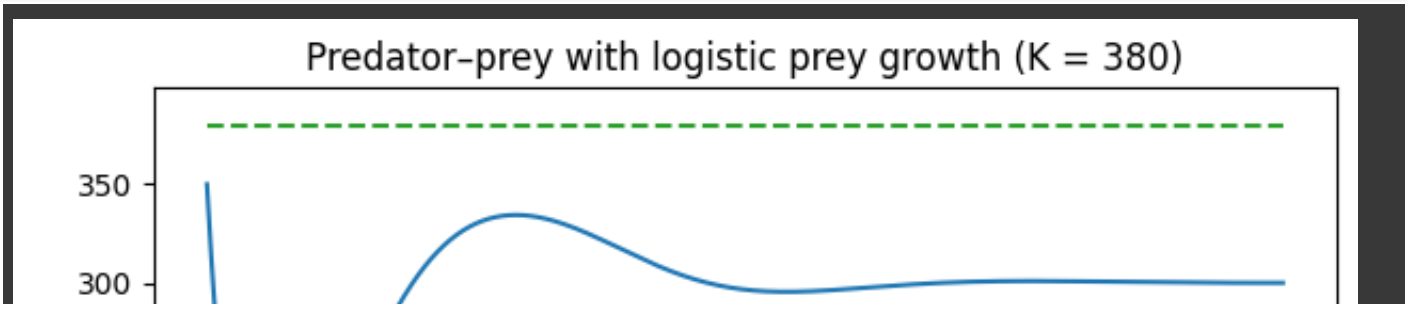
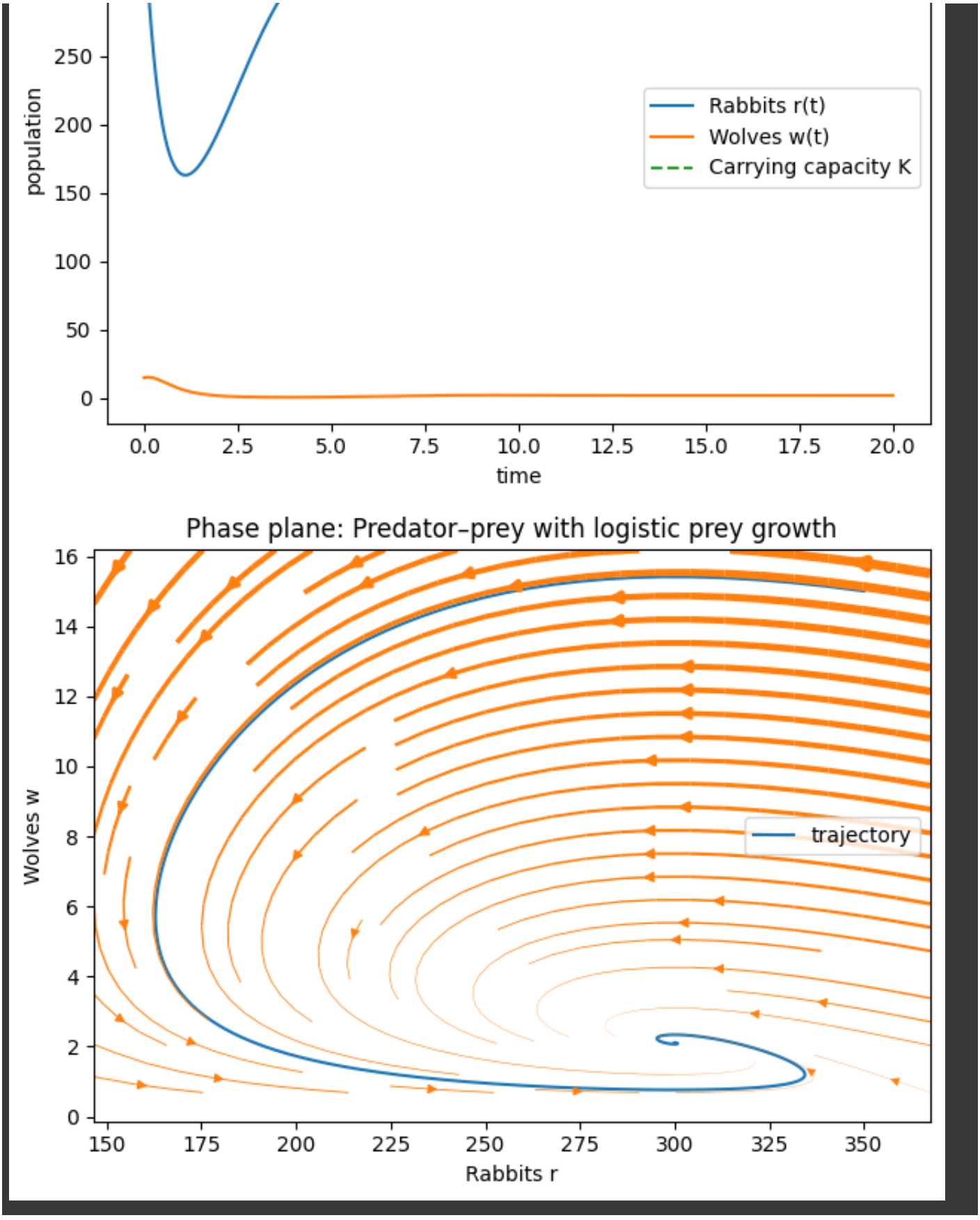
Predator–prey with logistic prey growth (Q5)

```
# dr/dt = a*r*(1 - r/K) - b*r*w
# dw/dt = c*r*w - d*w

from numpy import linspace, array, stack, meshgrid, sqrt
from scipy.integrate import solve_ivp
from matplotlib import pyplot as plt
import numpy as np

# --- parameters ---
a = 1        # Rabbit growth rate
b = 0.1      # Rabbit death rate from wolf
c = 0.01     # Wolf growth rate from rabbit
d = 3        # wolf natural death rate
K = 380    # rabbit carrying capacity

# --- ODE system ---
def prey_predator_logistic(t, var):
    r, w = var
    drdt = a * r * (1 - r / K) - b * r * w
    dwdt = c * r * w - d * w
    return stack([drdt, dwdt])

# --- initial conditions and time horizon ---
r0 = 350
w0 = 15
ic = array([r0, w0])

t0, tfinal = 0, 20

# --- solve ---
sol = solve_ivp(
    prey_predator_logistic,
    array([t0, tfinal]),
    ic,
    dense_output=True,
    rtol=1e-8,
    atol=1e-8
)
assert sol.success, "ODE solver failed."

# --- evaluate solution on a grid ---
```

```
t = linspace(t0, tfinal, 1000)
r = sol.sol(t)[0, :]
w = sol.sol(t)[1, :]

# --- Plot 1: time series ---
fig1, ax1 = plt.subplots()
ax1.plot(t, r, label="Rabbits r(t)")
ax1.plot(t, w, label="Wolves w(t)")
ax1.plot(t, 0*t + K, linestyle="--", label="Carrying capacity K")
ax1.set_xlabel("time")
ax1.set_ylabel("population")
ax1.set_title("Predator–prey with logistic prey growth (K = 380)")
ax1.legend()
fig1.tight_layout()
fig1.savefig("logistic_PP_timeseries.pdf")

# --- Plot 2: phase plane (r vs w) ---
fig2, ax2 = plt.subplots()
ax2.plot(r, w, label="trajectory")
ax2.set_xlabel("Rabbits r")
ax2.set_ylabel("Wolves w")
ax2.set_title("Phase plane: Predator–prey with logistic prey growth")

# vector field
xmin, xmax = max(0.0, r.min()*0.9), r.max()*1.05
ymin, ymax = max(0.0, w.min()*0.9), w.max()*1.05
xx = linspace(xmin, xmax, 17)
yy = linspace(ymin, ymax, 17)
X, Y = meshgrid(xx, yy, indexing="xy")
U, V = prey_predator_logistic(0.0, stack((X, Y)))

speed = sqrt((U/(xmax - xmin + 1e-12))**2 + (V/(ymax - ymin + 1e-12))**2)
lw = 5 * speed / (speed.max() + 1e-12)

ax2.streamplot(X, Y, U, V, density=0.8, linewidth=lw)
ax2.legend(loc="best")
fig2.tight_layout()
```

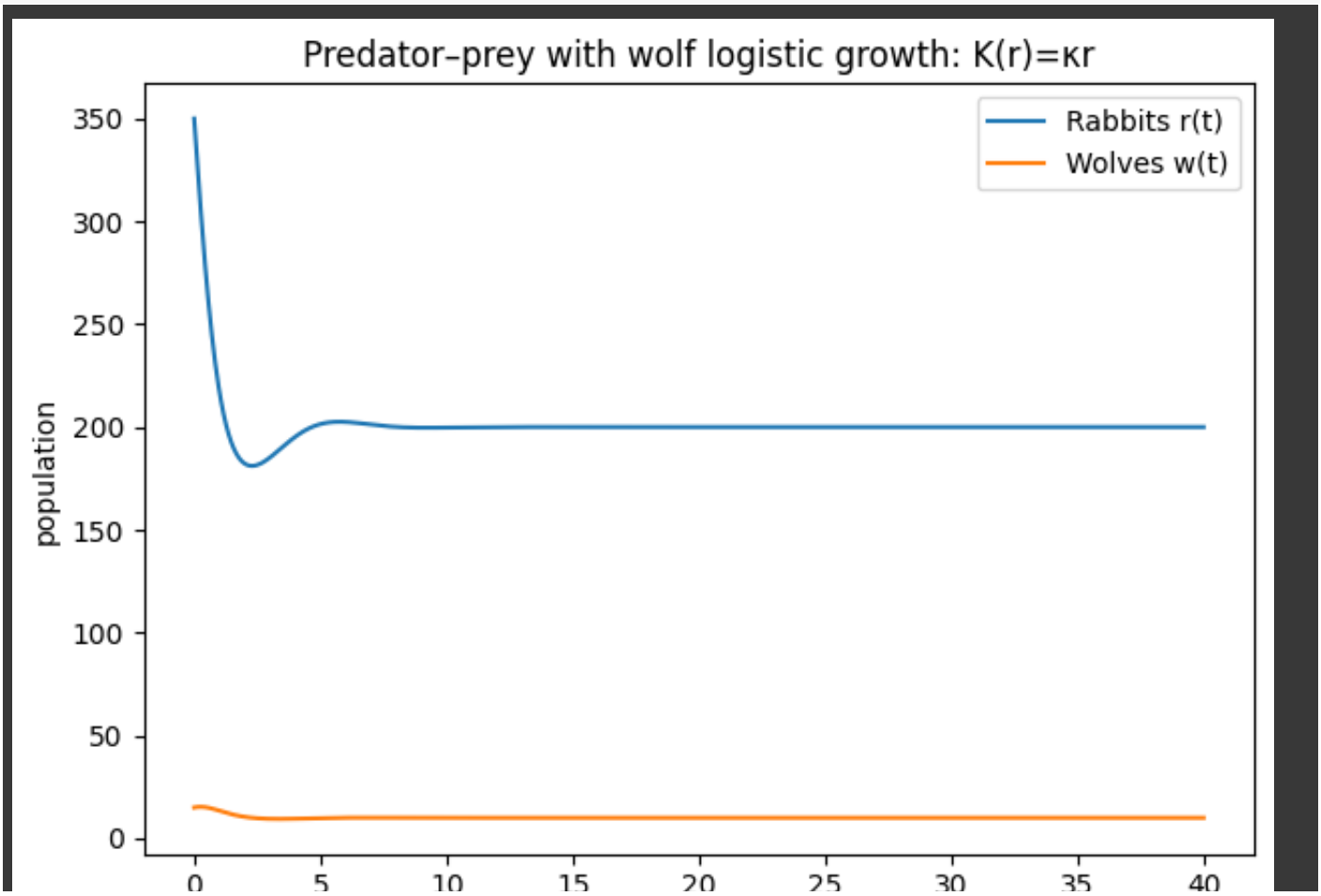


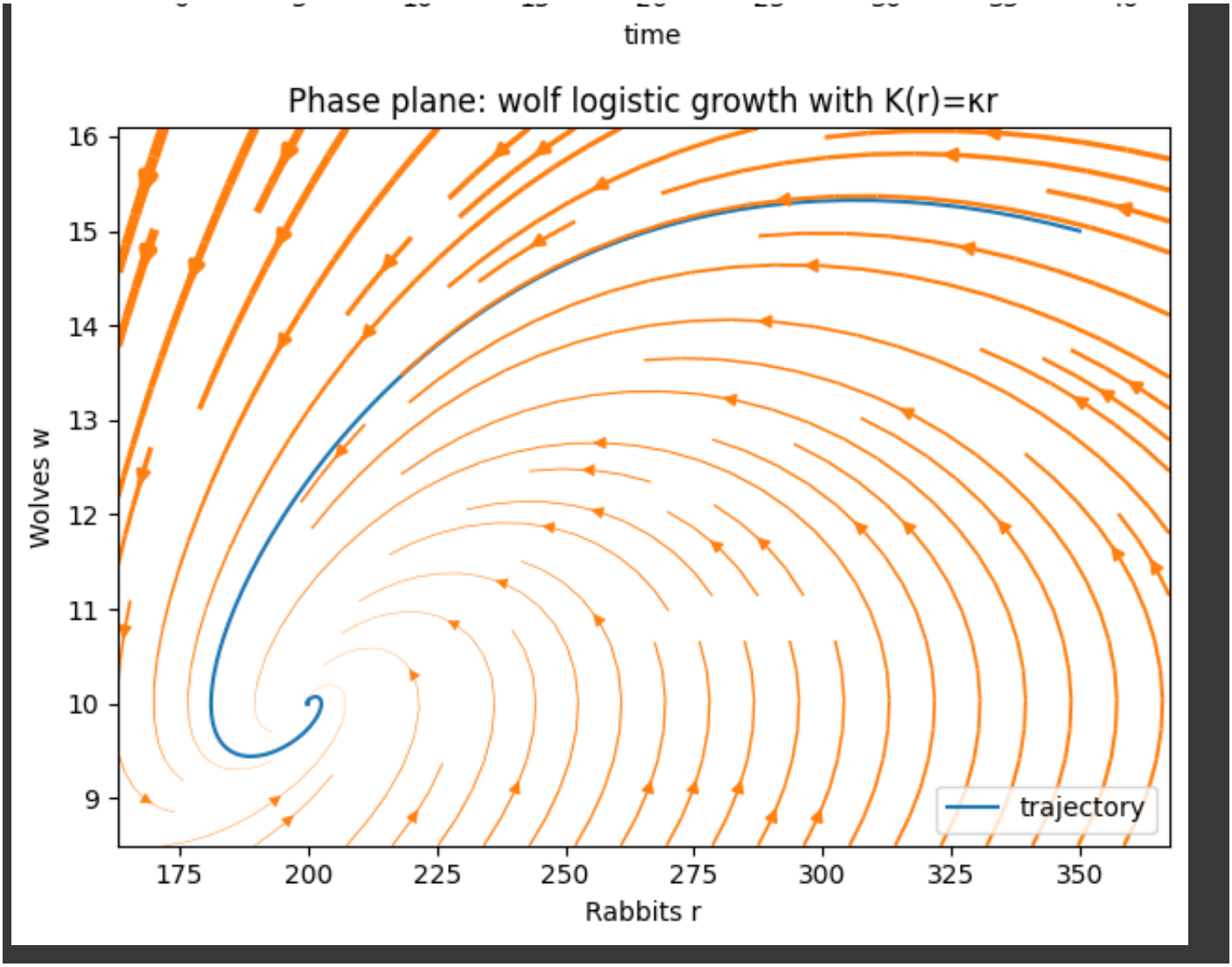

population
time
Rabbits r(t)
Wolves w(t)
Carrying capacity K
Phase plane: Predator–prey with logistic prey growth
Wolves w
Rabbits r
trajectory

Predator-prey with logistic form for wolves (Q7)

```
# Q7: Predator–prey with wolf constrained growth, K(r) = κ r
# dr/dt = a*r - b*r*w
# dw/dt = s*w*(1 - w/(κ*r))

from numpy import linspace, array, stack, meshgrid, sqrt
from scipy.integrate import solve_ivp
from matplotlib import pyplot as plt

# parameters
a = 1      # rabbit growth
b = 0.1      # predation death rate
s = 1.2      # wolf growth toward its carrying capacity
kappa = 0.05 # wolf carrying capacity per rabbit (so K = kappa * r)

# ODE system (kept in parenthesis form)
def f(t, z):
    r, w = z
    drdt = a*r - b*r*w
    dwdt = s*w*(1 - w/(kappa*r))
    return stack([drdt, dwdt])

# initial conditions and time span
r0, w0 = 350, 15
t0, tf = 0, 40
z0 = array([r0, w0])

sol = solve_ivp(f, array([t0, tf]), z0, dense_output=True, rtol=1e-8, atol=1e-8)
assert sol.success, "ODE solver failed."

# sample solution
t = linspace(t0, tf, 1200)
r = sol.sol(t)[0, :]
w = sol.sol(t)[1, :]

# Plot 1: time series
fig1, ax1 = plt.subplots()
ax1.plot(t, r, label="Rabbits r(t)")
ax1.plot(t, w, label="Wolves w(t)")
ax1.set_xlabel("time")
ax1.set_ylabel("population")
ax1.set_title("Predator–prey with wolf logistic growth: K(r)=κr")
ax1.legend()
```

```
fig1.tight_layout()

# Plot 2: phase plane with stream field
fig2, ax2 = plt.subplots()
ax2.plot(r, w, label="trajectory")
ax2.set_xlabel("Rabbits r")
ax2.set_ylabel("Wolves w")
ax2.set_title("Phase plane: wolf logistic growth with K(r)=κr")

xmin, xmax = max(0.0, r.min()*0.9), r.max()*1.05
ymin, ymax = max(0.0, w.min()*0.9), w.max()*1.05
xx = linspace(xmin, xmax, 17)
yy = linspace(ymin, ymax, 17)
X, Y = meshgrid(xx, yy, indexing="xy")
U, V = f(0.0, stack((X, Y)))
spd = sqrt((U/(xmax - xmin + 1e-12))**2 + (V/(ymax - ymin + 1e-12))**2)
lw = 5*spd/(spd.max() + 1e-12)
ax2.streamplot(X, Y, U, V, density=0.8, linewidth=lw)
ax2.legend(loc="best")
fig2.tight_layout()
```

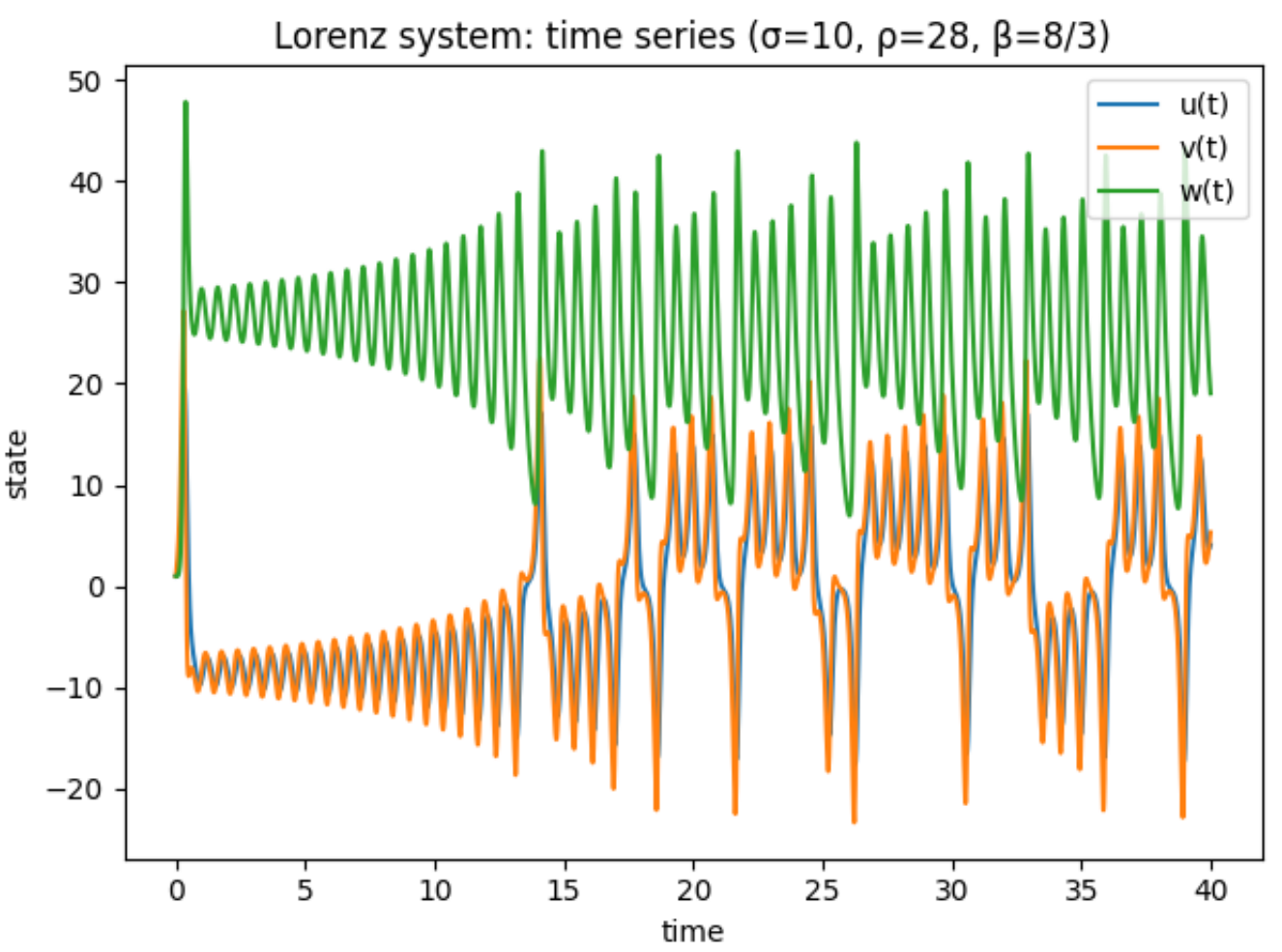


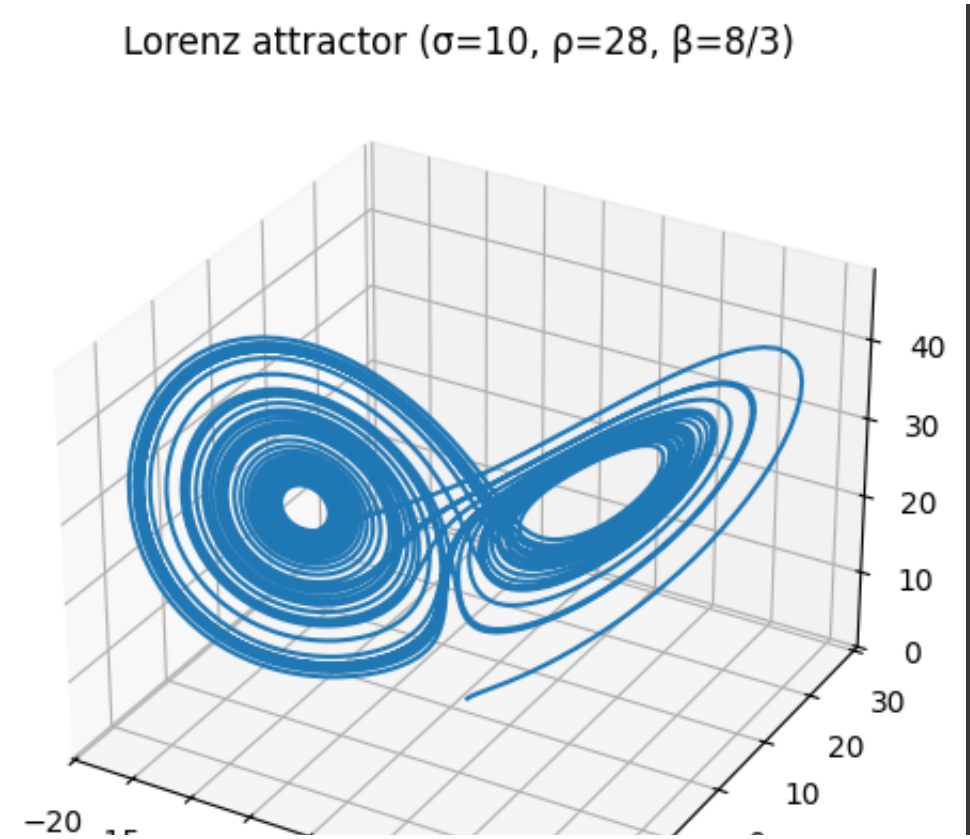


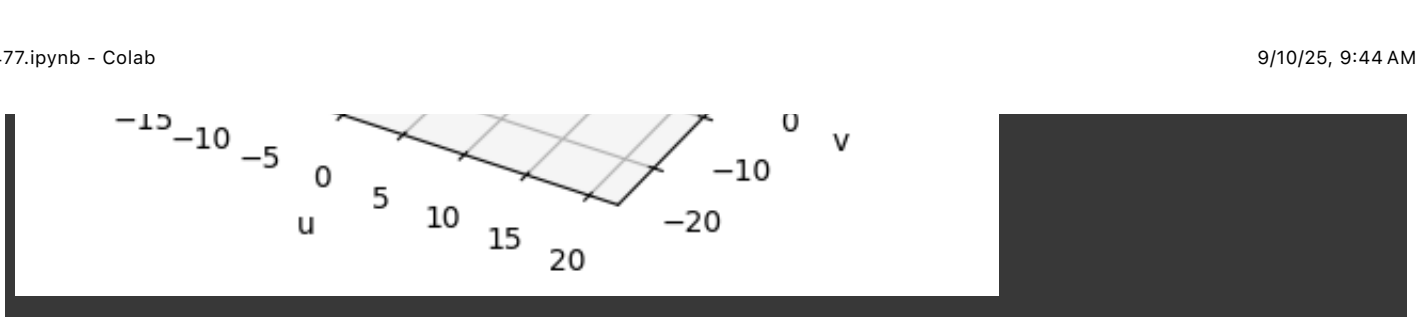
Lorenz system computational demo - (Q4)

```
# u' = sigma*(v - u)
# v' = u*(rho - w) - v
# w' = u*v - beta*w

from numpy import linspace, array, stack
from scipy.integrate import solve_ivp
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# --- parameters---
sigma = 10
rho   = 28
beta  = 8/3
```

```
# --- ODE system ---
def lorenz(t, z):
    u, v, w = z
    du = sigma*(v - u)
    dv = u*(rho - w) - v
    dw = u*v - beta*w
    return stack([du, dv, dw])

# --- initial condition & time span ---
z0 = array([1, 1, 1])
t0, tf = 0, 40

sol = solve_ivp(lorenz, array([t0, tf]), z0, dense_output=True, rtol=1e-8, atol=1
assert sol.success, "ODE solver failed."

# --- sample solution ---
t = linspace(t0, tf, 6000)
u = sol.sol(t)[0, :]
v = sol.sol(t)[1, :]
w = sol.sol(t)[2, :]

# --- Plot 1: time series ---
fig1, ax1 = plt.subplots()
ax1.plot(t, u, label="u(t)")
ax1.plot(t, v, label="v(t)")
ax1.plot(t, w, label="w(t)")
ax1.set_xlabel("time")
ax1.set_ylabel("state")
ax1.set_title("Lorenz system: time series (σ=10, ρ=28, β=8/3)")
ax1.legend(loc="upper right")
fig1.tight_layout()

# --- Plot 2: 3D Lorenz attractor ---
fig2 = plt.figure()
ax2 = fig2.add_subplot(111, projection="3d")
ax2.plot(u, v, w)
ax2.set_xlabel("u")
ax2.set_ylabel("v")
ax2.set_zlabel("w")
ax2.set_title("Lorenz attractor (σ=10, ρ=28, β=8/3)")
fig2.tight_layout()
```


Lorenz system: time series (σ=10, ρ=28, β=8/3)
u(t)
v(t)
w(t)
state
time
50
40
30
20
10
0
−10
−20
0
5
10
15
20
25
30
35
40


Lorenz attractor (σ=10, ρ=28, β=8/3)
40
30
20
10
0
30
20
10
−20

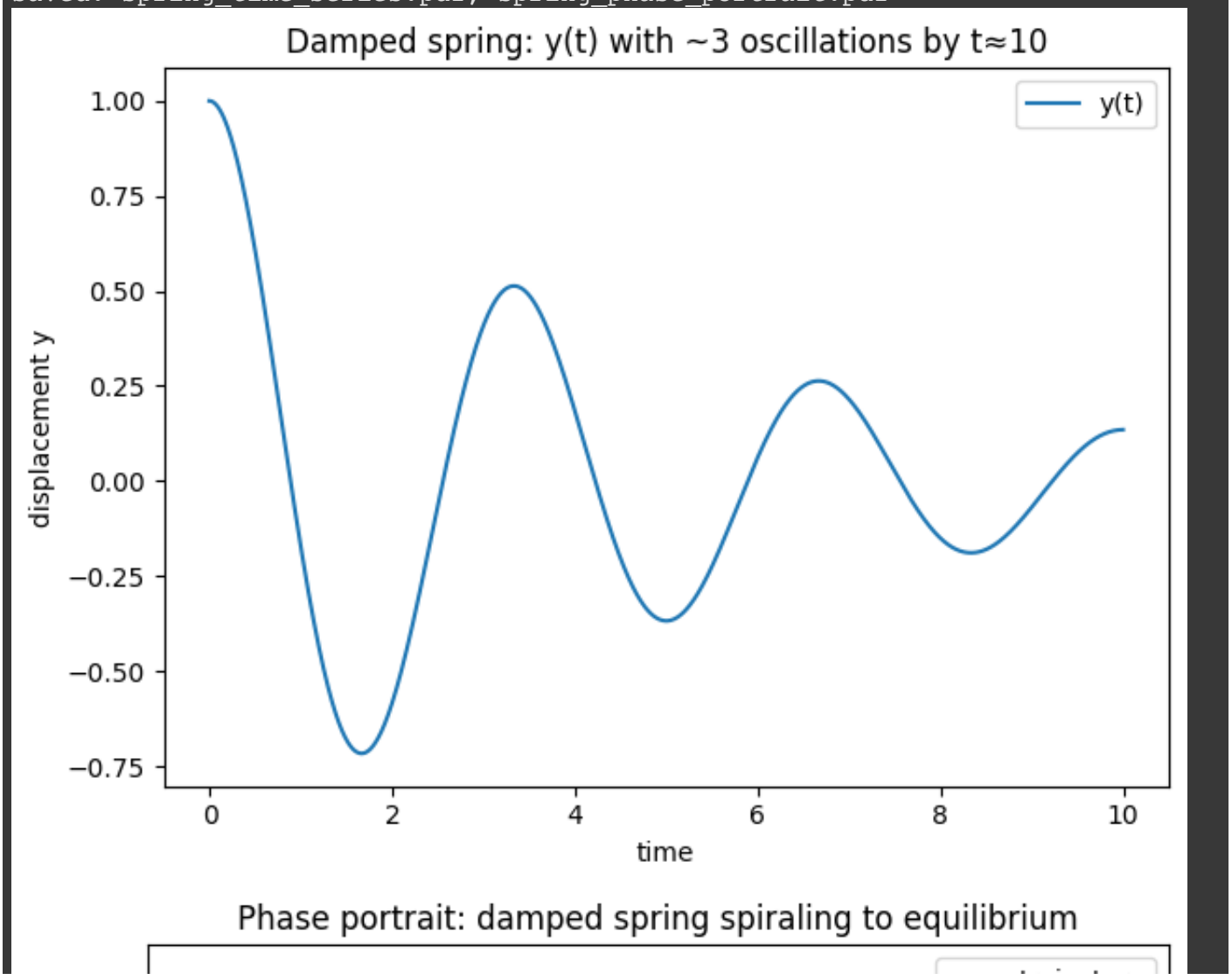


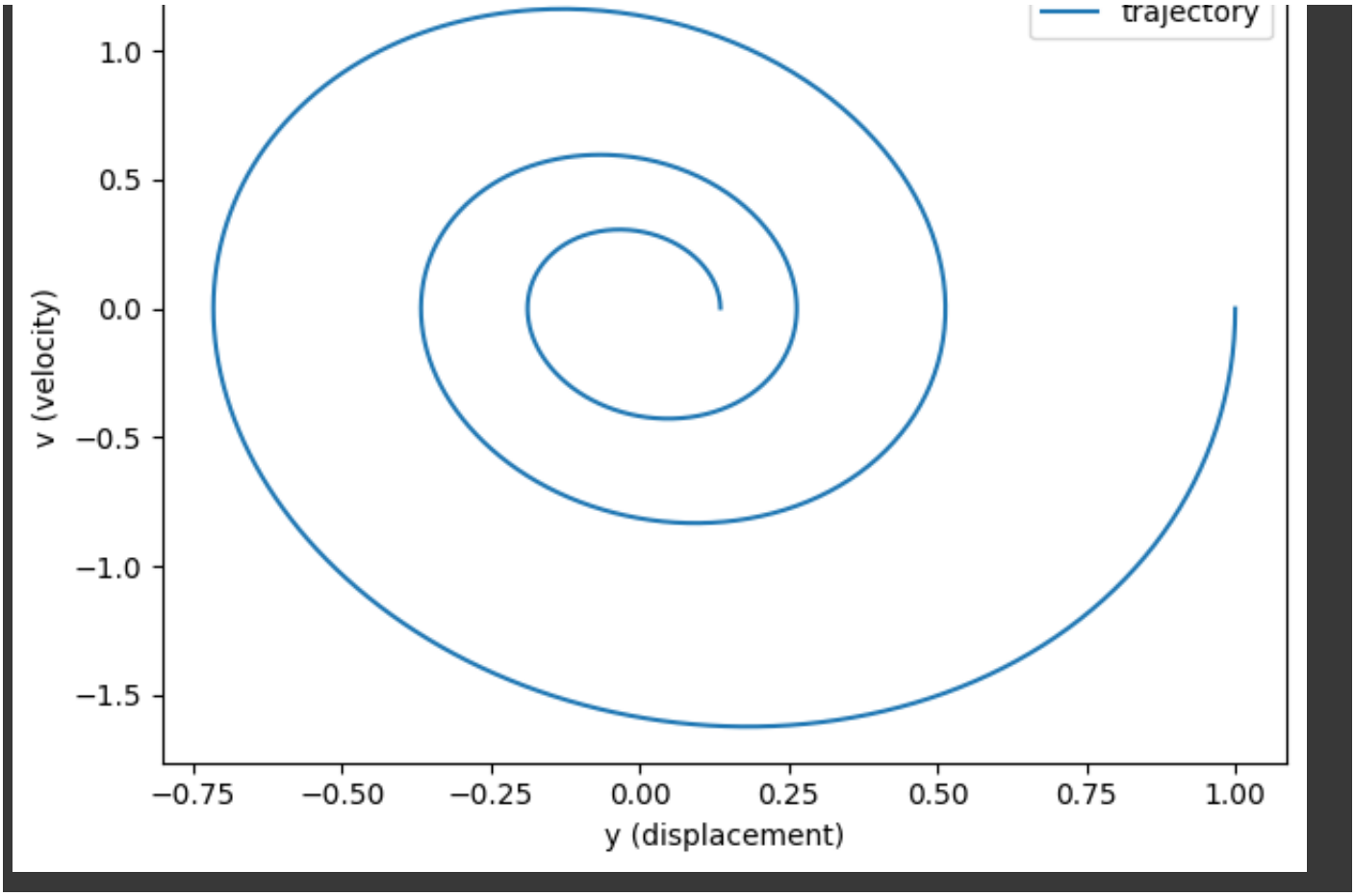
Spring equation as a first-order system (q3)

```
# y' = v
# v' = −(r/m)∗v − (k/m)∗y

from numpy import linspace, array, stack, meshgrid, sqrt
from scipy.integrate import solve_ivp
from matplotlib import pyplot as plt

# --- parameters ---
m = 1
r = 0.4
k = 3.59   # ≈ (2π/3.33)^2 + (r/2)^2

# --- system ---
def spring(t, z):
    y, v = z
    dy = v
    dv = −(r/m)∗v − (k/m)∗y
    return stack([dy, dv])

# --- initial condition ---
y0, v0 = 1, 0
t0, tf = 0, 10
z0 = array([y0, v0])

sol = solve_ivp(spring, array([t0, tf]), z0, dense_output=True, rtol=1e−8, atol=1
assert sol.success, "ODE solver failed."

# --- sample ---
t = linspace(t0, tf, 1200)
y = sol.sol(t)[0, :]
v = sol.sol(t)[1, :]

# Plot 1: displacement over time
fig1, ax1 = plt.subplots()
ax1.plot(t, y, label="y(t)")
```

```
ax1.set_xlabel("time")
ax1.set_ylabel("displacement y")
ax1.set_title("Damped spring: y(t) with ~3 oscillations by t≈10")
ax1.legend()
fig1.tight_layout()

# Plot 2: phase portrait (y vs v)
fig2, ax2 = plt.subplots()
ax2.plot(y, v, label="trajectory")
ax2.set_xlabel("y (displacement)")
ax2.set_ylabel("v (velocity)")
ax2.set_title("Phase portrait: damped spring spiraling to equilibrium")
ax2.legend()
fig2.tight_layout()
```

```
Saved: spring_time_series.pdf, spring_phase_portrait.pdf
```




trajectory
1.0
0.5
0.0
−0.5
−1.0
−1.5
v (velocity)
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
y (displacement)