1 Problem 1

In this section, the solutions to the questions around problem 1 is going to be presented. The system that is analyzed is the following:

$$\dot{N}(t) = rN(t)\left(1 - \frac{N(t-T)}{K}\right)\left(\frac{N(t)}{A} - 1\right) \tag{1}$$

The simulations of this system was ran with the following parameters for all the solutions: $A = 20$, $K = 100$, $r = 0.1$ and $N_0 = 50$

1.1 a)

Here examples of trajectories with the characteristics: ”no oscillations”, ”damped oscillations” and ”limit cycle” is shown.

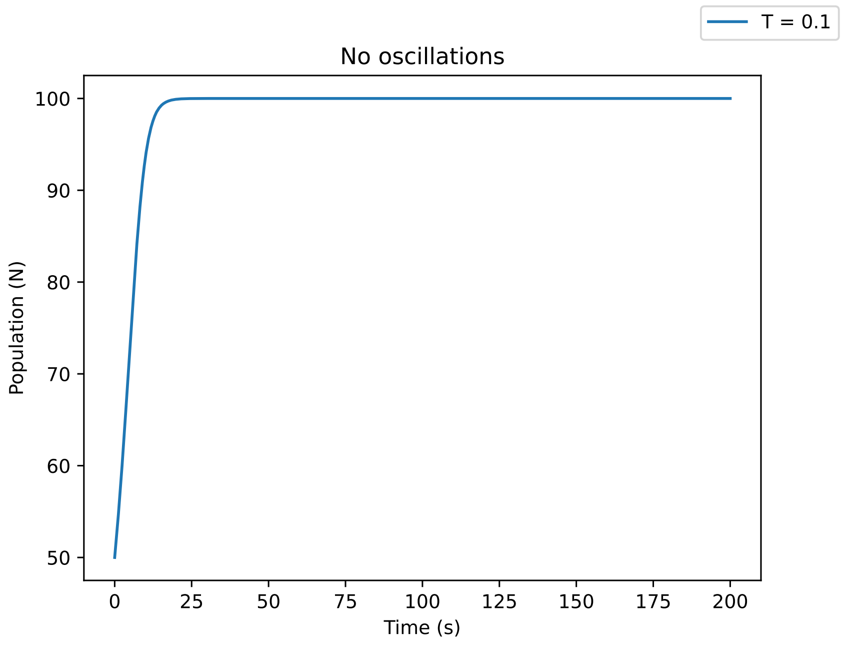

Figure 1: Example of a trajectory with no oscillations. In the figure we can see that the trajectory goes directly to 100 without no oscillations and stays there. The value of T for this run was 0.1

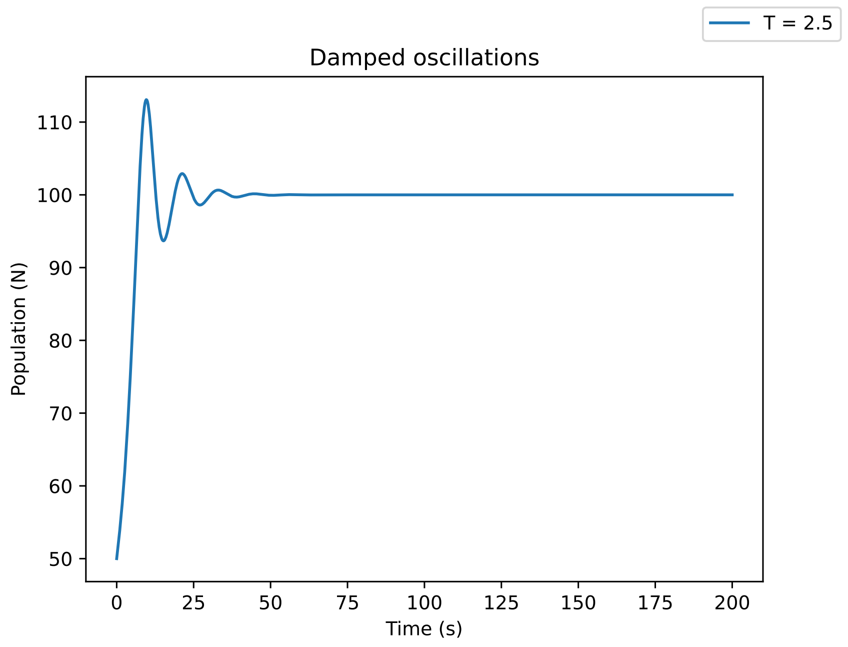

Figure 2: Example of a trajectory with damped oscillations. We can se that is starts to oscillate at the begining around the value 100 but starts to decrease in size and after 50 generations the oscillations are almost nonexistant. The T - value for this run where set to 2.5.

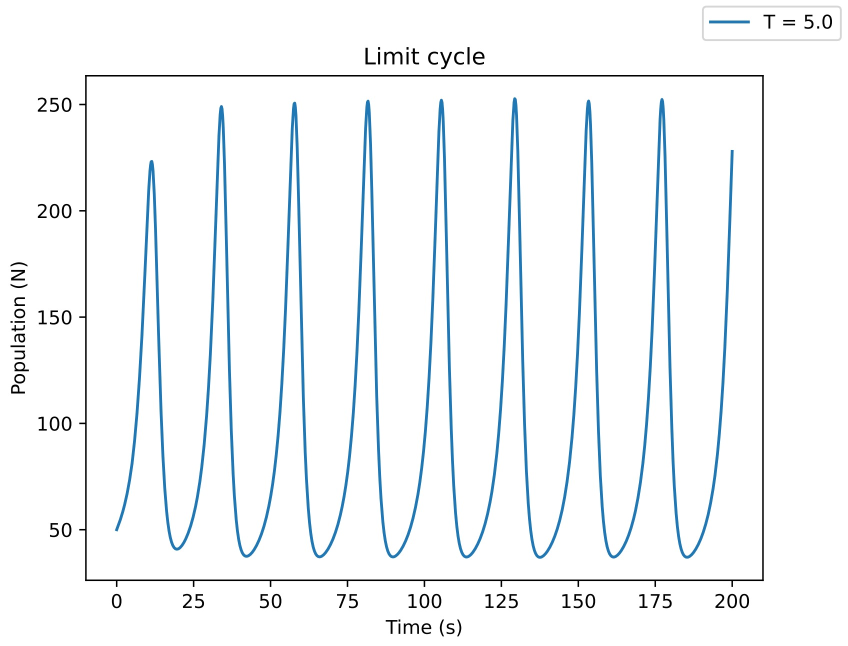

Figure 3: Example of a trajectory that creates a limit cycle. It starts to oscillate almost imediatly and keeps on going even after 200 generations. The value of T where 5.0 for this run.

1.2 b)

In this part the value of T where the no oscillation solutions went trough bifurcation to damped oscillations instead. To analyze this numerically we set a parameter called $\epsilon = 10^{-3}$ that we use as a thresh hold. To count as a oscillation we require the curve to first go over the midpoint = 100, and then also go under the midpoint with a gap larger than ϵ so that a full period is achived. The trajectory with the smallest T to achive this was $T = 1.2$ and this can be seen in figure 4.

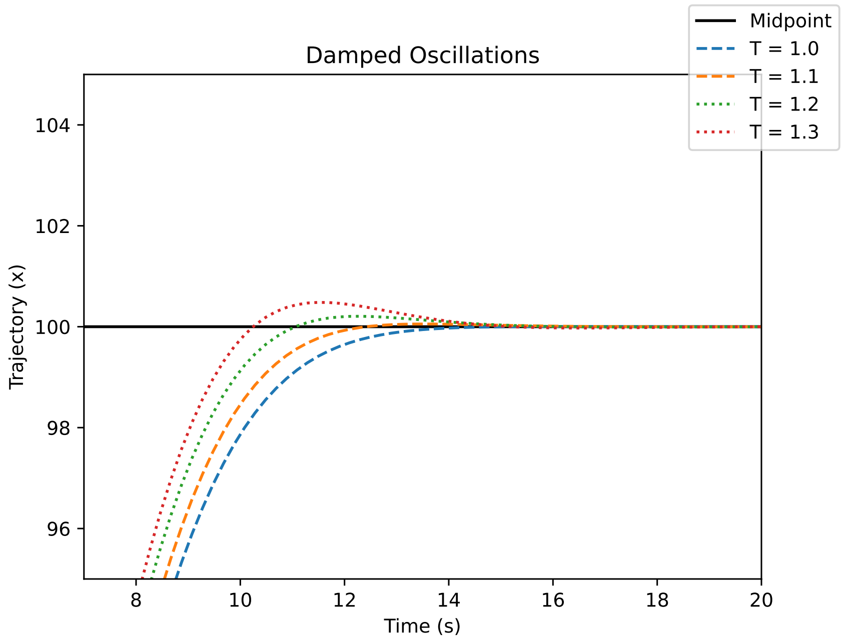

Figure 4: Figure over the trajectory with the smallest T to go from no oscillations to damped oscillations. The doted lines are trajectories that achive the criterion described above and therefore counts as damped oscillations and the dashed lines are no oscillations.

1.3 c)

In this part the value of T when the trajectory goes from damped oscillations to stable limit cycles is to be numerically estimated. For this to be done, another criterion for limit cycles is to be created. We choose to look at the peaks of every oscillation and fit a line to these points. Then the value of the slope of this line was compared to ϵ and if it wasless than it the trajectory was determined to be a limit cycle. To get more accurate values of this, the number of generations was increased to get more accurate values of the slope. The result of this was that $T = 4.0$ was the smallest value that created a stable oscillation around the midpoint, and graph from where this result was taken from can be seen in figure 5.

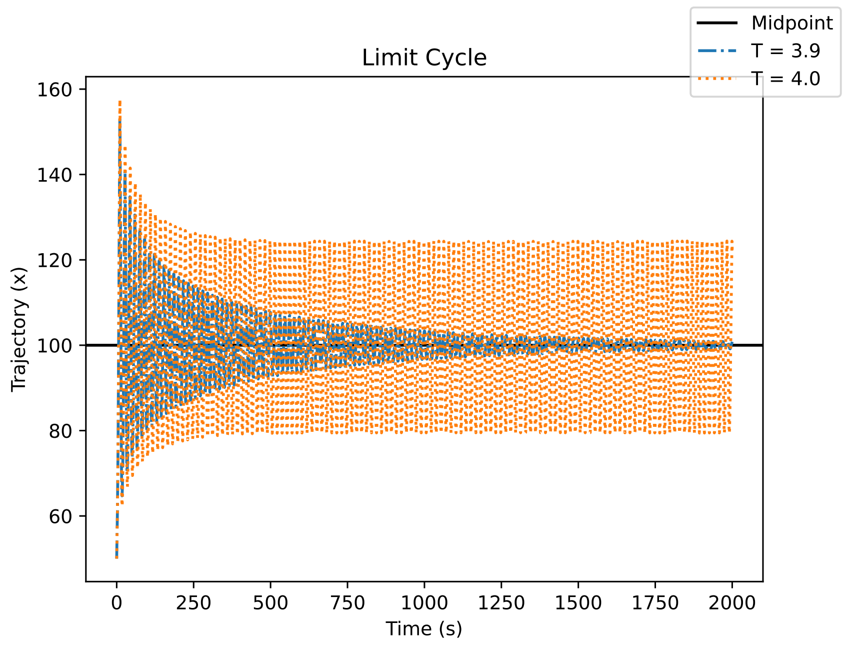

Figure 5: In this figure the smallest T where the trajectory goes from damped oscillations to a stable limit cycle can be refered.

1.4 d)

In this part the exact value of when the trajectory goes from damped oscillations to stable is to be analyticaly computed. This is done by a linear stability analyze and the computation can be found below.

We start by analyzing the state around the steady state $N^* = K$ and from there say the following:

$$N(t) = N^* + \eta(t) = K + \eta(t) \tag{2}$$

where $\eta(t) \propto exp(\lambda t)$. If we take the derivative of $N(t)$ with respect to t we get the following expression:

$$\dot{N}(t) = \frac{dN^*}{dt} + \frac{d\eta}{dt} = \lambda A exp(t\lambda) \tag{3}$$

If we insert this into equation 1, we get the following equation:

$$\lambda A exp(\lambda t) = r\left(N^* + \eta(t)\right)\left(1 - \frac{N^* + \eta(t)}{K}\right)\left(\frac{N^* + \eta(t)}{A} - 1\right) \tag{4}$$

with some simplification we get the following:

$$-\lambda exp(T\lambda) = \frac{2}{5} \tag{5}$$

Now we can use that exacly in the bifurcation we have that the eigenvalue goes from negative to positive so this means that $\lambda = i\omega$. If we put this into the equation we get two equation systems:

$$\omega T = \frac{1}{2}\pi \tag{6}$$

$$\omega = \frac{2}{5} \tag{7}$$

This gives the result:

$$T = \frac{5}{4}\pi \approx 3.93 \tag{8}$$

2 Problem 2

2.1 a)

In this section the steady states is to be determined analytically. We start with the condition for stable states in discrete systems:

$$N_{t+1} = N_t = \frac{(1+r)N}{1+N_t/K} \tag{9}$$

solving this equation gives the two solutions:

$$N_1^* = 0, N_2^* = Kr^{1/b} \tag{10}$$

2.2 b)

For this section a linear stability analyze is done on the system around the two stable points to determine the eigenvalues and how they depend on the parameters. We start with the following:

$$N_t = N^* + \epsilon_t \tag{11}$$

and we have also:

$$N_{t+1} = N^* + \epsilon_{t+1} \tag{12}$$

From this equation we define the eigenvalue as:

$$\Lambda = \frac{e_{t+1}}{e_t} \tag{13}$$

We start with the case $N^* = 0$ and solve equation 9 with substitution with equation 11 and 12 and get the following result:

$$\epsilon_{t+1}\left(1 + (\frac{\epsilon_t}{K})^b\right) = (r+1)\epsilon_t \tag{14}$$

because $\epsilon_t, e_{t+1} << 1$, the $(\frac{\epsilon_t}{K})^b$ term can be approximated to 0 because of $b \geq 1$. This can then be calculated to the following:

$$\Lambda_1 = \frac{\epsilon_{t+1}}{\epsilon_t} = r + 1 \tag{15}$$

For the case $N^* = Kr^{1/b}$ we get, after some cancellation the following equation:

$$\epsilon_{t+1} = \frac{(r+1)(Kr^{1/b} + \epsilon_t) - Kr^{1/b}(1 + (r^{1/b} + \frac{\epsilon_t}{K})^b)}{r+1} \tag{16}$$

For the same reason as before we can approximate $(r^{1/b} + \frac{\epsilon_t}{K})^b) \approx r(1 + \frac{b\epsilon_t}{Kr^{1/b}})$ because of higher terms is approximately zero in comparison. Therefore we get the following expression:

$$\epsilon_{t+1} = \frac{(r+1)(Kr^{1/b} + \epsilon_t) - (Kr^{1/b} + Kr^{1+1/b} + b\epsilon_t r)}{r+1} \tag{17}$$

with some cancellation we get the following answer:

$$\Lambda_2 = \frac{\epsilon_{t+1}}{\epsilon_t} = 1 - \frac{br}{r+1} \tag{18}$$

2.3 c)

Let

$$f(N) = \frac{(1+r)N}{1+(N/K)^b}, \qquad K > 0,\ r > 0,\ b \geq 1.$$

A fixed point N^* changes stability only when $|\lambda| = 1$ with $\lambda = f'(N^*)$.
Further, we have:

$$N = \frac{(1+r)N}{1+(N/K)^b} \;\Rightarrow\; N = 0 \quad \text{or} \quad 1+(N/K)^b = 1+r \Rightarrow (N/K)^b = r \Rightarrow N_+^* = Kr^{1/b}.$$

Differentiate

$$f(N) = (1+r)N\left(1+(N/K)^b\right)^{-1}:$$

$$f'(N) = (1+r)\left[\frac{1}{1+(N/K)^b} - \frac{b(N/K)^b}{\left(1+(N/K)^b\right)^2}\right].$$

Hence

$$\lambda_0 = f'(0) = (1+r) > 1 \quad \Rightarrow \quad N_0^* = 0 \text{ is always unstable (no bifurcation for } r > 0).$$

At N_+^*, use $(N_+^*/K)^b = r$:

$$\lambda_+ = f'(N_+^*) = (1+r)\left[\frac{1}{1+r} - \frac{br}{(1+r)^2}\right] = 1 - \frac{br}{1+r}.$$

Stability loss requires $|\lambda_+| = 1$. Since $\lambda_+ < 1$ for $r > 0$, the only possibility is $\lambda_+ = -1$:

$$1 - \frac{br}{1+r} = -1 \;\Rightarrow\; \frac{br}{1+r} = 2 \;\Rightarrow\; br = 2(1+r) \;\Rightarrow\; r(b-2) = 2.$$

Therefore a finite critical value exists only for $b > 2$:

$$r_c = \frac{2}{b-2} \quad (b > 2).$$

For $1 \leq b < 2$ there is no solution with $r > 0$ (so N_+ stays stable), and for $b = 2$ no finite-r solution exists.

2.4 d)

For $b = 1$ the discrete growth model reads

$$N_{t+1} = f(N_t) = \frac{(1+r)N_t}{1+N_t/K}, \qquad K = 10^3,\; r = 0.1. \tag{19}$$

The unstable steady state is $N_u^* = 0$. Writing $N_t = N_u^* + \eta_t = \eta_t$ and linearizing gives

$$\eta_{t+1} \approx f'(0)\,\eta_t = (1+r)\eta_t, \qquad \Rightarrow \qquad N_t^{\text{lin}} \approx (1+r)^t N_0 = 1.1^t N_0. \tag{20}$$

Figure 6 compares the exact iteration of (19) with the linear approximation (20) for $N_0 \in \{1, 2, 3, 10\}$, plotted on log–log axes as requested.

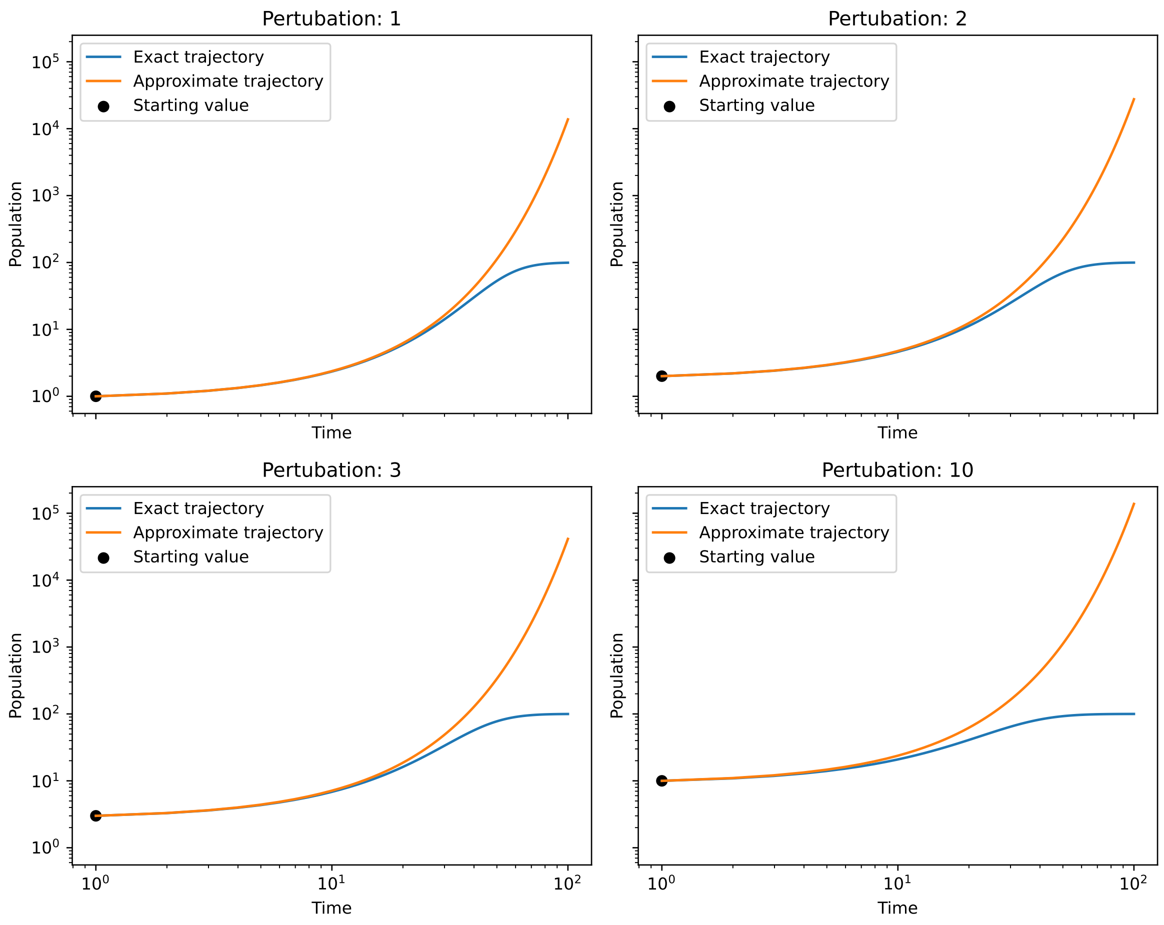

Figure 6: Part (d): Exact trajectories from (19) compared to the linear stability approximation near the unstable steady state $N_u^* = 0$, i.e. $N_t^{\text{lin}} = 1.1^t N_0$, for $N_0 = 1, 2, 3, 10$. Both axes are in log scale.

2.5 e)

Figure 6 shows that the linear approximation around $N_u^* = 0$ is accurate only at early times. This is expected because the linearization assumes $N_t \ll K$, so that $1 + N_t/K \approx 1$ and the nonlinear map (19) is well-approximated by $N_{t+1} \approx (1+r)N_t$. As N_t grows, the denominator in (19) reduces the effective growth rate, so the exact trajectories bend away from exponential growth and approach the stable equilibrium near $N \approx 100$, while the linear model (20) continues to grow exponentially. The initial condition determines how long the system remains in the neighborhood where the linearization is valid: larger N_0 leaves the vicinity of 0 sooner, hence the deviation between exact and linear curves appears earlier (most clearly for $N_0 = 10$) than for smaller N_0.

2.6 f)

The stable steady state for $b = 1$ is

$$N_s^* = rK = 0.1 \cdot 10^3 = 100. \tag{21}$$

Let $N_t = N_s^* + \eta_t$ with $\eta_t = N_t - N_s^*$. Linear stability analysis gives

$$\eta_{t+1} \approx f'(N_s^*)\,\eta_t, \qquad f'(N) = \frac{1+r}{(1+N/K)^2} \;\Rightarrow\; \lambda_s := f'(N_s^*) = \frac{1}{1+r} = \frac{1}{1.1} \approx 0.909. \tag{22}$$

Thus, for $N_0 = N_s^* + \delta N_0$ the linear approximation is

$$N_t^{\text{lin}} \approx N_s^* + \lambda_s^t \delta N_0 = 100 + (0.909)^t \delta N_0. \tag{23}$$

Figure 7 shows that the approximation is generally very accurate once the trajectory is close to N_s^*, with best agreement for small $|\delta N_0|$. The largest perturbations ($|\delta N_0| = 10$) can show small early-time discrepancies due to nonlinear effects away from the fixed point, but convergence toward N_s^* is captured well since $0 < \lambda_s < 1$.

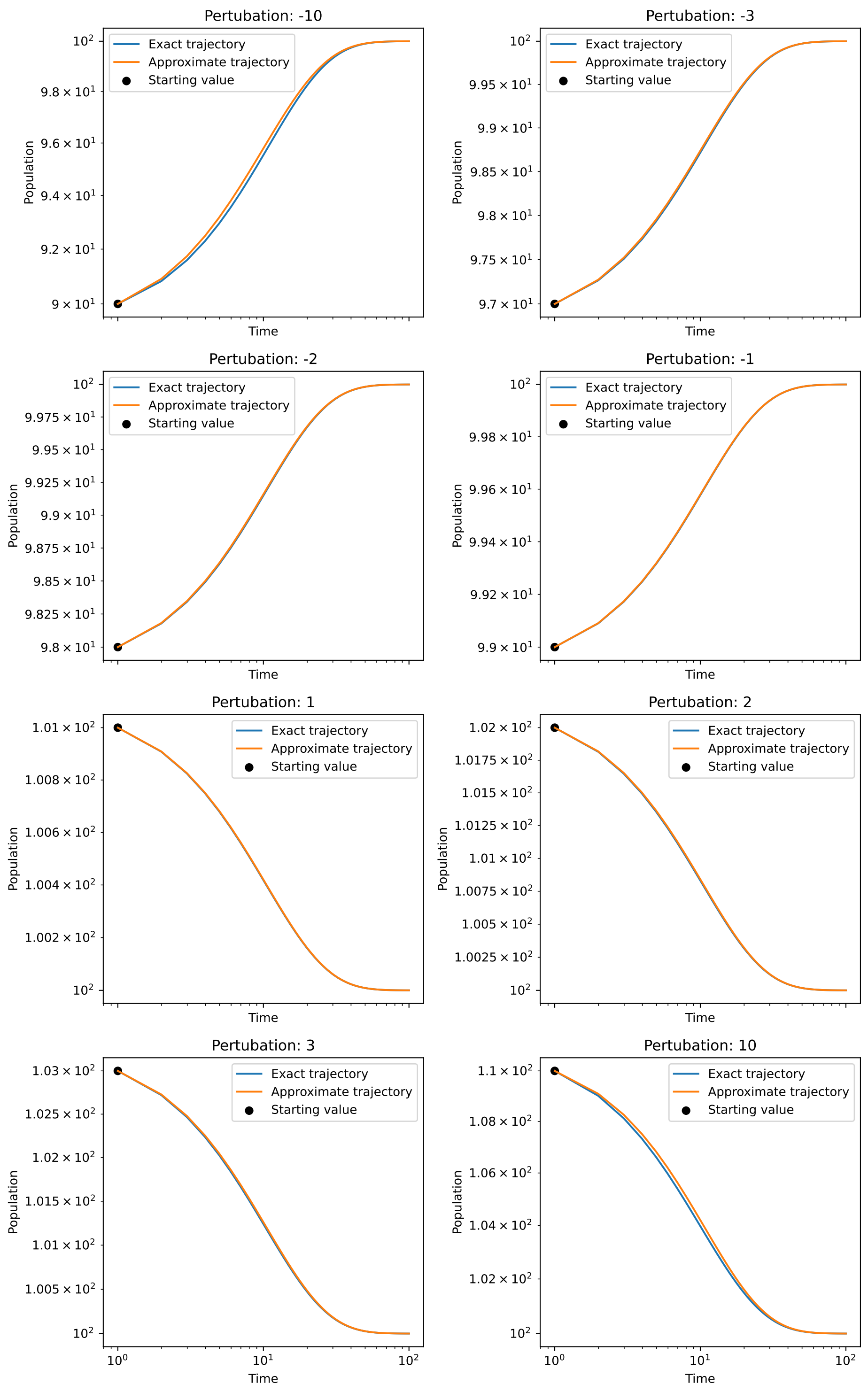

Figure 7: Part (f): Exact trajectories from (19) compared to the linear stability approximation near the stable steady state $N_s^* = 100$, i.e. $N_t^{\text{lin}} = 100 + \lambda_s^t \delta N_0$ with $\lambda_s \approx 0.909$, for $\delta N_0 \in \{-10, -3, -2, -1, 1, 2, 3, 10\}$. Both axes are in log scale.

3 Problem 3

We study the Ricker map

$$\eta_{t+1} = R\,\eta_t\,e^{-\alpha\eta_t}, \qquad t = 0, 1, 2, \ldots, \tag{24}$$

with $\alpha = 0.01$ and initial condition $\eta_0 = 900$. The parameter R controls the reproduction rate, while the factor $e^{-\alpha\eta_t}$ represents offspring survival in the presence of cannibalism.

3.1 a

A bifurcation diagram was constructed by sweeping $R \in [1, 30]$ in steps of 0.1. For each value of R, the map was iterated for 300 generations and the last 100 values of η_t were plotted against R (to remove transients).

Figure 8 shows the classic route to chaos via period-doubling. For small R, the dynamics converges to a single stable equilibrium (one point per R). As R increases, the equilibrium loses stability and the attractor bifurcates to a stable 2-cycle, further increases lead to successive period doublings (4-cycle, 8-cycle, etc.). After an accumulation point, the dynamics becomes chaotic, visible as a vertical "cloud" of values for a given R. Within the chaotic regime there are periodic windows after which chaos returns.

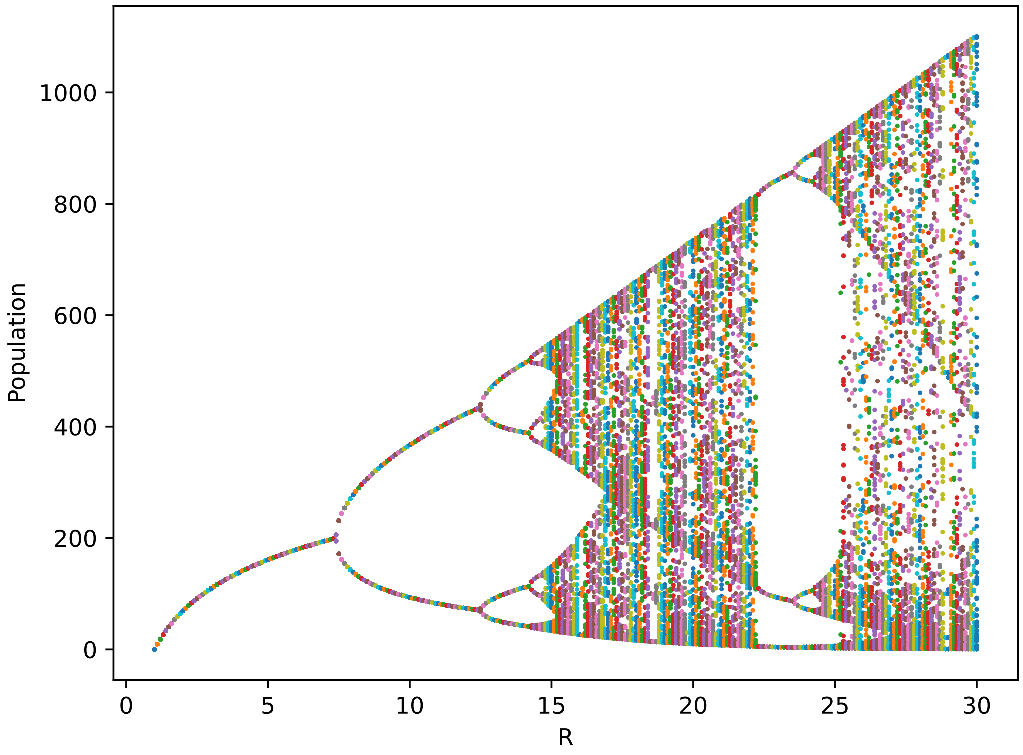

Figure 8: Bifurcation diagram of the Ricker map for $\alpha = 0.01$: for each R, the last 100 iterates (after 300 total generations) are plotted. The diagram shows period-doubling cascades leading to chaos and periodic windows inside chaos.

3.2 b

To illustrate the qualitative regimes, η_t was plotted versus generation t for $t = 0, \ldots, 40$ for four representative values of R (Figure 9).

For $R = 5$, the population rapidly converges to a stable fixed point (the curve settles to a constant level). For $R = 10$, after a short transient the trajectory alternates between two distinct levels, indicating a stable 2-point cycle. For $R = 13$, the long-term dynamics cycles through four distinct levels, consistent with a stable 4-point cycle. For $R = 23$, the dynamics exhibits a stable 3-point cycle with large amplitude: the population repeatedly visits three characteristic levels, including one very small value and one very large peak (a consequence of strong overcompensation at large R).

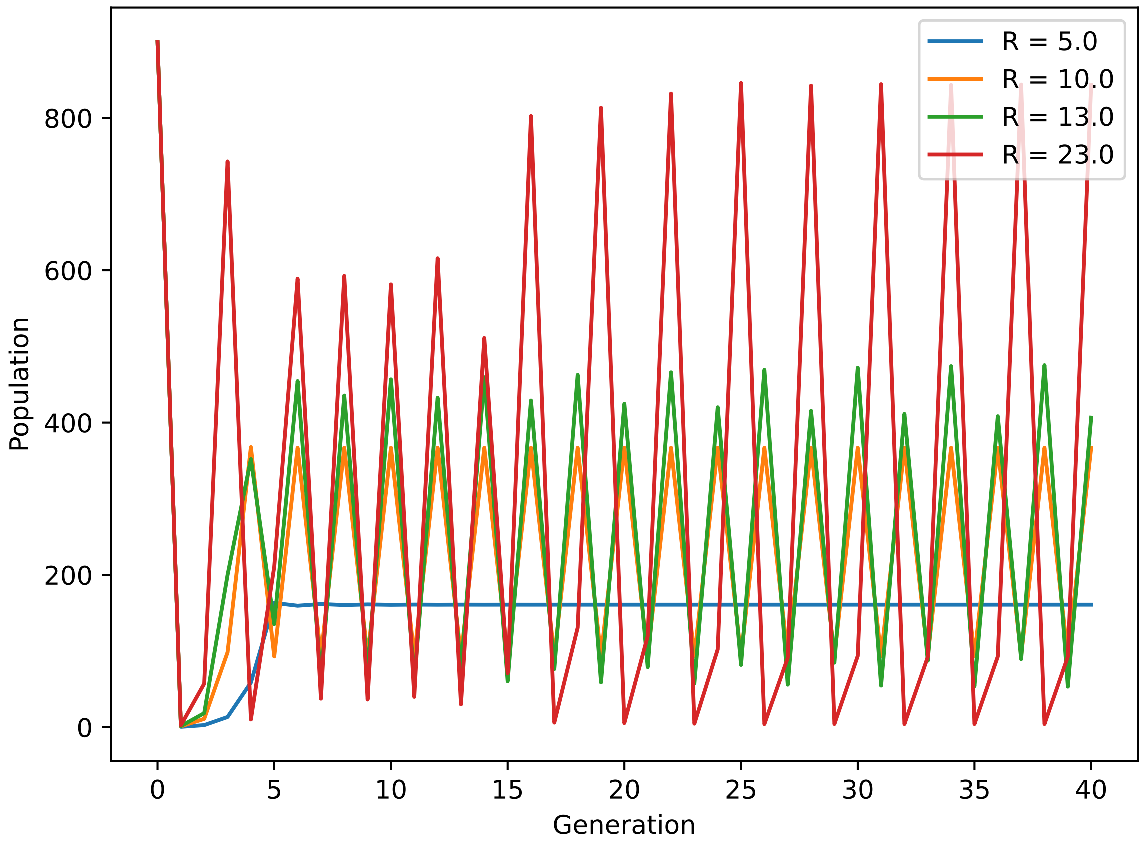

Figure 9: Population dynamics η_t versus generation t for four representative values of R: stable fixed point ($R = 5$), stable 2-cycle ($R = 10$), stable 4-cycle ($R = 13$), and stable 3-cycle ($R = 23$).

3.3 c

From the bifurcation diagram, the first bifurcation from a stable equilibrium to a stable 2-cycle occurs when the single branch splits into two. A refined zoom in the range $R \in [7.0, 8.0]$ (Figure 10) shows that the split occurs between $R = 7.3$ and $R = 7.4$. Hence, a reasonable estimate is

$$R_1 \approx 7.4. \tag{25}$$

The next requested transition is from a stable 2-cycle to a stable 4-cycle. A refined zoom in the range $R \in [12.0, 13.0]$ (Figure 11) shows that two branches split into four around $R = 12.5$. Hence, we estimate

$$R_2 \approx 12.5. \tag{26}$$

(These values are approximate and limited by the finite R-grid resolution.)

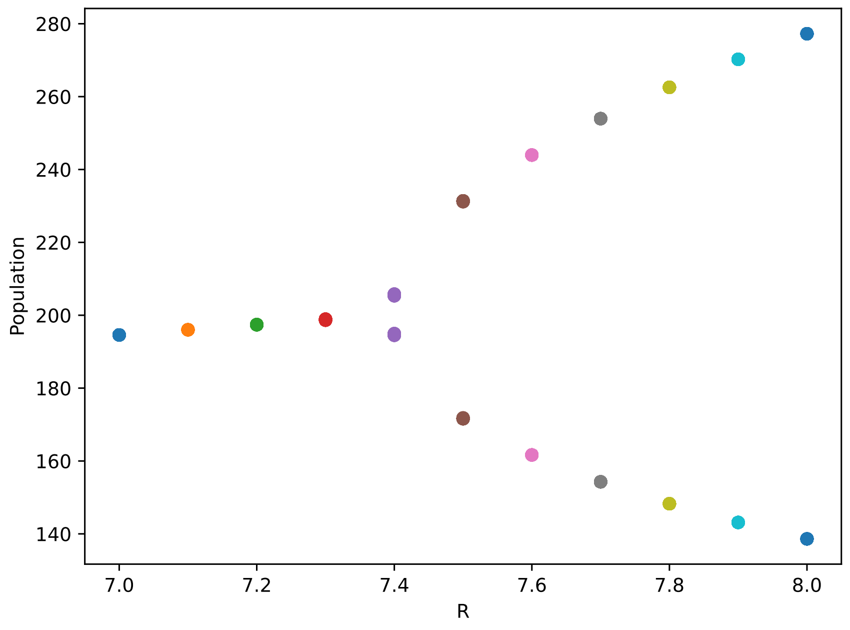

Figure 10: Zoom around the first period-doubling: the stable fixed point splits into two branches between $R = 7.3$ and $R = 7.4$, giving $R_1 \approx 7.4$.

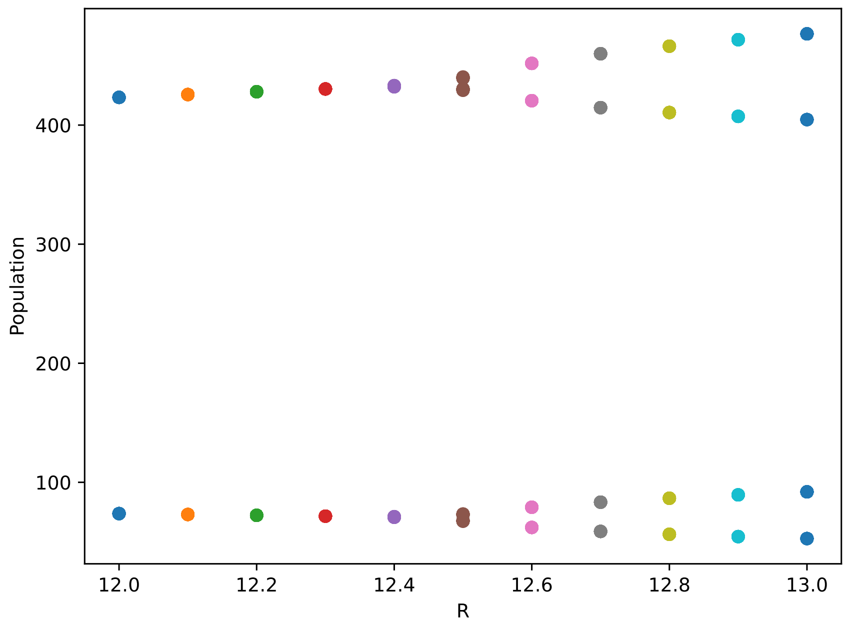

Figure 11: Zoom around the second period-doubling: the stable 2-cycle splits into a stable 4-cycle around $R = 12.5$, giving $R_2 \approx 12.5$.

3.4 d

To estimate R_∞, the R-grid was refined around the region where successive period doublings accumulate and the number of distinct points grows too large to resolve. In the zoomed bifurcation plot around $R \in$

$[14.74, 14.79]$ (Figure 12), the branches become extremely dense as R approaches ≈ 14.77–14.78, and beyond this the attractor starts to appear band-like rather than consisting of clearly separated discrete levels. This indicates that the period-doubling cascade has essentially accumulated.

A rough estimate is therefore

$$R_\infty \approx 14.77 \tag{27}$$

The estimate is obtained by locating the smallest R for which further splittings can no longer be resolved (branches crowd together), signalling the onset of an effectively infinite-period attractor and the transition to chaos.

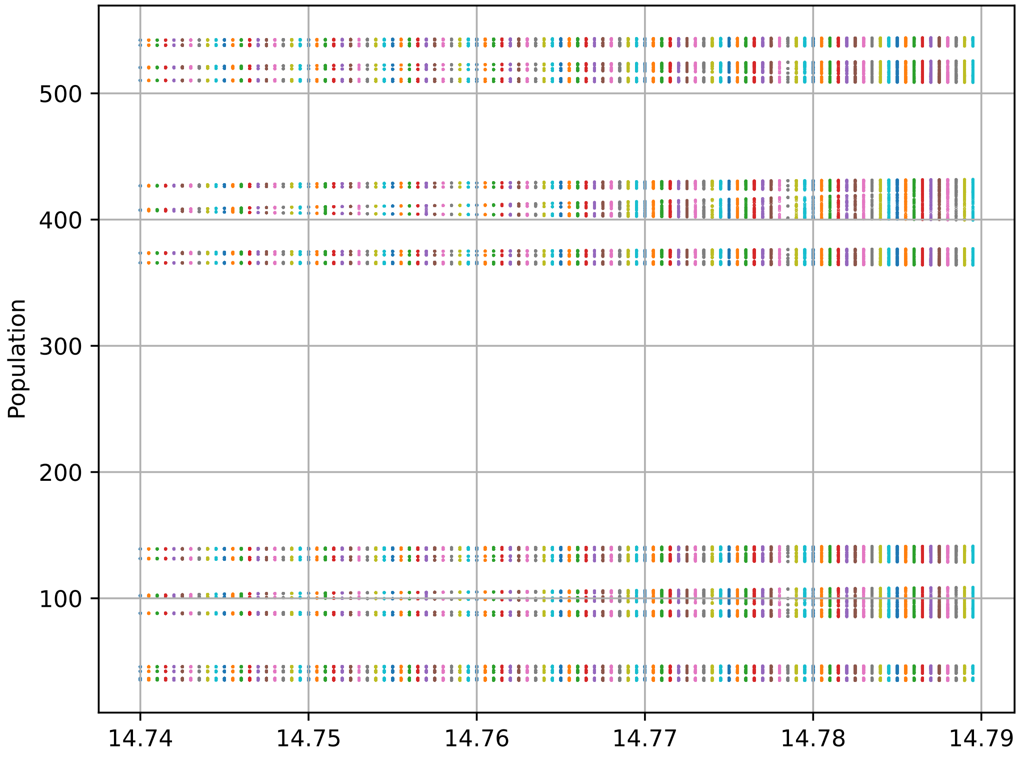

Figure 12: Zoom near the accumulation region: successive period doublings crowd together and become difficult to resolve around $R \approx 14.77$–14.78, motivating the estimate $R_\infty \approx 14.77$.

4 Code

4.1 Problem1

```
import numpy as np
import matplotlib.pyplot as plt
from ddeint import ddeint


def n_prim(
    n,
    t: float,
    T: float,
    A: float = 20.0,
    K: float = 100.0,
    r: float = 0.1,
) -> float:
    n_prim = r * n(t) * (1 - n(t - T) / K) * (n(t) / A - 1)
    return n_prim


def before_zero(t, n_0=50.0):
    return n_0


def check_if_oscillation(
    trajectory: np.ndarray, mid_point: float, thresh_hold: float = 1e-3
) -> bool:
    if np.max(trajectory) - mid_point > thresh_hold:
        index = np.argmax(trajectory)
        print(np.min(trajectory[index:]))
        if mid_point - np.min(trajectory[index:]) > thresh_hold:
            return True
        else:
            return False
    else:
        return False


def check_if_limit_cycle(trajectory: np.ndarray, thresh_hold: float = -1e-3) -> bool:
    index = []
    peaks = []

    for idx, point in enumerate(trajectory):
        if (idx != 0) and (idx != trajectory.shape[0] - 1):
            if (trajectory[idx - 1] < point) and (trajectory[idx + 1] < point):
                index.append(idx)
                peaks.append(point)
    k = np.polyfit(index, peaks, 1)[0]
    print(k)

    if k < thresh_hold:
        return True
    else:
        return False


def run_integration(
    different_T: np.ndarray,
    total_time: float,
    check_oscillation=False,
    check_limit_cycle=False,
):
    tt = np.linspace(0, total_time, 5 * int(total_time))
    fig, ax = plt.subplots()
    # ax.axhline(100.0, color="black", label="Midpoint")
    for T in different_T:
        print(T)
        yy = ddeint(n_prim, before_zero, tt, fargs=(T,))
        linestyle = "solid"

        if check_oscillation:
            ax.set_ylim(95, 105)
            ax.set_xlim(7, 20)
            oscillation = check_if_oscillation(yy, 100.0)
            if oscillation:
                linestyle = "dotted"
            else:
                linestyle = "dashed"

        if check_limit_cycle:
            limit_cycle = check_if_limit_cycle(yy)
            if limit_cycle:
                linestyle = "dashdot"
            else:
                linestyle = "dotted"

        ax.plot(tt, yy, label=f"T = {T:.1f}", linestyle=linestyle)
    ax.set_xlabel("Time (s)")
    ax.set_ylabel("Population (N)")
    ax.set_title("Limit cycle")
    fig.legend()
    fig.savefig("../report/figures/problem1/stable_oscillation.pdf")


def main():

    T = np.arange(5.0, 5.05, 0.1)
    total_runtime = 200.0
    run_integration(T, total_runtime, check_limit_cycle=False)


if __name__ == "__main__":
    main()
```

4.2 Problem2

```
    import numpy as np
import matplotlib.pyplot as plt


def get_next_pertubation(eta, k: float, r: float, b: float, stable=False):
    if stable:

        eta_next = (1 - b * r / (1 + r)) * eta
    else:
        eta_next = (r + 1) * eta

    return eta_next


def get_exact_step(n, k: float, r: float, b: float) -> float:
    n_next = (r + 1) * n / (1 + (n / k) ** b)
    return n_next


def get_exact_trajectory(
    time_steps: int, n_0, k: float, r: float, b: float
) -> np.ndarray:
    trajectory = np.zeros(time_steps + 1)
    trajectory[0] = n_0

    for idx in range(time_steps):
        trajectory[idx + 1] = get_exact_step(trajectory[idx], k, r, b)

    return trajectory


def get_approx_trajectory(
    time_steps: int,
    k: float,
    r: float,
    b: float,
    stable_point: float,
    pertubation,
    stable=False,
) -> np.ndarray:
    eta = np.zeros((time_steps + 1))
    trajectory = np.zeros((time_steps + 1))
    trajectory[0] = pertubation + stable_point
    eta[0] = pertubation

    for idx in range(time_steps):
        eta[idx + 1] = get_next_pertubation(eta[idx], k, r, b, stable=stable)
        trajectory[idx + 1] = stable_point + eta[idx + 1]

    return trajectory


def part1():
    number_of_timesteps = 100
    stable_point = 0
    pertubation = np.array([1, 2, 3, 10])
    start_pos = stable_point + pertubation
    k = 1e3
    r = 0.1
    b = 1

    fig, ax = plt.subplots(2, 2, figsize=(10, 8), sharey=True, sharex=True)
    ax = ax.flatten()

    time = np.linspace(1, number_of_timesteps, number_of_timesteps + 1)

    for idx, pert in enumerate(pertubation):
        traj_exact = get_exact_trajectory(number_of_timesteps, start_pos[idx], k, r, b)
        traj_approx = get_approx_trajectory(
            number_of_timesteps, k, r, b, stable_point, pert
        )
        ax[idx].plot(time, traj_exact, label="Exact trajectory")
        ax[idx].plot(time, traj_approx, label="Approximate trajectory")
        ax[idx].scatter(1, start_pos[idx], label="Starting value", color="black")
        ax[idx].set_xlabel("Time")
        ax[idx].set_ylabel("Population")
        ax[idx].set_title(f"Pertubation: {pert}")
        ax[idx].set_xscale("log")
        ax[idx].set_yscale("log")
        ax[idx].legend()

    fig.tight_layout()
    fig.savefig("../figures/d.pdf")


def part2():
    number_of_timesteps = 100
    pertubation = np.array([-10, -3, -2, -1, 1, 2, 3, 10])
    k = 1e3
    r = 0.1
    b = 1

    stable_point = k * (r ** (1 / b))
    start_pos = stable_point + pertubation

    fig, ax = plt.subplots(4, 2, figsize=(10, 16), sharey=False, sharex=True)
    ax = ax.flatten()

    time = np.linspace(1, number_of_timesteps, number_of_timesteps + 1)

    for idx, pert in enumerate(pertubation):
        traj_exact = get_exact_trajectory(number_of_timesteps, start_pos[idx], k, r, b)
        traj_approx = get_approx_trajectory(
            number_of_timesteps,
            k,
            r,
            b,
            stable_point,
            pert,
            stable=True,
        )
        ax[idx].plot(time, traj_exact, label="Exact trajectory")
        ax[idx].plot(time, traj_approx, label="Approximate trajectory")
        ax[idx].scatter(1, start_pos[idx], label="Starting value", color="black")
        ax[idx].set_xlabel("Time")
        ax[idx].set_ylabel("Population")
        ax[idx].set_title(f"Pertubation: {pert:.0f}")
        ax[idx].set_xscale("log")
        ax[idx].set_yscale("log")
        ax[idx].legend()

    fig.tight_layout()
    fig.savefig("../figures/e.pdf")
```

```
def main():
    part1()
    part2()


if __name__ == "__main__":
    main()
```

4.3 Problem3

```
    import numpy as np
import matplotlib.pyplot as plt


def next_population(n, r: np.ndarray, alpha: float) -> float:
    n_next = r * n * np.exp(-alpha * n)
    return n_next


def simulate_population(
    time_steps: int, n_0: float, r: np.ndarray, alpha: float
) -> np.ndarray:

    trajectories = np.zeros((r.shape[0], time_steps + 1))
    trajectories[:, 0] = n_0

    for idx in range(time_steps):
        trajectories[:, idx + 1] = next_population(trajectories[:, idx], r, alpha)
    return trajectories


def part1():
    time_steps = 300
    n_0 = 900
    alpha = 0.01
    r_values = np.arange(
        1,
        30.1,
        0.1,
    )
    trajectories = simulate_population(time_steps, n_0, r_values, alpha)
    fig, ax = plt.subplots()

    for idx, r in enumerate(r_values):
        ax.scatter(r * np.ones(100), trajectories[idx, -100:], s=1)
    ax.set_xlabel("R")
    ax.set_ylabel("Population")

    fig.tight_layout()
    fig.savefig("../figures/3a.pdf")
```

```
def part2():
    time_steps = 40
    n_0 = 900
    alpha = 0.01
    r_values = np.array([5.0, 10.0, 13.0, 23.0])
    trajectories = simulate_population(time_steps, n_0, r_values, alpha)
    fig, ax = plt.subplots()

    for idx, r in enumerate(r_values):
        ax.plot(trajectories[idx, :], label=f"R = {r}")
    ax.set_xlabel("Generation")
    ax.set_ylabel("Population")
    ax.legend()

    fig.tight_layout()
    fig.savefig("../figures/3b.pdf")


def part3():
    time_steps = 300
    n_0 = 900
    alpha = 0.01
    r_values = np.arange(
        12,
        13.1,
        0.1,
    )
    trajectories = simulate_population(time_steps, n_0, r_values, alpha)
    fig, ax = plt.subplots()

    for idx, r in enumerate(r_values):
        ax.scatter(r * np.ones(100), trajectories[idx, -100:])
    ax.set_xlabel("R")
    ax.set_ylabel("Population")

    fig.tight_layout()
    fig.savefig("../figures/3c2.pdf")


def part4():
    time_steps = 1000
    n_0 = 900
    alpha = 0.01
    r_values = np.arange(14.74, 14.79, 0.0005)
    trajectories = simulate_population(time_steps, n_0, r_values, alpha)
    fig, ax = plt.subplots()

    for idx, r in enumerate(r_values):
        ax.scatter(r * np.ones(800), trajectories[idx, -800:], s=0.1)

    ax.set_ylabel("R")
    ax.set_ylabel("Population")
    ax.grid()

    fig.tight_layout()
    fig.savefig("../figures/3d.pdf")


def main():
    # part1()
    # part2()
    # part3()
    part4()


if __name__ == "__main__":
    main()
```