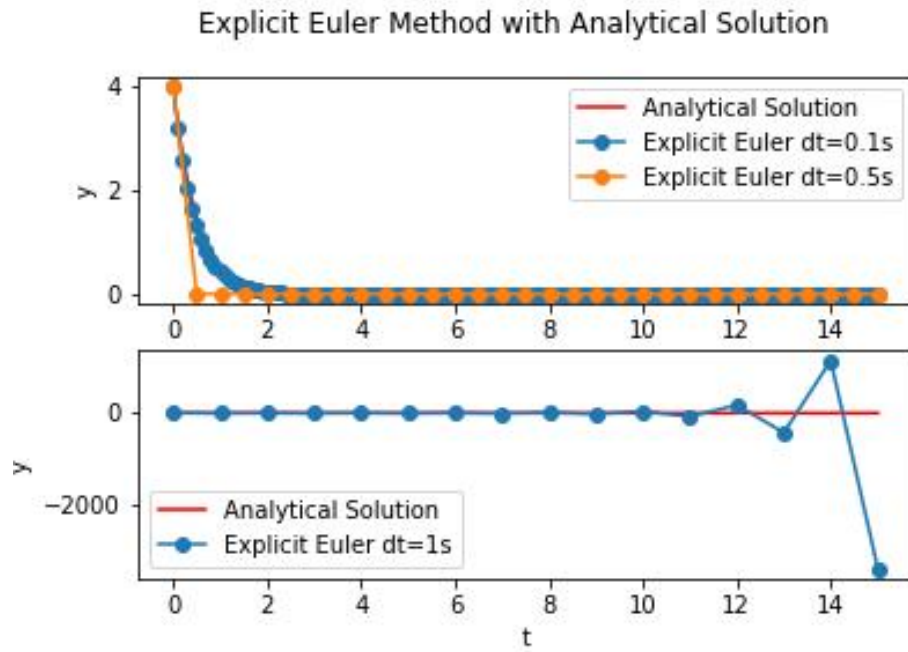


Figure 5: Explicit Euler at different time steps

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.

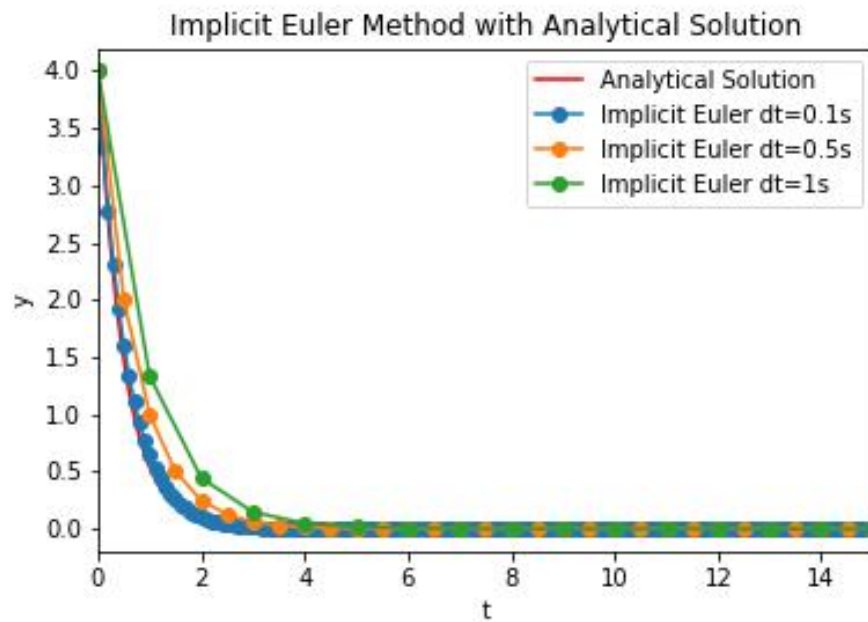


Figure 6: Implicit Euler at different time steps

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

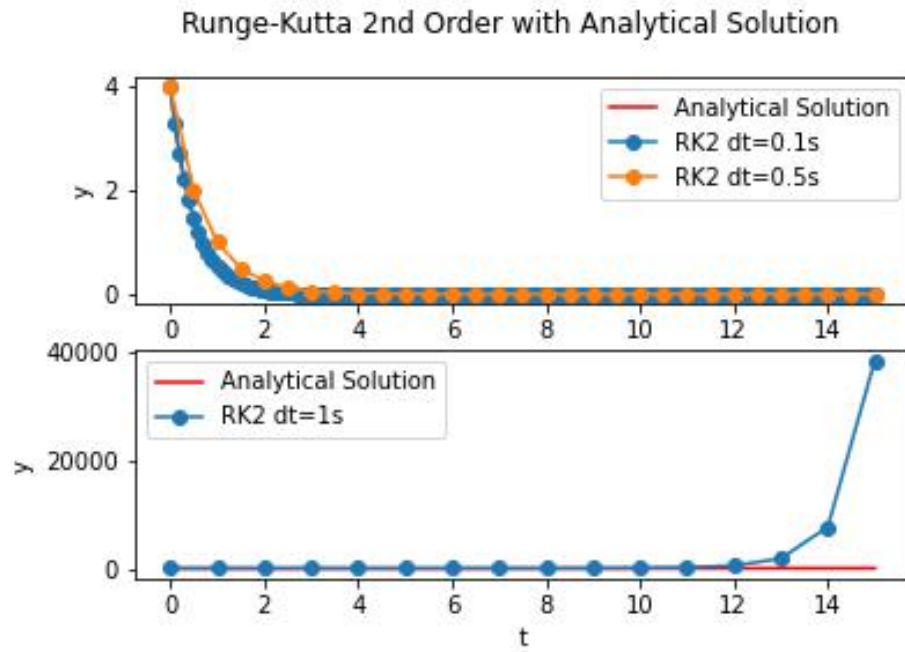


Figure 7: Runge-Kutta 2<sup>nd</sup> order at different time steps

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

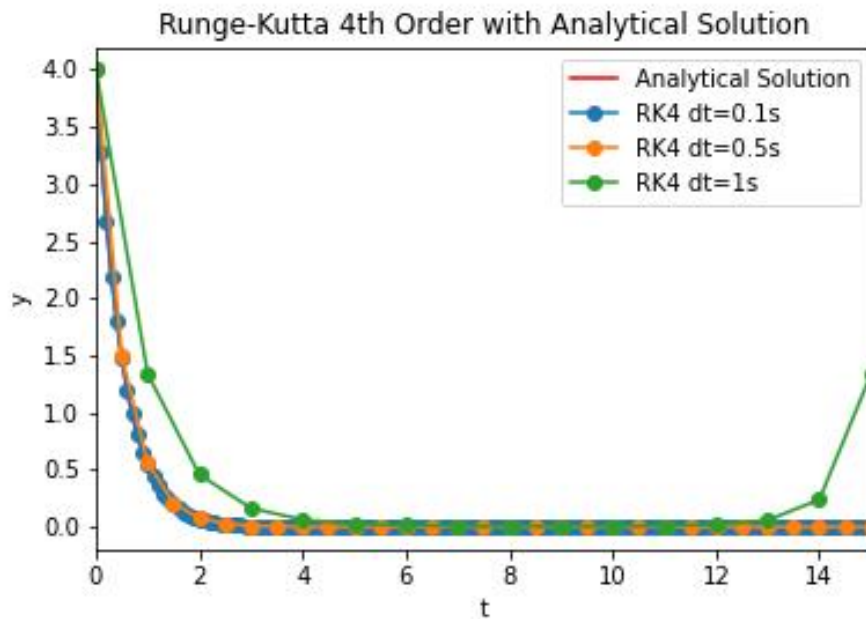


Figure 8: Runge-Kutta 4<sup>th</sup> order at different time steps

For the Runge-Kutta 4<sup>th</sup> 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.

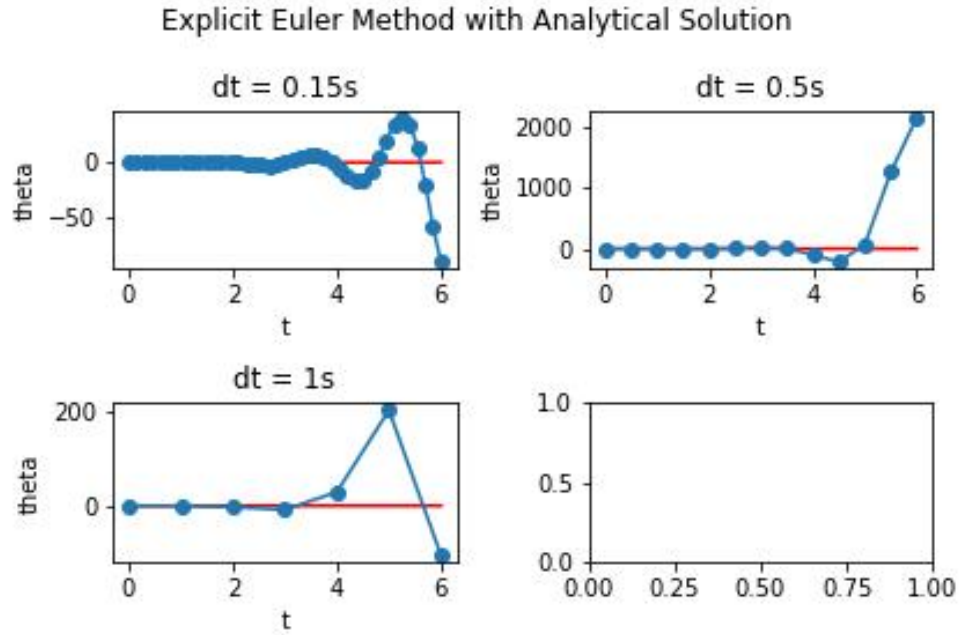


Figure 9: Explicit Euler at different time steps

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.

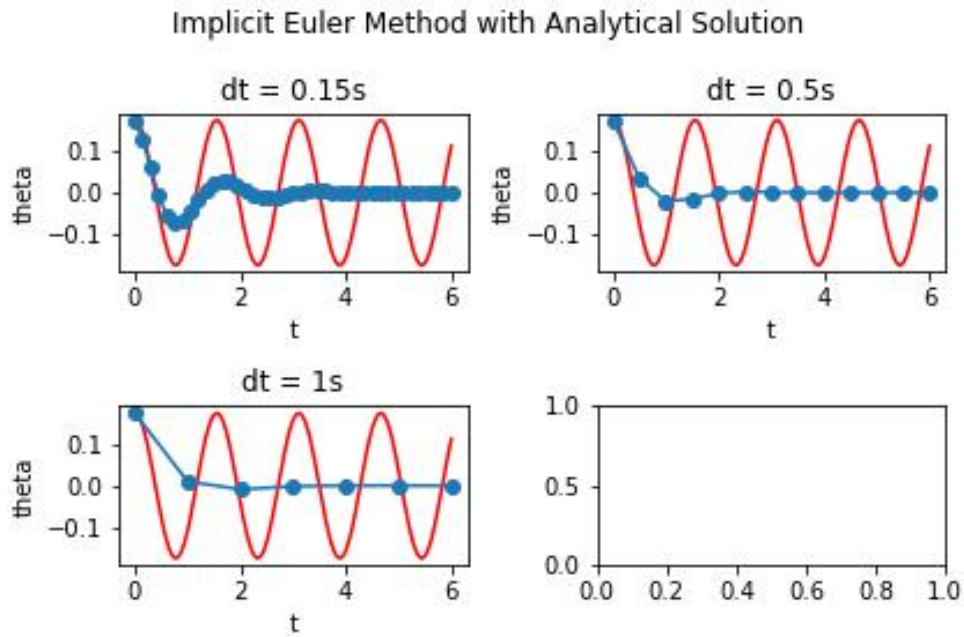


Figure 10: Implicit Euler at different time steps

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.

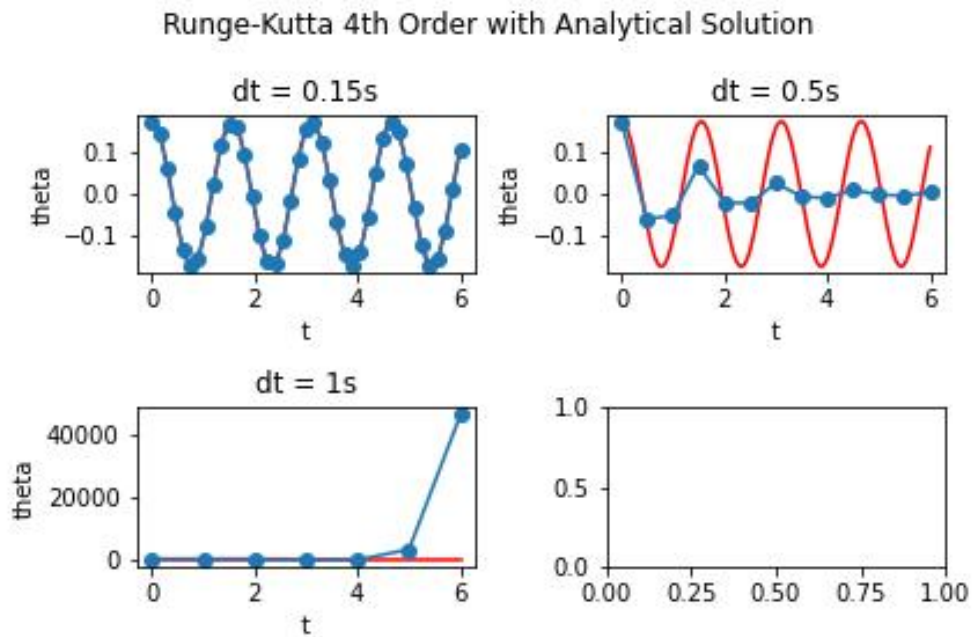


Figure 11: Runge-Kutta 4<sup>th</sup> order at different time steps

Lastly for the Runge-Kutta 4<sup>th</sup> 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.

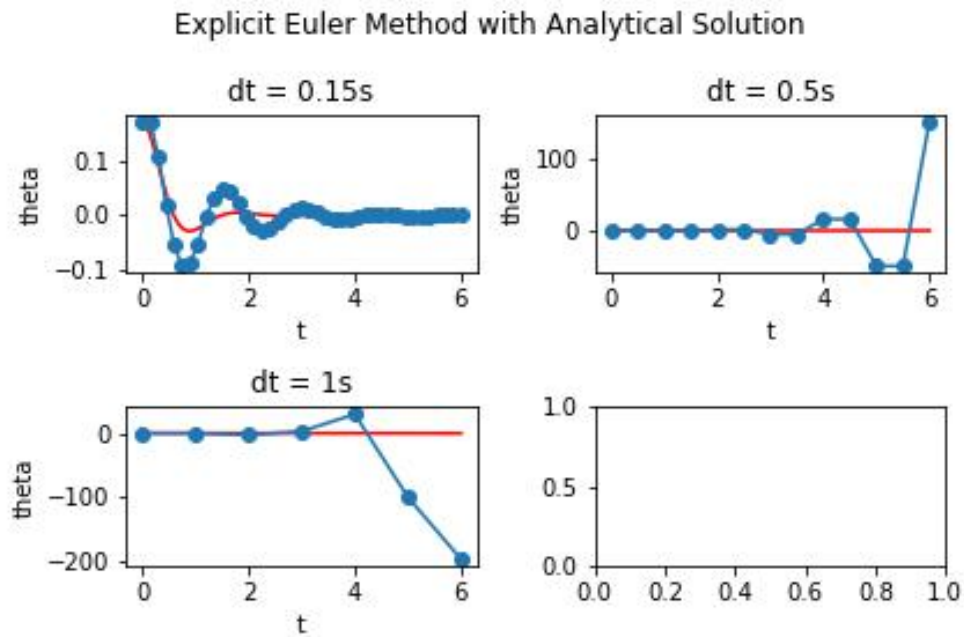


Figure 12: Explicit Euler at different time steps

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.

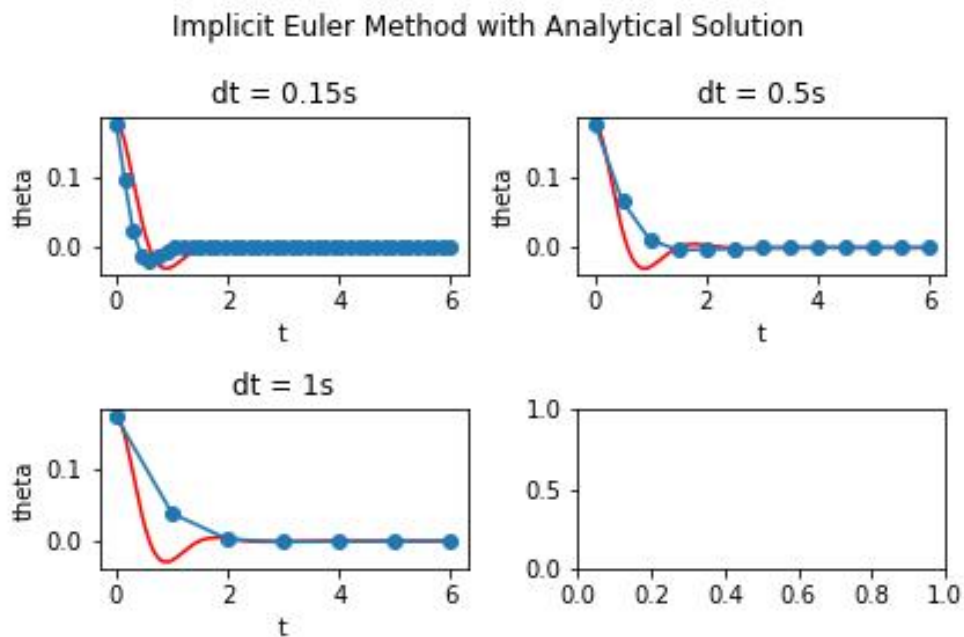


Figure 13: Implicit Euler at different time steps

For the implicit euler method, all of the time steps showed to be stable and even approached the right steady state solution. However its 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.

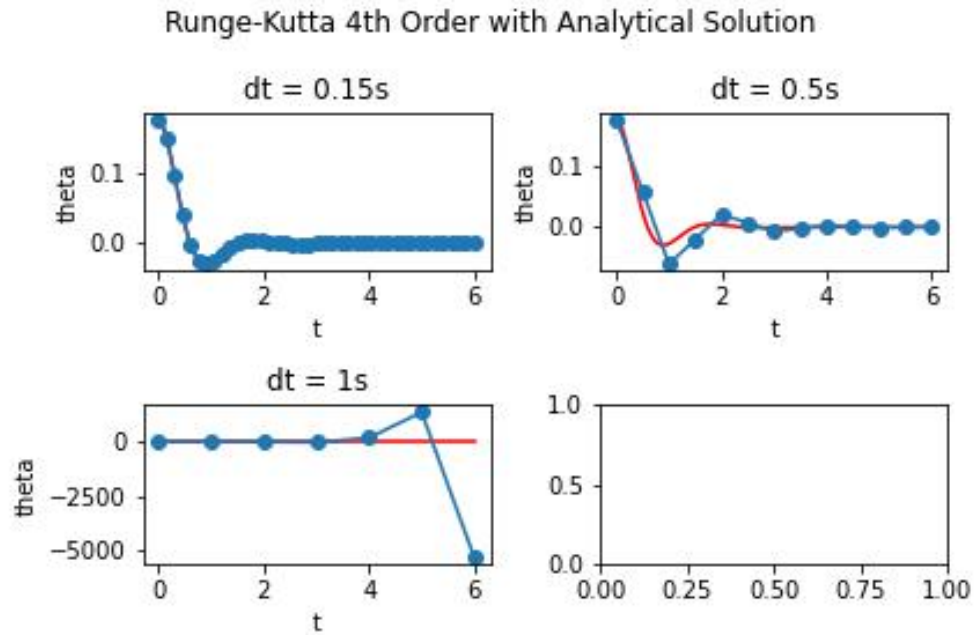


Figure 14: Runge-Kutta 4<sup>th</sup> order at different time steps

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)

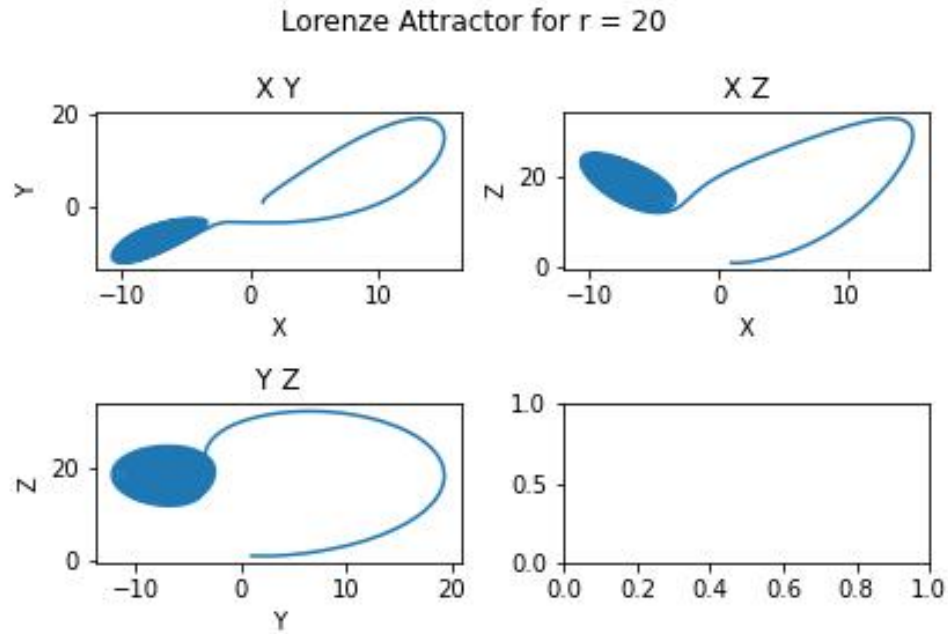


Figure 15: Lorenz attractor at  $r=20$  starting at  $(1,1,1)$

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)

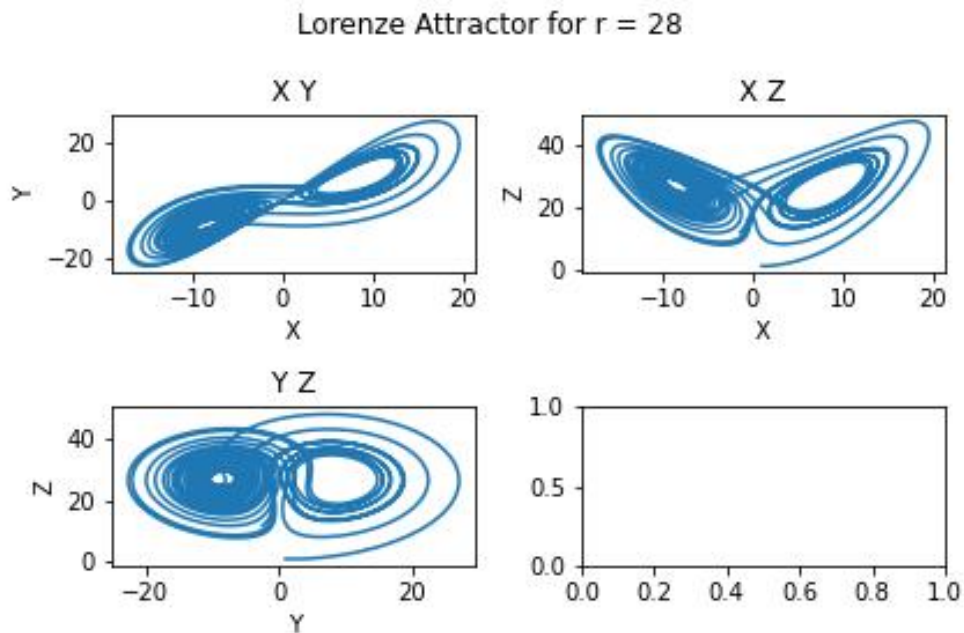


Figure 16: Lorenz attractor at  $r=28$  starting at  $(1,1,1)$

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)

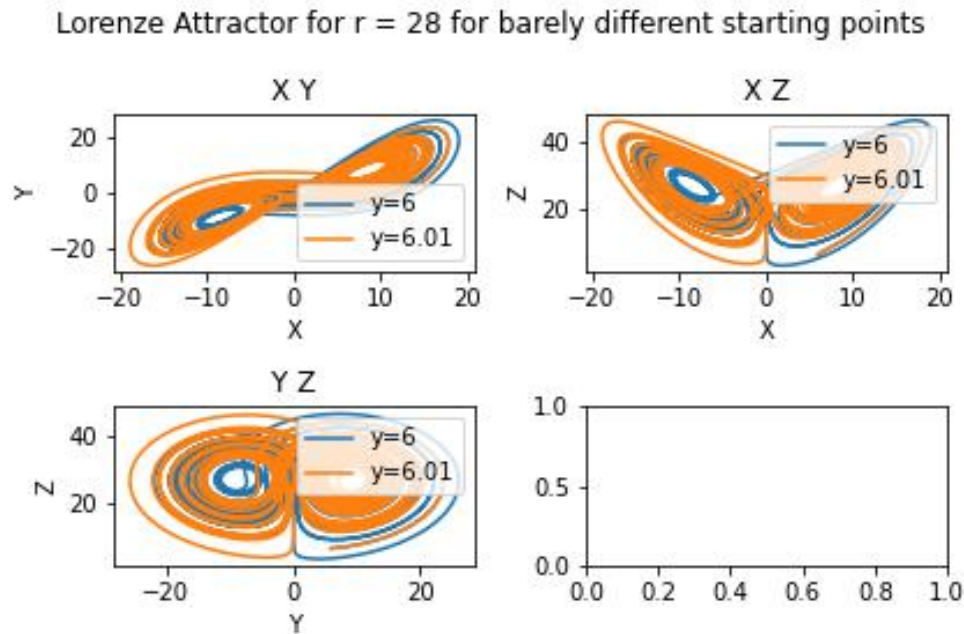


Figure 17: Lorenz attractor at  $r=28$  with two different starting point at  $(6,6,6)$  and  $(6,6.01,6)$

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  $A$  and  $B$  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  $A$  and  $B$  that as  $N \rightarrow \infty$ , the eigenvalues approach  $2j$  and  $-2j$  for matrix  $A$  and approaches 0 or 4 for matrix  $B$

## 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  $A$  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  $B$  has eigenvalues only on  $C_R \Delta t$  and can take advantage of probably all the different schemes.

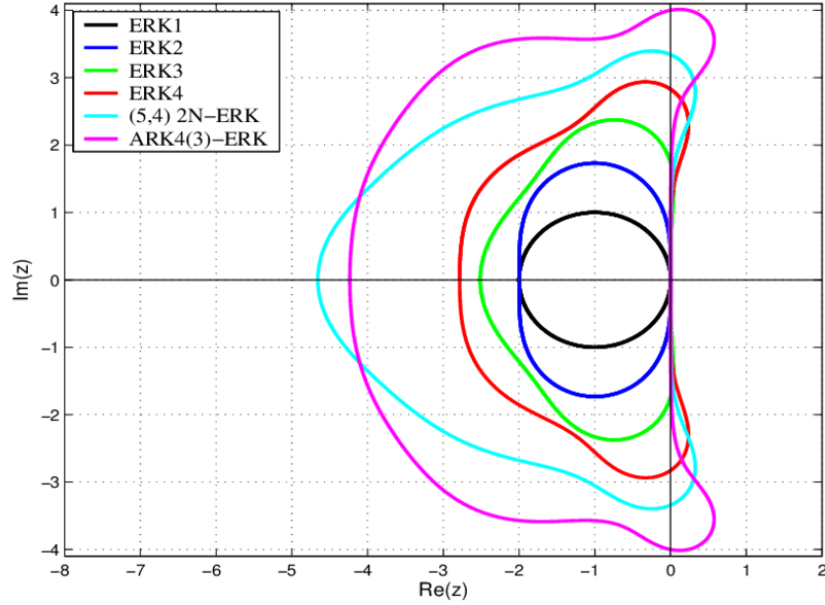


Figure 18: Stability Region [article]

## 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.

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

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

## Appendix)

### Python Code for 1 a)

```
1 """
2 Fall 2022 AEM 5253
3 Justine John "JJ" A. Serdoncillo
4 Homework 1 Number 1
5 """
6 import numpy as np
7 import matplotlib.pyplot as plt
8
9 dt = [0.1,0.5,1]
10
11 # g(y(t), t)
12 def g(y, t):
13     return - 2 * y;
14
15 # solvers
16 def anl(ti,tf,yi):
17     t = np.linspace(ti,tf,1000)
18     y = np.zeros(len(t))
19     y[0] = yi
20     y = yi * np.exp(-2*t)
21     return t, y
22 def exp(ti,tf,yi,dt):
23     N = int((tf-ti)/dt)
24     t = np.linspace(ti,tf,N+1)
25     y = np.zeros(len(t))
26     y[0] = yi
27     for i in range(len(t)-1):
28         y[i+1] = y[i] + dt * g(y[i],t[i])
29     return t, y
30 def imp(ti,tf,yi,dt):
31     N = int((tf-ti)/dt)
32     t = np.linspace(ti,tf,N+1)
33     y = np.zeros(len(t))
34     y[0] = yi
35     for i in range(len(t)-1):
36         nume = y[i]
37         denu = 1 + 2 * dt
38         y[i+1] = nume / denu
39     return t, y
40 def rk2(ti,tf,yi,dt):
41     N = int((tf-ti)/dt)
42     t = np.linspace(ti,tf,N+1)
43     y = np.zeros(len(t))
44     y[0] = yi
45     for i in range(len(t)-1):
46         k0 = g(y[i],t[i])
47         k1 = g(y[i]+dt/2*k0,t[i]+dt/2)
48         y[i+1] = y[i] + dt * k1
49     return t, y
50 def rk4(ti,tf,yi,dt):
51     N = int((tf-ti)/dt)
52     t = np.linspace(ti,tf,N+1)
53     y = np.zeros(len(t))
54     y[0] = yi
55     for i in range(len(t)-1):
56         k0 = g(y[i],t[i])
57         k1 = g(y[i]+dt/2*k0,t[i]+dt/2)
58         k2 = g(y[i]+dt/2*k1,t[i]+dt/2)
59         k3 = g(y[i]+dt * k2,t[i]+dt )
60         y[i+1] = y[i] + dt/6 * (k0+2*k1+2*k2+k3)
61     return t, y
62
63 # Analytical
64 t_anal, y_anal = anl(0,15,4)
65
66 # Explicit
67 fig1, ax1 = plt.subplots()
```



```

68 ax1.plot(t_anal,y_anal,'-r',label='Analytical Solution')
69 for i in dt:
70     t, y = exp(0,15,4,i)
71     label = 'Explicit Euler dt=' + str(i) + 's'
72     ax1.plot(t,y,'-o',label=label)
73 ax1.set_title('Explicit Euler Method with Analytical Solution')
74 ax1.set_xlabel('t')
75 ax1.set_ylabel('y')
76 ax1.set_xlim([0,15])
77 ax1.legend()
78 plt.savefig('images/exp1a.jpg')
79 plt.show()
80
81 # Implicit
82 fig1, ax1 = plt.subplots()
83 ax1.plot(t_anal,y_anal,'-r',label='Analytical Solution')
84 for i in dt:
85     t, y = imp(0,15,4,i)
86     label = 'Implicit Euler dt=' + str(i) + 's'
87     ax1.plot(t,y,'-o',label=label)
88 ax1.set_title('Implicit Euler Method with Analytical Solution')
89 ax1.set_xlabel('t')
90 ax1.set_ylabel('y')
91 ax1.set_xlim([0,15])
92 #ax1.set_ylim([0, 4])
93 ax1.legend()
94 plt.savefig('images/imp1a.jpg')
95 plt.show()
96
97 # RK2
98 fig1, ax1 = plt.subplots()
99 ax1.plot(t_anal,y_anal,'-r',label='Analytical Solution')
100 for i in dt:
101     t, y = rk2(0,15,4,i)
102     label = 'RK2 dt=' + str(i) + 's'
103     ax1.plot(t,y,'-o',label=label)
104 ax1.set_title('Runge-Kutta 2nd Order with Analytical Solution')
105 ax1.set_xlabel('t')
106 ax1.set_ylabel('y')
107 ax1.set_xlim([0,15])
108 ax1.legend()
109 plt.savefig('images/rk21a.jpg')
110 plt.show()
111
112 # RK4
113 fig1, ax1 = plt.subplots()
114 ax1.plot(t_anal,y_anal,'-r',label='Analytical Solution')
115 for i in dt:
116     t, y = rk4(0,15,4,i)
117     label = 'RK4 dt=' + str(i) + 's'
118     ax1.plot(t,y,'-o',label=label)
119 ax1.set_title('Runge-Kutta 4th Order with Analytical Solution')
120 ax1.set_xlabel('t')
121 ax1.set_ylabel('y')
122 ax1.set_xlim([0,15])
123 ax1.legend()
124 plt.savefig('images/rk41a.jpg')
125 plt.show()

```

## Python Code for 2 a)

```

1 """
2 Fall 2022 AEM 5253
3 Justine John "JJ" A. Serdoncillo
4 Homework 1 Number 2
5 """
6
7 import numpy as np
8 import matplotlib.pyplot as plt
9
10 dt = [0.1,0.5,1]
11

```

```

12 # solvers
13 def g(y, t):
14     return - (2+0.01*t**2) * y;
15 def anl(ti,tf,yi):
16     t = np.linspace(ti,tf,1000)
17     y = np.zeros(len(t))
18     y[0] = yi
19     y = yi * np.exp(-2*t - 0.01/3*t**3)
20     return t, y
21 def imp(ti,tf,yi,dt):
22     N = int((tf-ti)/dt)
23     t = np.linspace(ti,tf,N+1)
24     y = np.zeros(len(t))
25     y[0] = yi
26     for i in range(len(t)-1):
27         nume = y[i]
28         denu = 1 + dt * (2+0.01*t[i]**2)
29         y[i+1] = nume / denu
30     return t, y
31 #####
32 def exp(ti,tf,yi,dt):
33     N = int((tf-ti)/dt)
34     t = np.linspace(ti,tf,N+1)
35     y = np.zeros(len(t))
36     y[0] = yi
37     for i in range(len(t)-1):
38         y[i+1] = y[i] + dt * g(y[i],t[i])
39     return t, y
40 def rk2(ti,tf,yi,dt):
41     N = int((tf-ti)/dt)
42     t = np.linspace(ti,tf,N+1)
43     y = np.zeros(len(t))
44     y[0] = yi
45     for i in range(len(t)-1):
46         k0 = g(y[i],t[i])
47         k1 = g(y[i]+dt/2*k0,t[i]+dt/2)
48         y[i+1] = y[i] + dt * k1
49     return t, y
50 def rk4(ti,tf,yi,dt):
51     N = int((tf-ti)/dt)
52     t = np.linspace(ti,tf,N+1)
53     y = np.zeros(len(t))
54     y[0] = yi
55     for i in range(len(t)-1):
56         k0 = g(y[i],t[i])
57         k1 = g(y[i]+dt/2*k0,t[i]+dt/2)
58         k2 = g(y[i]+dt/2*k1,t[i]+dt/2)
59         k3 = g(y[i]+dt * k2,t[i]+dt)
60         y[i+1] = y[i] + dt/6 * (k0+2*k1+2*k2+k3)
61     return t, y
62
63 # Analytical
64 t_anal, y_anal = anl(0,15,4)
65
66 # Explicit
67 fig, ax = plt.subplots(2,1)
68 ax[0].plot(t_anal,y_anal,'-r',label='Analytical Solution')
69 ax[1].plot(t_anal,y_anal,'-r',label='Analytical Solution')
70 fig.suptitle('Explicit Euler Method with Analytical Solution')
71 i=dt[0]
72 t, y = exp(0,15,4,i)
73 label = 'Explicit Euler dt=' + str(i) + 's'
74 ax[0].plot(t,y,'-o',label=label)
75 i=dt[1]
76 t, y = exp(0,15,4,i)
77 label = 'Explicit Euler dt=' + str(i) + 's'
78 ax[0].plot(t,y,'-o',label=label)
79 i=dt[2]
80 t, y = exp(0,15,4,i)
81 label = 'Explicit Euler dt=' + str(i) + 's'
82 ax[1].plot(t,y,'-o',label=label)
83 ax[0].set_xlabel('t')

```

```

84 ax[0].set_ylabel('y')
85 ax[1].set_xlabel('t')
86 ax[1].set_ylabel('y')
87 ax[0].legend()
88 ax[1].legend()
89 plt.savefig('images/exp2a.jpg')
90 plt.show()
91
92 # Implicit
93 fig, ax = plt.subplots()
94 ax.plot(t_anal, y_anal, '-r', label='Analytical Solution')
95 for i in dt:
96     t, y = imp(0,15,4,i)
97     label = 'Implicit Euler dt=' + str(i) + 's'
98     ax.plot(t,y,'-o',label=label)
99 ax.set_title('Implicit Euler Method with Analytical Solution')
100 ax.set_xlabel('t')
101 ax.set_ylabel('y')
102 ax.set_xlim([0,15])
103 #ax.set_ylim([0, 4])
104 ax.legend()
105 plt.savefig('images/imp2a.jpg')
106 plt.show()
107
108 # RK2
109 fig, ax = plt.subplots(2,1)
110 ax[0].plot(t_anal, y_anal, '-r', label='Analytical Solution')
111 ax[1].plot(t_anal, y_anal, '-r', label='Analytical Solution')
112 fig.suptitle('Runge-Kutta 2nd Order with Analytical Solution')
113 i=dt[0]
114 t, y = rk2(0,15,4,i)
115 label = 'RK2 dt=' + str(i) + 's'
116 ax[0].plot(t,y,'-o',label=label)
117 i=dt[1]
118 t, y = rk2(0,15,4,i)
119 label = 'RK2 dt=' + str(i) + 's'
120 ax[0].plot(t,y,'-o',label=label)
121 i=dt[2]
122 t, y = rk2(0,15,4,i)
123 label = 'RK2 dt=' + str(i) + 's'
124 ax[1].plot(t,y,'-o',label=label)
125 ax[0].set_xlabel('t')
126 ax[0].set_ylabel('y')
127 ax[1].set_xlabel('t')
128 ax[1].set_ylabel('y')
129 ax[0].legend()
130 ax[1].legend()
131 plt.savefig('images/rk22a.jpg')
132 plt.show()
133
134 # RK4
135 fig, ax = plt.subplots()
136 ax.plot(t_anal, y_anal, '-r', label='Analytical Solution')
137 for i in dt:
138     t, y = rk4(0,15,4,i)
139     label = 'RK4 dt=' + str(i) + 's'
140     ax.plot(t,y,'-o',label=label)
141 ax.set_title('Runge-Kutta 4th Order with Analytical Solution')
142 ax.set_xlabel('t')
143 ax.set_ylabel('y')
144 ax.set_xlim([0,15])
145 ax.legend()
146 plt.savefig('images/rk42a.jpg')
147 plt.show()

```

## Python Code for 3 a)

```

1 """
2 Fall 2022 AEM 5253
3 Justine John "JJ" A. Serdoncillo
4 Homework 1 Number 3 Part B
5 """

```

```

6 import numpy as np
7 import matplotlib.pyplot as plt
8 import math
9
10 dt = [0.15, 0.5, 1]
11 grav = 9.81
12 Leng = 0.6
13
14 # g(y(t), t)
15 def g1(theta, dtheta, t):
16     return dtheta;
17 def g2(theta, dtheta, t):
18     return -grav/Leng * theta;
19
20 # solvers
21 def anl(ti,tf,yi,vi):
22     t = np.linspace(ti,tf,1000)
23     y = np.zeros(len(t))
24     y[0] = yi
25     for i in range(len(t)):
26         y[i] = yi * math.cos(t[i] * math.sqrt(grav/Leng))
27     return t, y
28 def exp(ti,tf,yi,vi,dt):
29     N = int((tf-ti)/dt)
30     t = np.linspace(ti,tf,N+1)
31     y = np.zeros(len(t))
32     v = np.zeros(len(t))
33     y[0] = yi
34     v[0] = vi
35     for i in range(len(t)-1):
36         y[i+1] = y[i] + dt * g1(y[i], v[i], t[i]);
37         v[i+1] = v[i] + dt * g2(y[i], v[i], t[i]);
38     return t, y
39 def imp(ti,tf,yi,vi,dt):
40     N = int((tf-ti)/dt)
41     t = np.linspace(ti,tf,N+1)
42     y = np.zeros(len(t))
43     v = np.zeros(len(t))
44     y[0] = yi
45     v[0] = vi
46     for i in range(len(t)-1):
47         # version 1; based on coupled equations
48         nume = y[i] + v[i] * dt
49         denu = 1 + grav/Leng * dt**2
50         y[i+1] = nume / denu
51         v[i+1] = (y[i+1] - y[i]) / dt;
52     return t, y
53 def rk4(ti,tf,yi,vi,dt):
54     N = int((tf-ti)/dt)
55     t = np.linspace(ti,tf,N+1)
56     y = np.zeros(len(t))
57     v = np.zeros(len(t))
58     y[0] = yi
59     v[0] = vi
60     for i in range(len(t)-1):
61         k01 = g1(y[i], v[i], t[i])
62         k02 = g2(y[i], v[i], t[i])
63         k11 = g1(y[i]+dt/2*k01, v[i]+dt/2*k02, t[i]+dt/2)
64         k12 = g2(y[i]+dt/2*k01, v[i]+dt/2*k02, t[i]+dt/2)
65         k21 = g1(y[i]+dt/2*k11, v[i]+dt/2*k12, t[i]+dt/2)
66         k22 = g2(y[i]+dt/2*k11, v[i]+dt/2*k12, t[i]+dt/2)
67         k31 = g1(y[i]+dt*k21, v[i]+dt*k22, t[i]+dt)
68         k32 = g2(y[i]+dt*k21, v[i]+dt*k22, t[i]+dt)
69         y[i+1] = y[i] + dt/6 * (k01+2*k11+2*k21+k31)
70         v[i+1] = v[i] + dt/6 * (k02+2*k12+2*k22+k32)
71     return t, y
72
73 # Analytical
74 t_anal, y_anal = anl(0,6,math.radians(10),0)
75
76 # Explicit
77 fig, ax = plt.subplots(2,2)

```

```

78 fig.suptitle('Explicit Euler Method with Analytical Solution')
79 ax[0,0].plot(t_anal,y_anal,'-r')
80 ax[0,1].plot(t_anal,y_anal,'-r')
81 ax[1,0].plot(t_anal,y_anal,'-r')
82 ax[0,0].set_title("dt = 0.15s")
83 ax[0,1].set_title("dt = 0.5s")
84 ax[1,0].set_title("dt = 1s")
85 t, y = exp(0,6,math.radians(10),0,0.15)
86 ax[0,0].plot(t,y,'-o')
87 t, y = exp(0,6,math.radians(10),0,0.5)
88 ax[0,1].plot(t,y,'-o')
89 t, y = exp(0,6,math.radians(10),0,1)
90 ax[1,0].plot(t,y,'-o')
91
92 ax[0,0].set_xlabel('t')
93 ax[0,0].set_ylabel('theta')
94 ax[0,1].set_xlabel('t')
95 ax[0,1].set_ylabel('theta')
96 ax[1,0].set_xlabel('t')
97 ax[1,0].set_ylabel('theta')
98
99 fig.tight_layout()
100 plt.savefig('images/exp3a.jpg')
101 plt.show()
102
103 # Implicit
104 fig, ax = plt.subplots(2,2)
105 fig.suptitle('Implicit Euler Method with Analytical Solution')
106 ax[0,0].plot(t_anal,y_anal,'-r')
107 ax[0,1].plot(t_anal,y_anal,'-r')
108 ax[1,0].plot(t_anal,y_anal,'-r')
109 ax[0,0].set_title("dt = 0.15s")
110 ax[0,1].set_title("dt = 0.5s")
111 ax[1,0].set_title("dt = 1s")
112 t, y = imp(0,6,math.radians(10),0,0.15)
113 ax[0,0].plot(t,y,'-o')
114 t, y = imp(0,6,math.radians(10),0,0.5)
115 ax[0,1].plot(t,y,'-o')
116 t, y = imp(0,6,math.radians(10),0,1)
117 ax[1,0].plot(t,y,'-o')
118
119 ax[0,0].set_xlabel('t')
120 ax[0,0].set_ylabel('theta')
121 ax[0,1].set_xlabel('t')
122 ax[0,1].set_ylabel('theta')
123 ax[1,0].set_xlabel('t')
124 ax[1,0].set_ylabel('theta')
125
126 fig.tight_layout()
127 plt.savefig('images/imp3a.jpg')
128 plt.show()
129
130 # Runge-Kutta 4
131 fig, ax = plt.subplots(2,2)
132 fig.suptitle('Runge-Kutta 4th Order with Analytical Solution')
133 ax[0,0].plot(t_anal,y_anal,'-r')
134 ax[0,1].plot(t_anal,y_anal,'-r')
135 ax[1,0].plot(t_anal,y_anal,'-r')
136 ax[0,0].set_title("dt = 0.15s")
137 ax[0,1].set_title("dt = 0.5s")
138 ax[1,0].set_title("dt = 1s")
139 t, y = rk4(0,6,math.radians(10),0,0.15)
140 ax[0,0].plot(t,y,'-o')
141 t, y = rk4(0,6,math.radians(10),0,0.5)
142 ax[0,1].plot(t,y,'-o')
143 t, y = rk4(0,6,math.radians(10),0,1)
144 ax[1,0].plot(t,y,'-o')
145
146 ax[0,0].set_xlabel('t')
147 ax[0,0].set_ylabel('theta')
148 ax[0,1].set_xlabel('t')
149 ax[0,1].set_ylabel('theta')

```

```

150 ax[1,0].set_xlabel('t')
151 ax[1,0].set_ylabel('theta')
152
153 fig.tight_layout()
154 plt.savefig('images/rk43a.jpg')
155 plt.show()

```

## Python Code for 3 b)

```

1  """
2  Fall 2022 AEM 5253
3  Justine John "JJ" A. Serdoncillo
4  Homework 1 Number 3 Part B
5  """
6  import numpy as np
7  import matplotlib.pyplot as plt
8  import math
9
10 dt = [0.15, 0.5, 1]
11 grav = 9.81
12 Leng = 0.6
13 c = 4
14
15 # g(y(t), t)
16 def g1(theta, dtheta, t):
17     return dtheta;
18 def g2(theta, dtheta, t):
19     return -grav/Leng * theta - c * dtheta;
20
21 # solvers
22 def anl(ti,tf,yi,vi):
23     t = np.linspace(ti,tf,1000)
24     y = np.zeros(len(t))
25     y[0] = yi
26     alph = math.sqrt(4*grav/Leng - c**2)/2
27     beta = c/math.sqrt(4*grav/Leng - c**2)
28     for i in range(len(t)):
29         y[i] = yi * math.exp(-c*t[i]/2) * (math.cos(alph*t[i]) + beta * math.sin(alph*t[i]))
30     return t, y
31 def exp(ti,tf,yi,vi,dt):
32     N = int((tf-ti)/dt)
33     t = np.linspace(ti,tf,N+1)
34     y = np.zeros(len(t))
35     v = np.zeros(len(t))
36     y[0] = yi
37     v[0] = vi
38     for i in range(len(t)-1):
39         y[i+1] = y[i] + dt * g1(y[i], v[i], t[i]);
40         v[i+1] = v[i] + dt * g2(y[i], v[i], t[i]);
41     return t, y
42 def imp(ti,tf,yi,vi,dt):
43     N = int((tf-ti)/dt)
44     t = np.linspace(ti,tf,N+1)
45     y = np.zeros(len(t))
46     v = np.zeros(len(t))
47     y[0] = yi
48     v[0] = vi
49     for i in range(len(t)-1):
50         nume = y[i] * (1 + c * dt) + v[i] * dt
51         denu = 1 + (1 + c * dt) + grav/Leng * dt**2
52         y[i+1] = nume / denu
53         v[i+1] = (y[i+1] - y[i]) / dt;
54     return t, y
55 def rk4(ti,tf,yi,vi,dt):
56     N = int((tf-ti)/dt)
57     t = np.linspace(ti,tf,N+1)
58     y = np.zeros(len(t))
59     v = np.zeros(len(t))
60     y[0] = yi
61     v[0] = vi
62     for i in range(len(t)-1):
63         k01 = g1(y[i], v[i], t[i])

```

```

64     k02 = g2(y[i], v[i], t[i])
65     k11 = g1(y[i]+dt/2*k01, v[i]+dt/2*k02, t[i]+dt/2)
66     k12 = g2(y[i]+dt/2*k01, v[i]+dt/2*k02, t[i]+dt/2)
67     k21 = g1(y[i]+dt/2*k11, v[i]+dt/2*k12, t[i]+dt/2)
68     k22 = g2(y[i]+dt/2*k11, v[i]+dt/2*k12, t[i]+dt/2)
69     k31 = g1(y[i]+dt*k21, v[i]+dt*k22, t[i]+dt)
70     k32 = g2(y[i]+dt*k21, v[i]+dt*k22, t[i]+dt)
71     y[i+1] = y[i] + dt/6 * (k01+2*k11+2*k21+k31)
72     v[i+1] = v[i] + dt/6 * (k02+2*k12+2*k22+k32)
73     return t, y
74
75 # Analytical
76 t_anal, y_anal = anl(0,6,math.radians(10),0)
77
78 # Explicit
79 fig, ax = plt.subplots(2,2)
80 fig.suptitle('Explicit Euler Method with Analytical Solution')
81 ax[0,0].plot(t_anal,y_anal,'-r')
82 ax[0,1].plot(t_anal,y_anal,'-r')
83 ax[1,0].plot(t_anal,y_anal,'-r')
84 ax[0,0].set_title("dt = 0.15s")
85 ax[0,1].set_title("dt = 0.5s")
86 ax[1,0].set_title("dt = 1s")
87 t, y = exp(0,6,math.radians(10),0,0.15)
88 ax[0,0].plot(t,y,'-o')
89 t, y = exp(0,6,math.radians(10),0,0.5)
90 ax[0,1].plot(t,y,'-o')
91 t, y = exp(0,6,math.radians(10),0,1)
92 ax[1,0].plot(t,y,'-o')
93
94 ax[0,0].set_xlabel('t')
95 ax[0,0].set_ylabel('theta')
96 ax[0,1].set_xlabel('t')
97 ax[0,1].set_ylabel('theta')
98 ax[1,0].set_xlabel('t')
99 ax[1,0].set_ylabel('theta')
100
101 fig.tight_layout()
102 plt.savefig('images/exp3b.jpg')
103 plt.show()
104
105 # Implicit
106 fig, ax = plt.subplots(2,2)
107 fig.suptitle('Implicit Euler Method with Analytical Solution')
108 ax[0,0].plot(t_anal,y_anal,'-r')
109 ax[0,1].plot(t_anal,y_anal,'-r')
110 ax[1,0].plot(t_anal,y_anal,'-r')
111 ax[0,0].set_title("dt = 0.15s")
112 ax[0,1].set_title("dt = 0.5s")
113 ax[1,0].set_title("dt = 1s")
114 t, y = imp(0,6,math.radians(10),0,0.15)
115 ax[0,0].plot(t,y,'-o')
116 t, y = imp(0,6,math.radians(10),0,0.5)
117 ax[0,1].plot(t,y,'-o')
118 t, y = imp(0,6,math.radians(10),0,1)
119 ax[1,0].plot(t,y,'-o')
120
121 ax[0,0].set_xlabel('t')
122 ax[0,0].set_ylabel('theta')
123 ax[0,1].set_xlabel('t')
124 ax[0,1].set_ylabel('theta')
125 ax[1,0].set_xlabel('t')
126 ax[1,0].set_ylabel('theta')
127
128 fig.tight_layout()
129 plt.savefig('images/imp3b.jpg')
130 plt.show()
131
132 # Runge-Kutta 4
133 fig, ax = plt.subplots(2,2)
134 fig.suptitle('Runge-Kutta 4th Order with Analytical Solution')
135 ax[0,0].plot(t_anal,y_anal,'-r')

```

```

136 ax[0,1].plot(t_anal,y_anal,'-r')
137 ax[1,0].plot(t_anal,y_anal,'-r')
138 ax[0,0].set_title("dt = 0.15s")
139 ax[0,1].set_title("dt = 0.5s")
140 ax[1,0].set_title("dt = 1s")
141 t, y = rk4(0,6,math.radians(10),0,0.15)
142 ax[0,0].plot(t,y,'-o')
143 t, y = rk4(0,6,math.radians(10),0,0.5)
144 ax[0,1].plot(t,y,'-o')
145 t, y = rk4(0,6,math.radians(10),0,1)
146 ax[1,0].plot(t,y,'-o')
147
148 ax[0,0].set_xlabel('t')
149 ax[0,0].set_ylabel('theta')
150 ax[0,1].set_xlabel('t')
151 ax[0,1].set_ylabel('theta')
152 ax[1,0].set_xlabel('t')
153 ax[1,0].set_ylabel('theta')
154
155 fig.tight_layout()
156 plt.savefig('images/rk43b.jpg')
157 plt.show()

```

## Python Code for 4

```

1  """
2  Fall 2022 AEM 5253
3  Justine John "JJ" A. Serdoncillo
4  Homework 1 Number 4
5  """
6  import numpy as np
7  import matplotlib.pyplot as plt
8  import math
9
10 ### Number 1 ###
11 sigm = 10
12 b = 8/3.
13
14 def gx(x, y, z, r):
15     return sigm*(y-x);
16 def gy(x, y, z, r):
17     return r*x - y - x*z;
18 def gz(x, y, z, r):
19     return x*y - b*z;
20
21 # solvers
22 def rk4(ti,tf,xi,yi,zi,r,dt):
23     N = int((tf-ti)/dt)
24     t = np.linspace(ti,tf,N+1)
25     x = np.zeros(len(t))
26     y = np.zeros(len(t))
27     z = np.zeros(len(t))
28     x[0] = xi
29     y[0] = yi
30     z[0] = zi
31     for i in range(len(t)-1):
32         k0x = gx(x[i], y[i], z[i], r)
33         k0y = gy(x[i], y[i], z[i], r)
34         k0z = gz(x[i], y[i], z[i], r)
35         k1x = gx(x[i]+dt/2*k0x, y[i]+dt/2*k0y, z[i]+dt/2*k0z, r)
36         k1y = gy(x[i]+dt/2*k0x, y[i]+dt/2*k0y, z[i]+dt/2*k0z, r)
37         k1z = gz(x[i]+dt/2*k0x, y[i]+dt/2*k0y, z[i]+dt/2*k0z, r)
38         k2x = gx(x[i]+dt/2*k1x, y[i]+dt/2*k1y, z[i]+dt/2*k1z, r)
39         k2y = gy(x[i]+dt/2*k1x, y[i]+dt/2*k1y, z[i]+dt/2*k1z, r)
40         k2z = gz(x[i]+dt/2*k1x, y[i]+dt/2*k1y, z[i]+dt/2*k1z, r)
41         k3x = gx(x[i]+dt*k2x, y[i]+dt*k2y, z[i]+dt*k2z, r)
42         k3y = gy(x[i]+dt*k2x, y[i]+dt*k2y, z[i]+dt*k2z, r)
43         k3z = gz(x[i]+dt*k2x, y[i]+dt*k2y, z[i]+dt*k2z, r)
44         x[i+1] = x[i] + dt/6 * (k0x+2*k1x+2*k2x+k3x)
45         y[i+1] = y[i] + dt/6 * (k0y+2*k1y+2*k2y+k3y)
46         z[i+1] = z[i] + dt/6 * (k0z+2*k1z+2*k2z+k3z)
47     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')

```

```
120 plt.show()
```

## Python Code for 5

```
1 """
2 Fall 2022 AEM 5253
3 Justine John "JJ" A. Serdoncillo
4 Homework 1 Number 5
5 """
6 import numpy as np
7 from numpy import linalg as LA
8
9 Num = np.linspace(11,51,5)
10
11 def Agen(N):
12     A = np.zeros((N,N))
13     A[0, -1] = -1
14     A[-1, 0] = 1
15     for i in range(N):
16         A[i-1,i ] = 1
17         A[i ,i-1] = -1
18     return A
19 def Bgen(N):
20     B = np.zeros((N,N))
21     B[0, -1] = 1
22     B[-1, 0] = 1
23     for i in range(N):
24         B[i-1,i ] = 1
25         B[i ,i-1] = 1
26         B[i-1,i-1] = 2
27     return B
28
29 for i in range(len(Num)):
30     print('for N = ', str(Num[i]))
31     Aeig = LA.eigvals(Agen(int(Num[i])))
32     for j in range(4):
33         AeigYeah = round(Aeig[j].real,2) + round(Aeig[j].imag,2) * 1j
34     print(AeigYeah)
35 for i in range(len(Num)):
36     print('for N = ', str(Num[i]))
37     Beig = LA.eigvals(Bgen(int(Num[i])))
38     Beig.argsort()[::-1]
39     for j in range(4):
40         BeigYeah = round(Beig[j].real,2) + round(Beig[j].imag,2) * 1j
41     print(BeigYeah)
```