Implementation of Perelson et al.

Please indicate at the top of your assignment whether or not you used any AI tools, such as MS Copilot. If you did use one of these tools, please provide a very brief explanation alongside each answer for how you confirmed the correctness of your solution.

- Used MS Copilot for some of the questions
- Used ChatGPT and Claude for plot making and organization of code

In this implementation we're going to evaluate the properties of a dynamical model.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

(1) Reproduce all four panels from Figure 1 shown in Perelson et al. using eq. 6

Since we are not given any empirical data, use the fitted parameters and the appropriate constants to generate the curves shown on these plots. Remember to account for the time delays for each pat.

(i) Implement eq. 6 from the paper as a function.

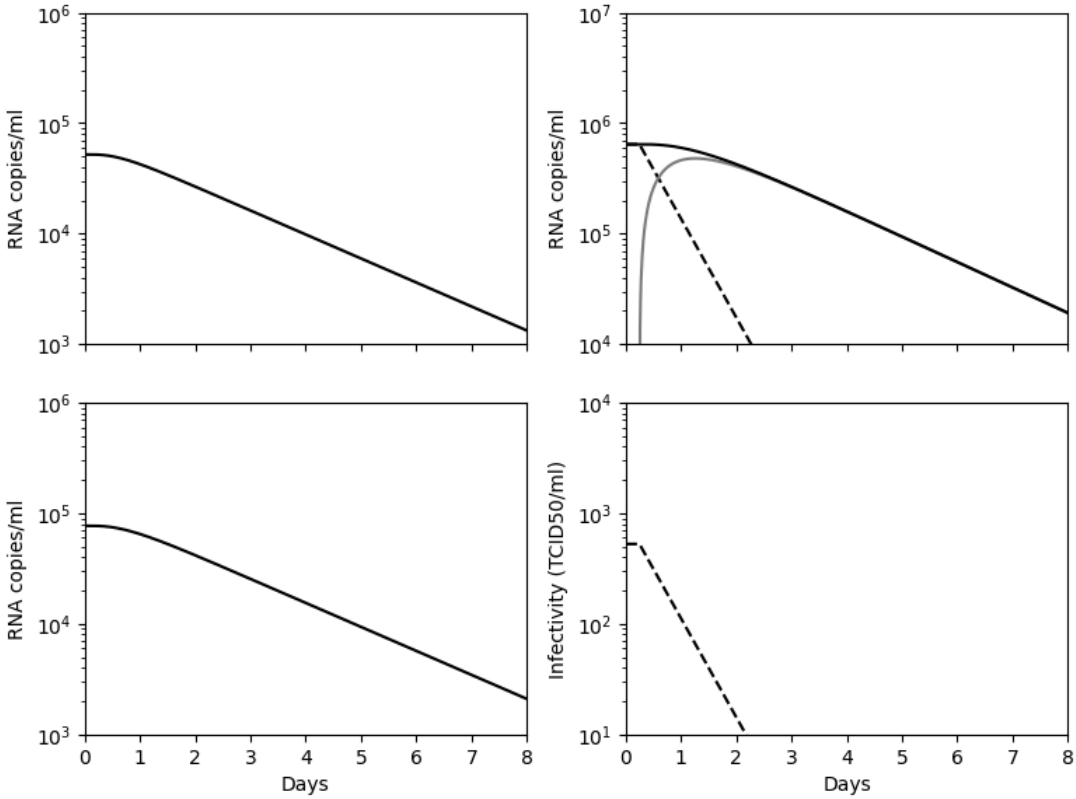
In [2]:
```
# Answer

import numpy as np
import matplotlib.pyplot as plt

def equation_6(tps: np.ndarray, v0: float, c: float, delta: float, delay: float, scale: float = 1.0):
    """
    Simulate viral load dynamics using Equation 6 from Perelson et al.

    Parameters:
    - tps (np.ndarray): Array of time points in days.
    - v0 (float): Initial viral concentration.
    - c (float): Clearance rate constant of free virions.
    - delta (float): Death rate constant of infected cells.
    - delay (float): Pharmacokinetic delay in hours, converted to days.
    - scale (float): Scaling factor for fitting purposes, default is 1.0.

    Returns:
    - V_i (np.ndarray): Concentration of infected virions over time.
    - V_ni (np.ndarray): Concentration of non-infected virions over time.
    - V_tot (np.ndarray): Total viral concentration over time.
    """

    # Normalize initial concentration with the scaling factor
    v0 *= scale

    # Convert delay from hours to days for consistency with time units
    tps_adjusted = tps - (delay/24)  # Apply the delay to the timepoints

    # Calculate infected and non-infected virion concentrations
    V_i = v0 * np.exp(-c * tps_adjusted)
    V_ni = (c * v0 / (c - delta)) * ((c / (c - delta)) * (np.exp(-delta * tps_adjusted) - np.exp(-c * tps_adjusted)) - delta * tps_ac

    # Apply conditions to simulate delay effects
    V_i[tps_adjusted < 0] = v0  # Before the delay, infected virion concentration is constant
    V_ni[tps_adjusted < 0] = 0  # Before the delay, non-infected virions are absent

    # Total viral concentration is the sum of infected and non-infected virions
    V_tot = V_i + V_ni

    return V_i, V_ni, V_tot



timepoints = np.linspace(0.01, 10, num=1000)

pat_102 = equation_6(tps=timepoints,v0=294E3, c=3.81, delta=0.26, delay=2, scale=1)
pat_103 = equation_6(tps=timepoints,v0=12E3, c=2.73, delta=0.68, delay=6, scale=1)

pat_104 = equation_6(tps=timepoints,v0=52E3, c=3.68, delta=0.50, delay=2, scale=1)
pat_105 = equation_6(tps=timepoints,v0=643E3, c=2.06, delta=0.53, delay=6, scale=1)
pat_107 = equation_6(tps=timepoints,v0=77E3, c=3.09, delta=0.50, delay=2, scale=1)
```

(ii) Make the figure panels from this function's values over time.

Note that some constants are given in other papers published by this lab. An estimate of k, the viral infectivity rate, can be found in Wein et al. (J. Theor. Biol. 192:81-98) to be 3.43×10^{-8} mL/(virion•day). Note that you will need to solve ordinary differential equations for T^*, V_I, and V_{NI} to reproduce the data in Figure 1.

In [3]:

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

# Configure plot limits and ticks for all subplots
def configure_axes(axs, x_lim=(0, 8), x_ticks=range(9), y_lims=None):
    for i, ax_row in enumerate(axs):
        for j, ax in enumerate(ax_row):
            ax.set_xlim(x_lim)
            ax.set_xticks(x_ticks)
            if y_lims and (i, j) in y_lims:
                ax.set_ylim(y_lims[(i, j)])


V_i0 = 643E3
TCID50_0 = V_i0 * 8.2E-4 # Recommended to use by prof
infect_coeff = TCID50_0 / V_i0


fig, axs = plt.subplots(2, 2, sharex=True, figsize=(8, 6))
fig.tight_layout(pad=2)


def plot_pat(ax, time, data, color='k', linestyle='-', ylim=None, label=None):
    ax.semilogy(time, data, color=color, linestyle=linestyle, label=label)
    if ylim:
        ax.set_ylim(ylim)
    if label:
        ax.legend()

plot_pat(axs[0, 0], timepoints, pat_104[2], color='k', ylim=[1e3, 1e6])
axs[0, 0].set_ylabel('RNA copies/ml')

# Plot pat 107 (bottom-left panel)
plot_pat(axs[1, 0], timepoints, pat_107[2], color='k', ylim=[1e3, 1e6])
axs[1, 0].set_xlabel('Days')
axs[1, 0].set_ylabel('RNA copies/ml')

# Plot pat 105 (top-right panel)
plot_pat(axs[0, 1], timepoints, pat_105[0], color='k', linestyle='--', ylim=[1e4, 1e7])
plot_pat(axs[0, 1], timepoints, pat_105[1], color='grey')
plot_pat(axs[0, 1], timepoints, pat_105[2], color='k')
axs[0, 1].set_ylabel('RNA copies/ml')

# Plot pat 105 infectivity (bottom-right panel)
plot_pat(axs[1, 1], timepoints, pat_105[0] * infect_coeff, color='k', linestyle='--', ylim=[10, 1e4])
axs[1, 1].set_xlabel('Days')
axs[1, 1].set_ylabel('Infectivity (TCID50/ml)')

# Set axis limits and ticks
configure_axes(axs, y_lims={(0, 0): [1e3, 1e6], (1, 0): [1e3, 1e6], (0, 1): [1e4, 1e7], (1, 1): [10, 1e4]})


plt.show()
```



(2) Can the ODE model of virion production before treatment show stable or unstable oscillations?

Describe and justify your answer.

To evaluate whether the ODE model of virion production before treatment shows stable or unstable oscillations, we can analyze the behavior of the model in its steady state (e.g., $T*$=0 and V =0) by linearizing the system around this point using a Taylor series approximation. This involves computing the Jacobian matrix of the system at the steady state and examining the eigenvalues of this matrix to determine the stability of the model.

The steady-state equations from the paper are:

$$\frac{dT^*}{dt} = kVT - \delta T^*$$

$$\frac{dV}{dt} = N\delta T^* - cV$$

The Jacobian matrix for this system is:

$$\begin{bmatrix} -\delta & kT \\ N\delta & -c \end{bmatrix}$$

where:

T represents the concentration of target cells, T$*$ is the concentration of infected cells, V is the concentration of free virions, δ is the death rate of infected cells, c is the clearance rate of free virions, k is the infection rate constant, and N is the number of virions produced per infected cell.

To assess the stability of the system, we find the eigenvalues of the Jacobian matrix. If all eigenvalues have negative real parts, the steady state is stable (indicating no oscillations).

(3) Reimplement the model relaxing the assumption about T remaining constant by numerically solving the system of ODEs.

i. Define a function that takes each parameter as a vector and outputs the time derivative of each species.

In [4]:
```
# Answer

def ode_func(y: np.ndarray, t: float, k: float, T0: float, delta: float, c: float, N: float, eta: float, delay: float):


    if t < delay:
        return 0, 0, 0

    T_star, V_i, V_ni = y

    dT_star = (k * V_i * (T0 - T_star)) - (delta * T_star)
    dV_i = -c * V_i + (1 - eta) * N * delta * T_star
    dV_ni = eta * N * delta * T_star - (c * V_ni)
```

```
    return dT_star, dV_i, dV_ni
```

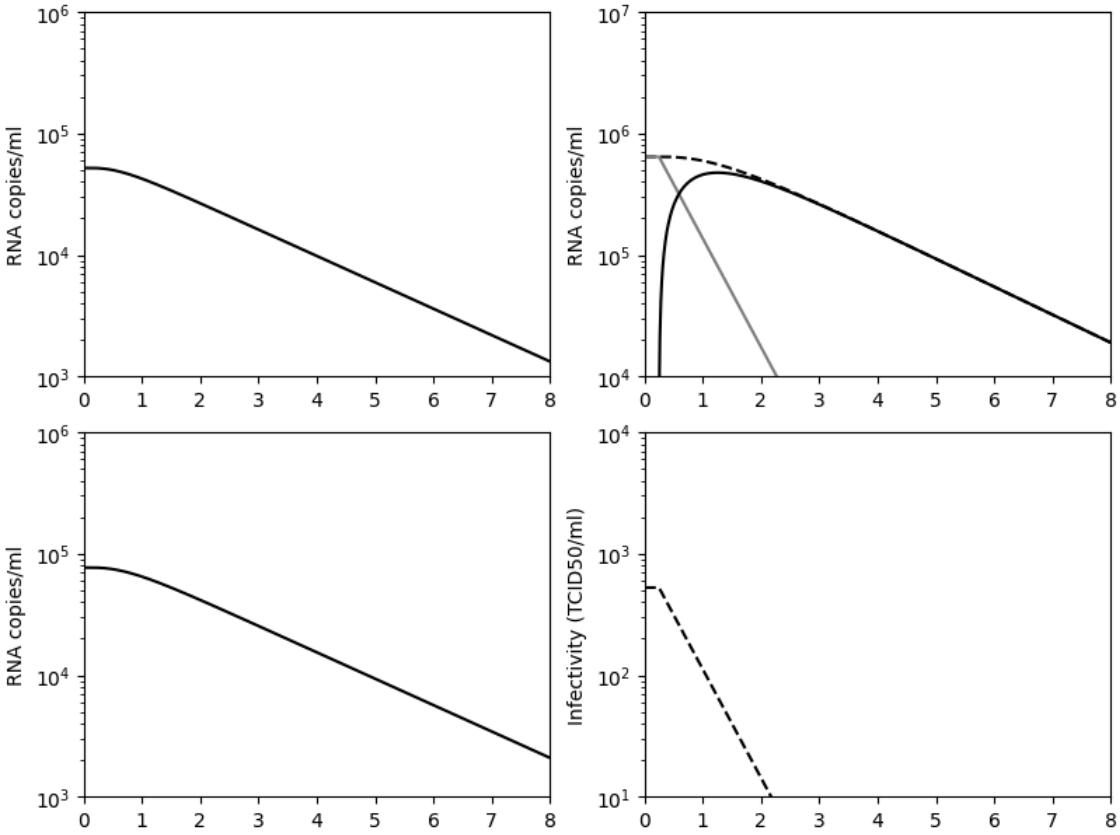
ii. Use odeint to solve this system of equations numerically for pat 105.

Hint: For ease later, it may be helpful to make a function here that handles everything and takes in η, c, and δ.

In [5]:
```
from scipy.integrate import odeint

def run_ode_solver(timepoints, V_i0, k, T0, delta, c, eta, delay):

    """Function template to help."""
    N = c / (k * T0)
    args = (
        k,
        T0,
        delta,
        c,
        N,
        eta,
        delay
    )
    V_ni0 = 0
    T_star0 = k * V_i0 * T0 / delta
    y0 = (
        T_star0,
        V_i0,
        V_ni0
    )

    return odeint(ode_func, y0, timepoints, args=args)

timepoints = np.linspace(0.01, 8, num=1000)


pat_105_sol = run_ode_solver(timepoints, 643E3, 3.43E-8, 11E3, 0.53, 2.06, 1, 6/24)
pat_104_sol = run_ode_solver(timepoints, 52E3, 3.43E-8, 2E3, 0.5, 3.68, 1, 2/24)
pat_107_sol = run_ode_solver(timepoints, 77E3, 3.43E-8, 412E3, 0.5, 3.09, 1, 2/24)
```

iii. Reproduce the subpanels of Fig. 1 with this updated model.

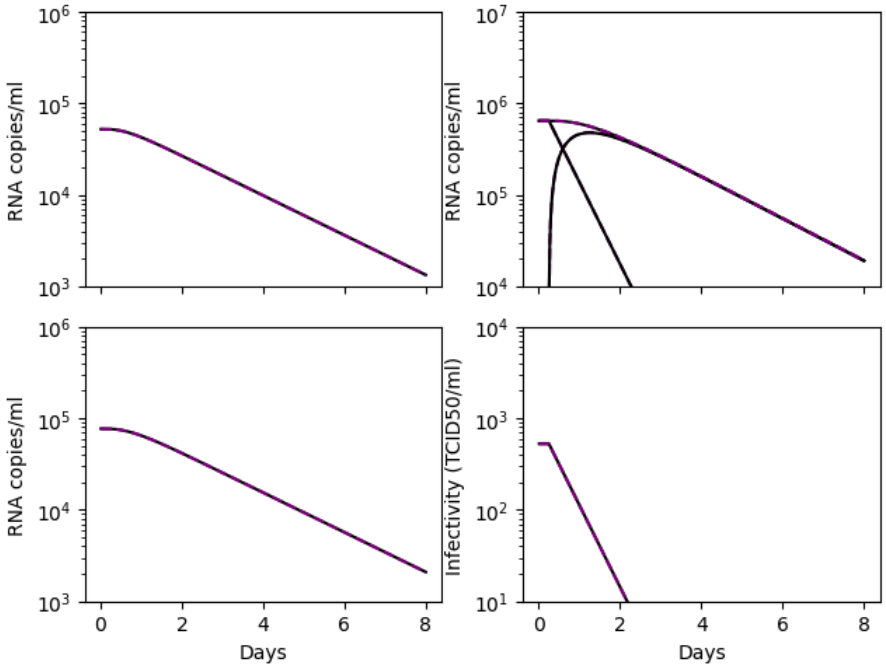
In [6]:
```
# Solve ODEs for each pat using the previous function
solutions = {
    "pat 105": run_ode_solver(timepoints, 643E3, 3.43E-8, 11E3, 0.53, 2.06, 1, 6/24),
    "pat 104": run_ode_solver(timepoints, 52E3, 3.43E-8, 2E3, 0.5, 3.68, 1, 2/24),
    "pat 107": run_ode_solver(timepoints, 77E3, 3.43E-8, 412E3, 0.5, 3.09, 1, 2/24)
}

# Set up figure with shablack x-axis for days and log scale for y-axis
fig, axs = plt.subplots(2, 2, figsize=(8, 6), constrained_layout=True)

# Helper function to plot the total RNA copies/ml for each pat
def plot_total_rna(ax, solution, label):
    total_rna = solution[:, 1] + solution[:, 2]
    ax.semilogy(timepoints, total_rna, color='k', label=label)
    ax.set_ylim([1E3, 1E7])
    ax.set_ylabel('RNA copies/ml')

# Plot total RNA for each pat in the left column
plot_total_rna(axs[0, 0], solutions["pat 104"], "pat 104")
plot_total_rna(axs[1, 0], solutions["pat 107"], "pat 107")

# Plot detailed RNA components for pat 105 in the top-right panel
axs[0, 1].semilogy(timepoints, solutions["pat 105"][:, 1] + solutions["pat 105"][:, 2], 'k--', label="Total RNA")
axs[0, 1].semilogy(timepoints, solutions["pat 105"][:, 1], color='grey', label="Infected Cells")
axs[0, 1].semilogy(timepoints, solutions["pat 105"][:, 2], color='k', label="Free Virions")
axs[0, 1].set_ylim([1E4, 1E7])
axs[0, 1].set_ylabel('RNA copies/ml')

# Plot infectivity curve for pat 105 in the bottom-right panel
infectivity = solutions["pat 105"][:, 1] * infect_coeff
axs[1, 1].semilogy(timepoints, infectivity, 'k--', label="Infectivity")
axs[1, 1].set_ylim([10, 1E4])
axs[1, 1].set_ylabel('Infectivity (TCID50/ml)')

# Set axis limits and ticks
configure_axes(axs, y_lims={(0, 0): [1e3, 1e6], (1, 0): [1e3, 1e6], (0, 1): [1e4, 1e7], (1, 1): [10, 1e4]})


plt.show()
```



iv. Are the results essentially the same as in (1)?

Show quantitative evidence of your conclusion.

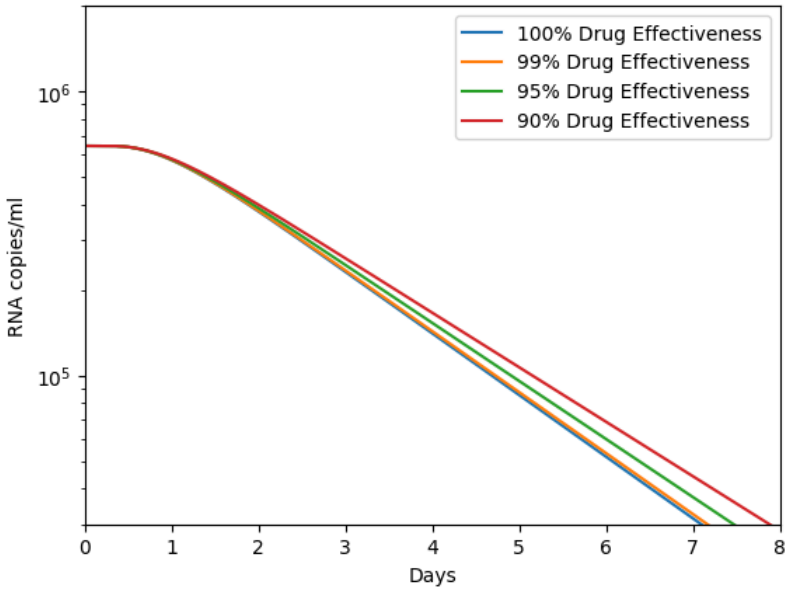
In [7]:
```
# Analytical solutions
pat_104 = equation_6(tps=timepoints,v0=52E3, c=3.68, delta=0.50, delay=2, scale=1)
pat_105 = equation_6(tps=timepoints,v0=643E3, c=2.06, delta=0.53, delay=6, scale=1)
pat_107 = equation_6(tps=timepoints,v0=77E3, c=3.09, delta=0.50, delay=2, scale=1)

pat_104_sol = run_ode_solver(timepoints, 52E3, 3.43E-8, 2E3, 0.5, 3.68, 1, 2/24)
pat_105_sol = run_ode_solver(timepoints, 643E3, 3.43E-8, 11E3, 0.53, 2.06, 1, 6/24)
pat_107_sol = run_ode_solver(timepoints, 77E3, 3.43E-8, 412E3, 0.5, 3.09, 1, 2/24)


fig, axs = plt.subplots(2, 2, sharex=True)
fig.tight_layout()

axs[0][0].semilogy(timepoints, pat_104_sol[:, 1] + pat_104_sol[:, 2], color='black')
axs[0][0].semilogy(timepoints, pat_104[2], color='purple', linestyle='--')
axs[1][0].semilogy(timepoints, pat_107_sol[:, 1] + pat_107_sol[:, 2], color='black')
axs[1][0].semilogy(timepoints, pat_107[2], color='purple', linestyle='--')
axs[0][0].set_ylim([1E3, 1E6])
axs[1][0].set_ylim([1E3, 1E6])

axs[0][1].semilogy(timepoints, pat_105_sol[:, 1] + pat_105_sol[:, 2], color='black')
axs[0][1].semilogy(timepoints, pat_105[0], color='purple', linestyle='--')
axs[0][1].semilogy(timepoints, pat_105_sol[:, 1], color='black')
axs[0][1].semilogy(timepoints, pat_105[1], color='purple', linestyle='--')
axs[0][1].semilogy(timepoints, pat_105_sol[:, 2], color='black')
axs[0][1].semilogy(timepoints, pat_105[2], color='purple', linestyle='--')
axs[0][1].set_ylim([1E4, 1E7])

axs[1][1].semilogy(timepoints, pat_105_sol[:, 1] * infect_coeff, color='black')
axs[1][1].semilogy(timepoints, pat_105[0] * infect_coeff, color='purple', linestyle='--')
axs[1][1].set_ylim([10, 1E4])


axs[1][0].set_xlabel('Days')
axs[1][1].set_xlabel('Days')
axs[0][0].set_ylabel('RNA copies/ml')
axs[1][0].set_ylabel('RNA copies/ml')
axs[0][1].set_ylabel('RNA copies/ml')
axs[1][1].set_ylabel('Infectivity (TCID50/ml)')

plt.show()
```



(4) Repeat the work that the authors describe in item 12 of their “References and Notes”.

(i) Namely, vary the effectiveness factor of the drug (η=1.0, 0.99, 0.95, and 0.90) and simulate the viral load that would result from the modified differential equations using the parameters c=3.0 1/days and δ=0.5 1/days as described in item 12 and the values of T0, V0, and the time delay provided for pat 105.

In [8]:
```
"""
Parameter estimates are based on the assumption of complete inhibition of
the production of new infectious virions and no increase in target
cells, we expect our parameter estimates,to be minimal estimates.
Generalizing the model to relax these two assumptions,
we can show that 8 is always a minimal estimate and that,
with target cell growth, c is typically a minimal estimate
"""

timepoints = np.linspace(0.01, 8, num=1000)

# Run the ODE solver for different drug effectiveness values
pat_105_100 = run_ode_solver(timepoints, V_i0, 3.43E-8, 11E3, 0.5, 3, 1, 6/24)
pat_105_99 = run_ode_solver(timepoints, V_i0, 3.43E-8, 11E3, 0.5, 3, 0.99, 6/24)
pat_105_95 = run_ode_solver(timepoints, V_i0, 3.43E-8, 11E3, 0.5, 3, 0.95, 6/24)
pat_105_90 = run_ode_solver(timepoints, V_i0, 3.43E-8, 11E3, 0.5, 3, 0.90, 6/24)

# Plot RNA copies/ml over time for each drug effectiveness value
plt.semilogy(timepoints, pat_105_100[:, 1] + pat_105_100[:, 2], label='100% Drug Effectiveness')
plt.semilogy(timepoints, pat_105_99[:, 1] + pat_105_99[:, 2], label='99% Drug Effectiveness')
plt.semilogy(timepoints, pat_105_95[:, 1] + pat_105_95[:, 2], label='95% Drug Effectiveness')
plt.semilogy(timepoints, pat_105_90[:, 1] + pat_105_90[:, 2], label='90% Drug Effectiveness')


plt.ylim([3E4, 2E6])
plt.xlim([0, 8])
plt.xlabel('Days')
plt.ylabel('RNA copies/ml')
plt.legend()
plt.show()
```



(ii) Then, use the function `scipy.optimize.leastsq` to fit this data to the equation for V(t) given in the paper and find the estimates of c and δ that result. Fit only the portion of the curve after the pharmacokinetic delay. Compare these estimates to the actual values for c and δ and discuss how an imperfect drug would affect the clearance time estimates.

In [9]:
```
from scipy.optimize import leastsq

import pandas as pd

delay = 6/24
p0 = [3.09, 0.53]
baseline = run_ode_solver(timepoints, V_i0, 3.43E-8, 11E3, 0.53, 2.06, 1.0, delay)

def lsq_function(params, gnu):
    c, delta = params
    one = run_ode_solver(timepoints, V_i0, 3.43E-8, 11E3, delta, c, gnu, delay)
    return one[:, 1] + one[:, 2] - baseline[:, 1] - baseline[:, 2]


drug_effectiveness_values = [1.0, 0.99, 0.95, 0.9]
results = []

# Run least squares for each drug effectiveness value and store the results
for gnu in drug_effectiveness_values:
    lsq_results = leastsq(lsq_function, p0, args=(gnu,), full_output=True)
    c, delta = lsq_results[0]
    results.append([int(gnu * 100), round(c, 2), round(delta, 2)])


df = pd.DataFrame(results, columns=['Drug Effectiveness (%)', 'c', 'delta'])


print(df)
```

```
   Drug Effectiveness (%)     c  delta
0                     100  2.06   0.53
1                      99  2.06   0.54
2                      95  2.04   0.57
3                      90  2.02   0.61
```

In [ ]:
```

```