DYS HW3-3 Homoclinicbifurcation

See exercise 8.4.3 and 8.4.12 in Strogatz

Consider the dynamical system

In[273]:=

$$\begin{aligned} \dot{x} &= \mu x + y - x^2 \\ \dot{y} &= -x + \mu y + 2x^2 \end{aligned} \quad , \qquad (1)$$

where μ is a dimensionless parameter.

See exercise 8.4.3 and 8.4.12 in Strogatz

Consider the dynamical system

Out[273]=

$$\begin{aligned} \dot{x} &= \mu x + y - x^2 \\ \dot{y} &= -x + \mu y + 2x^2 \end{aligned} \quad , \qquad (1)$$

where μ is a dimensionless parameter.

---

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

This point need to be

In[274]:=

```
x'[t] == m * x[t] + y[t] - x[t]^2;
y'[t] == -x[t] + m * y[t] + 2 * x[t]^2;
```

In[276]:=
```
(*ClearAll["Global'*"]
eq1 = m*x[t] + y[t] - x[t]^2 == 0;
eq2 = -x[t] + m*y[t] + 2*x[t]^2 == 0;

xStar = (1+m^2)/(m+2);
yStar = (1/m) * (xStar - 2*xStar^2) // Expand // Simplify

detFP2 = ((m-2*xStar) * (m)) - ((-1 + 4* xStar)*(1));

Solve[((m - 2*xStar)*m) - ((-1 + 4*xStar)*1) == 1, m];

A = {{m - 2 * xStar, 1},{-1+4*xStar, 2 * xStar}};
DetA = Det[A]
r = DetA /. m→-1

TraceA = Trace[A] // Simplify;
sol = NSolve[DetA[[1]] == 0,m] // Simplify

(1-2*u + u^2 - 2*u^3)/(2+u)^2*)
```

Implemented in a numerical solving in Python. With the manipulate plot below, are range for \mu_{critical} could be found. In the diff. equation is computed and evaluated at time = 0 and time = 100. Time 100 is truly just a arbitrary number that in the end gave a okayish solution. The two evaluated points are compared with each other and if the distance is smaller than a certain threshold, it is assumed that the critical \mu got found.

```python
import numpy as np
from scipy.integrate import solve_ivp
import tqdm
import matplotlib.pyplot as plt
```

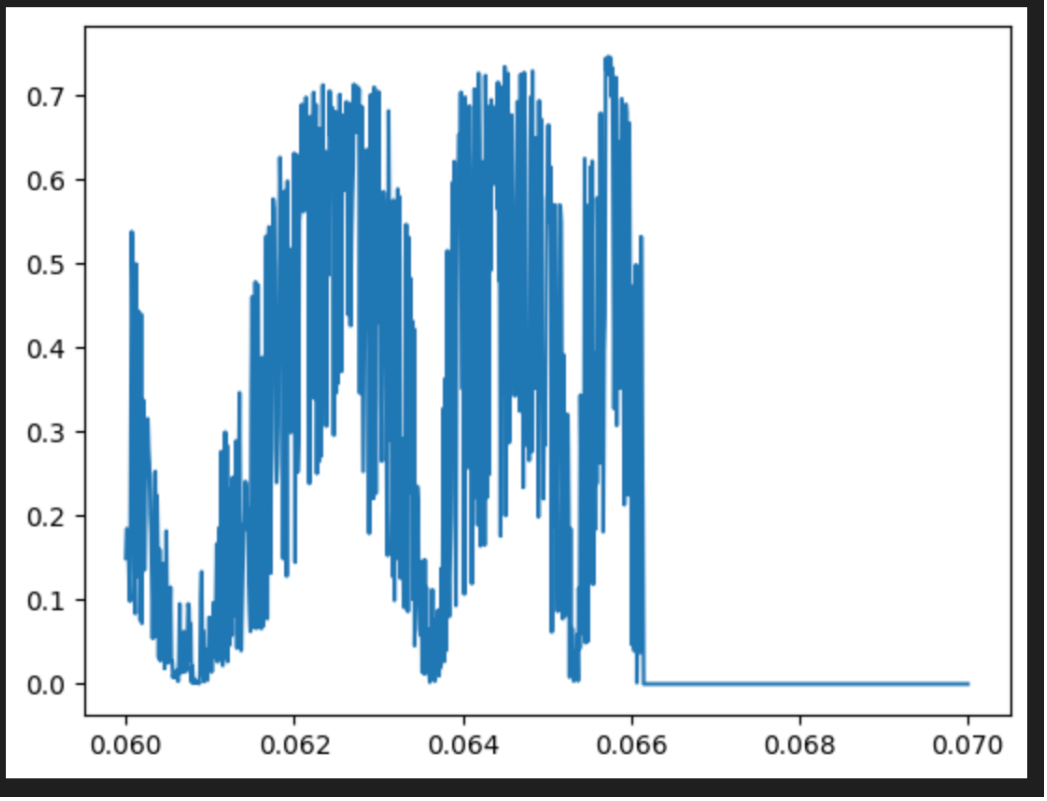
✓ 0.4s

```python
stepSize = 0.00001
Us_ = np.arange(0.06, 0.07, stepSize)

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

distanceTracker = np.zeros(len(Us_))

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(Us_):

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

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

    solution = solve_ivp(system, t_span, initG, 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)

    distanceTracker[Us_ == u] = distance

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

plt.plot(Us_, distanceTracker)
```

```
1  plt.plot(Us_, distanceTracker)
2  print(u)
```

✓ 0.1s

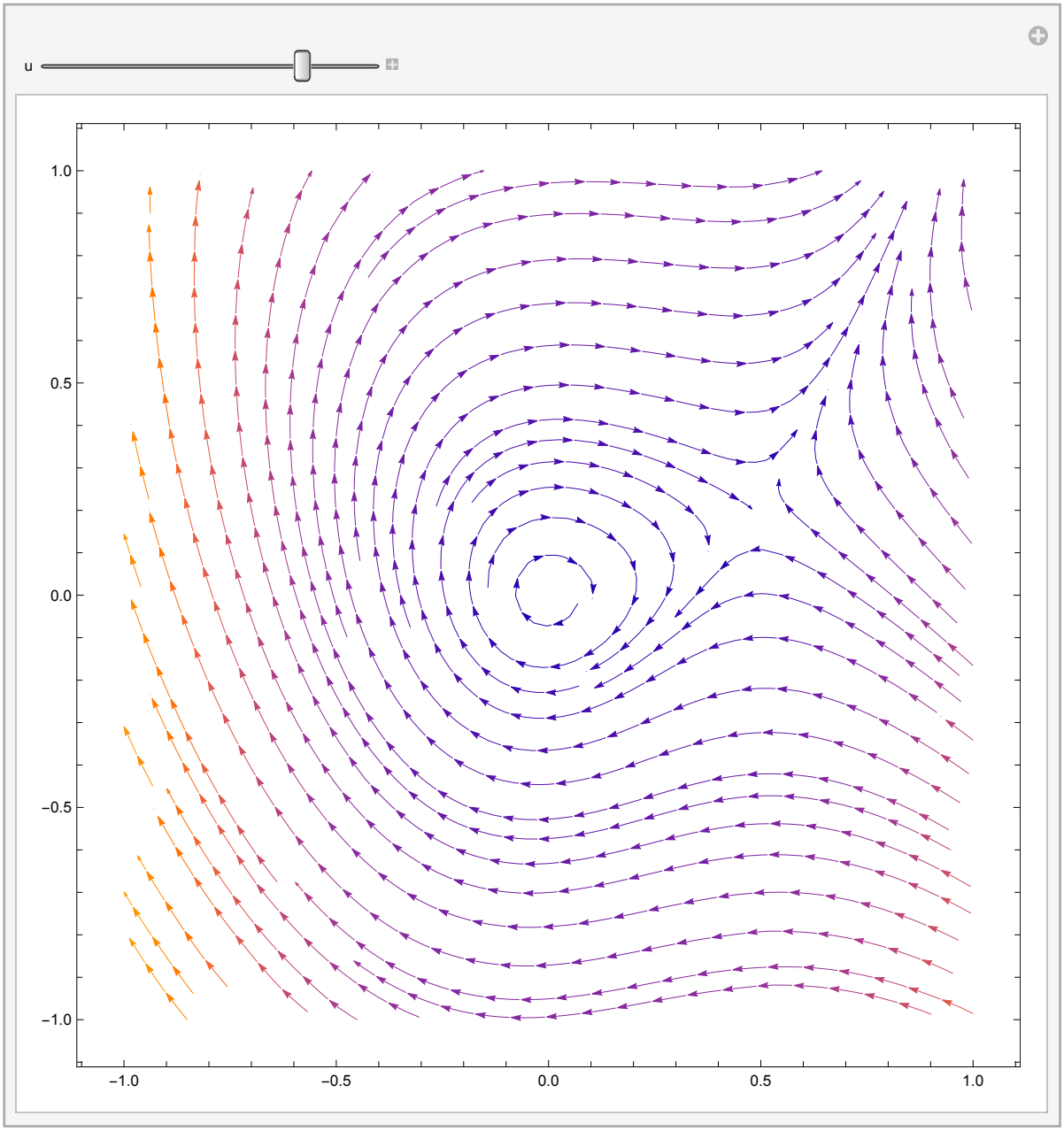
```
0.06615000000000187
```



In[277]:=

```
ClearAll["Global'*"]

Manipulate[
  StreamPlot[{u*x + y - x^2, -x + u*y + 2x^2}, {x, -1, 1}, {y, -1, 1},
    PlotRange → All, ImageSize → Large, ColorFunction → (Black), StreamScale→Tiny,
    StreamStyle → Directive[Black]],
  {{u, 0.066, "u"}, 0.05, .07, 0.001}
]
```

Out[278]=



b.) b) Plot the phase portraits that occur for the cases m < 0, m=0, 0< m < m_c , m = m_c and m > m_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). Upload the portraits as .png images or as .pdf.

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). *Upload the portraits as .png images or as .pdf.*

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

$$\begin{aligned} \dot{x} &= u\,x \\ \dot{y} &= s\,y \end{aligned},$$

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.

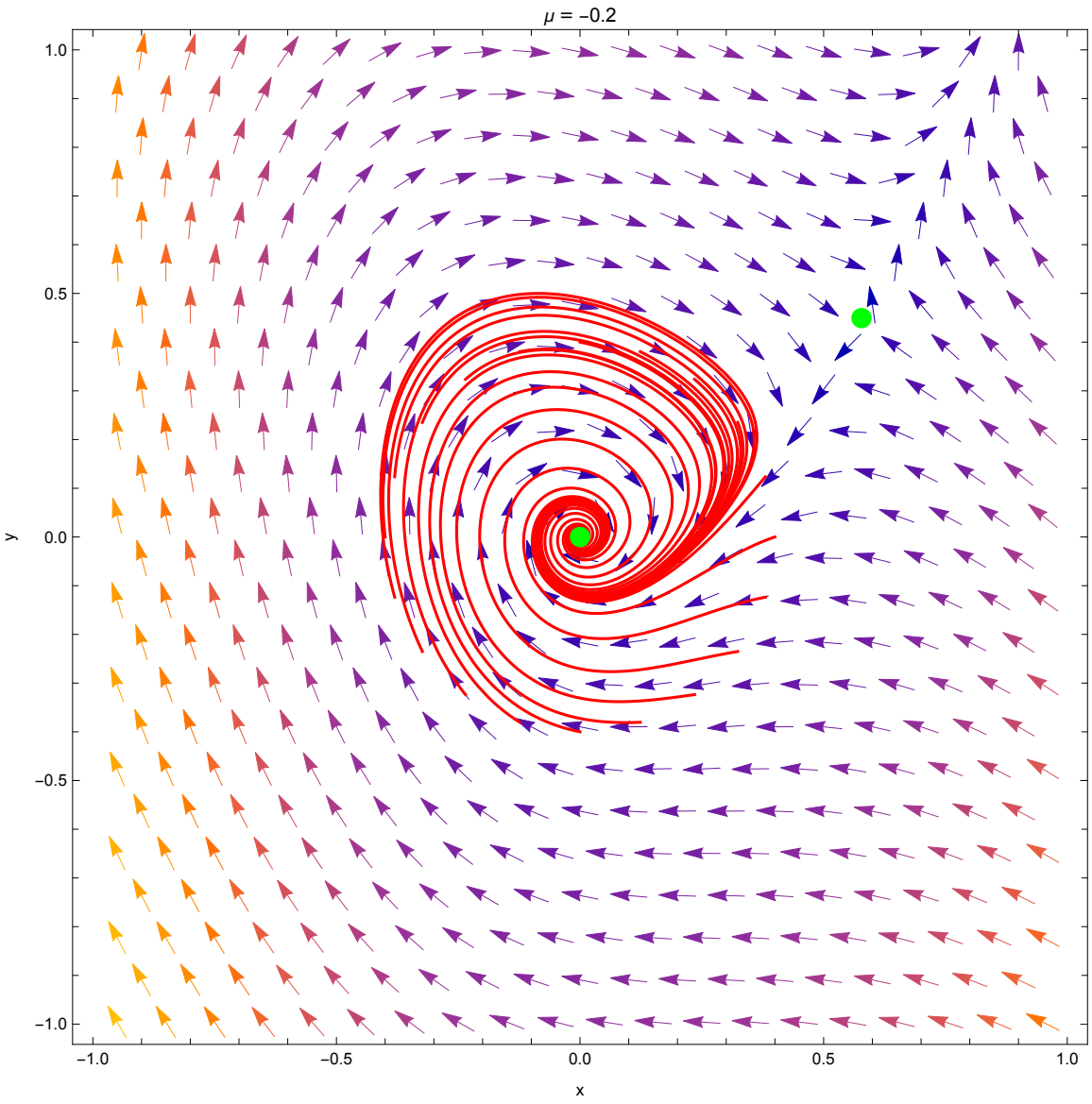
In[1208]:=

```
ClearAll["Global`*"]

u = -0.2;

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 = 50;

(* Place init points around the FP*)
numInitPoints = 20;
radiusMin = 0.4;
radiusMax = radiusMin + 0.01

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

(* 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
```

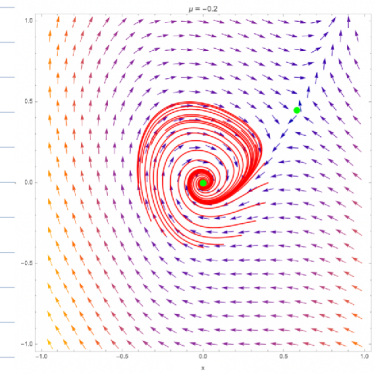
```
(* Plot the trajectories for each solution with arrows indicating the direction *)
TrajectoryPlotArrows = Table[
   ParametricPlot[Evaluate[{x[t], y[t]} /. solution], {t, t0, tMax}, PlotStyle → Red],
   {solution, solutions}
];

(* Add arrows indicating the direction using VectorPlot *)
VectorPlotArrows = VectorPlot[{u*x + y - x^2, -x + u*y + 2*x^2}, {x, -1, 1}, {y, -1, 1
   VectorPoints → 20, VectorScale → 0.03, VectorStyle → Directive[Black]];

(* Show the combined plot *)
Show[VectorPlotArrows, TrajectoryPlotArrows, FrameLabel → {"x", "y"},
  PlotLabel → "μ=-0.2", ImageSize → 600,
  Epilog → {Green, PointSize[0.02], {Point[{FixPoint2[[1]], FixPoint2[[2]]}],Point[{FixPoi
```

Out[1219]= 0.41

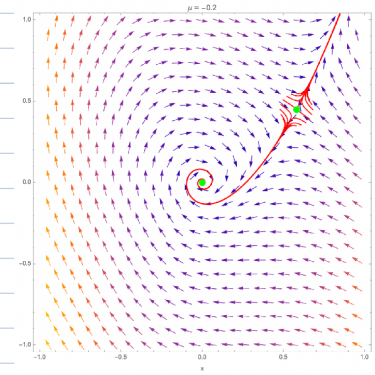
Out[1225]=



$\mu = -0.2$



Starting very close to FP_1.
The trajectory spirals into the stable FP_1 @ $\{0,0\}$.



When starting the trajectories around FP_2 they either spiral into the stable FP_1 or escape towards NorthEast, with a way higher velocity than the ones going into FP_1. FP_2 seems to be a saddle point, since attracting in one direction, and repelling in the other (Eigenvectors)

So stable FP_1 at $\{0,0\}$ and a saddle node FP_2 (μ).

In[1460]:=
```
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 = 200;

(* Place init points around the FP*)
numInitPoints = 20;
radiusMin = 0.2;
radiusMax = radiusMin + 0.01

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

(* 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 → Red],
   {solution, solutions}
];

(* Add arrows indicating the direction using VectorPlot *)
VectorPlotArrows = VectorPlot[{u*x + y - x^2, -x + u*y + 2*x^2}, {x, -1, 1}, {y, -1, 1
   VectorPoints → 20, VectorScale → 0.03, VectorStyle → Directive[Black]];

(* Show the combined plot *)
Show[VectorPlotArrows, TrajectoryPlotArrows, FrameLabel → {"x", "y"},
  PlotLabel → "μ=0", ImageSize → 600,
  Epilog → {Green, PointSize[0.02], {Point[{FixPoint2[[1]], FixPoint2[[2]]}],Point[{FixPoi
```

Out[1471]= 0.21

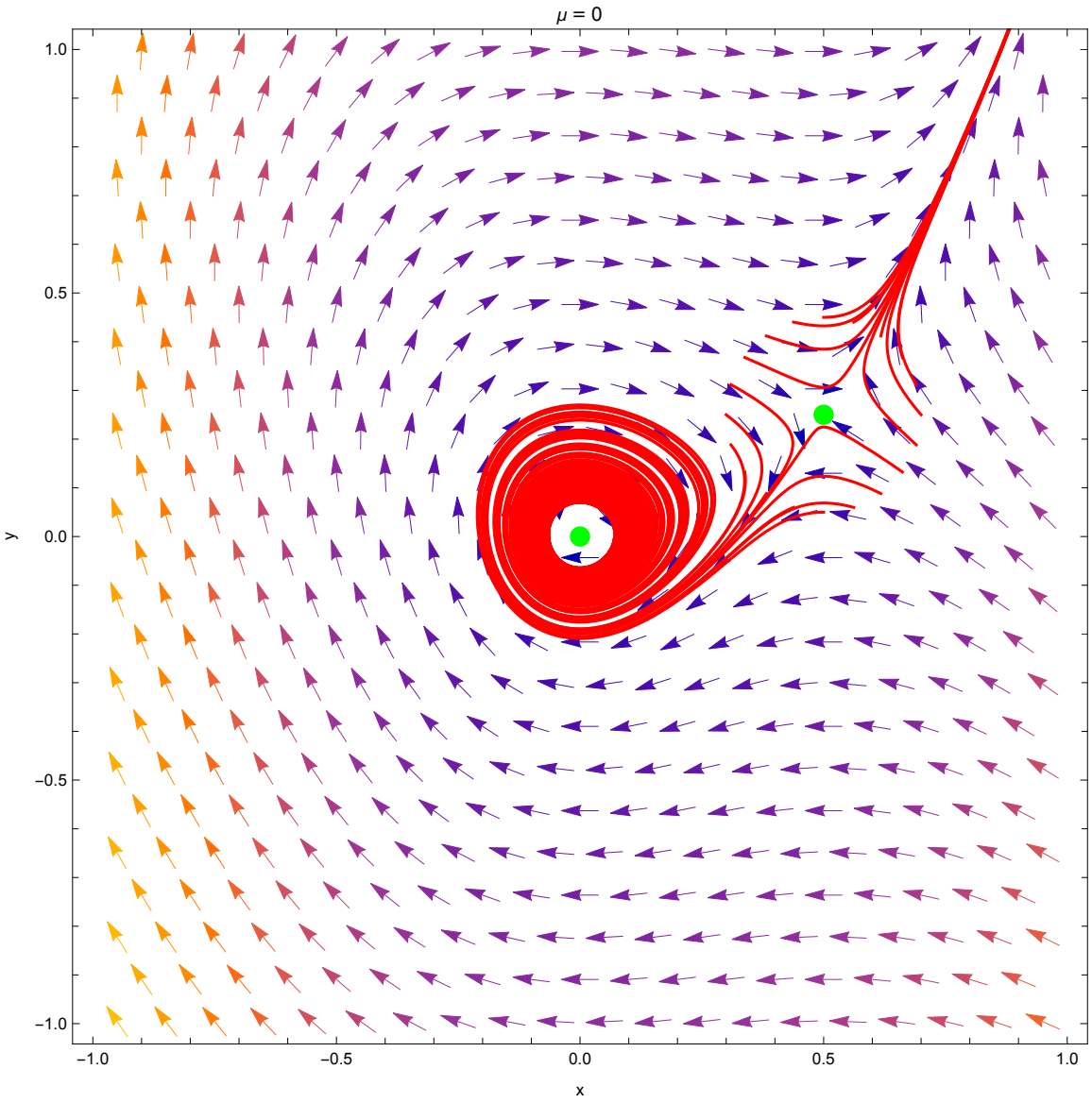
NDSolve : Error test failure at t == 28.235085537388585` ; unable to continue.

NDSolve : Error test failure at t == 28.010370412325354` ; unable to continue.

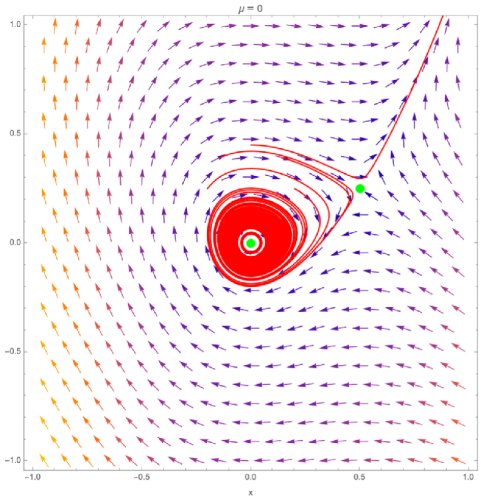
NDSolve : Error test failure at t == 27.9260038909556` ; unable to continue.

General : Further output of NDSolve::nderr will be suppressed during this calculation.

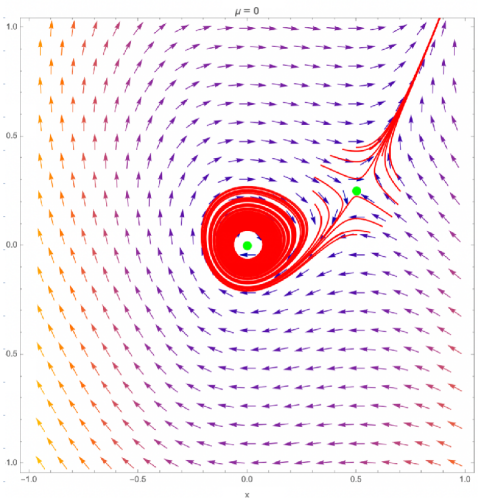
Out[1477]=



$\mu = 0$



Trajectories starting close around FP_1 spiral into FP_1. FP_1 stable. Into FP_1 the trajectories are very slow.



FP_2 is again as in $\mu = -0,2$ a saddle point

In[1568]:=
```
ClearAll["Global`*"]

u = 0.02;

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 = 200;

(* Place init points around the FP*)
numInitPoints = 20;
radiusMin = 0.2;
radiusMax = radiusMin + 0.01

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

(* 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 → Red],
   {solution, solutions}
];

(* Add arrows indicating the direction using VectorPlot *)
VectorPlotArrows = VectorPlot[{u*x + y - x^2, -x + u*y + 2*x^2}, {x, -1, 1}, {y, -1, 1
   VectorPoints → 20, VectorScale → 0.03, VectorStyle → Directive[Black]];

(* Show the combined plot *)
Show[VectorPlotArrows, TrajectoryPlotArrows, FrameLabel → {"x", "y"},
  PlotLabel → "μ=0.02}", ImageSize → 600,
  Epilog → {Green, PointSize[0.02], {Point[{FixPoint2[[1]], FixPoint2[[2]]}],Point[{FixPoi
```

Out[1579]= 0.21

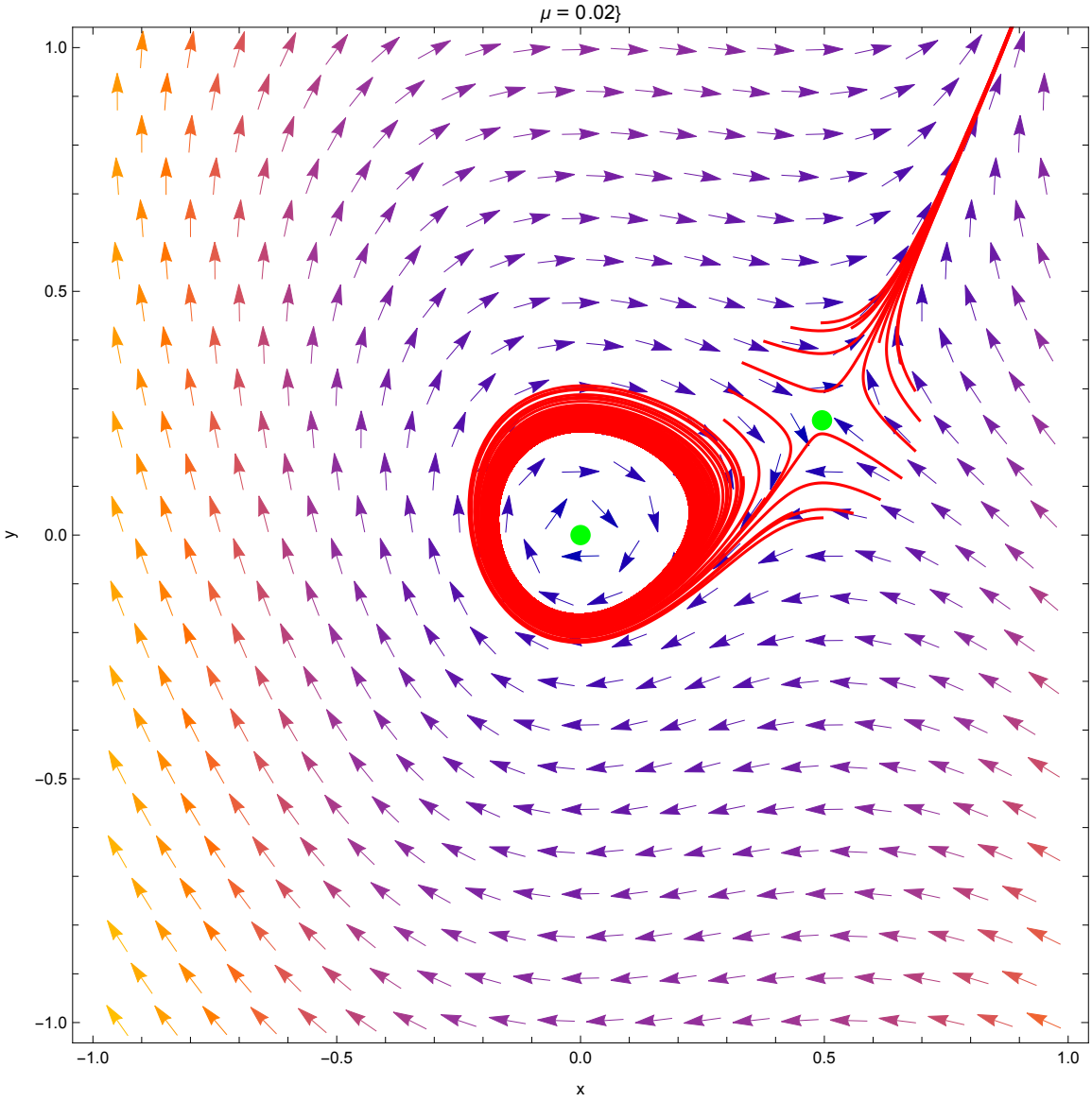
NDSolve : Error test failure at t == 27.910097723743238` ; unable to continue.

NDSolve : Error test failure at t == 27.76434314687637` ; unable to continue.

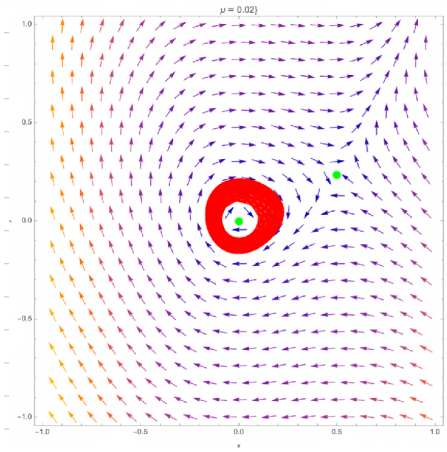
NDSolve : Error test failure at t == 27.735878991780677` ; unable to continue.

General : Further output of NDSolve::nderr will be suppressed during this calculation.

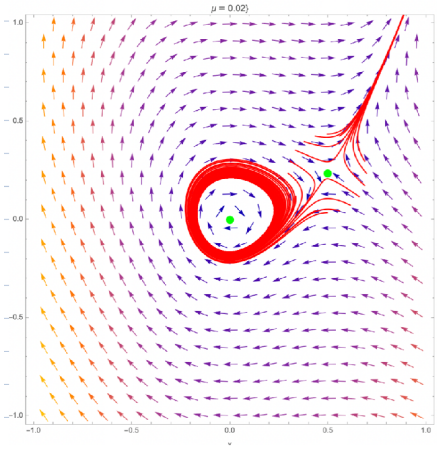
Out[1585]=



$\mu = 0,02$



Trj. starting close around FP_1 are repelled from the FP_1, hence FP_1 seems unstable. They are trapped inside the "egg" shape, limit cycle.



When starting around FP_2, the traj are attracted either to limit cycle or escape towards NorthEast. FP_2 saddle point.

In[1712]:=
```
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;

(* Place init points around the FP*)
numInitPoints = 4;
radiusMin = 0.05;
radiusMax = radiusMin + 0.0001

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

(* 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 → Red],
   {solution, solutions}
];

(* Add arrows indicating the direction using VectorPlot *)
VectorPlotArrows = VectorPlot[{u*x + y - x^2, -x + u*y + 2*x^2}, {x, -1, 1}, {y, -1, 1
   VectorPoints → 20, VectorScale → 0.03, VectorStyle → Directive[Black]];

(* Show the combined plot *)
Show[VectorPlotArrows, TrajectoryPlotArrows, FrameLabel → {"x", "y"},
  PlotLabel → "μ=0.066 -> Homoclinic", ImageSize → 600,
  Epilog → {Green, PointSize[0.02], {Point[{FixPoint2[[1]], FixPoint2[[2]]}],Point[{FixPoi
```

Out[1723]= 0.0501

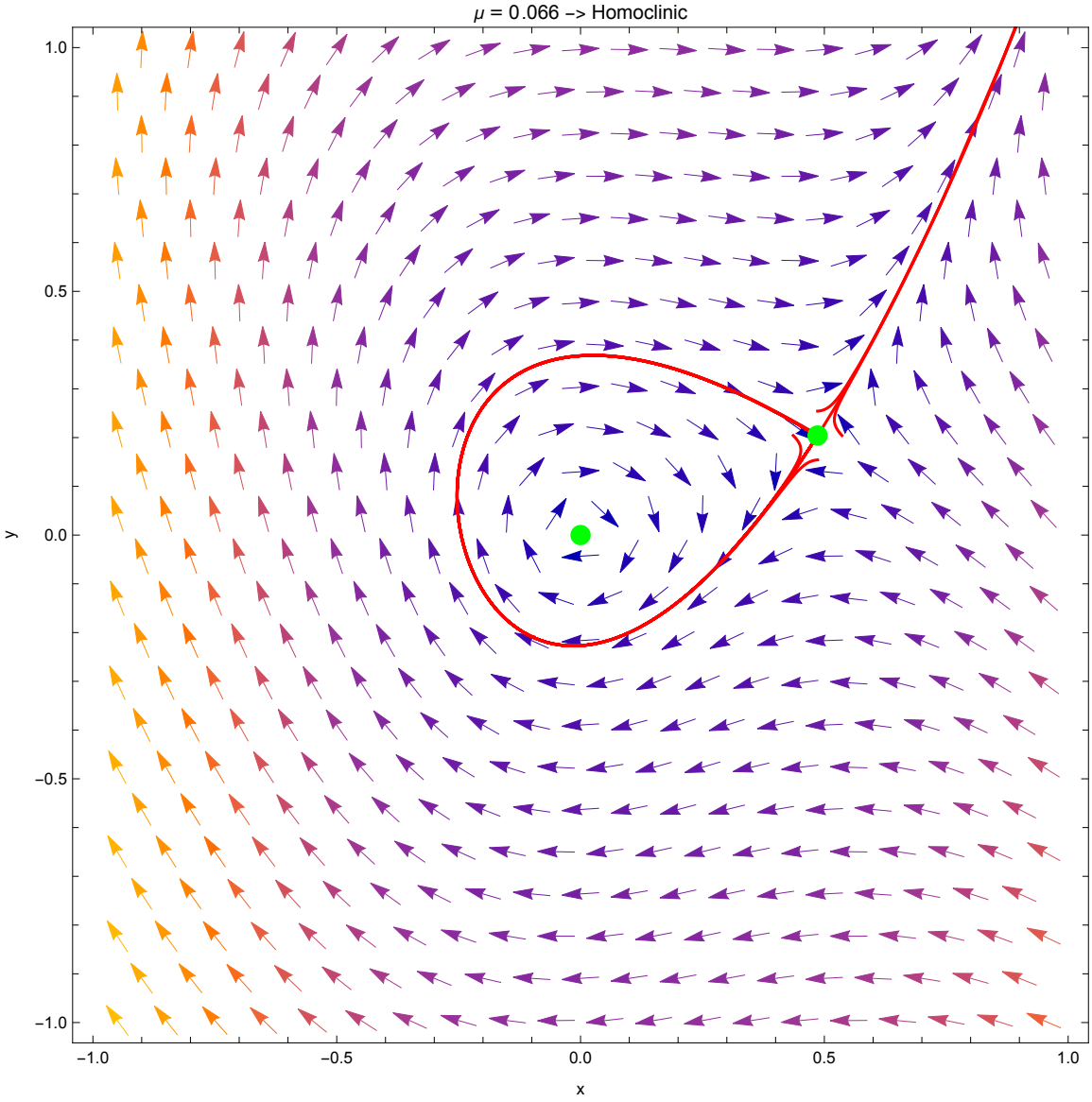
NDSolve : Error test failure at t == 29.208131297485682` ; unable to continue.

NDSolve : Error test failure at t == 44.749583973206654` ; unable to continue.

NDSolve : Error test failure at t == 29.778555367806362` ; unable to continue.

General : Further output of NDSolve::nderr will be suppressed during this calculation.

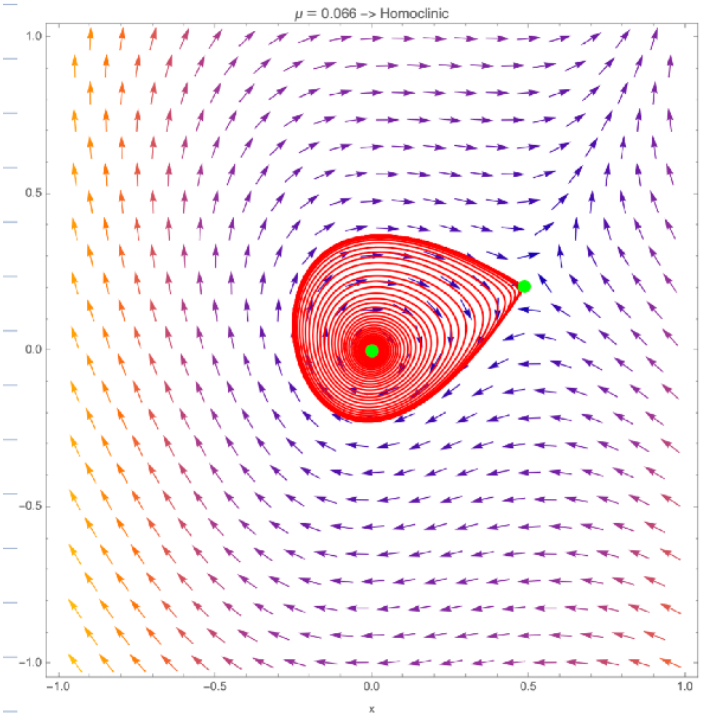
Out[1729]=



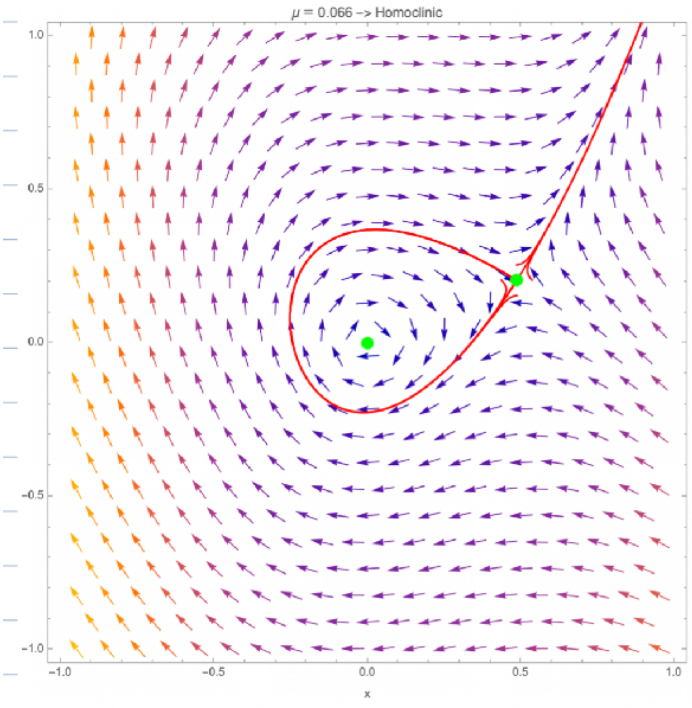
In[351]:=

Out[351]=

$\mu = \mu_c$



Starting trj. cl
the get trapped by
FP_1 unstable spiral



Starting just at the F
the trajectories escap
come into it again.

In[2018]:=
```
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 = 200;

(* Place init points around the FP*)
numInitPoints = 6;
radiusMin = 0.3;
radiusMax = radiusMin + 0.01

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

(* 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 → Red],
   {solution, solutions}
];

(* Add arrows indicating the direction using VectorPlot *)
VectorPlotArrows = VectorPlot[{u*x + y - x^2, -x + u*y + 2*x^2}, {x, -1, 1}, {y, -1, 1
   VectorPoints → 20, VectorScale → 0.03, VectorStyle → Directive[Black]];

(* Show the combined plot *)
Show[VectorPlotArrows, TrajectoryPlotArrows, FrameLabel → {"x", "y"},
  PlotLabel → "μ=0.08", ImageSize → 600,
  Epilog → {Green, PointSize[0.02], {Point[{FixPoint2[[1]], FixPoint2[[2]]}],Point[{FixPoi
```

Out[2029]= 0.31

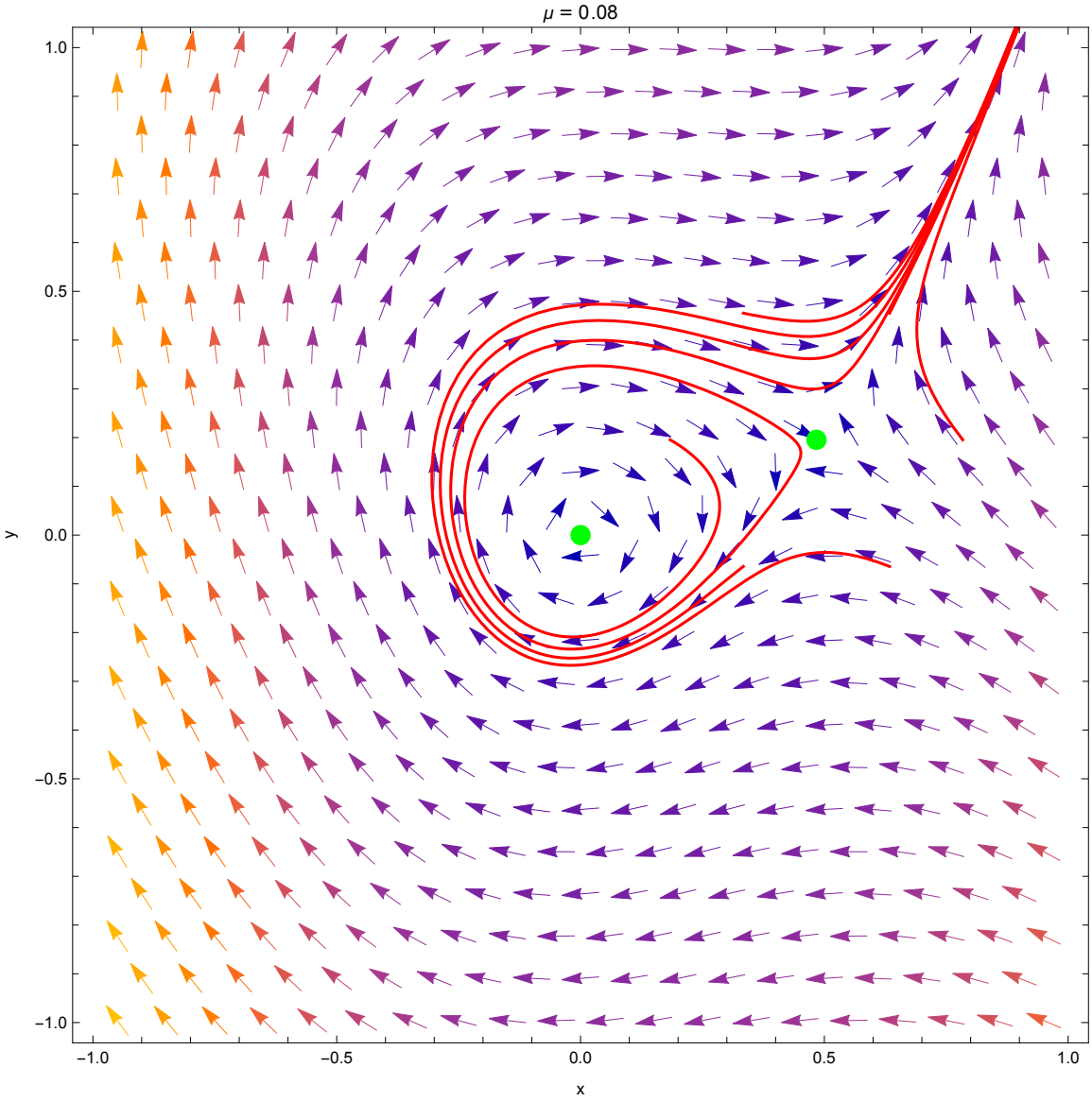
NDSolve : Error test failure at t == 26.1790464824705` ; unable to continue.

NDSolve : Error test failure at t == 27.29286744041605` ; unable to continue.

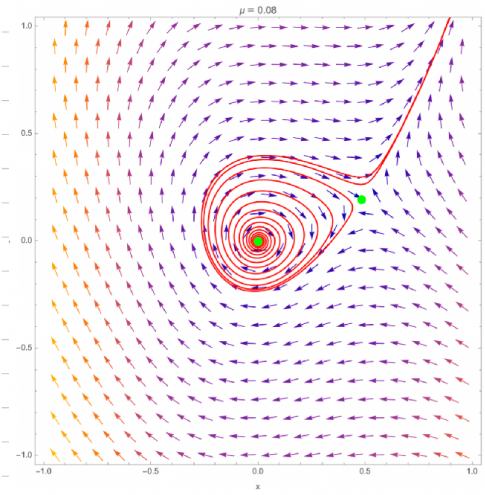
NDSolve : Error test failure at t == 46.68575708513769` ; unable to continue.

General : Further output of NDSolve::nderr will be suppressed during this calculation.

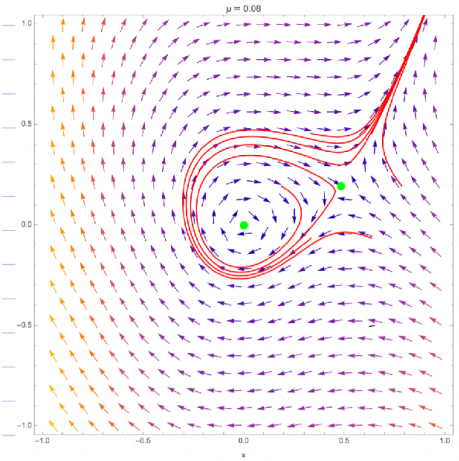
Out[2035]=



$\mu = 0{,}08$



Starting traj. close to FP_1 the escape from FP_1. So around FP_1 unstable spiral.



FP_2 saddle point.

FP1 is referred to the FP at {0,0}, FP2 the one depending on mu.

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

$(x(0), y(0)) = (\gamma, 1)$, where $\gamma > 0$ is small.

Idea is to find a solution for x(t) and y(t), depending on m and g, and integrate from t = 0 to t = t1 until x(t1) == 1

In[370]:=
```
ClearAll["Global`*"]

A = {{m - 2*x, 1},{-1+4*x, m}}
DetA = Det[A]
EigVal = Eigenvalues[A] // Simplify
EigVec = Eigenvectors[A]
```

Out[371]= $\{\{m - 2\,x,\ 1\},\ \{-1 + 4\,x,\ m\}\}$

Out[372]= $1 + m^2 - 4\,x - 2\,m\,x$

Out[373]= $\left\{m - x - \sqrt{-1 + 4\,x + x^2},\ m - x + \sqrt{-1 + 4\,x + x^2}\right\}$

Out[374]= $\left\{\left\{-\frac{x + \sqrt{-1 + 4\,x + x^2}}{-1 + 4\,x},\ 1\right\},\ \left\{-\frac{x - \sqrt{-1 + 4\,x + x^2}}{-1 + 4\,x},\ 1\right\}\right\}$

In[375]:=

c)

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

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

↓ linearise

$\dot{x} = ux$ $\qquad \frac{dx}{dt} = ux$

$\dot{y} = sy$

find the time t_1 to escape from the saddle to $x(t_1) = 1$!

Init conditions:

$x(0) = \mu$

$y(0) = 1$

Seperation of variable

$\frac{dx}{dt} = u \cdot x$

$\frac{1}{x}\,dx = u \cdot dt$ $\qquad \int$

$\int \frac{1}{x}\,dx = \int u \cdot dt$

$\ln|x| = u \cdot t + C$

Put in init conditions: $x(0) = g \rightarrow \ln|g| = C$

$\ln|x| = u \cdot t + \ln|g|$

$x = g \cdot e^{(u \cdot t)}$

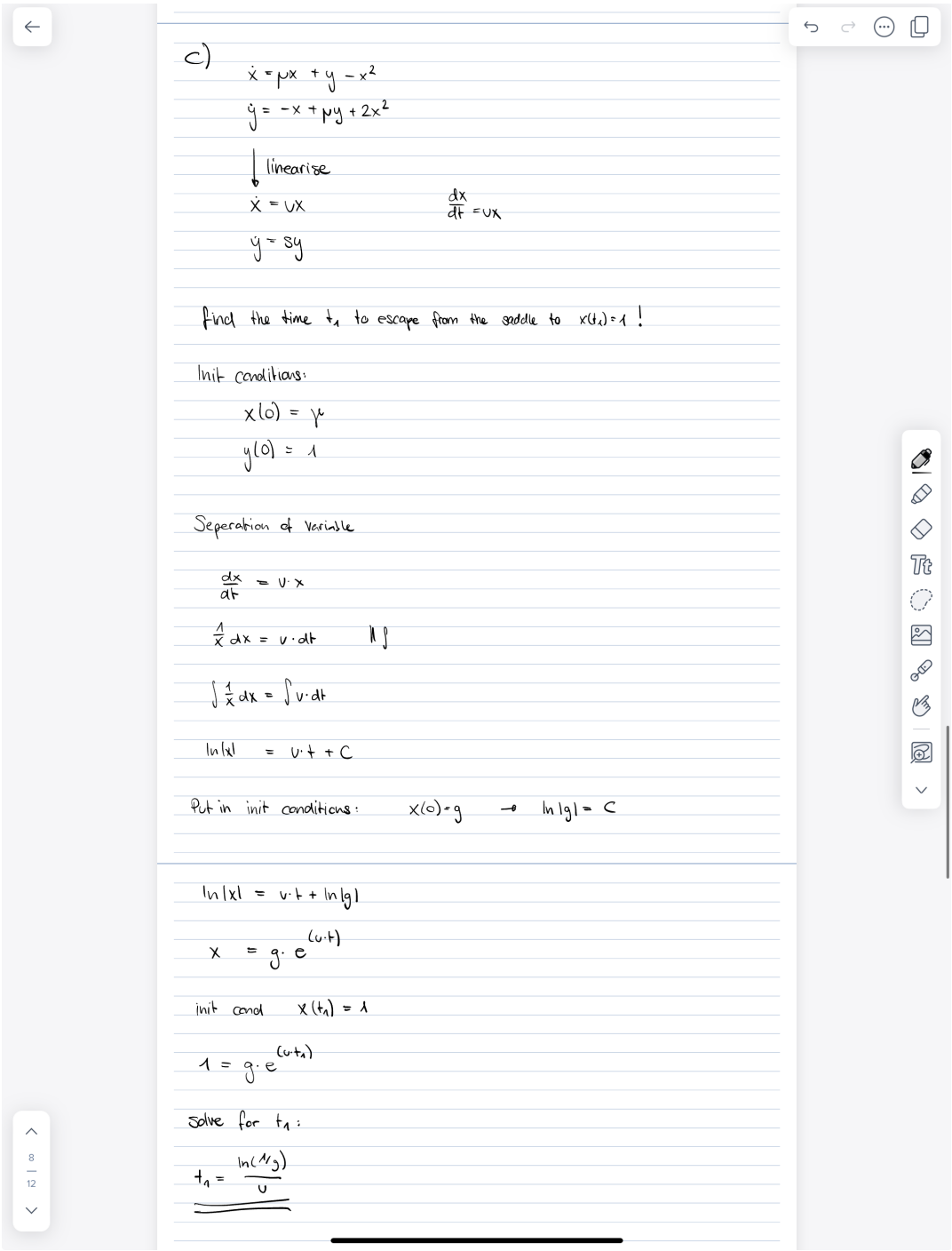
init cond $x(t_1) = 1$

$1 = g \cdot e^{(u \cdot t_1)}$

Solve for t_1:

$t_1 = \frac{\ln(1/g)}{u}$

Out[375]=



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

Find u in terms of μ. For this find the unstable Eigenvector (where Eigenvalue is > 0). Evaluate the Jacobian at the FP xStar(\mu)

In[376]:=
```
ClearAll["Global`*"]

xStar = (1+m^2)/(m+2);
A = {{m - 2 * x, 1},{4 * x-1, m}}

(* Eigenvalues *)
EigVal = Eigenvalues[A] /. x→xStar //Simplify
(* EigVec = Eigenvectors[A] // Simplify *)
```

Out[378]= $\{\{m-2x,\ 1\},\ \{-1+4x,\ m\}\}$

Out[379]= $\left\{m-\frac{1}{2+m}-\frac{m^2}{2+m}-\sqrt{\frac{5+9m^2+4m^3+m^4}{(2+m)^2}},\ m-\frac{1}{2+m}-\frac{m^2}{2+m}+\sqrt{\frac{5+9m^2+4m^3+m^4}{(2+m)^2}}\right\}$

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 $|\mu - \mu_c| \ll 1$.

e.)

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 $\sim$ denotes asymptotic equivalence, meaning that the expression $A \sim |\mu - \mu_c|^a$ is approached in the limit of $\mu \to \mu_c$.

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

```python
import numpy as np
from scipy.integrate import solve_ivp
import tqdm
import matplotlib.pyplot as plt
```

✓ 0.6s

$$\gamma \sim A|\mu - \mu_c|$$

```python
def compGamma(mu):
    return (1 + mu**2) / (mu + 2)
```

✓ 0.0s

```python
mc = 0.066
mStart = 0.064
n = 100

delta = (mc - mStart) / n

# approach mc from below
m = np.linspace(mStart, mc-delta, n)
mabs = np.abs(m - mc)

# calculate gamma -> xStar depending on mu
gamma = np.zeros(n)
for i in range(n):
    gamma[i] = compGamma(m[i])

x = (mabs)
y = np.log(gamma)

# # ln gamma y axis, mc-m x axis
plt.plot(x, y, 'o')
plt.xlabel("ln |m-mc|")
plt.ylabel("ln gamma")

# # # compute regression of degree 1
coeff = np.polyfit(x, y, 1)
print("Coeff a: ", coeff[0], "\nCoeff b: ", coeff[1])

# # # plot regression
xreg = mabs
yreg = coeff[0] * xreg + coeff[1]
plt.plot(xreg, yreg, '--r')

plt.show
```