3.3 Homoclinic bifurcation

Consider the dynamical system

$$\dot{x} = \mu x + y - x^2$$
$$\dot{y} = -x + \mu y + 2x^2$$

where μ is a dimensionless parameter.

a) Using a numerical method of your choice, find the value $\mu = \mu_c$ where the system undergoes a homoclinic bifurcation. Give your result with two significant digits.

```
In[ ]:= x = (1 + u^2)/(u + 2)
        y = (3((1+u^2)/(2+u))^2-u((1+u^2)/(2+u))-(1+u^2)/(2+u))/(1-u)//Simplify
```

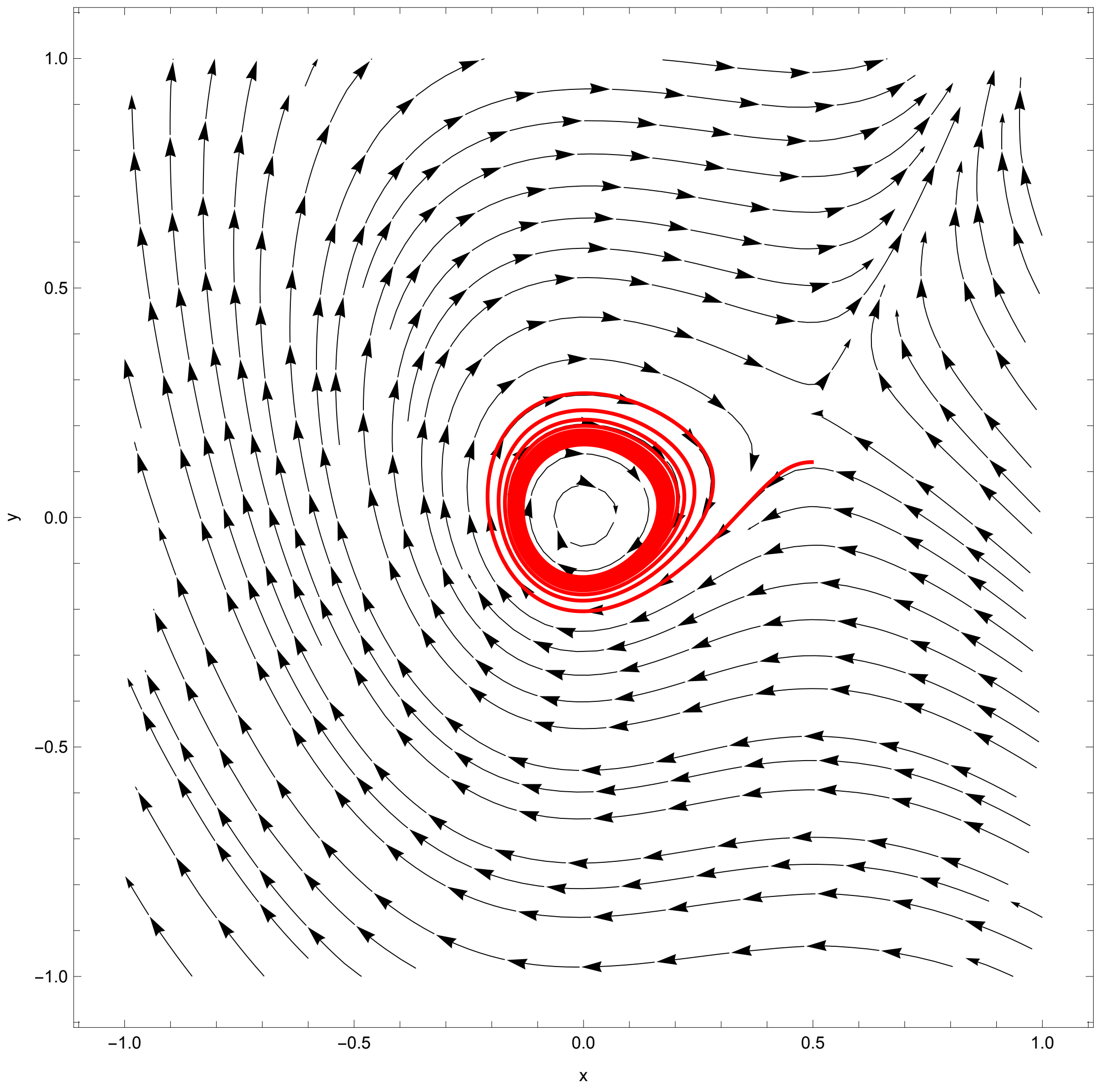
Out[ ]=
$$\frac{1+u^2}{2+u}$$

Out[ ]=
$$\frac{1-2u+u^2-2u^3}{(2+u)^2}$$

Starting off by finding the fixed points, fixed point 1 and 2. One can see that fixed point 2 is only dependent on μ. Then using the code from OpenTA 2.3 to plot this system for arbitrary μ and looking at the behaviour of the fixed points. One can then easily see that fixed point 2 is the one which undergoes a homoclinic bifurcation. With the knowledge, that in the case of a homoclinic bifurcation the starting position (fixed point 2 $\pm\epsilon$) must again end up in fixed point 2, one can leverage this to calculate the euclidian distance between starting position and end position. With this method it is possible to iteratively test for different μ which minimize the distance between the two points. This has not been implemented in Mathematica, but in Python due to lack of coding skills in Mathematica (Python code included below).

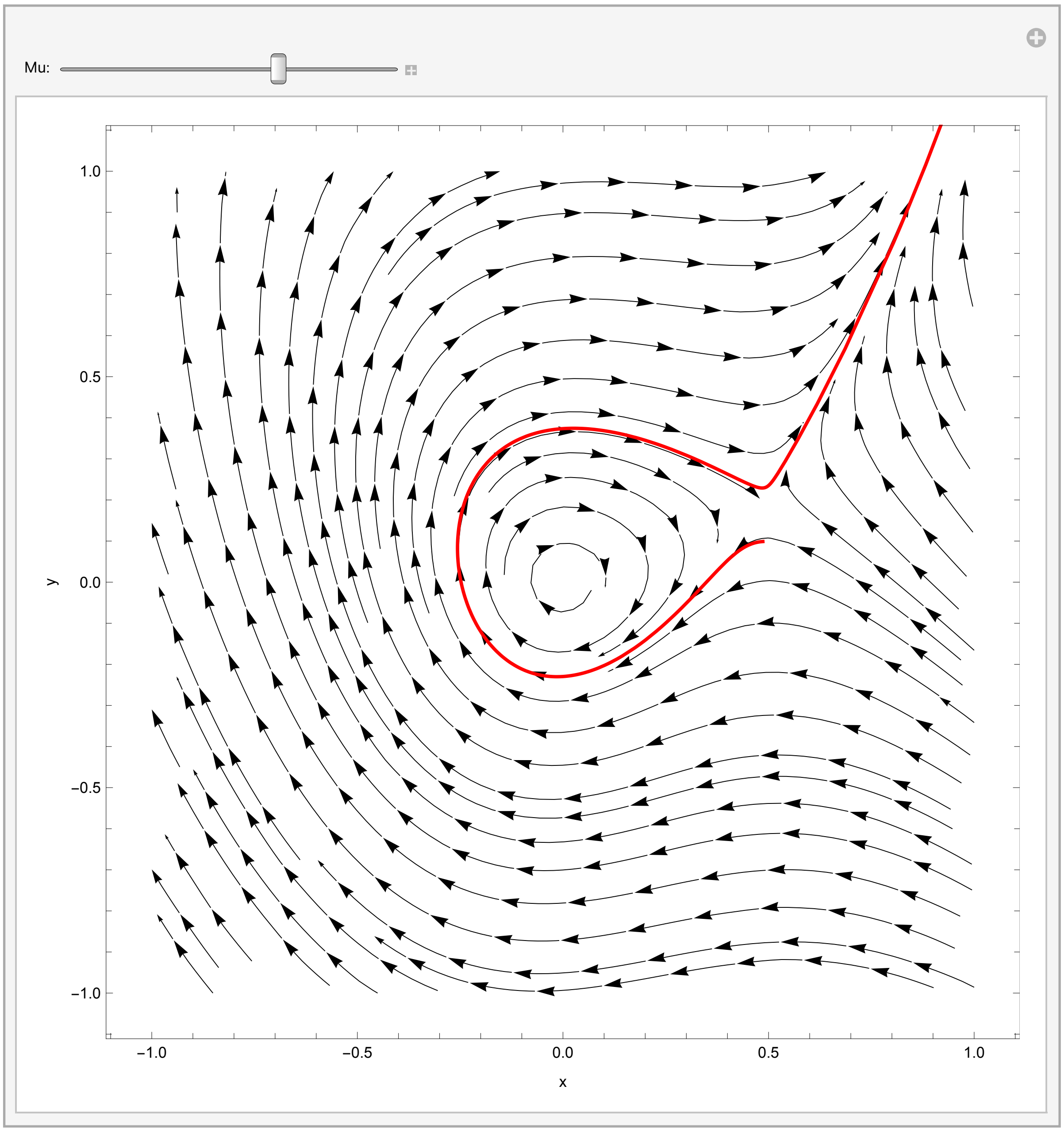
In[ ]:=
```
ClearAll["Global`*"]

u = 0.01;

Equation1 = x'[t] == u*x[t] + y[t] - x[t]^2;
Equation2 = y'[t] == -x[t] + u*y[t] + 2*x[t]^2;

System1 = {Equation1, Equation2};

FixPoint1 = {0,0};
FixPoint2 = {(1 + u^2)/(u + 2), (1-2u+u^2-2u^3)/(2+u)^3} // Expand // Simplify;

t0 = 0;
tMax = 100;

StartingPoint1 = {x[0] == FixPoint2⟦1⟧-0.00001, y[0] == FixPoint2⟦2⟧-0.00001};

Solution = NDSolve[{System1, StartingPoint1}, {x, y}, {t, t0, tMax}];

startT = 0;
endT = 100;

StartingPosition = {xValue1, yValue1} = {x[startT], y[startT]} /. Solution⟦1⟧;
EndPosition = {xValue, yValue} = {x[endT], y[endT]} /. Solution⟦1⟧;
EuclideanDistance[EndPosition,StartingPosition];



StreamPlot10 = StreamPlot[{u*x + y - x^2, -x + u*y + 2*x^2}, {x, -1, 1}, {y, -1, 1},
   PlotRange → All, ImageSize → Large, ColorFunction → (Black &),
   StreamStyle → Directive[Black], StreamColorFunction → None];

TrajectoryPlot10 = ParametricPlot[Evaluate[{x[t], y[t]} /. Solution], {t, t0, tMax}, P

Show[StreamPlot10, TrajectoryPlot10, FrameLabel → {"x", "y"}]
```

Out[ ]=



In[ ]:=
```
Manipulate[
  Module[{u, Equation1, Equation2, System1, FixPoint2, t0, tMax, StartingPoint1, Solut
    StartingPosition, EndPosition, StreamPlot10, TrajectoryPlot10},

    u = mu;

    Equation1 = x'[t] == u*x[t] + y[t] - x[t]^2;
    Equation2 = y'[t] == -x[t] + u*y[t] + 2*x[t]^2;
    System1 = {Equation1, Equation2};

    FixPoint2 = {(1 + u^2)/(u + 2), (1-2u+u^2-2u^3)/(2+u)^3} // Expand // Simplify;

    t0 = 0;
    tMax = 30;

    StartingPoint1 = {x[0] == FixPoint2⟦1⟧ - 0.00001, y[0] == FixPoint2⟦2⟧ - 0.00001};

    Solution = NDSolve[{System1, StartingPoint1}, {x, y}, {t, t0, tMax}];

    startT = 0;
    endT = 30;

    StartingPosition = {xValue1, yValue1} = {x[startT], y[startT]} /. Solution⟦1⟧;
    EndPosition = {xValue, yValue} = {x[endT], y[endT]} /. Solution⟦1⟧;

    StreamPlot10 = StreamPlot[{u*x + y - x^2, -x + u*y + 2*x^2}, {x, -1, 1}, {y, -1, 1
      PlotRange → All, ImageSize → Large, ColorFunction → (Black &),
      StreamStyle → Directive[Black], StreamColorFunction → None];

    TrajectoryPlot10 = ParametricPlot[Evaluate[{x[t], y[t]} /. Solution], {t, t0, tMax

    Show[StreamPlot10, TrajectoryPlot10, FrameLabel → {"x", "y"}]
  ],

  {{mu, 0.066, "Mu:"}, 0.00, 0.1, 0.00001}
]
```

Out[ ]=



Python code to find μ that minimises the euclidian distance between the starting point and end point close to the fixed point 2.

In[ ]:=
```
import numpy as np
from scipy.integrate import solve_ivp
import tqdm


u_values = np.arange(0.05, 0.1, 0.00001)

t_span = [0, 1000]
initial_condition_offset = np.array([0.00001, 0.00001])
tolerance = 0.0001

def system(t, z, u):
    x, y = z
    dxdt = u * x + y - x**2
    dydt = -x + u * y + 2 * x**2
    return [dxdt, dydt]

prev_solution = None

for u in tqdm.tqdm(u_values):

    fix_point = [(1 + u**2)/(u + 2), (1-2*u+u**2-2*u**3)/((2+u)**2)]

    if prev_solution is not None:
        initial_guess = prev_solution.y[:, -1]
    else:
        initial_guess = fix_point - initial_condition_offset

    solution = solve_ivp(system, t_span, initial_guess, args=(u,), dense_output=True)

    prev_solution = solution

    start_position = solution.sol(0)
    end_position = solution.sol(100)
    distance = np.linalg.norm(end_position - start_position)

    if distance < tolerance:
        print(f"Euclidean Distance is smaller than {tolerance} for u = {u}")
        break
```

b) Plot the phase portraits that occur for the cases $\mu < 0$, $\mu = 0$, $0 < \mu < \mu_c$, $\mu = \mu_c$ and $\mu > \mu_c$, marking and labelling fixed points, closed orbits, limit cycles and homoclinic orbits. Use a numerical solver, for example NDSolve[] in Mathematica (using StreamPlot[] will not give enough

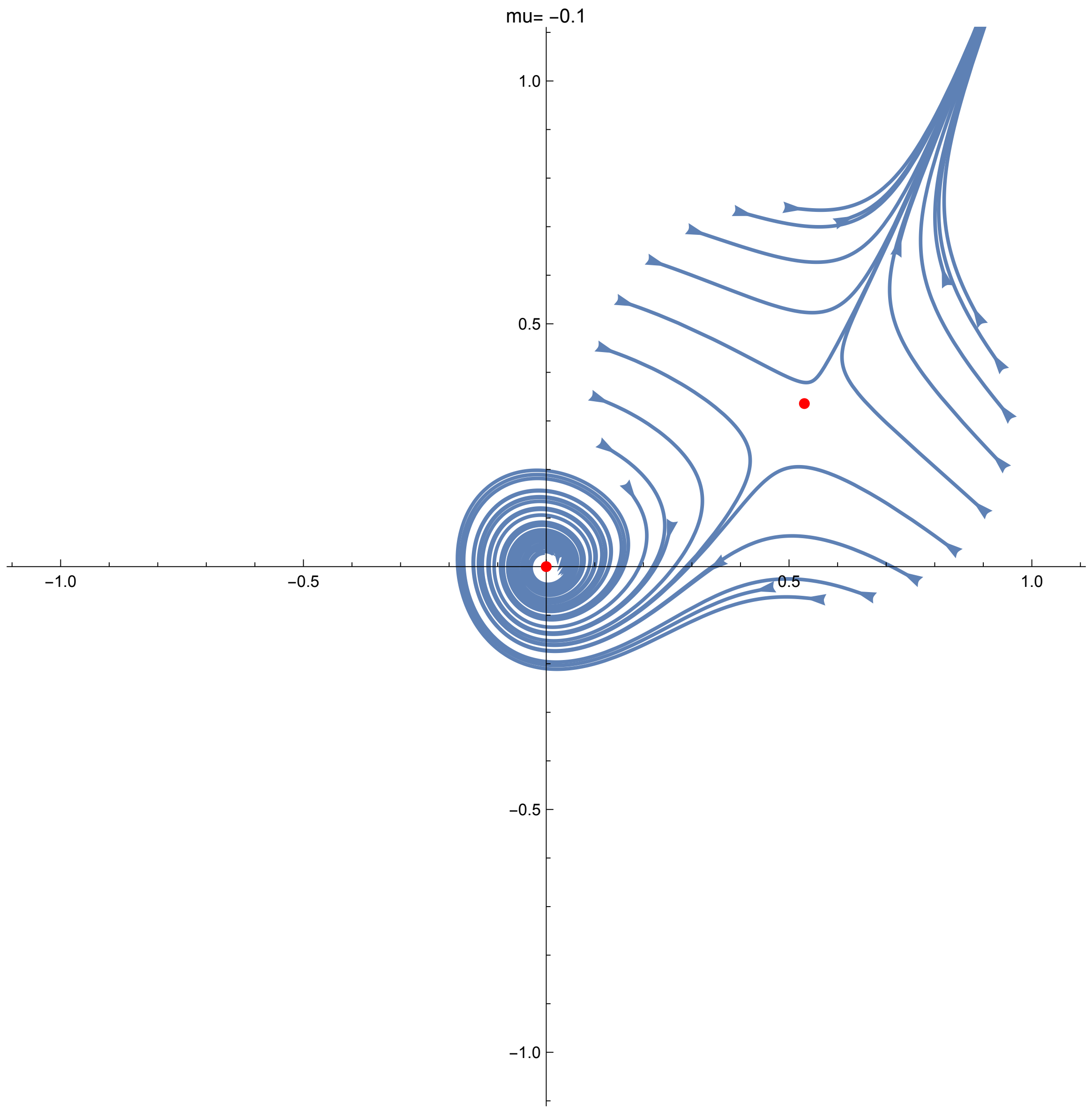
resolution for this task).

For this task explanations were used to each case.

For $\mu = -0.1$ we have a stable fixed point at (0,0) (stable spiral) and a saddle node at fixed point 2 which depends on μ.

In[ ]:=
```
ClearAll["Global`*"]

u = -0.1;

Equation1 = x'[t] == u*x[t] + y[t] - x[t]^2;
Equation2 = y'[t] == -x[t] + u*y[t] + 2*x[t]^2;

System1 = {Equation1, Equation2};

FixPoint1 = {0, 0};
FixPoint2 = {(1 + u^2)/(u + 2), (1-2*u+u^2-2*u^3)/((2+u)^2)} // Expand // Simplify;

t0 = 0;
tMax = 100;

(* Create 10 circles with different radii on which the initialPoints lie *)
nrOfInitialPoints = 1;
radiusMin = 0.1;
radiusMax = 1;

initialPoints = Table[
   {x[0] == FixPoint1[[1]] + radius*Cos[2 Pi k/nrOfInitialPoints],
    y[0] == FixPoint1[[2]] + radius*Sin[2 Pi k/nrOfInitialPoints]},
   {radius, Range[radiusMin, radiusMax, 1]},
   {k, nrOfInitialPoints}
];

(* Flatten the list to get a single list of initial points *)
initialPoints = Flatten[initialPoints, 1];

(* Use NDSolve for each initial point and store the solutions in a list *)
solutions = Table[NDSolve[{System1, initialPoints[[i]]}, {x, y}, {t, t0, tMax}], {i, Len

(* Plot the trajectories for each solution with arrows indicating the direction *)
TrajectoryPlotArrows = Table[
    ParametricPlot[Evaluate[{x[t], y[t]} /. solution], {t, t0, tMax},
    PlotStyle → style] /. Line[x_] :→ {Arrowheads[{0.02, 0.02}], Arrow[x]},{solution, s

(* Assuming x and y are the variables you want to use for the plot range *)
Show[TrajectoryPlotArrows,
  FrameLabel → {"x", "y"}, PlotRange → {{-1,1},{-1,1}}, PlotLabel → "mu= -0.1", ImageS
```

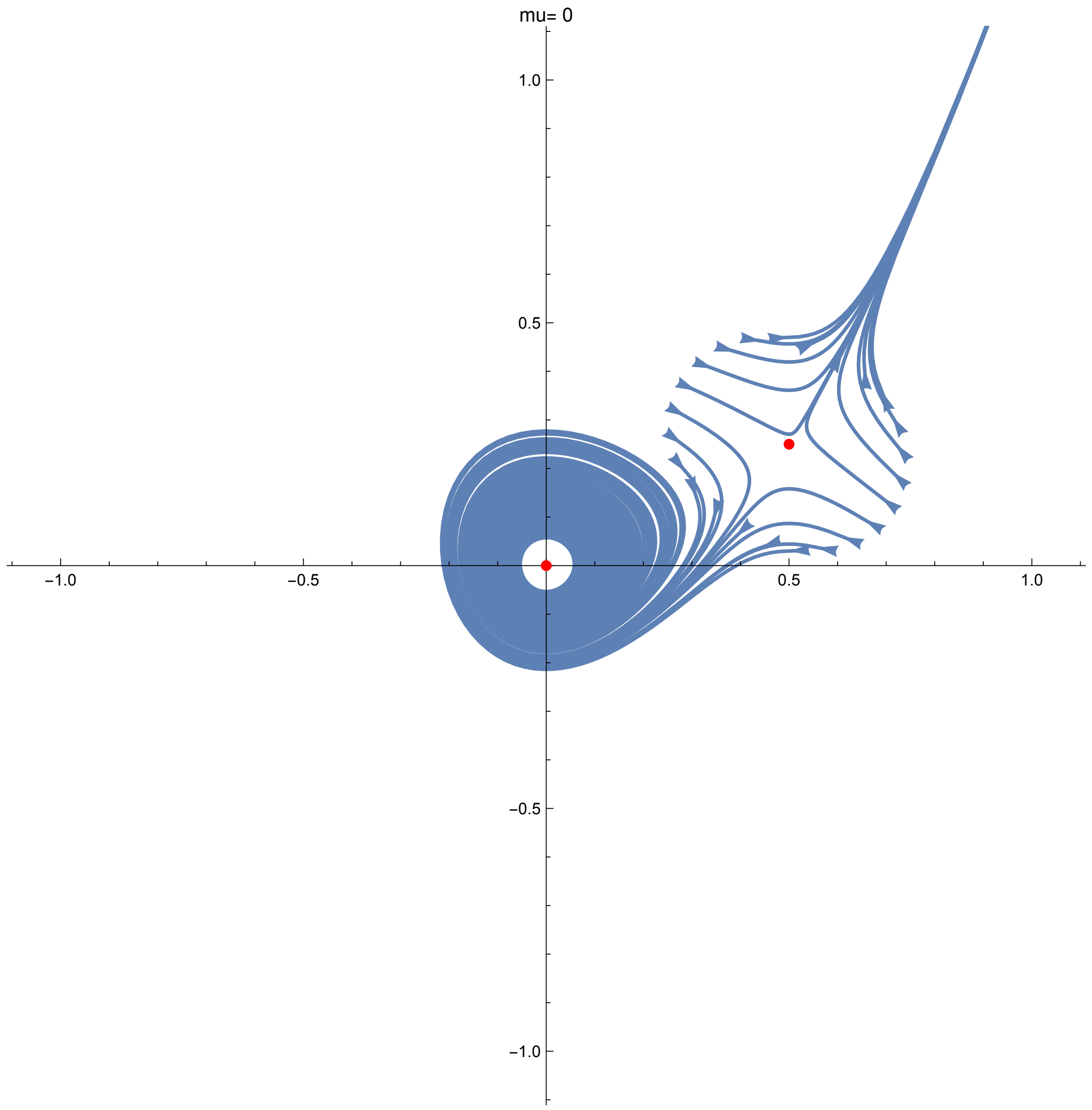
Out[ ]=



This has a stable fixed point at (0,0) (stable spiral) and a saddle point at fixed point 2 which depends on μ.

In[ ]:=
```
ClearAll["Global`*"]

u = 0;

Equation1 = x'[t] == u*x[t] + y[t] - x[t]^2;
Equation2 = y'[t] == -x[t] + u*y[t] + 2*x[t]^2;

System1 = {Equation1, Equation2};

FixPoint1 = {0, 0};
FixPoint2 = {(1 + u^2)/(u + 2), (1-2*u+u^2-2*u^3)/((2+u)^2)} // Expand // Simplify;

t0 = 0;
tMax = 300;

(* Create 10 circles with different radii on which the initialPoints lie *)
nrOfInitialPoints = 24;
radiusMin = 0.22;
radiusMax = 0.5;

initialPoints = Table[
   {x[0] == FixPoint2[[1]] + radius*Cos[2 Pi k/nrOfInitialPoints],
    y[0] == FixPoint2[[2]] + radius*Sin[2 Pi k/nrOfInitialPoints]},
   {radius, Range[radiusMin, radiusMax, 1]},
   {k, nrOfInitialPoints}
];

(* Flatten the list to get a single list of initial points *)
initialPoints = Flatten[initialPoints, 1];

(* Use NDSolve for each initial point and store the solutions in a list *)
solutions = Table[NDSolve[{System1, initialPoints[[i]]}, {x, y}, {t, t0, tMax}], {i, Len

(* Plot the trajectories for each solution with arrows indicating the direction *)
TrajectoryPlotArrows = Table[
    ParametricPlot[Evaluate[{x[t], y[t]} /. solution], {t, t0, tMax},
    PlotStyle → style] /. Line[x_] :→ {Arrowheads[{0.02, 0.0}], Arrow[x]},{solution, so

(* Assuming x and y are the variables you want to use for the plot range *)
Show[TrajectoryPlotArrows, PlotRange → (*Full*){{-1, 1}, {-1, 1}},
  FrameLabel → {"x", "y"}, PlotLabel → "mu= 0", ImageSize → 600, Epilog → {Red, Point


  (*This plot has a limit cycle around (0,0), *)
```

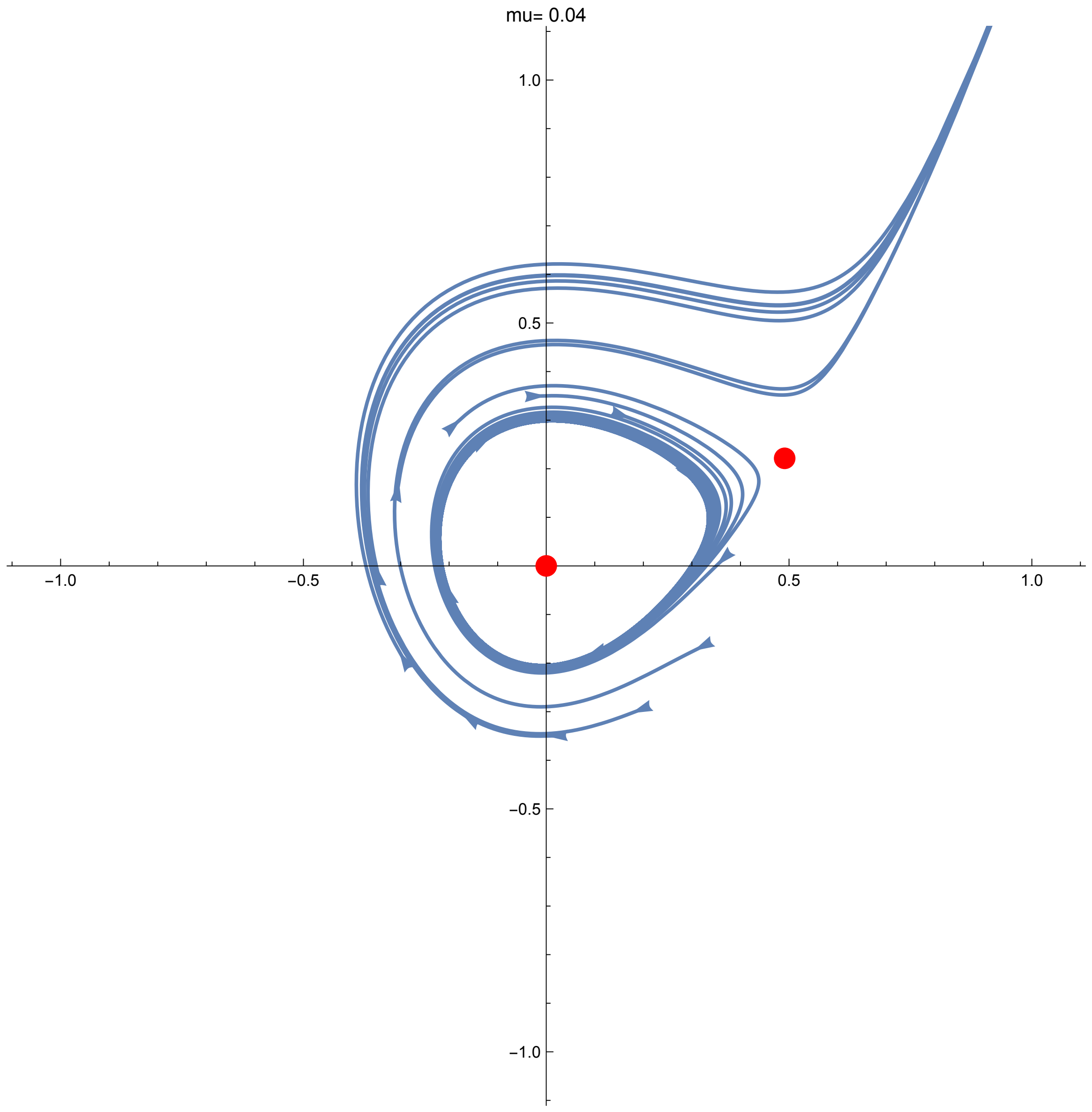
Out[ ]=



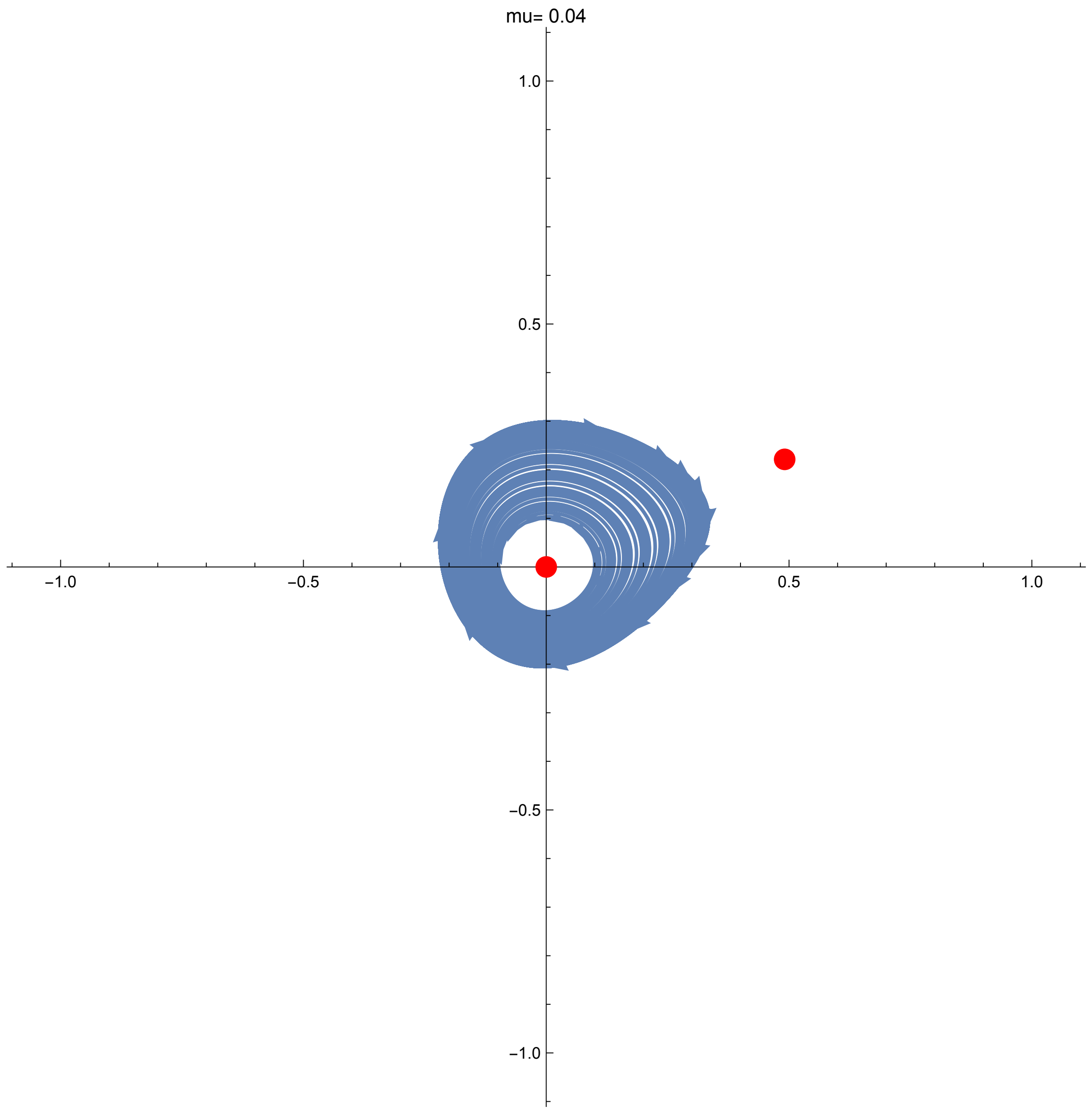
At (0,0) there is a unstable fixed point (unstable spiral). Around it there is stable limit cycle that traps the trajectories on the inside and limits the trajectories to the limit cycle.

In[ ]:=
```
ClearAll["Global`*"]

u = 0.04;

Equation1 = x'[t] == u*x[t] + y[t] - x[t]^2;
Equation2 = y'[t] == -x[t] + u*y[t] + 2*x[t]^2;

System1 = {Equation1, Equation2};

FixPoint1 = {0, 0};
FixPoint2 = {(1 + u^2)/(u + 2), (1-2*u+u^2-2*u^3)/((2+u)^2)} // Expand // Simplify;

t0 = 0;
tMax = 100;

(* Create 10 circles with different radii on which the initialPoints lie *)
nrOfInitialPoints = 12;
radiusMin = 0.35;
radiusMax = 1;

initialPoints = Table[
   {x[0] == FixPoint1[[1]] + radius*Cos[2 Pi k/nrOfInitialPoints],
    y[0] == FixPoint1[[2]] + radius*Sin[2 Pi k/nrOfInitialPoints]},
   {radius, Range[radiusMin, radiusMax, 1]},
   {k, nrOfInitialPoints}
];

(* Flatten the list to get a single list of initial points *)
initialPoints = Flatten[initialPoints, 1];

(* Use NDSolve for each initial point and store the solutions in a list *)
solutions = Table[NDSolve[{System1, initialPoints[[i]]}, {x, y}, {t, t0, tMax}], {i, Len

(* Plot the trajectories for each solution with arrows indicating the direction *)
TrajectoryPlotArrows = Table[
    ParametricPlot[Evaluate[{x[t], y[t]} /. solution], {t, t0, tMax},
    PlotStyle → style] /. Line[x_] :→ {Arrowheads[{0.02, 0.02}], Arrow[x]},{solution, s

(* Assuming x and y are the variables you want to use for the plot range *)
Show[TrajectoryPlotArrows, PlotRange → {{-1, 1}, {-1, 1}},
  FrameLabel → {"x", "y"}, PlotLabel → "mu= 0.04", ImageSize → 600, Epilog → {Red, Po
```

Out[ ]=



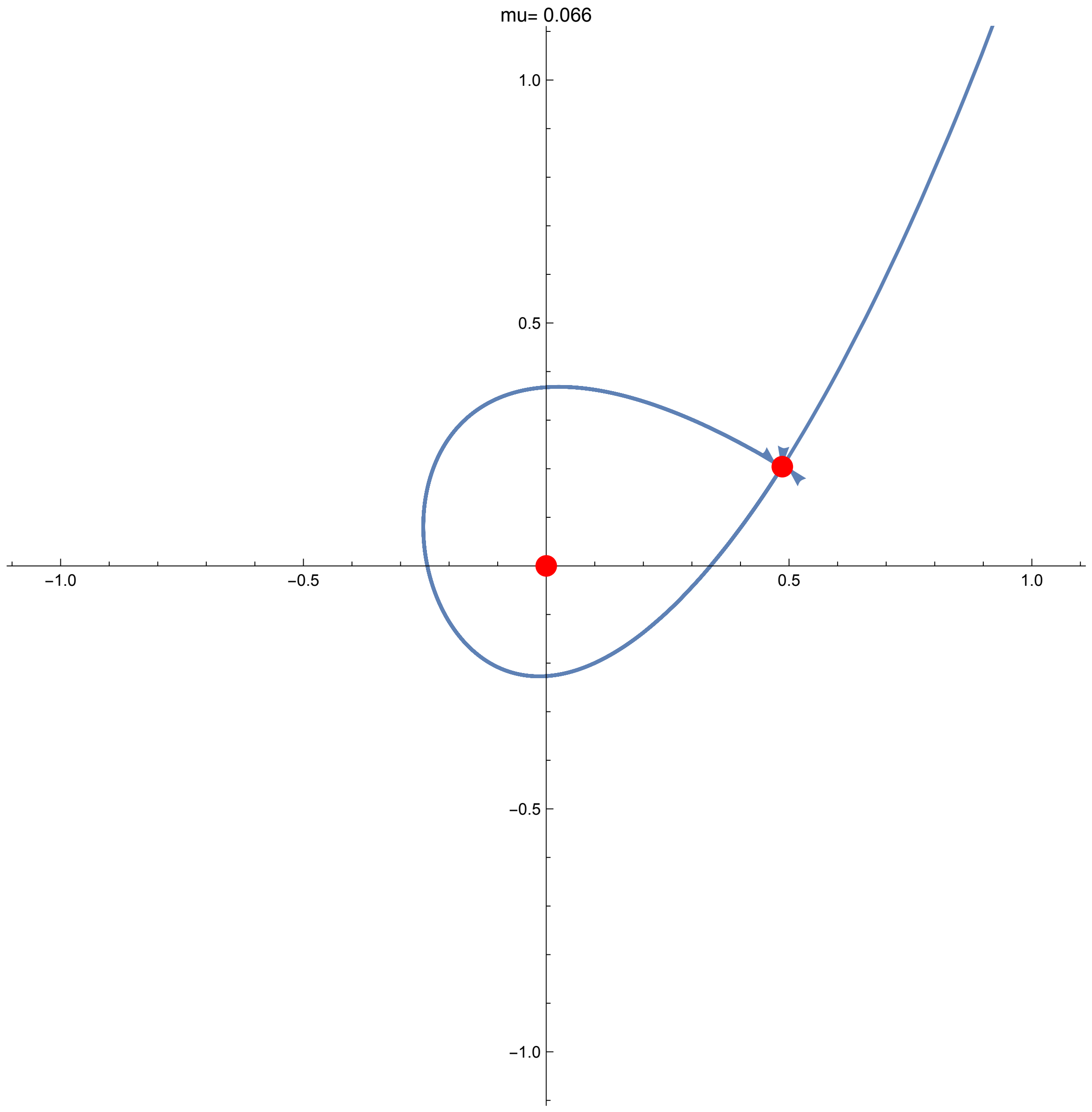
Out[ ]=



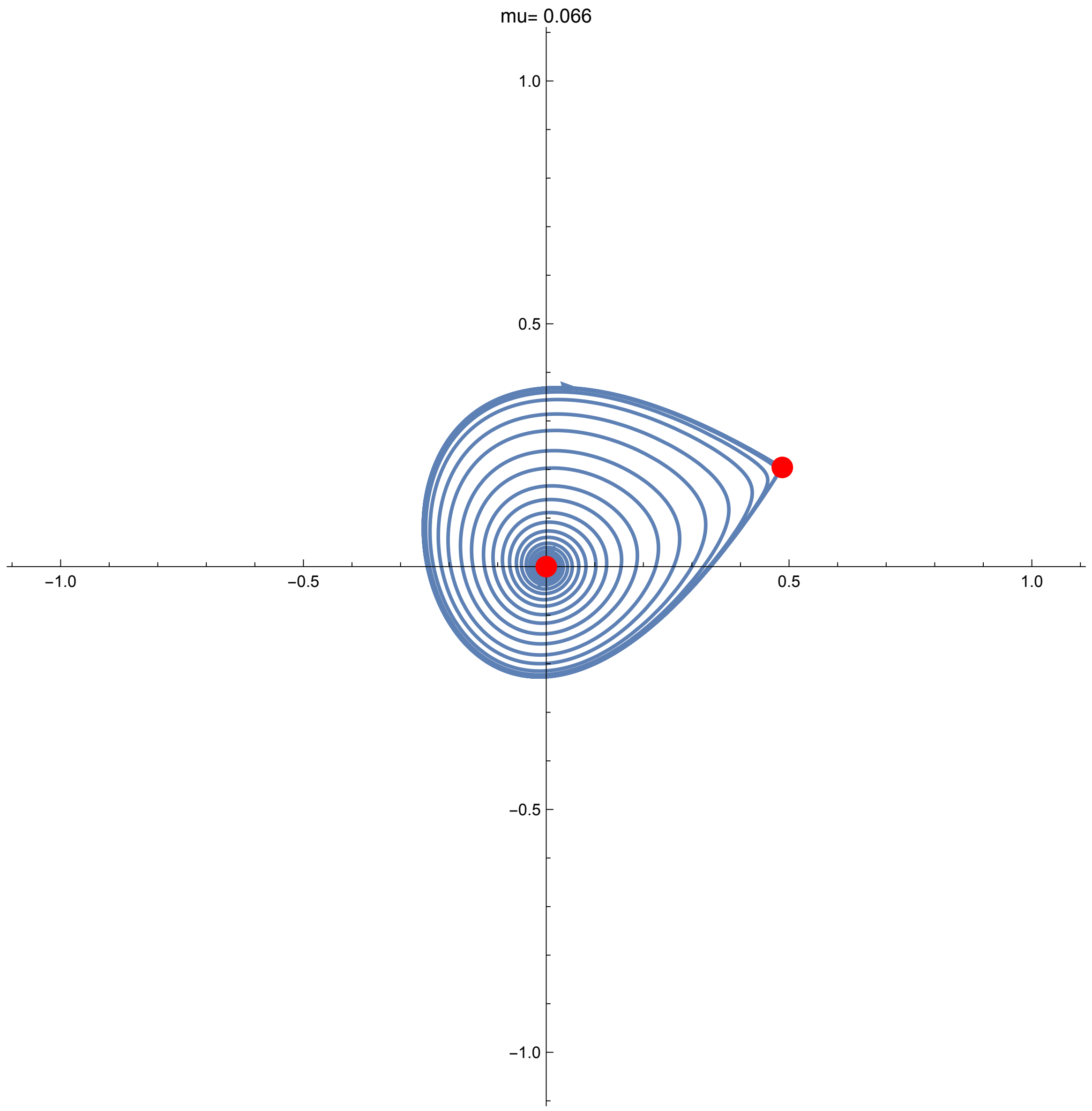
Trajectories (unstable spiral) around unstable fixed point 1 at (0,0) are trapped by the homoclinic orbit as can be seen in the plot. Also one can see that the manifold returns into the fixed point 2 as a homoclinic should.

In[ ]:=
```
ClearAll["Global`*"]

u = 0.066;

Equation1 = x'[t] == u*x[t] + y[t] - x[t]^2;
Equation2 = y'[t] == -x[t] + u*y[t] + 2*x[t]^2;

System1 = {Equation1, Equation2};

FixPoint1 = {0, 0};
FixPoint2 = {(1 + u^2)/(u + 2), (1-2*u+u^2-2*u^3)/((2+u)^2)} // Expand // Simplify;

t0 = 0;
tMax = 100;

(* Create 10 circles with different radii on which the initialPoints lie *)
nrOfInitialPoints = 2;
radiusMin = 0.01;
radiusMax = 0.3;

initialPoints = Table[
   {x[0] == FixPoint2[[1]] + radius*Cos[2 Pi k/nrOfInitialPoints],
    y[0] == FixPoint2[[2]] + radius*Sin[2 Pi k/nrOfInitialPoints]},
   {radius, Range[radiusMin, radiusMax, 1]},
   {k, nrOfInitialPoints}
];

(* Flatten the list to get a single list of initial points *)
initialPoints = Flatten[initialPoints, 1];

(* Use NDSolve for each initial point and store the solutions in a list *)
solutions = Table[NDSolve[{System1, initialPoints[[i]]}, {x, y}, {t, t0, tMax}], {i, Len

(* Plot the trajectories for each solution with arrows indicating the direction *)
TrajectoryPlotArrows = Table[
    ParametricPlot[Evaluate[{x[t], y[t]} /. solution], {t, t0, tMax},
    PlotStyle → style] /. Line[x_] :→ {Arrowheads[{0.02, 0.02}], Arrow[x]},{solution, s

(* Assuming x and y are the variables you want to use for the plot range *)
(* Assuming x and y are the variables you want to use for the plot range *)
Show[TrajectoryPlotArrows, PlotRange → {{-1, 1}, {-1, 1}},
  FrameLabel → {"x", "y"}, PlotLabel → "mu= 0.066", ImageSize → 600, Epilog → {Red, P
```

Out[ ]=



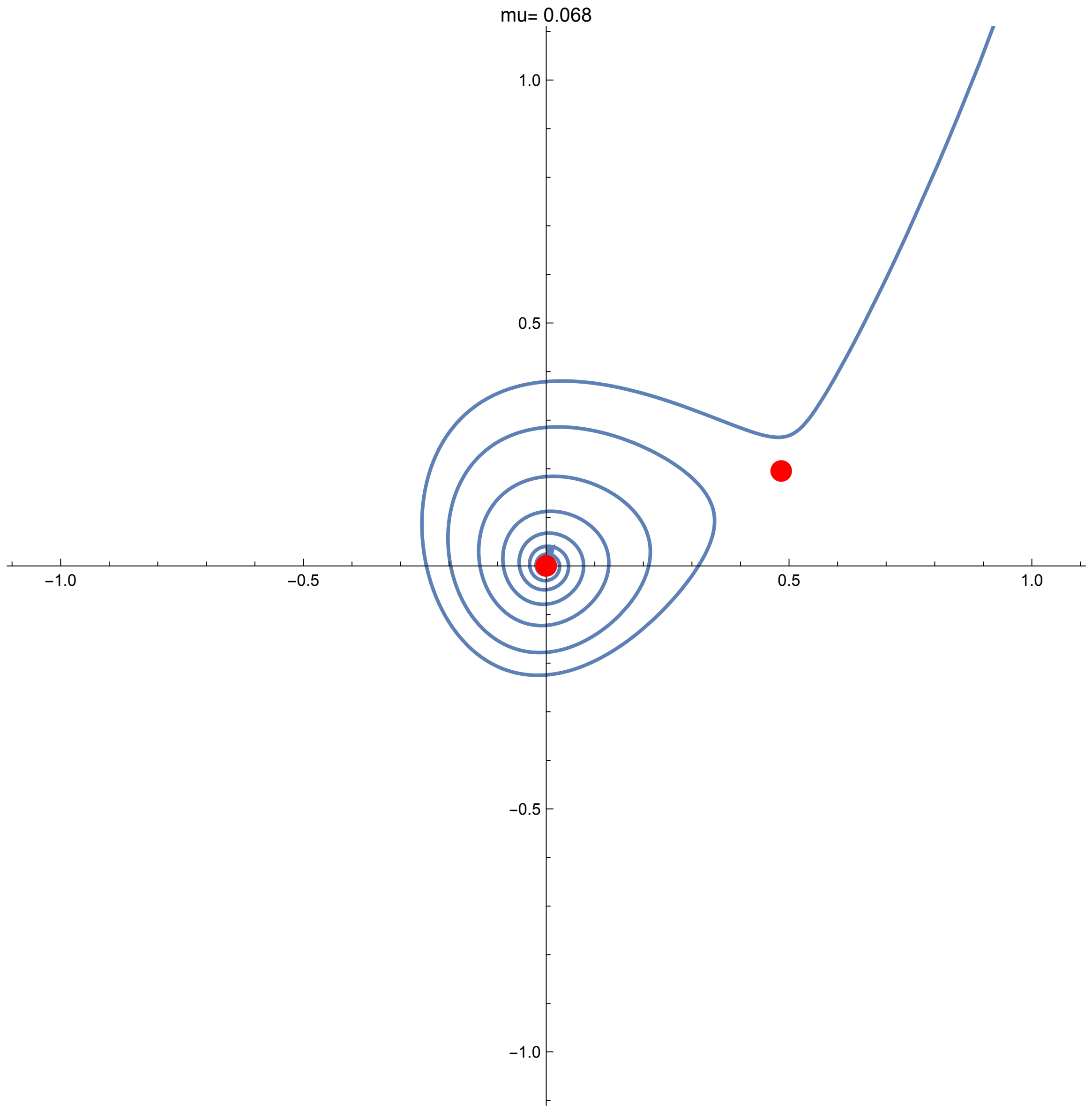
Out[ ]=



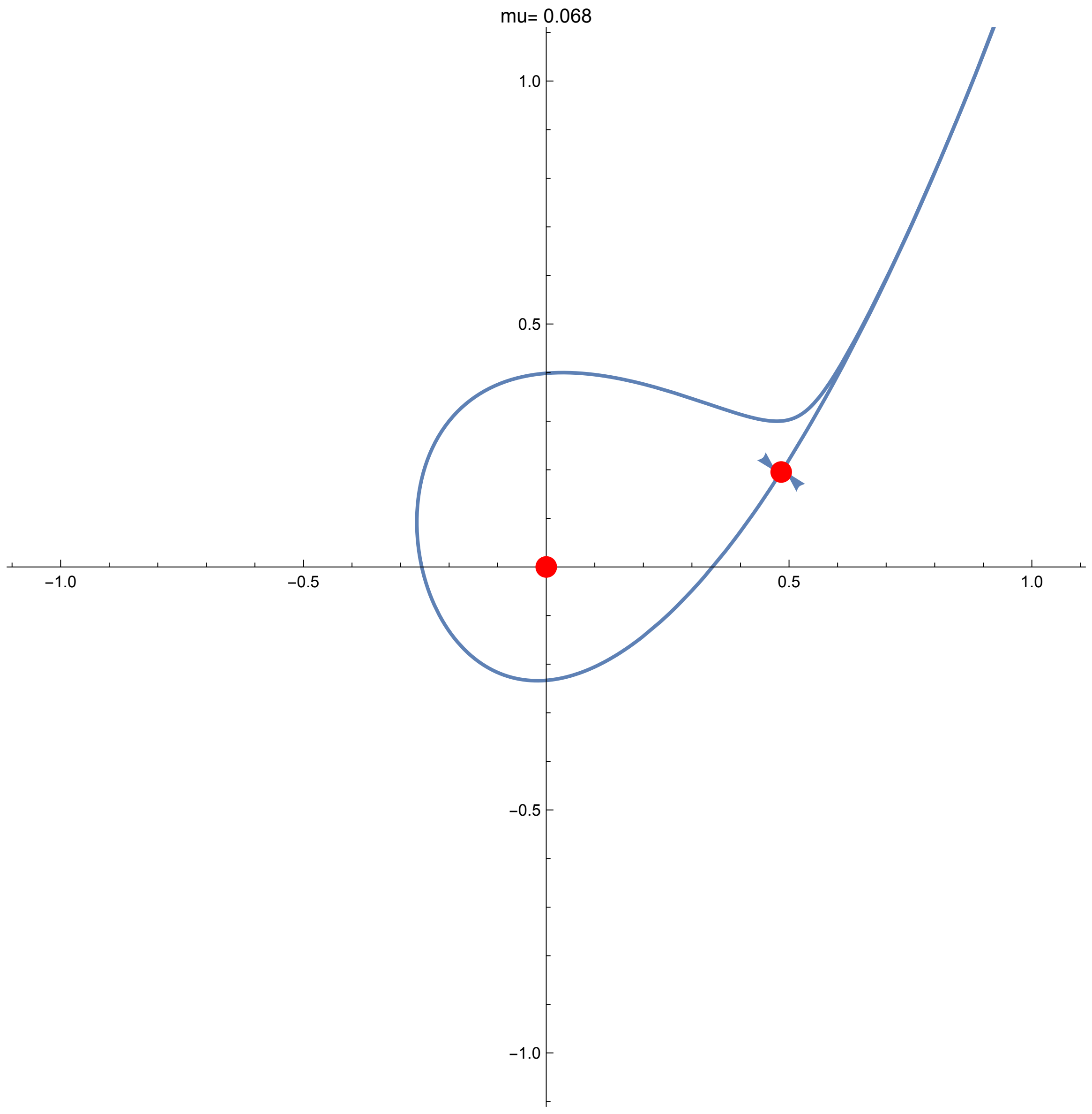
As one can see the manifold moves away from the saddle point at fixed point 2 which only depends on μ. At fixed point 1 (0,0) we have an unstable spiral.

In[ ]:=
```
ClearAll["Global`*"]

u = 0.08;

Equation1 = x'[t] == u*x[t] + y[t] - x[t]^2;
Equation2 = y'[t] == -x[t] + u*y[t] + 2*x[t]^2;

System1 = {Equation1, Equation2};

FixPoint1 = {0, 0};
FixPoint2 = {(1 + u^2)/(u + 2), (1-2*u+u^2-2*u^3)/((2+u)^2)} // Expand // Simplify

t0 = 0;
tMax = 60;

(* Create 10 circles with different radii on which the initialPoints lie *)
nrOfInitialPoints = 1;
radiusMin = 0.01;
radiusMax = 0.3;

initialPoints = Table[
   {x[0] == FixPoint1[[1]] + radius*Cos[2 Pi k/nrOfInitialPoints],
    y[0] == FixPoint1[[2]] + radius*Sin[2 Pi k/nrOfInitialPoints]},
   {radius, Range[radiusMin, radiusMax, 1]},
   {k, nrOfInitialPoints}
];


(* Flatten the list to get a single list of initial points *)
initialPoints = Flatten[initialPoints, 1];

(* Use NDSolve for each initial point and store the solutions in a list *)
solutions = Table[NDSolve[{System1, initialPoints[[i]]}, {x, y}, {t, t0, tMax}], {i, Len

(* Plot the trajectories for each solution with arrows indicating the direction *)
TrajectoryPlotArrows = Table[
    ParametricPlot[Evaluate[{x[t], y[t]} /. solution], {t, t0, tMax},
    PlotStyle → style] /. Line[x_] :→ {Arrowheads[{0.02, 0.02}], Arrow[x]},{solution, s

(* Assuming x and y are the variables you want to use for the plot range *)
Show[TrajectoryPlotArrows, PlotRange → {{-1, 1}, {-1, 1}},
  FrameLabel → {"x", "y"}, PlotLabel → "mu= 0.068", ImageSize → 600, Epilog → {Red, P
```

Out[ ]=



Out[ ]=



In order to find out how the period of a closed orbit scales as a homoclinic bifurcation is approached, it is useful to first estimate the time it takes for an orbit to pass a saddle point. To estimate the time to pass a generic saddle, consider the linearised dynamics

$$\dot{x} = u\,x$$

$$\dot{y} = s\, y$$

where u and s are the eigenvalues of the unstable and stable directions respectively, i.e. $u > 0$ and $s < 0$.
Now, let a trajectory start from the point

$$(x(0), y(0)) = (\gamma, 1)$$

where $\gamma > 0$ is small.

---

c) Find an analytical expression for the time t_1 to escape from the saddle to $x(t_1) = 1$.

Start by explicit the equation $\dot{x} = u\, x$ to

$$\frac{d\,x}{d\,t} = u\, x$$

Now use the Ansatz of separation of variables

$$\frac{d\,x}{x} = u\, d\, t$$

Now integrate the equivalent

$$\int_\gamma^1 \frac{1}{x}\, d\,x = \int_0^{t_1} u\, d\, t$$

which leaves you with

$$-\ln \gamma = t_1\, u$$

With basic rearrangement it follows that

$$t_1 = -\frac{\ln \gamma}{u}.$$

---

d) Find an analytical expression for u suitable for the system in Eq. (1).

Apply the same reasoning as in c) to the differential equation system (1). For this calculate the eigenvectors of the system and then put in the coordinates for a fixed point. Since we know that the $x(0) = \gamma > 0$, the previously calculated FixPoint1 = {0,0}={$x(0)$, $y(0)$} does not work, this leaves only

FixPoint2 = $\{(1+u^2)/(u+2), (1-2u+u^2-2u^3)/(2+u)^3\} = \{x(0), y(0)\}$.

Substitute the x in the eigenvalues with $(1+u^2)/(u+2)$ and then find which of the eigenvalues is the unstable one by comparing them. The unstable eigenvalue is the one which is larger than zero.

In[ ]:=
```
ClearAll["Global`*"]

FixPoint2 = {(1 + u^2)/(u + 2), (1-2u+u^2-2u^3)/(2+u)^3} // Expand // Simplify;
xCheck = FixPoint2[[1]];

JacobiA = {{u-2*x,1},{-1+4*x,u}};

EigVal = Eigenvalues[JacobiA];
EigVal /. x → xCheck // Simplify
```

Out[ ]=

$$\left\{u-\frac{1}{2+u}-\frac{u^2}{2+u}-\sqrt{\frac{5+9u^2+4u^3+u^4}{(2+u)^2}},\ u-\frac{1}{2+u}-\frac{u^2}{2+u}+\sqrt{\frac{5+9u^2+4u^3+u^4}{(2+u)^2}}\right\}$$

The time of the periodic orbit, T_μ, just below the homoclinic bifurcation where $\mu < \mu_c$ and $|\mu-\mu_c| \ll 1$ is well approximated by the time it takes to pass the saddle point, i.e. it depends only on u and γ. Hence, to find T_μ we first need to investigate how γ depends on $|\mu-\mu_c|$ for $|\mu-\mu_c| \ll 1$.

Numerically find this dependence. It should be given by the scaling law

$$\gamma \sim A\,|\mu-\mu_c|^a$$

for some constants a and A. Here ~ denotes asymptotic equivalence, meaning that the expression $A \sim |\mu-\mu_c|$ is approached in the limit of $\mu \to \mu_c$.

e) Give your estimate for a with one digit accuracy.

In[ ]:=
```
import numpy as np
import matplotlib.pyplot as plt

def macher(mu):
    #return (mu-(1/(2+mu))-(mu**2/(2+mu))+np.sqrt((5+9*mu**2+4*mu**3+mu**4)/((2+mu)**2
    return np.log((1 + mu**2)/(mu + 2))

uCritical = 0.066
uOffside = np.linspace(0.001,uCritical,100)
uDelta = np.abs(uOffside-uCritical)

yValues = macher(uDelta)
xValues = np.log(uDelta)

plt.plot(xValues,yValues,'o')

# regression
coeff = np.polyfit(xValues,yValues,1)
# print(coeff)
print(coeff)
```

f) First, numerically evaluate the period time of the periodic orbit and plot it against $\mu - \mu_c$. Second, from the estimate in e), together with the results of c) and d), you should be able to find an estimate of the period time, T_μ, as a function of $|\mu - \mu_c|$. Plot this estimate in the same figure as the numerically evaluated period time. To show that the scaling for small $|\mu - \mu_c|$ comes out correctly, you may need to multiply T_μ by a constant to make the curves overlap for small $|\mu - \mu_c|$.