Project_assignment

October 24, 2023

1 Honours Differential Equations

1.1 Project Assignment

Due: Friday 2nd December 2022, noon

2 Orlagh Keane

3 S2084384

3.1 Question 1

[1]:
```
import sympy as sym
sym.init_printing()
from IPython.display import display_latex
import sympy.plotting as sym_plot
import matplotlib.pyplot as plt
```

[2]:
```
x = sym.Function('x')
y = sym.Function('y')
a = sym.symbols('a')
t = sym.symbols('t')
eq1 = sym.Eq(x(t).diff(t), y(t))
eq2 = sym.Eq(y(t).diff(t), -x(t) + a*(y(t)-((y(t))**3)/3))
```

Part a

[3]:
```
EQS = sym.Matrix([eq1.rhs, eq2.rhs])
EQS

def lin_matrix(system, vec0):
    X, Y = sym.symbols('X, Y')
    FG = sym.Matrix([system[0].rhs, system[1].rhs]).subs({x(t):X, y(t):Y})
    matJ = FG.jacobian([X, Y])
    return matJ.subs({X:vec0[0], Y:vec0[1]})

def linearise(system, vec0):
    u = sym.Function('u')
```

```
    v = sym.Function('v')
    lin_mat = lin_matrix(system, vec0)
    lin_rhs = lin_mat * sym.Matrix([u(t), v(t)])
    linsys = [sym.Eq(u(t).diff(t), lin_rhs[0]),
              sym.Eq(v(t).diff(t), lin_rhs[1])]
    return linsys
```

[4]:
```
import numpy as np
print('The equations')
display_latex(list(EQS))
vec0 = [0,0]
print("have a critical point at "+ str(vec0))
linmat = lin_matrix([eq1, eq2],vec0)
linsys = linearise([eq1, eq2],vec0)
print("and the eigenvectors:")
display_latex(list(linmat.eigenvects()))
print("and the eigenvalues:")
eigen = list(linmat.eigenvals().keys())
display_latex(eigen)
```

```
The equations
```

$$\left[y(t),\ a\left(-\frac{y^3(t)}{3}+y(t)\right)-x(t)\right]$$

```
have a critical point at [0, 0]
and the eigenvectors:
```

$$\left[\left(\frac{a}{2}-\frac{\sqrt{(a-2)\,(a+2)}}{2},\ 1,\ \left[\left[\begin{matrix}\frac{a}{2}+\frac{\sqrt{(a-2)(a+2)}}{2}\\1\end{matrix}\right]\right]\right),\ \left(\frac{a}{2}+\frac{\sqrt{(a-2)\,(a+2)}}{2},\ 1,\ \left[\left[\begin{matrix}\frac{a}{2}-\frac{\sqrt{(a-2)(a+2)}}{2}\\1\end{matrix}\right]\right]\right)\right]$$

```
and the eigenvalues:
```

$$\left[\frac{a}{2}-\frac{\sqrt{(a-2)\,(a+2)}}{2},\ \frac{a}{2}+\frac{\sqrt{(a-2)\,(a+2)}}{2}\right]$$

The behaiviour of the critical point changes at a=0, 0<a<2, a=2 and a>2.

For a<2, the system will be oscillatory in the form a+bi, where a and b are real scalars and i is the imaginary number.

At a=0, the real part of the eigenvalue is zero, the system is unstable and behaves as an undamped oscillator.

At 0<a<=2, the real part of the eigenvalue is positive, the system is unstable and behaves as an unstable oscillator.

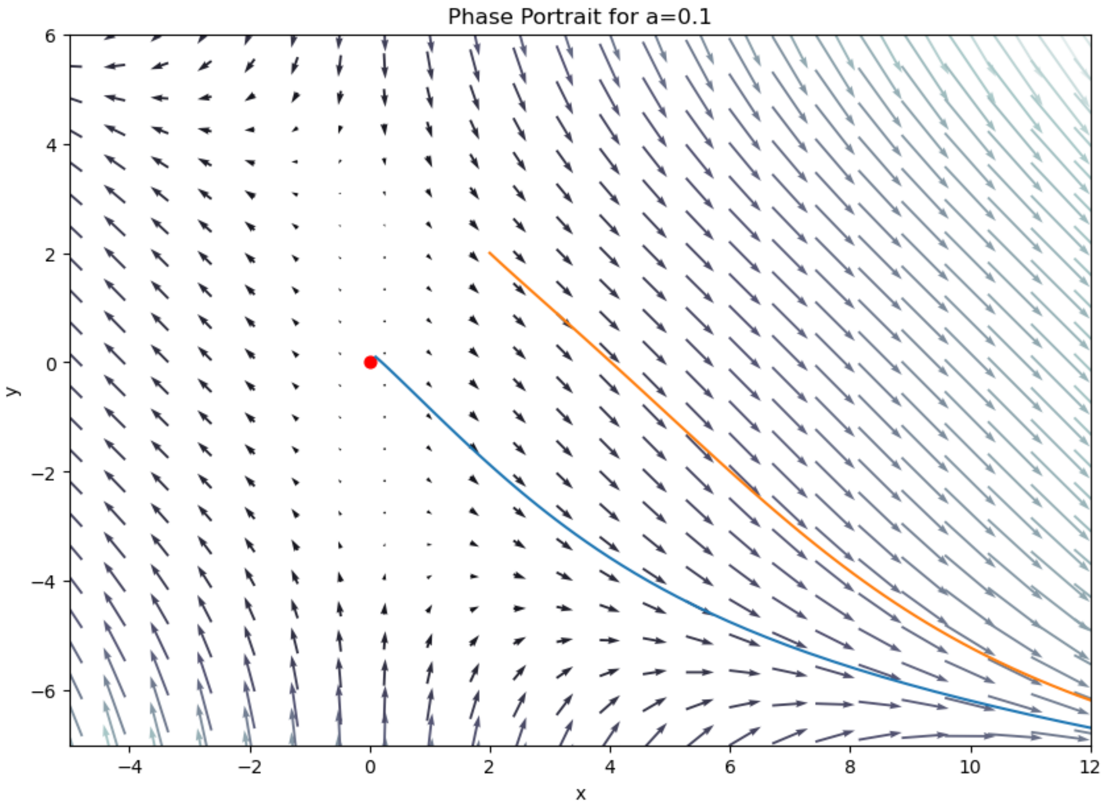
At a>2, both parts of the eigenvalue are real, and the system is unstable.

The system behaves the same as above for the opposite, -a.

Part b

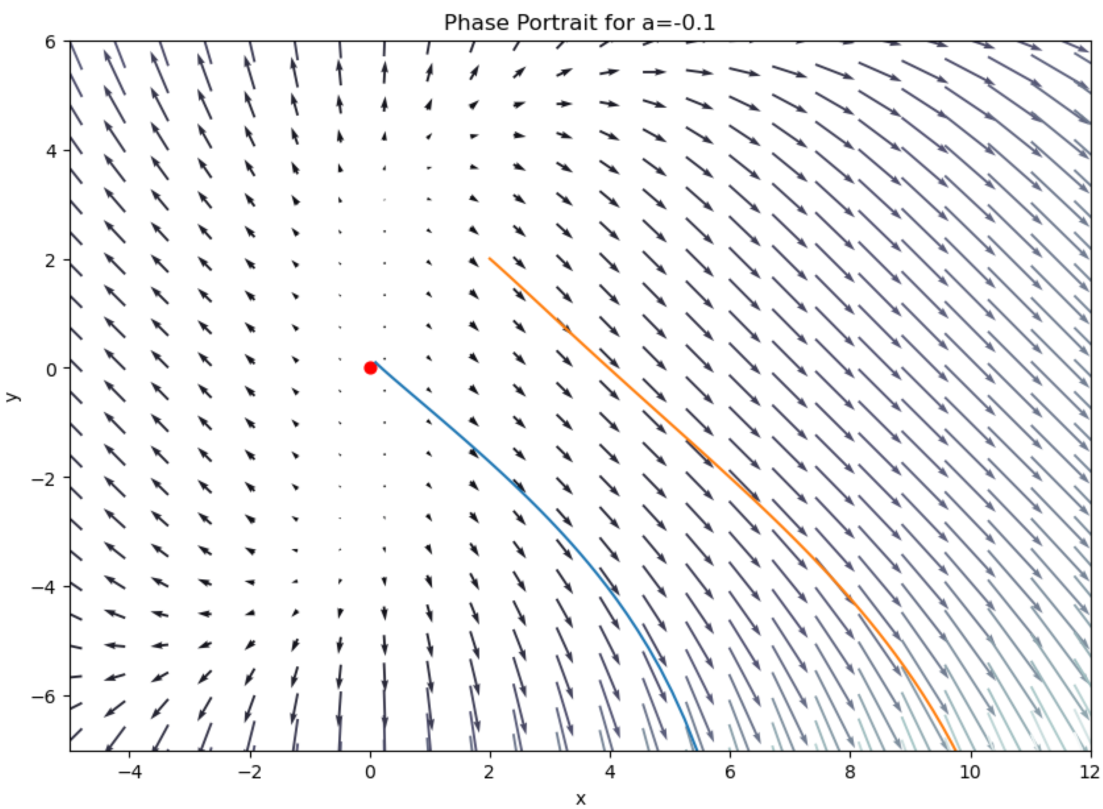
[5]:

```
#part b
import numpy as np
from matplotlib import pyplot as plt
from scipy.integrate import odeint
%matplotlib inline

# Define vector field
def vField1(x,t):
    u = x[0]
    v = -x[0]+(0.1)*(x[1]-(x[1]**3)/3)
    return [u,v]

# Plot vector field
X, Y = np.mgrid[-5:13:25j,-8:6:25j]
U1, V1 = vField1([X,Y],0)

# define colours for each vector based on their lengths
M1 = np.hypot(U1, V1)

fig, ax = plt.subplots(figsize=(10, 7))
ax.quiver(X, Y, U1, V1, M1, scale=1/0.005, pivot = 'mid', cmap = plt.cm.bone)

# Settings for trajectories
# for [0,0], a= 0.1
ics = [[0.1,0.1], [2,2]]
durations = [5,2]

# plot trajectories
for i, ic in enumerate(ics):
    t = np.linspace(0, durations[i], 100)
    x1 = odeint(vField1, ic, t)
    #x2 = odeint(vField2, ic, t)
    ax.plot(x1[:,0], x1[:,1], label='X0=(%.2f, %.2f)' % (ic[0], ic[1]) )
    #ax.plot(x2[:,0], x2[:,1], label='X0=(%.2f, %.2f)' % (ic[0], ic[1]) )

ax.scatter(0, 0, color='red', s=40)

plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-5,12)
plt.ylim(-7,6)
#plt.legend()
plt.title('Phase Portrait for a=0.1')
plt.show()
```



[6]:
```
import numpy as np
from matplotlib import pyplot as plt
from scipy.integrate import odeint
%matplotlib inline

def vField2(x,t):
    u = x[0]
    v = -x[0]+(-0.1)*(x[1]-(x[1]**3)/3)
    return [u,v]

X, Y = np.mgrid[-5:13:25j,-8:6:25j]
U2, V2 = vField2([X,Y],0)

M2 = np.hypot(U2, V2)

ics = [[0.1,0.1], [2,2]]
durations = [4,1.7]


fig, ax = plt.subplots(figsize=(10, 7))
```

```
ax.quiver(X, Y, U2, V2, M2, scale=1/0.005, pivot = 'mid', cmap = plt.cm.bone)
for i, ic in enumerate(ics):
    t = np.linspace(0, durations[i], 100)
    x2 = odeint(vField2, ic, t)
    ax.plot(x2[:,0], x2[:,1], label='X0=(%.2f, %.2f)' % (ic[0], ic[1]) )

ax.scatter(0, 0, color='red', s=40)

plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-5,12)
plt.ylim(-7,6)
#plt.legend()
plt.title('Phase Portrait for a=-0.1')
plt.show()
```



The behaviour is consistent with the eigenvalues you found for the linearised system. For both a=0.1 and a=-0.1, the imaginary part of the eigenvalue does not exist thus both systems are behave based on the real part of the eigenvalues. The postive eigenvalues produce a plot with some arrows pointing inwards to the critcal point and the negative eigenvalues produce a plot with arrows pointing away.Write your written solution here. You may have to include extra markdown cells.

3.2 Question 2

Part a

[7]:
```
#part a
x = sym.Function('x')
y = sym.Function('y')
a = sym.symbols('a')
b = sym.symbols('b')
t = sym.symbols('t')
eq1 = sym.Eq(x(t).diff(t), a - x(t) - b*x(t) + (x(t)**2)*y(t))
eq2 = sym.Eq(y(t).diff(t), b*x(t) -(x(t)**2)*y(t) )
EQS = sym.Matrix([eq1.rhs, eq2.rhs])
EQS
CP = sym.solve(EQS)

def lin_matrix(system, vec):
    X, Y = sym.symbols('X, Y')
    FG = sym.Matrix([system[0].rhs, system[1].rhs]).subs({x(t):X, y(t):Y})
    matJ = FG.jacobian([X, Y])
    return matJ.subs({X:vec[0], Y:vec[1]})

def linearise(system, vec):
    x = sym.Function('x')
    y = sym.Function('y')
    lin_mat = lin_matrix(system, vec)
    lin_rhs = lin_mat * sym.Matrix([x(t), y(t)])
    linsys = [sym.Eq(x(t).diff(t), lin_rhs[0]),
              sym.Eq(y(t).diff(t), lin_rhs[1])]
    return linsys

vec1 = list(CP[0].values())
print("Critical point ")
display_latex(list(vec1))
print("Linearised system:")
linmat = lin_matrix([eq1, eq2], vec1)
display_latex(linmat)
linsys = linearise([eq1, eq2], vec1)
display_latex(linsys)
print("Eigenvectors:")
display_latex(list(linmat.eigenvects()))
print("Eigenvalues:")
display_latex(list(linmat.eigenvals().keys()))
```

```
Critical point
```

$[x(t),\ x(t)y(t)]$

```
Linearised system:
```

$$\begin{bmatrix} -b+2x^2(t)y(t)-1 & x^2(t) \\ b-2x^2(t)y(t) & -x^2(t) \end{bmatrix}$$

$$\left[\frac{d}{dt}x(t) = \left(-b+2x^2(t)y(t)-1\right)x(t)+x^2(t)y(t),\ \frac{d}{dt}y(t) = \left(b-2x^2(t)y(t)\right)x(t)-x^2(t)y(t)\right]$$

Eigenvectors:

$$\left[\left(-\frac{b}{2}-\frac{\sqrt{\left(b-2x^2(t)y(t)+x^2(t)+1\right)^2-4x^2(t)}}{2}+x^2(t)y(t)-\frac{x^2(t)}{2}-\frac{1}{2},\ 1,\ \left[\left[\frac{}{b-\sqrt{b^2-4bx^2(t)y(t)+2bx^2(t)+2b+4x^4(t)y}}\right.\right.\right.\right.$$

Eigenvalues:

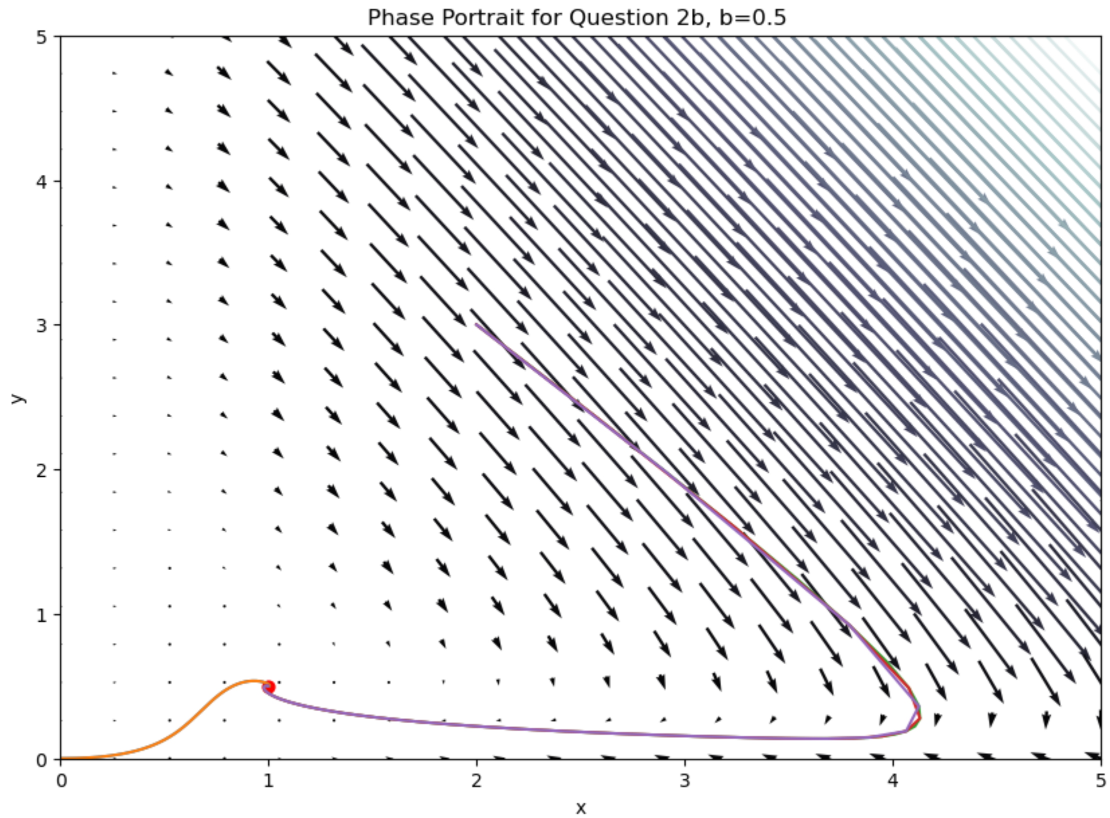
$$\left[-\frac{b}{2}-\frac{\sqrt{\left(b-2x^2(t)y(t)+x^2(t)+1\right)^2-4x^2(t)}}{2}+x^2(t)y(t)-\frac{x^2(t)}{2}-\frac{1}{2},\ -\frac{b}{2}+\frac{\sqrt{\left(b-2x^2(t)y(t)+x^2(t)+1\right)^2-}}{2}\right.$$

Part b

[8]:
```
x = sym.Function('x')
y = sym.Function('y')
a = sym.symbols('a')
b = sym.symbols('b')
t = sym.symbols('t')
eq1 = sym.Eq(x(t).diff(t), 1 - x(t) - 0.5*x(t) + (x(t)**2)*y(t))
eq2 = sym.Eq(y(t).diff(t), 0.5*x(t) -(x(t)**2)*y(t) )
EQS = sym.Matrix([eq1.rhs, eq2.rhs])
CPS = sym.solve(EQS)
eq3 = sym.Eq(x(t).diff(t), 1 - x(t) - 3*x(t) + (x(t)**2)*y(t))
eq4 = sym.Eq(y(t).diff(t), 3*x(t) -(x(t)**2)*y(t) )
EQS = sym.Matrix([eq3.rhs, eq4.rhs])
CP = sym.solve(EQS)
CPS.append(CP)
#CPS
```

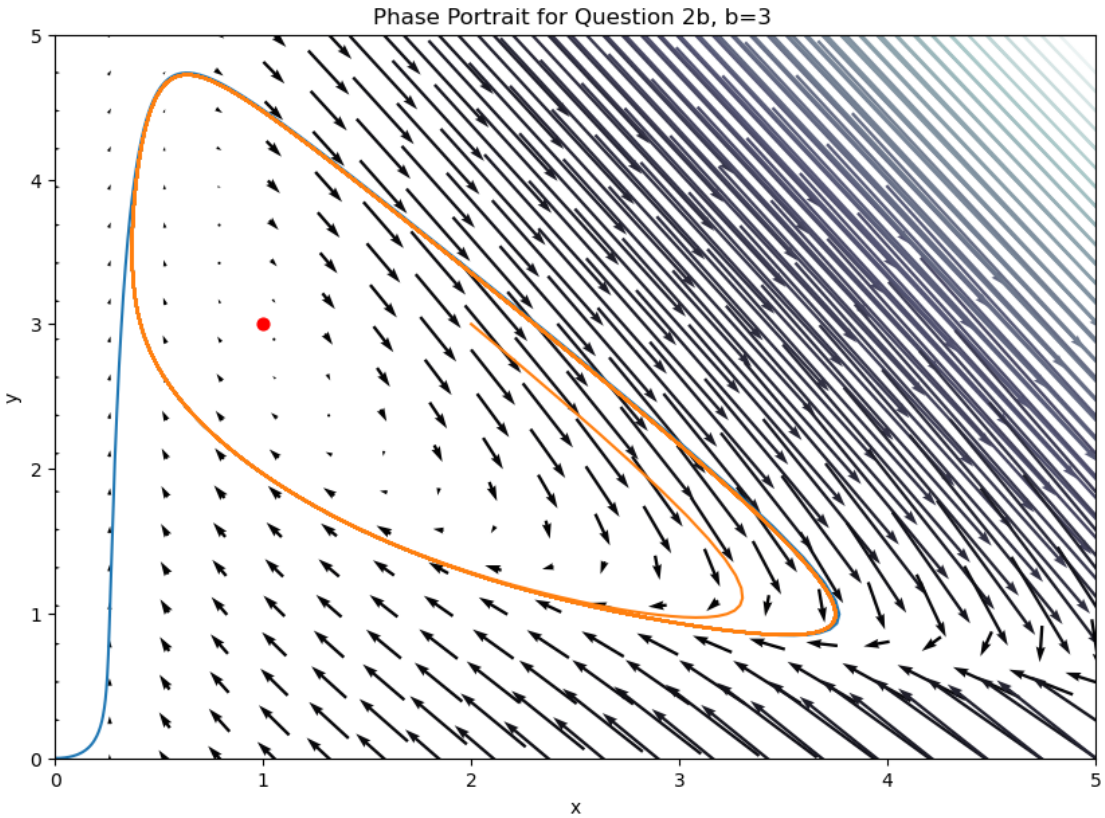
[9]:
```
import numpy as np
from matplotlib import pyplot as plt
from scipy.integrate import odeint
%matplotlib inline

# Define vector field
def vField1(x,t):
    u = 1 - x[0] - (0.5)*x[0] + (x[0]**2)*x[1]
    v = (0.5)*x[0] -(x[0]**2)*x[1]
    return [u,v]

# Plot vector field
X, Y = np.mgrid[0:5:20j,0:5:20j]
U1, V1 = vField1([X,Y],0)
```

```python
# define colours for each vector based on their lengths
M1 = np.hypot(U1, V1)

fig, ax = plt.subplots(figsize=(10, 7))
ax.quiver(X, Y, U1, V1, M1,  scale=1/0.005, pivot = 'mid', cmap = plt.cm.bone)

# Settings for trajectories
# for [xy,x],
ics = [[0,0], [0,0], [2,3], [2,3], [2,3]]
durations = [40,10,10, 20, 30]

# plot trajectories
for i, ic in enumerate(ics):
    t = np.linspace(0, durations[i], 400)
    x1 = odeint(vField1, ic, t)
    ax.plot(x1[:,0], x1[:,1], label='X0=(%.2f, %.2f)' % (ic[0], ic[1]) )


ax.scatter(1, 0.5, color='red', s=40)

plt.xlabel('x')
plt.ylabel('y')
plt.xlim(0,5)
plt.ylim(0,5)
#plt.legend()
plt.title('Phase Portrait for Question 2b, b=0.5 ')
plt.show()
```




```
[10]: import numpy as np
from matplotlib import pyplot as plt
from scipy.integrate import odeint
%matplotlib inline

# Define vector field
def vField2(x,t):
    u = 1 - x[0] - (3)*x[0] + (x[0]**2)*x[1]
    v = (3)*x[0] -(x[0]**2)*x[1]
    return [u,v]

# Plot vector field
X, Y = np.mgrid[0:5:20j,0:5:20j]
U2, V2 = vField2([X,Y],0)

# define colours for each vector based on their lengths
M2 = np.hypot(U2, V2)

fig, ax = plt.subplots(figsize=(10, 7))
ax.quiver(X, Y, U2, V2, M2, scale=1/0.01, pivot = 'mid', cmap = plt.cm.bone)
```

```
# Settings for trajectories
# for [xy,x],
ics = [[0,0], [2,3]]
durations = [100,100]

# plot trajectories
for i, ic in enumerate(ics):
    t = np.linspace(0, durations[i], 3700)
    x2 = odeint(vField2, ic, t)
    ax.plot(x2[:,0], x2[:,1], label='X0=(%.2f, %.2f)' % (ic[0], ic[1]) )


ax.scatter(1, 3, color='red', s=40)

plt.xlabel('x')
plt.ylabel('y')
plt.xlim(0,5)
plt.ylim(0,5)
#plt.legend()
plt.title('Phase Portrait for Question 2b, b=3 ')

plt.show()
```

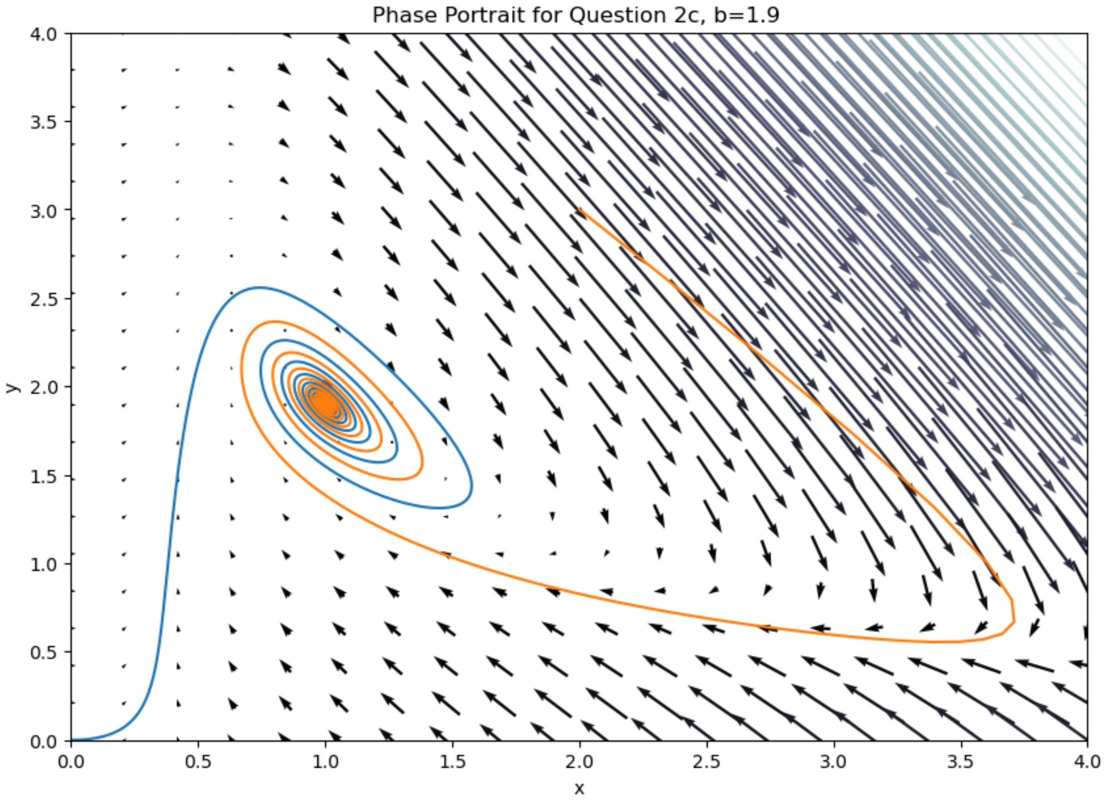


Both the phase portraits above are very different. For b=0.5, the critial point is sink node and all trajectories go to the node. Whereas, for b = 3, a bifurcation occurs since some trajectories close to critial point go outwards and then spiral, and other trajectories travel inwards and spiral in the same ring.

Part c

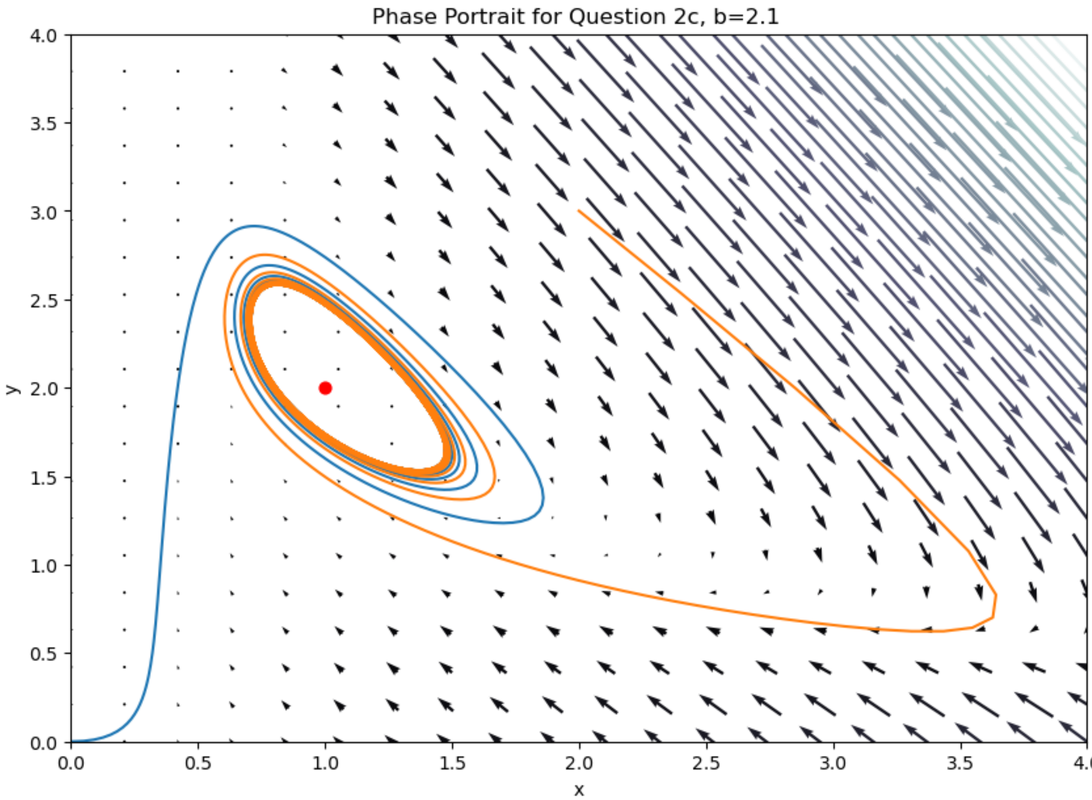
[11]:
```
#for b=1.9
import numpy as np
from matplotlib import pyplot as plt
from scipy.integrate import odeint
%matplotlib inline

# Define vector field
def vField2(x,t):
    u = 1 - x[0] - (1.9)*x[0] + (x[0]**2)*x[1]
    v = (1.9)*x[0] -(x[0]**2)*x[1]
    return [u,v]

# Plot vector field
X, Y = np.mgrid[0:4:20j,0:4:20j]
U2, V2 = vField2([X,Y],0)

# define colours for each vector based on their lengths
M2 = np.hypot(U2, V2)

fig, ax = plt.subplots(figsize=(10, 7))
ax.quiver(X, Y, U2, V2, M2, scale=1/0.01, pivot = 'mid', cmap = plt.cm.bone)

# Settings for trajectories
# for [xy,x],
ics = [[0,0], [2,3]]
durations = [80,80]

# plot trajectories
for i, ic in enumerate(ics):
    t = np.linspace(0, durations[i], 1900)
    x2 = odeint(vField2, ic, t)
    ax.plot(x2[:,0], x2[:,1], label='X0=(%.2f, %.2f)' % (ic[0], ic[1]) )


ax.scatter(1, 2, color='red', s=40)

plt.xlabel('x')
plt.ylabel('y')
plt.xlim(0,4)
plt.ylim(0,4)
```

```
#plt.legend()
plt.title('Phase Portrait for Question 2c, b=1.9 ')

plt.show()
```



[12]:
```
#for b=3.1
import numpy as np
from matplotlib import pyplot as plt
from scipy.integrate import odeint
%matplotlib inline

# Define vector field
def vField2(x,t):
    u = 1 - x[0] - (2.1)*x[0] + (x[0]**2)*x[1]
    v = (2.1)*x[0] -(x[0]**2)*x[1]
    return [u,v]

# Plot vector field
X, Y = np.mgrid[0:4:20j,0:4:20j]
U2, V2 = vField2([X,Y],0)
```

```python
# define colours for each vector based on their lengths
M2 = np.hypot(U2, V2)

fig, ax = plt.subplots(figsize=(10, 7))
ax.quiver(X, Y, U2, V2, M2, scale=1/0.005, pivot = 'mid', cmap = plt.cm.bone)

# Settings for trajectories
# for [xy,x],
ics = [[0,0], [2,3]]
durations = [80,80]

# plot trajectories
for i, ic in enumerate(ics):
    t = np.linspace(0, durations[i], 1500)
    x2 = odeint(vField2, ic, t)
    ax.plot(x2[:,0], x2[:,1], label='X0=(%.2f, %.2f)' % (ic[0], ic[1]) )


ax.scatter(1, 2, color='red', s=40)

plt.xlabel('x')
plt.ylabel('y')
plt.xlim(0,4)
plt.ylim(0,4)
#plt.legend()
plt.title('Phase Portrait for Question 2c, b=2.1 ')

plt.show()
```

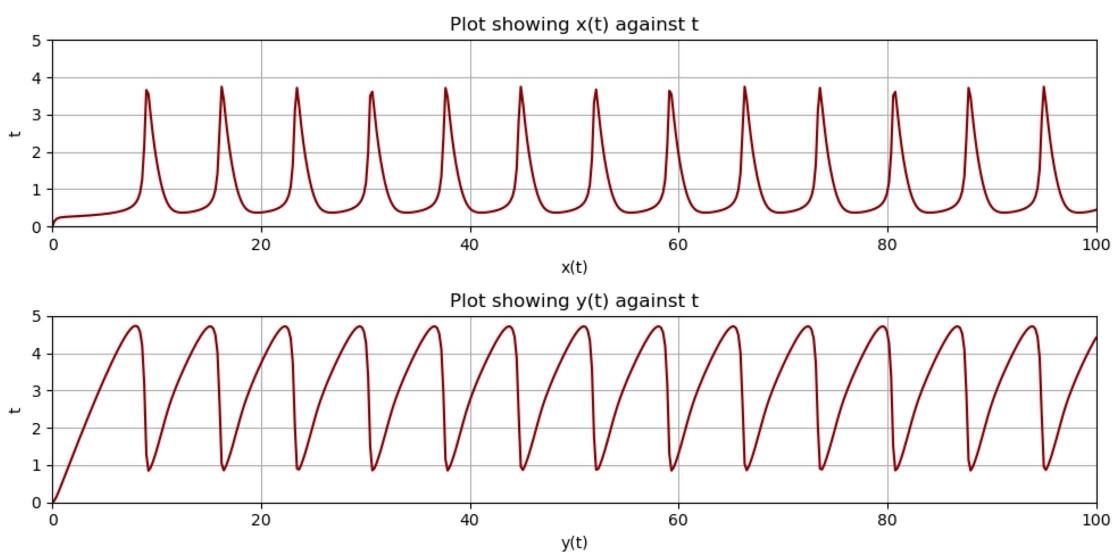


When a=1 and b = 2, the eigenvalues are imaginary, [i, -i], causing this bifurcation. Above are the plots for b = 1.9 and b = 2.1.

Part d

[13]:
```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import numpy as np

x = sym.symbols('x')
y = sym.symbols('y')

def vField(x,t):
    u = 1 - x[0] - (3)*x[0] + (x[0]**2)*x[1]
    v = (3)*x[0] -(x[0]**2)*x[1]
    return [u,v]


fig, ax = plt.subplots(2, 1, figsize=(10, 5))

t=np.linspace(0, 100, 500)
```

```
x0, y0 = 0,0
x = odeint(vField,[x0,y0],t)

ax[0].plot(t, x[:,0], 'maroon')


ax[1].plot(t, x[:,1], 'maroon')

ax[0].set_ylim(0, 5)
ax[1].set_ylim(0, 5)
ax[0].set_xlim(0, 100)
ax[1].set_xlim(0, 100)
ax[0].set_xlabel('x(t)')
ax[1].set_xlabel('y(t)')
ax[0].set_ylabel('t')
ax[1].set_ylabel('t')
ax[0].grid(True)
ax[1].grid(True)
ax[0].set_title('Plot showing x(t) against t')
ax[1].set_title('Plot showing y(t) against t')
fig.tight_layout()


plt.show()
```



3.3 Question 3

Part a

[14]:
```
#part a
def ModifiedEuler(func, times, y0):
    times = np.array(times)
    y0 = np.array(y0)
    n = y0.size       # the dimension of ODE
    nT = times.size   # the number of time steps
    y = np.zeros([nT,n])
    y[0, :] = y0
    # loop for timesteps
    for k in range(nT-1):
        h = times[k+1]-times[k]
        y[k+1, :] = y[k, :] + h*func(y[k, :] + (1/2)*h*func(y[k, :],times[k]) ,␣
↪times[k] + (1/2)*h  )
    return y

def ode_Euler(func, times, y0):
    times = np.array(times)
    y0 = np.array(y0)
    n = y0.size       # the dimension of ODE
    nT = times.size   # the number of time steps
    y = np.zeros([nT,n])
    y[0, :] = y0
    # loop for timesteps
    for k in range(nT-1):
        y[k+1, :] = y[k, :] + (times[k+1]-times[k])*func(y[k, :], times[k])
    return y
```

Part b

[15]:
```
#partb
def eq3_dy_dt(y, t):
    return 5*t - 2*((y)**0.5)

times = np.linspace(0,2,41)
eq3_modified_euler = ModifiedEuler(eq3_dy_dt, times, 2)
eq3_euler = ode_Euler(eq3_dy_dt, times, 2)

print(eq3_modified_euler)
```

```
[[2.        ]
 [1.86735114]
 [1.7517406 ]
 [1.65268081]
 [1.56965729]
 [1.50213273]
 [1.44955252]
 [1.41135153]
 [1.38696173]
```

```
[1.3758202 ]
[1.37737709]
[1.39110278]
[1.416494  ]
[1.45307839]
[1.50041766]
[1.5581091 ]
[1.62578582]
[1.70311598]
[1.78980106]
[1.88557378]
[1.99019558]
[2.10345402]
[2.22516024]
[2.35514638]
[2.49326335]
[2.63937863]
[2.79337433]
[2.9551455 ]
[3.12459855]
[3.30164989]
[3.48622477]
[3.67825617]
[3.87768396]
[4.08445402]
[4.29851758]
[4.5198306 ]
[4.74835326]
[4.98404941]
[5.22688628]
[5.476834  ]
[5.73386537]]
```

Part c

[16]:
```
#part c
t = sym.symbols('t')
y = sym.Function('y')
eq8 = sym.Eq(y(t).diff(t), 5*t - 2*sym.sqrt(y(t)))
print('The equation')
display_latex(eq8)
# Solve the ODE
eq8_sol = sym.dsolve(eq8, y(t), ics={y(0):2}, hint = 'best')
print('has the exact solutions: ')
display_latex(eq8_sol)
#eq8_sol
```

```
The equation
```

$$\frac{d}{dt}y(t) = 5t - 2\sqrt{y(t)}$$

has the exact solutions:

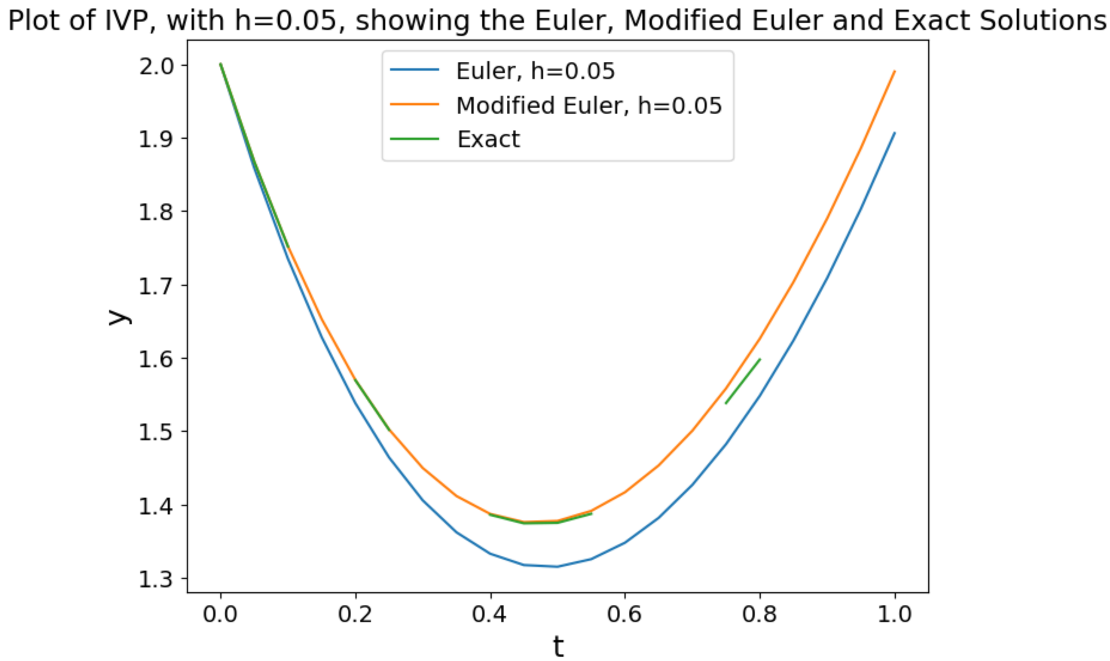
$$y(t) = 2 - 2\sqrt{2}t + \frac{7t^2}{2} - \frac{5\sqrt{2}t^3}{12} - \frac{5t^4}{24} + \frac{\sqrt{2}t^5}{64} + O\left(t^6\right)$$

Part d

[17]:
```
import math
from pandas import DataFrame

def timesteps(start, stop, h):
    num_steps = math.ceil((stop - start)/h)
    return np.linspace(start, start+num_steps*h, num_steps+1)
def Euler_step(func, start, stop, h, ics):
    times = timesteps(start, stop, h)
    values = ode_Euler(func, times, ics)
    title = ('Euler')
    return (values, times, title)
def Modified_Euler_step(func, start, stop, h, ics):
    times = timesteps(start, stop, h)
    title = ('Modified Euler')
    values = ModifiedEuler(func, times, ics)
    return (values, times, title)
def produce_df(method, vectorField, start, stop, h, ics):
    values, times, this_title = method(vectorField, start, stop, h, ics)
    return DataFrame(data = values, index = times, columns = [this_title +",␣
↪h="+str(h)])
```

[18]:
```
#part d
def eq8(t):
    return (2 - (2*((2**0.5))*t) + (3.5*( **2)) - (((5/12)*(2**0.5))*(t**3)) -␣
↪((5/24)*(t**4)) + ((2**0.5)/64*(t**5)))

df1 = produce_df(Euler_step, eq3_dy_dt, 0, 1, 0.05, 2)
df2 = produce_df(Modified_Euler_step, eq3_dy_dt, 0, 1, 0.05, 2)
df3 = DataFrame(data = [eq8(t) for t in timesteps(0,1,0.05)],
                        index = np.round(timesteps(0,1,0.05),3),
                        columns = ["Exact"])
plt.figure(figsize=(8, 6))
plt.rcParams['font.size'] = '14'
ax = plt.gca()
ax.set_xlabel('t',fontsize=18)
ax.set_ylabel('y',fontsize=18)
table_solutions = df1.join([df2, df3])
table_solutions.plot(ax=ax)
```

```
plt.title('Plot of IVP, with h=0.05, showing the Euler, Modified Euler and␣
 ↪Exact Solutions')
plt.show()
```



[19]:
```
#partd
print('Table of solutions for h = 0.05')
ts=[0,0.1,0.2,0.3,0.4,0.5,1.0]
table_solutions.filter(items=ts, axis=0)
```

```
Table of solutions for h = 0.05
```

[19]:

	Euler, h=0.05	Modified Euler, h=0.05	Exact
0.0	2.000000	2.000000	2.000000
0.1	1.734749	1.751741	1.751547
0.2	1.537944	1.569657	1.569274
0.4	1.332687	1.386962	1.385810
0.5	1.314973	1.377377	1.374799
1.0	1.906060	1.990196	1.896081

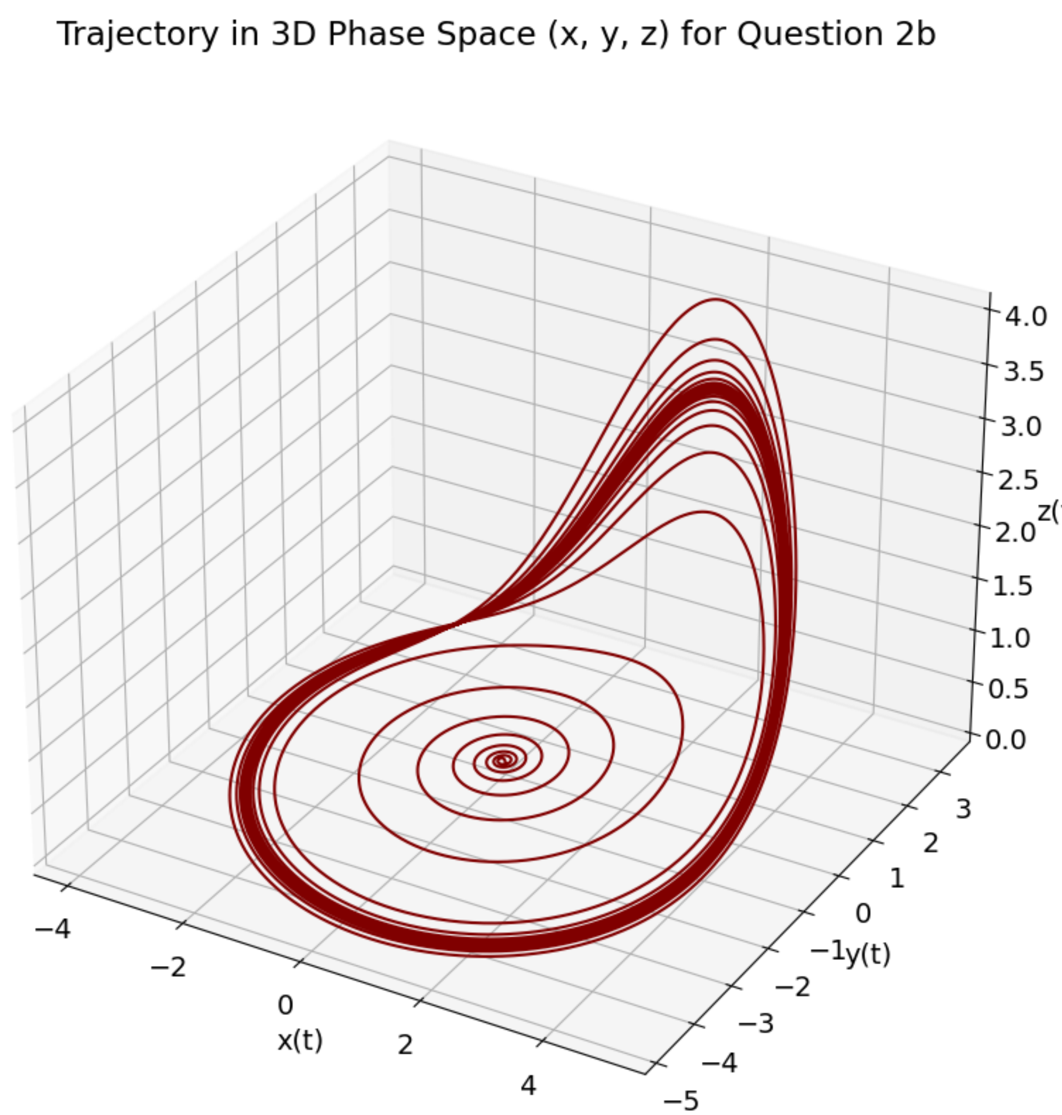
3.4 Question 4

Part a

[20]:
```
x = sym.Function('x')
y = sym.Function('y')
z = sym.Function('z')
```

```python
eq1 = sym.Eq(x(t).diff(t), -y(t)-z(t))
eq2 = sym.Eq(y(t).diff(t), x(t)+y(t)*(1/5) )
eq3 = sym.Eq(z(t).diff(t), 1/5 + (x(t) - 5/2)*z(t) )
```

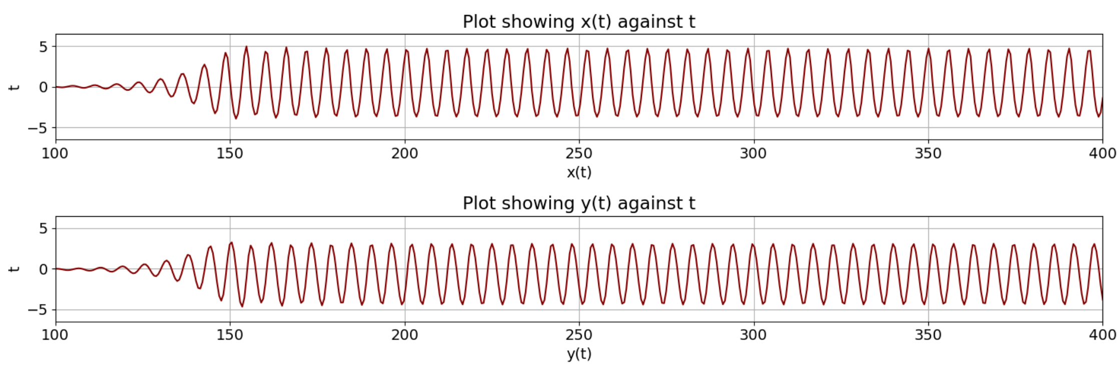
[21]:
```python
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import numpy as np

x = sym.symbols('x')
y = sym.symbols('y')
z = sym.symbols('z')

def vField(x,t):
    u = -x[1] - x[2]
    v = x[0] + (1/5)*x[1]
    w = 1/5 + (x[0] - 5/2)*x[2]
    return [u,v,w]

fig = plt.figure(figsize=(10, 10))
ax = plt.axes(projection='3d')

t=np.linspace(0, 1000, 90000)
x0, y0, z0 = 0,0,0
x = odeint(vField,[x0,y0,z0],t)
ax.plot3D(x[:,0],x[:,1],x[:,2], 'maroon');

ax.set_title('Trajectory in 3D Phase Space (x, y, z) for Question 2b')

ax.set_xlabel('x(t)')
ax.set_ylabel('y(t)')
ax.set_zlabel('z(t)')


plt.show()
```



Part b

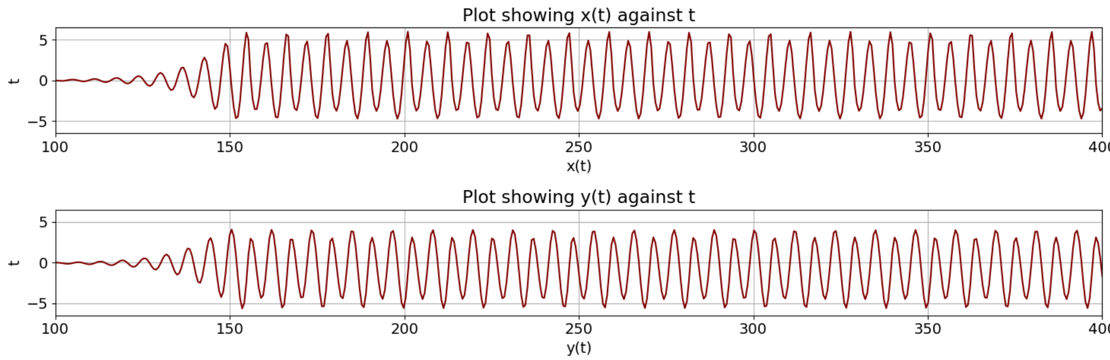
[22]:
```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import numpy as np

x = sym.symbols('x')
y = sym.symbols('y')
z = sym.symbols('z')

def vField(x,t):
    u = -x[1] - x[2]
```

```
    v = x[0] + (1/5)*x[1]
    w = 1/5 + (x[0] - 5/2)*x[2]
    return [u,v,w]



fig, ax = plt.subplots(2, 1, figsize=(15, 5))

t=np.linspace(100, 400, 500)
x0, y0, z0 = 0,0,0
x = odeint(vField,[x0,y0,z0],t)

ax[0].plot(t, x[:,0], 'maroon')

ax[1].plot(t, x[:,1], 'maroon')

ax[0].set_ylim(-6.5, 6.5)
ax[1].set_ylim(-6.5, 6.5)
ax[0].set_xlim(100, 400)
ax[1].set_xlim(100, 400)
ax[0].set_xlabel('x(t)')
ax[1].set_xlabel('y(t)')
ax[0].set_ylabel('t')
ax[1].set_ylabel('t')
ax[0].grid(True)
ax[1].grid(True)
ax[0].set_title('Plot showing x(t) against t')
ax[1].set_title('Plot showing y(t) against t')
fig.tight_layout()


plt.show()
```



Part c

[23]:
```python
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import numpy as np

x = sym.symbols('x')
y = sym.symbols('y')
z = sym.symbols('z')

def vField(x,t):
    u = -x[1] - x[2]
    v = x[0] + (1/5)*x[1]
    w = 1/5 + (x[0] - 3)*x[2]
    return [u,v,w]



fig, ax = plt.subplots(2, 1, figsize=(15, 5))

t=np.linspace(100, 400, 500)
x0, y0, z0 = 0,0,0
x = odeint(vField,[x0,y0,z0],t)

ax[0].plot(t, x[:,0], 'maroon')

ax[1].plot(t, x[:,1], 'maroon')

ax[0].set_ylim(-6.5, 6.5)
ax[1].set_ylim(-6.5, 6.5)
ax[0].set_xlim(100, 400)
ax[1].set_xlim(100, 400)
ax[0].set_xlabel('x(t)')
ax[1].set_xlabel('y(t)')
ax[0].set_ylabel('t')
ax[1].set_ylabel('t')
ax[0].grid(True)
ax[1].grid(True)
ax[0].set_title('Plot showing x(t) against t')
ax[1].set_title('Plot showing y(t) against t')
fig.tight_layout()


plt.show()
```



When the co-efficient 5/2 is replaced with 3, every other peak has increased and every other trough is decreased. As with the co-efficient 5/2 the oscillations are approximately at the same value continuously. The coefficient 3 changes the values of t produced.