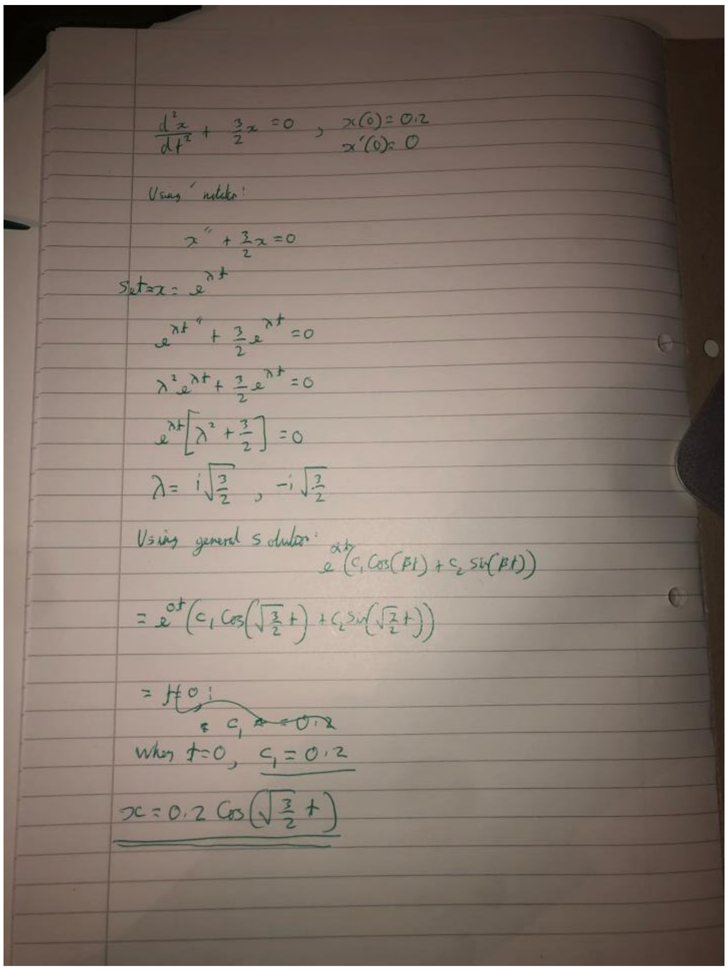
Python – MAS2806 assignment 2 written report by Rishik Kumar

Question 1

Analytical solution for part 1a)



$$\frac{d^2x}{dt^2} + \frac{3}{2}x = 0 \quad , \quad x(0) = 0.2, \quad x'(0) = 0$$

Using ' notation:

$$x'' + \frac{3}{2}x = 0$$

Set $x = e^{\lambda t}$

$$e^{\lambda t''} + \frac{3}{2}e^{\lambda t} = 0$$

$$\lambda^2 e^{\lambda t} + \frac{3}{2}e^{\lambda t} = 0$$

$$e^{\lambda t}\left[\lambda^2 + \frac{3}{2}\right] = 0$$

$$\lambda = i\sqrt{\frac{3}{2}} \, , \, -i\sqrt{\frac{3}{2}}$$

Using general solution: $e^{\alpha t}(c_1 \cos(\beta t) + c_2 \sin(\beta t))$

$$= e^{0t}\left(c_1 \cos\left(\sqrt{\frac{3}{2}}t\right) + c_2 \sin\left(\sqrt{\frac{3}{2}}t\right)\right)$$

= $t = 0$:

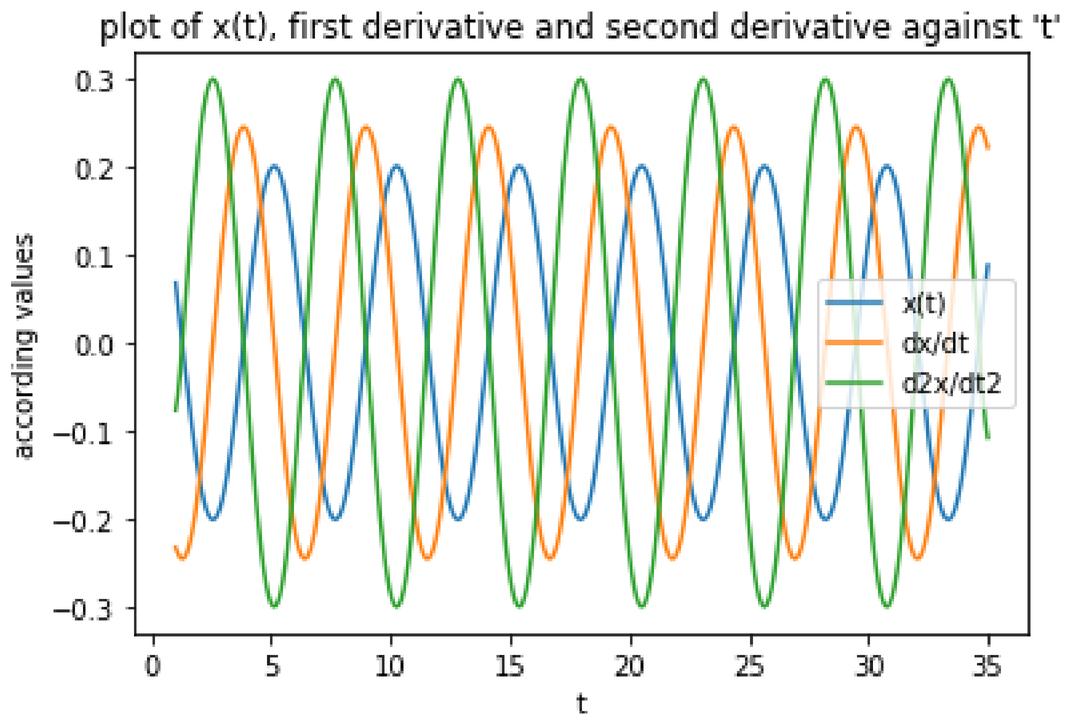
~~$c_1 = 0.2$~~

When $t = 0$, $\underline{c_1 = 0.2}$

$$\underline{\underline{x = 0.2 \cos\left(\sqrt{\frac{3}{2}}t\right)}}$$

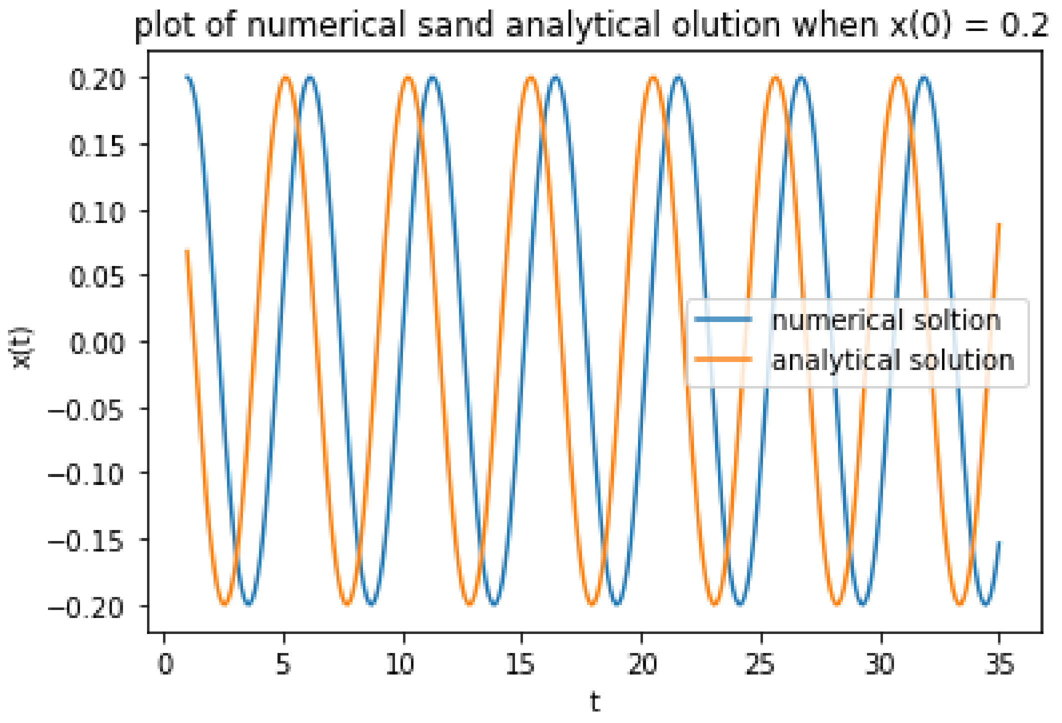
This image is show working to show that x = 0.2 * cos((sqrt(3/2)) * t) as required.

Plot of x(t), first derivative and second derivative against t.



Question 3
In this question we will be comparing the analytical and numerical solution.

To start with let's look at when x(0) = 0.2.

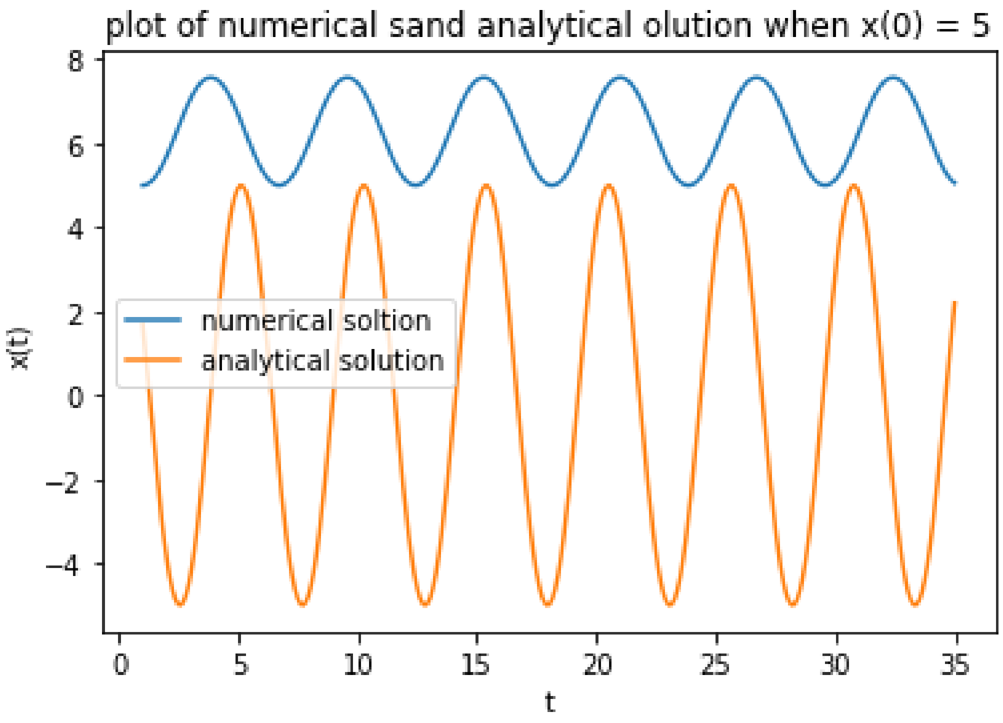


As we can see, both graphs tend to have the same shape.
As we set a and b to zero, we used the small angle approximation in the analytical solution plot. We can also observe that the time interval is very slightly different as

well due to using the small angle approximation. However, they both at peaks and troughs at 0.2 and -0.2 respectively.

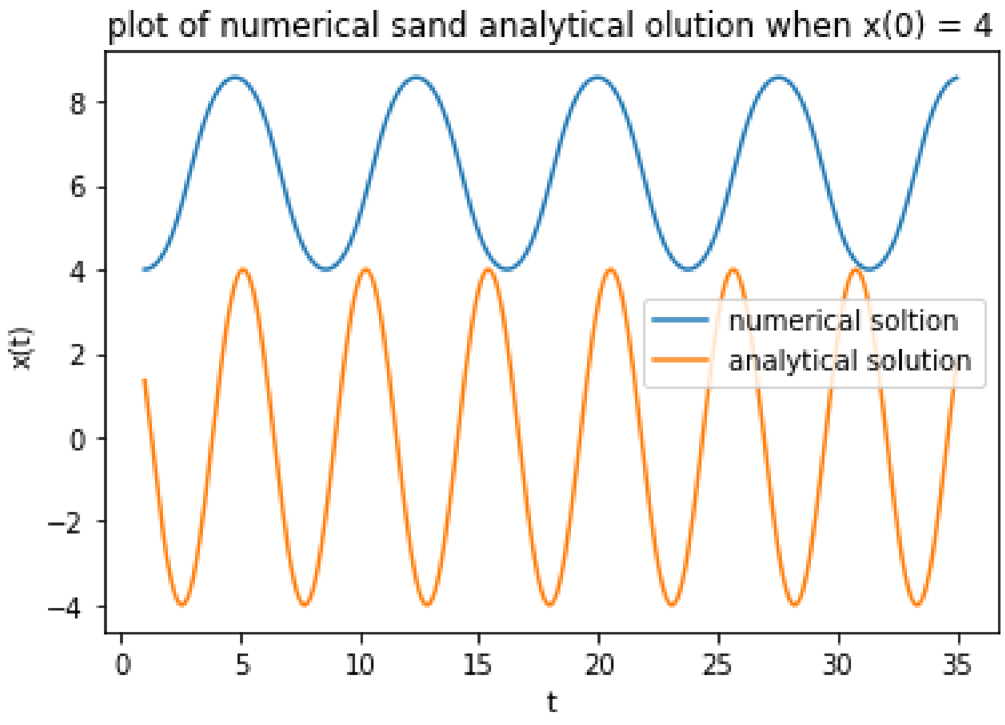
Now lets look at higher values of x(0).

This shows a plot for the analytical and numerical solution have x(0) = 5



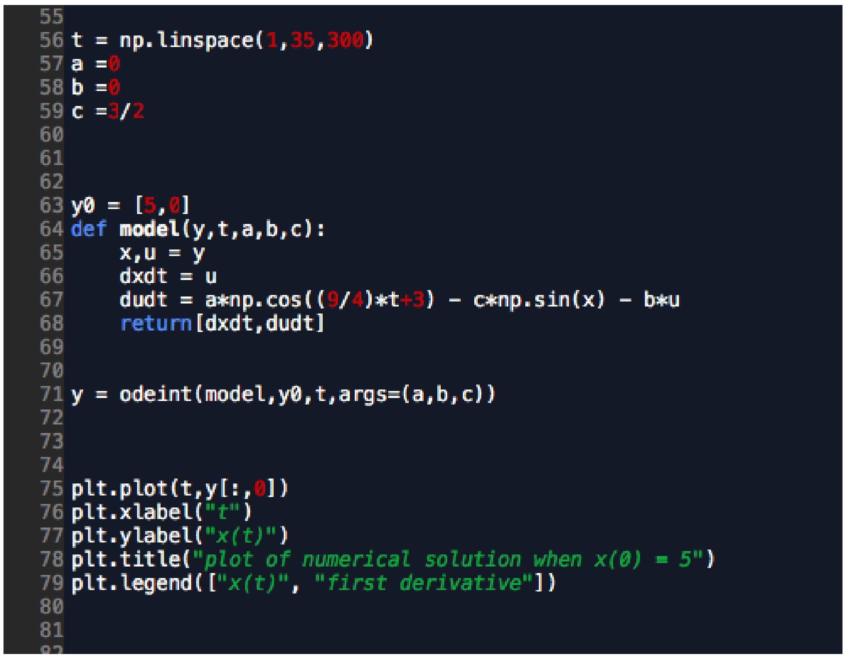
We can infer that the numerical solution has become slightly different and no longer has any negative values, but still holds the same shape of a sine curve. The analytical solution has just increased the peaks and troughs to reach the new x(0) being 5. The time period of both curves are slightly different to each other. The numerical solution has its trough at x(t) = 5 which is worth noting.

I then tried to find which value of x(0) results in the change with the numerical solution.
I found out that just after x(0) = 3 the graph for numerical solution changes. As shown below.



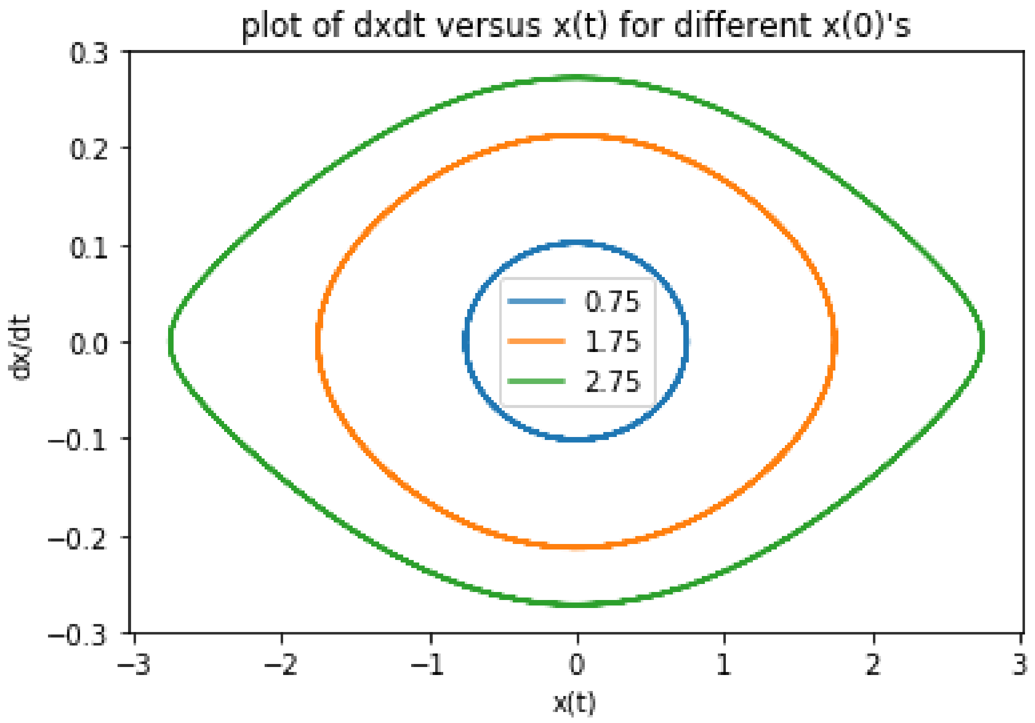
Here we can clearly see that when x(0) = 4 The numerical value only takes positive values. This is due to my specific equation given.

Code for question 3a), where we change y0 accordingly



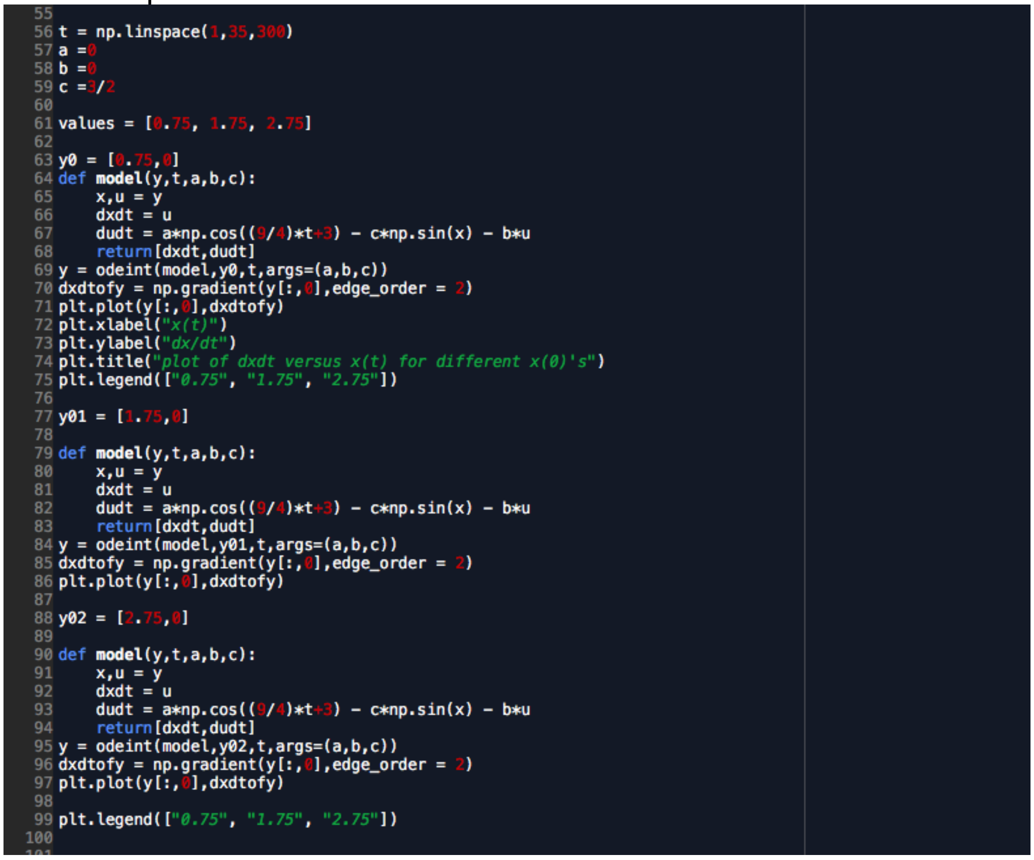
```
t = np.linspace(1,35,300)
a =0
b =0
c =3/2



y0 = [5,0]
def model(y,t,a,b,c):
    x,u = y
    dxdt = u
    dudt = a*np.cos((9/4)*t+3) - c*np.sin(x) - b*u
    return[dxdt,dudt]


y = odeint(model,y0,t,args=(a,b,c))



plt.plot(t,y[:,0])
plt.xlabel("t")
plt.ylabel("x(t)")
plt.title("plot of numerical solution when x(0) = 5")
plt.legend(["x(t)", "first derivative"])
```

For part 3b)



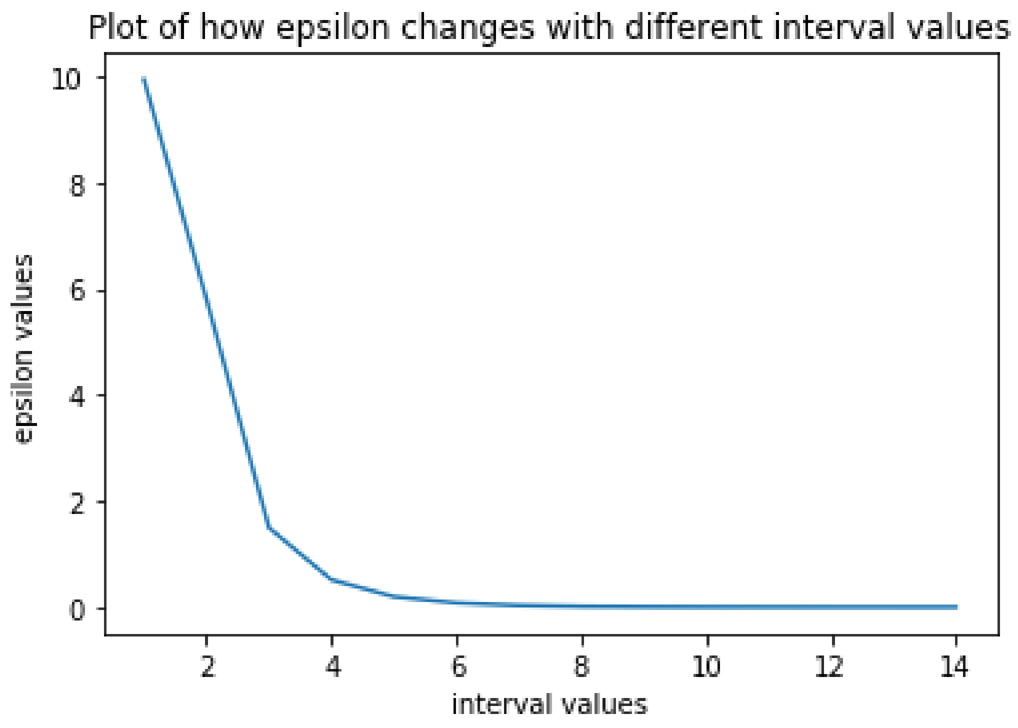
Here we can see that for smaller x(0)'s it looks more like an oval/ellipse however, for larger values of x(0)'s it gets increasingly distorted.

Code for question 3b below



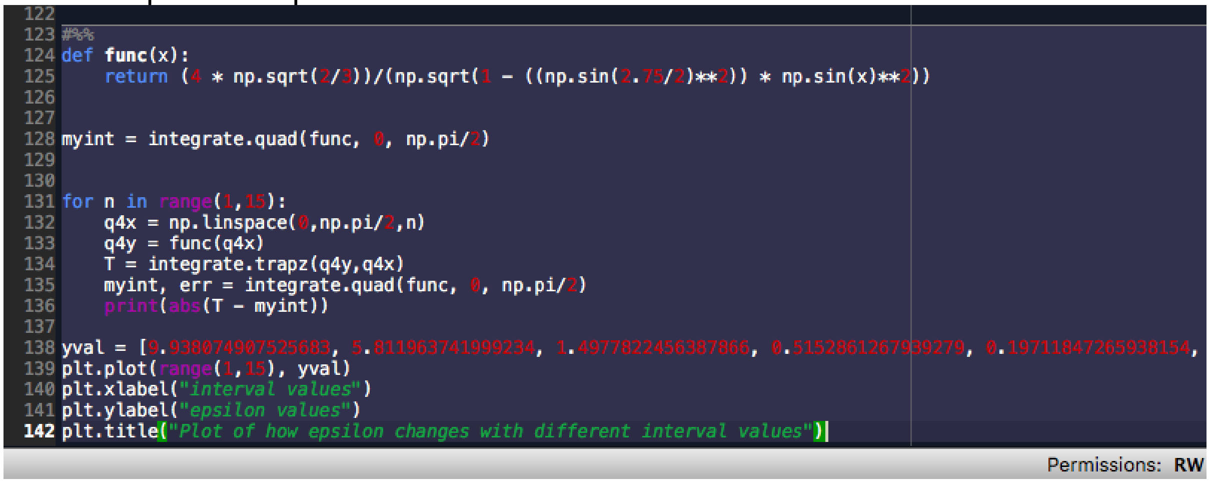
```

t = np.linspace(1,35,300)
a =0
b =0
c =3/2

values = [0.75, 1.75, 2.75]

y0 = [0.75,0]
def model(y,t,a,b,c):
    x,u = y
    dxdt = u
    dudt = a*np.cos((9/4)*t+3) - c*np.sin(x) - b*u
    return[dxdt,dudt]
y = odeint(model,y0,t,args=(a,b,c))
dxdtofy = np.gradient(y[:,0],edge_order = 2)
plt.plot(y[:,0],dxdtofy)
plt.xlabel("x(t)")
plt.ylabel("dx/dt")
plt.title("plot of dxdt versus x(t) for different x(0)'s")
plt.legend(["0.75", "1.75", "2.75"])

y01 = [1.75,0]

def model(y,t,a,b,c):
    x,u = y
    dxdt = u
    dudt = a*np.cos((9/4)*t+3) - c*np.sin(x) - b*u
    return[dxdt,dudt]
y = odeint(model,y01,t,args=(a,b,c))
dxdtofy = np.gradient(y[:,0],edge_order = 2)
plt.plot(y[:,0],dxdtofy)

y02 = [2.75,0]

def model(y,t,a,b,c):
    x,u = y
    dxdt = u
    dudt = a*np.cos((9/4)*t+3) - c*np.sin(x) - b*u
    return[dxdt,dudt]
y = odeint(model,y02,t,args=(a,b,c))
dxdtofy = np.gradient(y[:,0],edge_order = 2)
plt.plot(y[:,0],dxdtofy)

plt.legend(["0.75", "1.75", "2.75"])

```

Question 4



This plot shows how the epsilon value changes due to different interval values. Where epsilon is the difference between the 'trap' method and the 'quad' method. The graph shows us that with larger interval values the smaller the error is between the 'trap' method and the actual value. When interval size is large the 'trap' method is more accurate.
This shows that the error in the 'trap' method depends on the interval size.

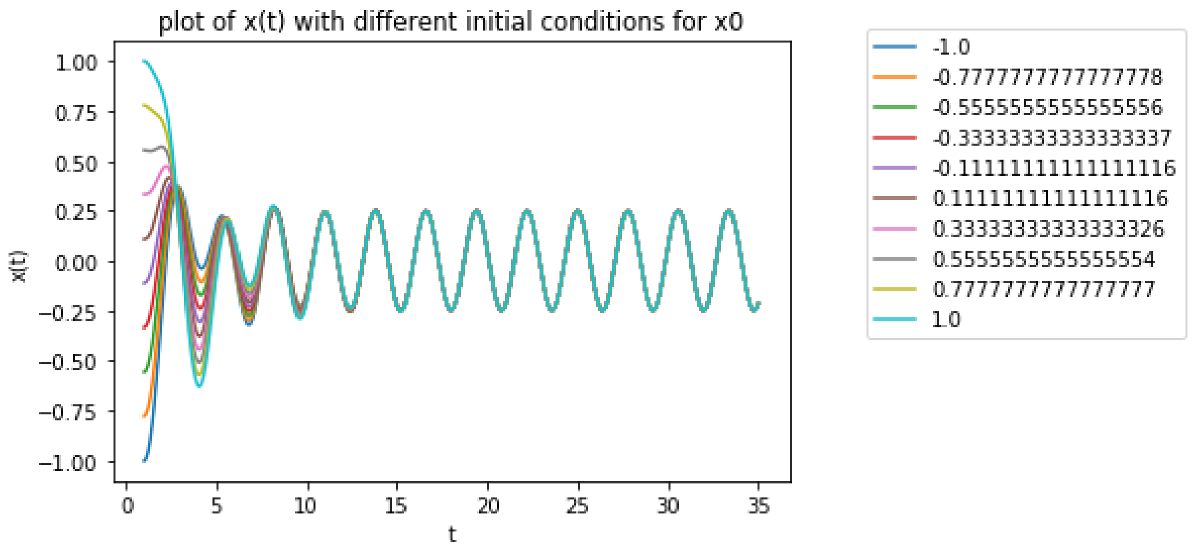
Code for question 4 provided below



```
#%%
def func(x):
    return (4 * np.sqrt(2/3))/(np.sqrt(1 - ((np.sin(2.75/2)**2)) * np.sin(x)**2))


myint = integrate.quad(func, 0, np.pi/2)


for n in range(1,15):
    q4x = np.linspace(0,np.pi/2,n)
    q4y = func(q4x)
    T = integrate.trapz(q4y,q4x)
    myint, err = integrate.quad(func, 0, np.pi/2)
    print(abs(T - myint))

yval = [9.938074907525683, 5.811963741999234, 1.4977822456387866, 0.5152861267939279, 0.19711847265938154,
plt.plot(range(1,15), yval)
plt.xlabel("interval values")
plt.ylabel("epsilon values")
plt.title("Plot of how epsilon changes with different interval values")
```

Permissions: RW

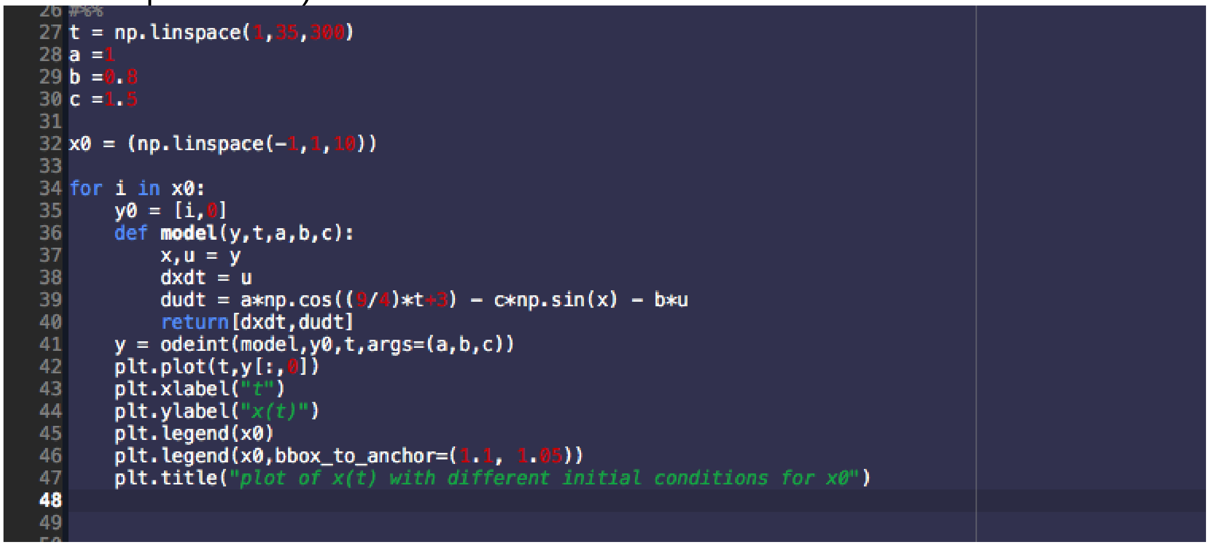
Question 5)



As we can see the initial condition affects the ‘Damping’, where the higher the initial condition the faster the damping takes place, i.e. gets to equilibrium quicker.

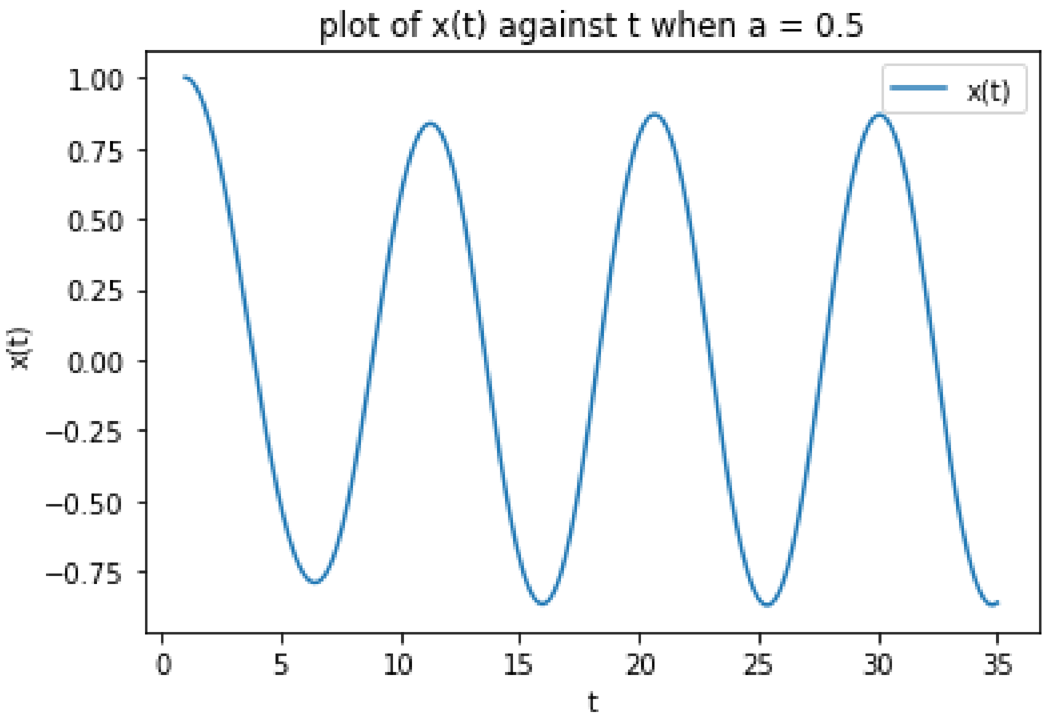
This plot shows the values of x(t) with different initial conditions ranging from -1 to 1. As we can see there is a distinct ‘sin-type’ curve. The different lines tend to have similar time period of oscillations. As we set a = 1, we allowed the damping term to have an affect. With higher values of x(0) the damping term is also larger therefore having greater affect on the curve.

Code for question 5a)



```
#%%
t = np.linspace(1,35,300)
a =1
b =0.8
c =1.5

x0 = (np.linspace(-1,1,10))

for i in x0:
    y0 = [i,0]
    def model(y,t,a,b,c):
        x,u = y
        dxdt = u
        dudt = a*np.cos((9/4)*t+3) - c*np.sin(x) - b*u
        return[dxdt,dudt]
    y = odeint(model,y0,t,args=(a,b,c))
    plt.plot(t,y[:,0])
    plt.xlabel("t")
    plt.ylabel("x(t)")
    plt.legend(x0)
    plt.legend(x0,bbox_to_anchor=(1.1, 1.05))
    plt.title("plot of x(t) with different initial conditions for x0")

```

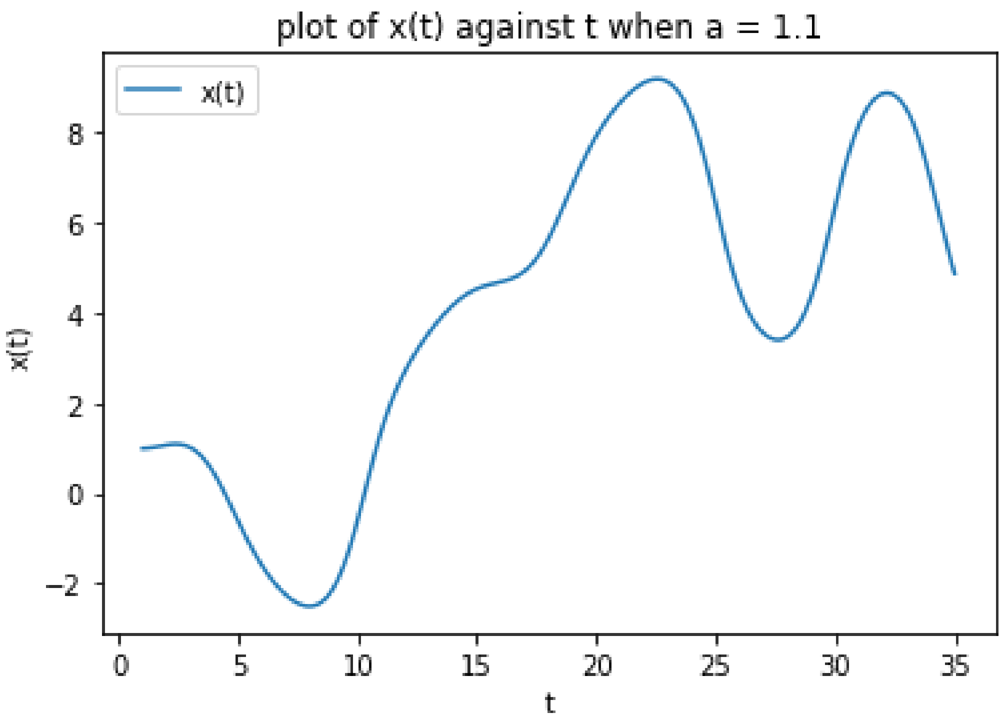
For question 5 part b)
We are asked to experiment with the ‘forcing’ term of the equation.



This above is a plot when a = 0.5.

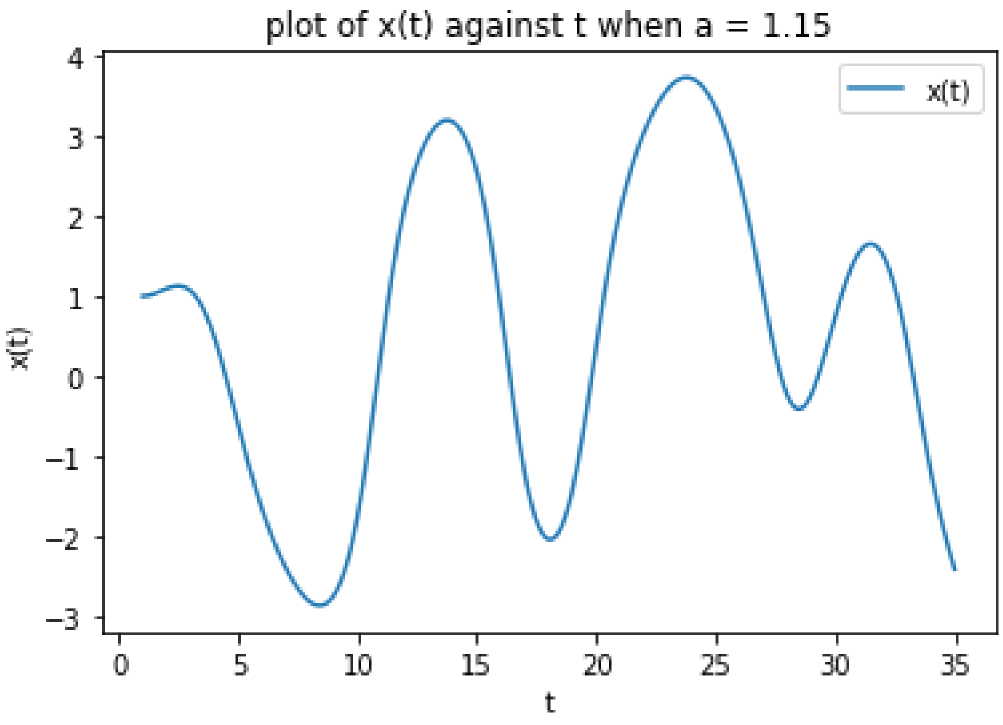
This looks like a normal sine curve after a few oscillations.
However, this changes when we increase the value of a

This next plot is for a = 1.1



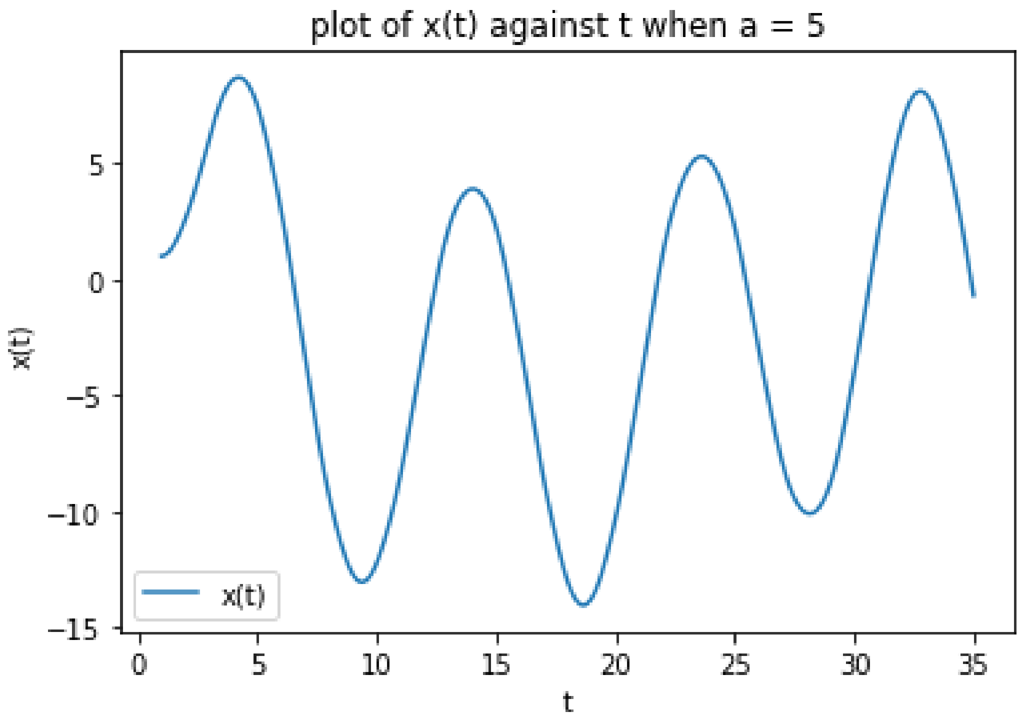
As you can see, the curve is very distorted with multiple peaks.
This is due to increasing the forcing term.

The next plot is 1.15



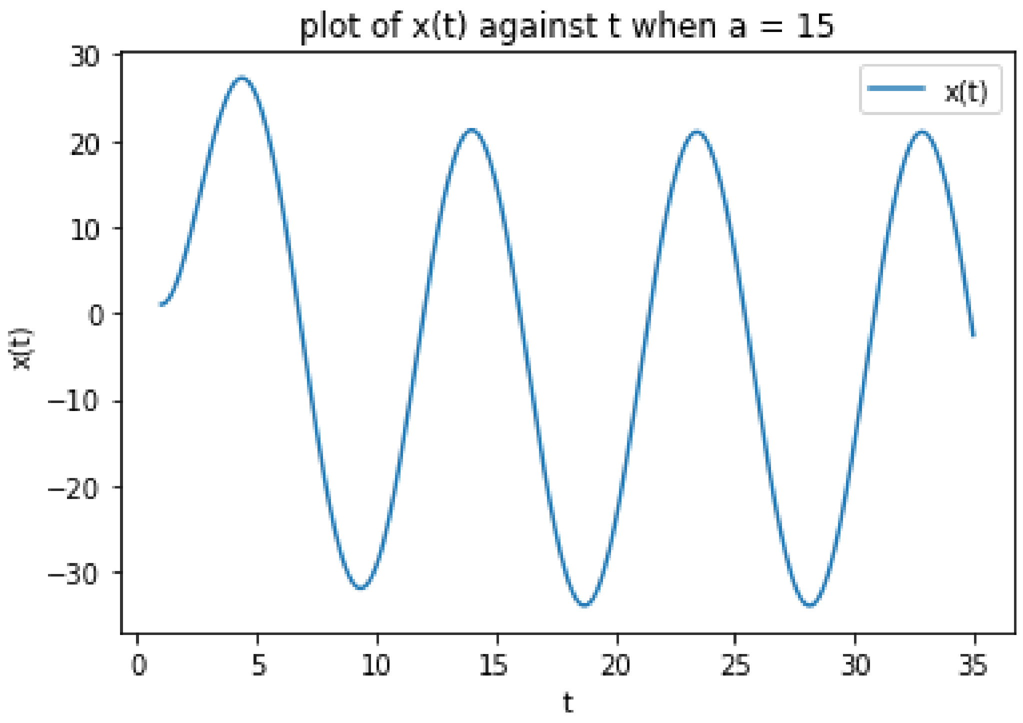
As we see here that just by changing the forcing term, by changing the a value by only 0.05, still has an impact. This shows that the forcing term is very sensitive.

This distortion continues up to around a = 5, where you can see that the graph is slightly looking like a sine curve again.
I will provide a plot below



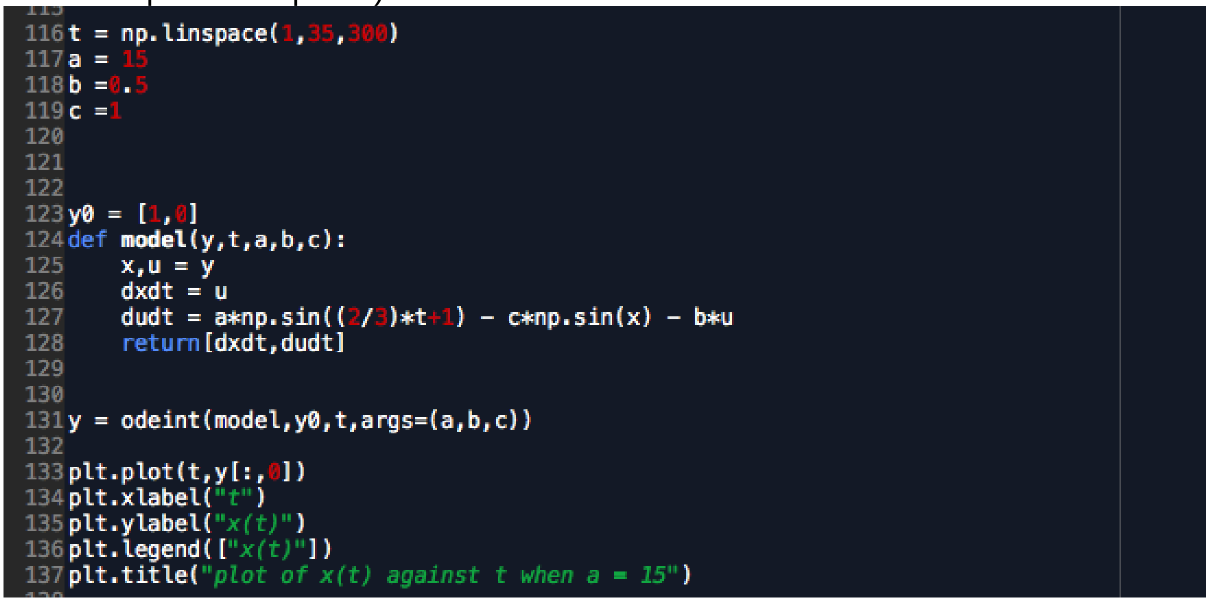
Then setting a to a large value such as a = 15, The graph is looking very similar to a sin curve again.

Graph of a = 15



This shows that the period between 1 and 4 is an anomaly compared to the other values, as the curve continues back to normal when a increases past 5.

Code for question 5 part b)



```
t = np.linspace(1,35,300)
a = 15
b =0.5
c =1



y0 = [1,0]
def model(y,t,a,b,c):
    x,u = y
    dxdt = u
    dudt = a*np.sin((2/3)*t+1) - c*np.sin(x) - b*u
    return[dxdt,dudt]


y = odeint(model,y0,t,args=(a,b,c))

plt.plot(t,y[:,0])
plt.xlabel("t")
plt.ylabel("x(t)")
plt.legend(["x(t)"])
plt.title("plot of x(t) against t when a = 15")
```