Multi-scale Hybrid Algorithm NonConst_WH (Ver1)

Yuan Yin (Rebecca)

April 13, 2021

1 Population Level, $S(t), I(t)$

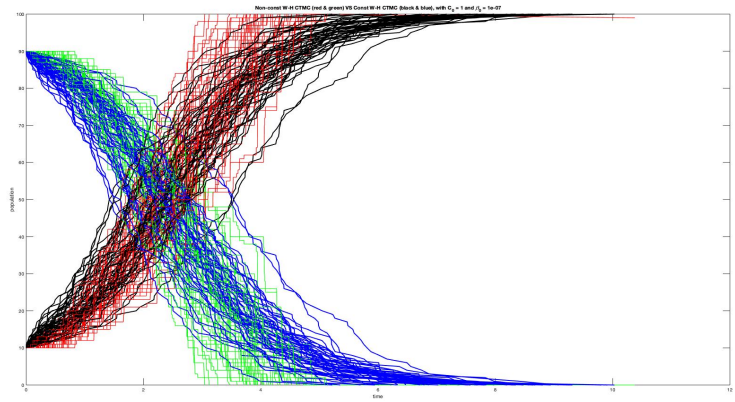

Figure 1: Comparisons between constant within-host multi-scale system and non-constant within host multi-scale system. The red curves represent $I_{non-const}(t)$, and the green curves represent $S_{non-const}(t)$, while the black curves denote $I_{const}(t)$, and the blue curve denote $S_{const}(t)$. For the non-constant within-host multi-scale system, $C_0 = 10^0$ and $\beta_0^{non-const} = 10^{-7}$. For the constant within-host multi-scale system, $\beta_0^{const} = 10^{-7}$. Note that $\beta_{const}(t) = V_{SS} \times \beta_0^{const}$ while $\beta_{non-const}(t) = C_0 \times \beta_0 \times V(\tau)$, where τ is the time from infection.

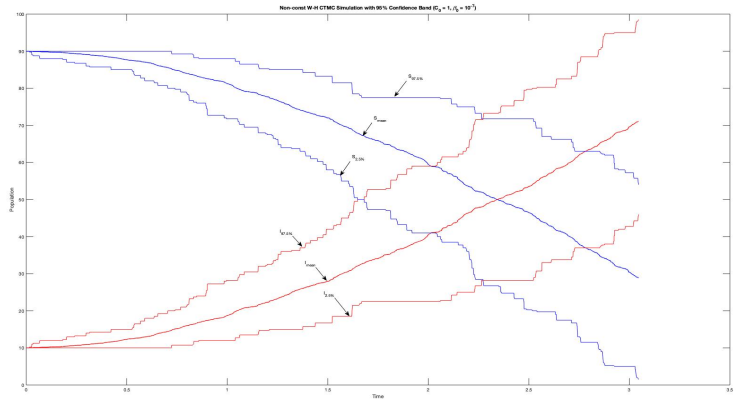

Figure 2: Non-constant within host multi-scale system with 95% confidence band (produced from Figure (1)).

2 The Plots for $\Theta(I) \propto \frac{\sum_{j\geq 1}^{I(t)} V_j T_j}{V_{ave} T_{ave}}$

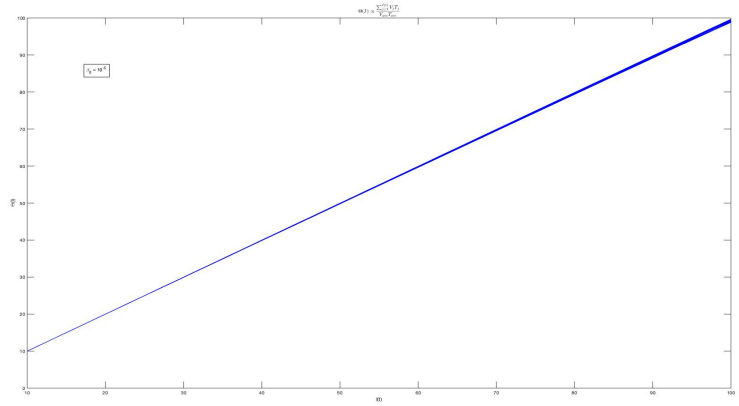

Figure 3: We set $C_0 = 10^0$, $\beta_0^{non-const} = 10^{-5}$, and the number of repetitions to be 50. Note that for each infectious individual, i, $\beta_i(t) = C_0 \times \beta_0 \times V(\tau)$, where τ is the time from infection.

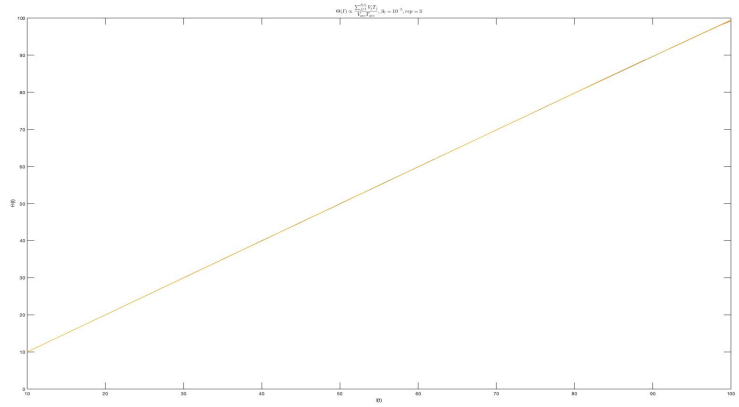

Figure 4: $C_0 = 10^0$, $\beta_0^{non-const} = 10^{-5}$, and the number of repetitions is 3.

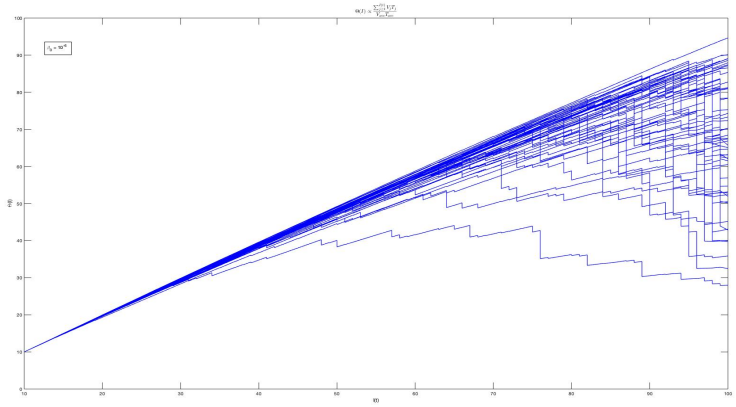

Figure 5: We set $C_0 = 10^0$, $\beta_0^{non-const} = 10^{-6}$, and the number of repetitions to be 50. Note that for each infectious individual, i, $\beta_i(t) = C_0 \times \beta_0 \times V(\tau)$, where τ is the time from infection.

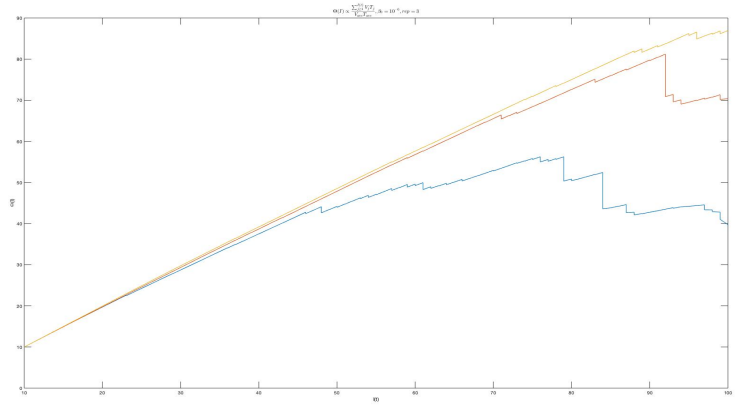

Figure 6: $C_0 = 10^0$, $\beta_0^{non-const} = 10^{-6}$, and the number of repetitions is 3.

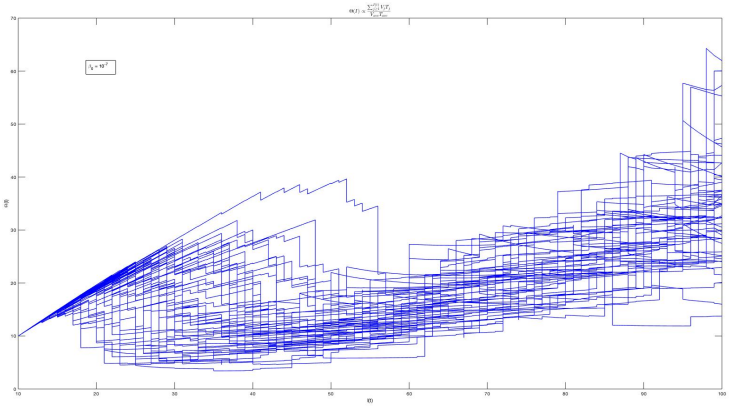

Figure 7: We set $C_0 = 10^0$, $\beta_0^{non-const} = 10^{-7}$, and the number of repetitions to be 50. Note that for each infectious individual, i, $\beta_i(t) = C_0 \times \beta_0 \times V(\tau)$, where τ is the time from infection.

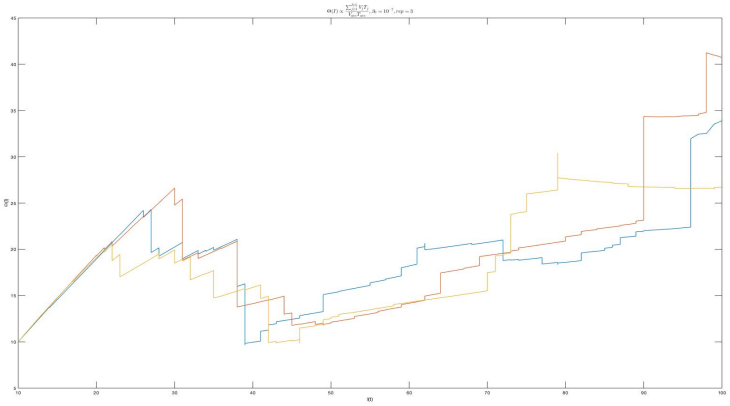

Figure 8: $C_0 = 10^0$, $\beta_0^{non-const} = 10^{-7}$, and the number of repetitions is 3.

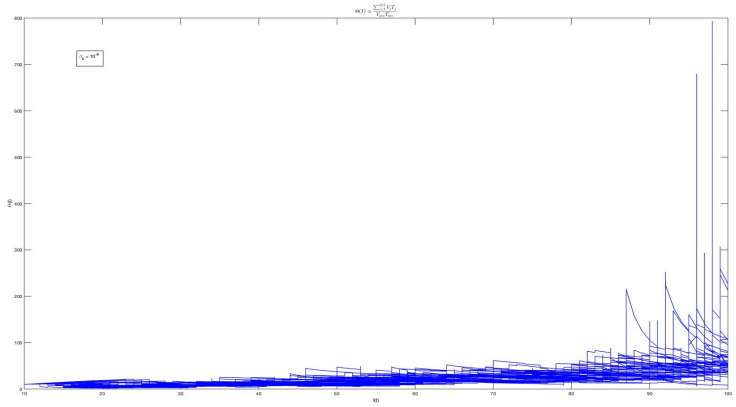

Figure 9: We set $C_0 = 10^0$, $\beta_0^{non-const} = 10^{-8}$, and the number of repetitions to be 50. Note that for each infectious individual, i, $\beta_i(t) = C_0 \times \beta_0 \times V(\tau)$, where τ is the time from infection.

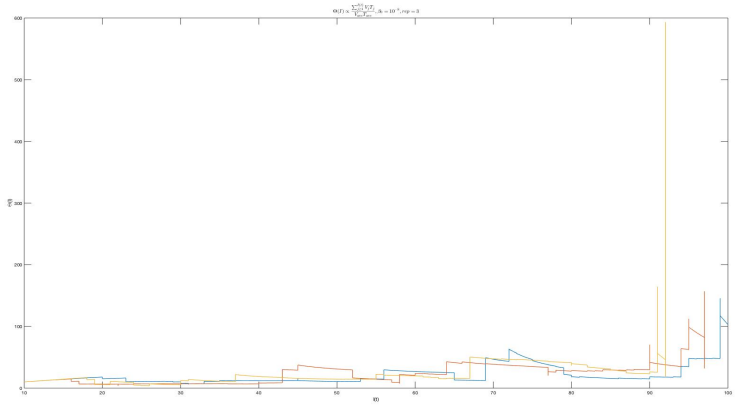

Figure 10: $C_0 = 10^0$, $\beta_0^{non-const} = 10^{-8}$, and the number of repetitions is 3.

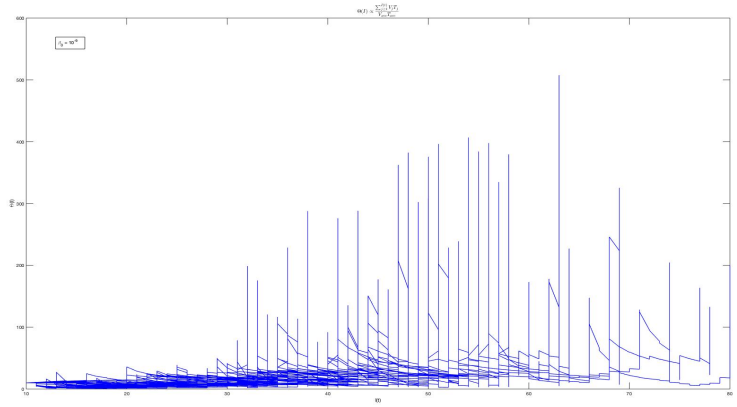

Figure 11: We set $C_0 = 10^0$, $\beta_0^{non-const} = 10^{-9}$, and the number of repetitions to be 50. Note that for each infectious individual, i, $\beta_i(t) = C_0 \times \beta_0 \times V(\tau)$, where τ is the time from infection.

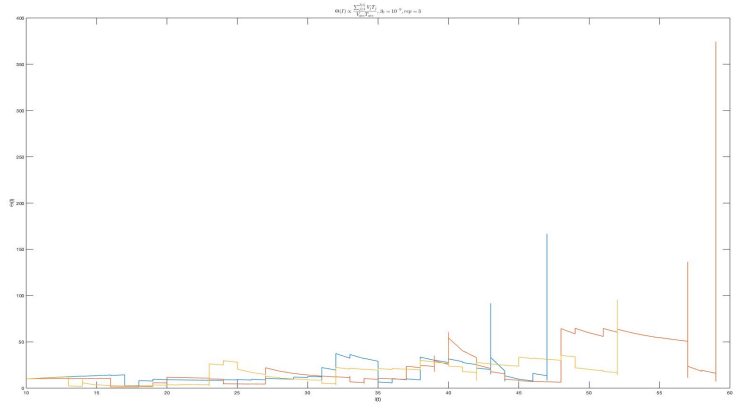

Figure 12: $C_0 = 10^0$, $\beta_0^{non-const} = 10^{-9}$, and the number of repetitions is 3.

Appendix A: Codes for Generating Θ Plots

```
function Theta(S0, I0, Lambda, mu, C0, beta0, t_end)

global Sdie Idie Infect_timepoint infect_timelength Q C S I T_poly V_poly t_array
Theta

Sdie = []; Idie = []; Infect_timepoint = []; infect_timelength = []; Theta = [];
Q = []; C =[]; S = []; S(1) = S0; I = []; I(1) = I0; t_array = []; t_array(1) = 0;

% Initialise all the arrays needed
t0 = 0; init_system(Lambda, mu, t0, C0); t_index = 1;

% Solve for the W-H system once:
[t_WH, T, ~, V] = Determ_WH_driver();
T_poly = interp1(t_WH,T, 'linear','pp'); V_poly = interp1(t_WH,V, 'linear','pp');

while t_array(length(t_array)) < t_end
    if ~ isempty(Q)
        event(t_index, mu, Lambda, beta0, C0);
        t_index = t_index + 1;
    else
        % Regenerate the queue Q:
        regenerate_Q(Lambda, mu, C0);
    end
end

% Make sure that t < t_end
if t_array(length(t_array)) > t_end
    t_array = t_array(1 : length(t_array) - 1);
    S = S(1 : length(t_array));
    I = I(1 : length(t_array));
    Theta = Theta(1 : length(t_array));
end

% plot(t_array, S, 'g', t_array, I, 'r', 'LineWidth', 1.2); hold on
% plot(I, Theta, 'b', 'LineWidth', 1.2); hold on
plot(I, Theta, 'LineWidth', 1.2); hold on


end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Debugged!
function init_system(Lambda, mu, t0, C0)

global Sdie Idie Infect_timepoint infect_timelength Q C S I Theta

% The waiting time for 1 susceptible to enter the population:
u1 = rand;
T1 = -log(u1) / Lambda;

% Create an array, 'Sdie', to store the waiting time for each susceptible
% to die:
for i = 1 : S(length(S))
    u2 = rand;
    Sdie(i) = -log(u2) / mu;
end

% Create an array, 'Idie', to store the waiting time for each infectious
% to die:
```

```
for i = 1 : I(length(I))
    u3 = rand;
    Idie(i) = -log(u3) / mu;
end

% Create an array, 'Infect_timepoint', to record the time stamp at which
% each infectious person gets infectious:
for i = 1 : I(length(I))
    Infect_timepoint(i) = t0;
end

% Create another array, 'infect_timelength', to record the time length for
% which each infectious person stays infectious:
for i = 1 : I(length(I))
    infect_timelength(i) = t0 - t0;
end

% Create an array, 'C', to record the waiting time for each infectious
% individual to interact with a susceptible person:
for i = 1 : I(length(I))
    u4 = rand;
    C(i) = -log(u4) / (C0 * S(length(S)));
end

% Create the event queues, 'Q':
Q = [C, Sdie, Idie, T1];

% Initialise Theta at t = 0:
Theta(1) = I(length(I));

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CHECK FURTHER FOR LOGIC ERRORS!!!!!!!!!!!!!!!
function event(t_index, mu, Lambda, beta0, C0)

global Q S I C Sdie Idie V_poly infect_timelength Infect_timepoint t_array T1 Theta
T_poly
%[T_SS, ~, V_SS] = const_WH();


% Find the minimal value in the event queue Q:
[DeltaT, index] = min(Q);

% If a new susceptible will enter the population:
if index == length(Q)
    S(t_index + 1) = S(t_index) + 1; I(t_index + 1) = I(t_index);

    % Update 'Sdie' and 'C':
    Sdie(S(t_index + 1)) = -log(rand) / mu;
    C = update_C(S, I, C, C0);

    % Add a new T1 into Q and update Q:
    T1_new = -log(rand) / Lambda;
    Q = [C, Sdie, Idie, T1_new];
end

% If some susceptible person dies:
if (length(Idie) + 1 <= index) && (index <= length(Idie) + length(Sdie))
    S(t_index + 1) = S(t_index) - 1; I(t_index + 1) = I(t_index);

    % Remove that susceptible from the system and update 'C':
    j = index - length(Idie);
```

```
    Sdie = delete(j, Sdie);
    C = update_C(S, I, C, C0);

    Q = [C, Sdie, Idie, T1];
end

% If some infectious person dies:
if (length(Idie) + length(Sdie) + 1 <= index) && (index <= 2 * length(Idie) +
length(Sdie))
    S(t_index + 1) = S(t_index); I(t_index + 1) = I(t_index) - 1;

    % Remove that infectious from the system:
    j = index - length(Idie) - length(Sdie);
    Idie = delete(j, Idie);
    C = delete(j, C);
    Infect_timepoint = delete(j, Infect_timepoint);
    infect_timelength = delete(j, infect_timelength);

    Q = [C, Sdie, Idie, T1];
end

% If an infectious person i=index interacts with a susceptible person
if (1 <= index) && (index <= length(C))
    V_1 = ppval(V_poly, infect_timelength(index));
    TV = beta0 * V_1; % is TV in [0, 1]??????????????????????????????
    u5 = rand;

    if u5 <= min(TV, 1)
        % A susceptible becomes infectious
        S(t_index + 1) = S(t_index) - 1; I(t_index + 1) = I(t_index) + 1;

        % The susceptible is randomly selected and deleted from the group
        % of susceptible people:
        Sj = randi(S(t_index));
        Sdie = delete(Sj, Sdie);

        % Add the new infectious individual to the system:
        new_death_time = -log(rand) / mu;
        Idie(length(Idie) + 1) = new_death_time;
        % Update information
        infect_timelength = infect_timelength + DeltaT;
        Infect_timepoint(I(t_index + 1)) = t_array(t_index) + DeltaT;
        infect_timelength(length(infect_timelength) + 1) = 0;
        C = update_C(S, I, C, C0);
    else
        % Nonthing happens
        S(t_index + 1) = S(t_index); I(t_index + 1) = I(t_index);
        infect_timelength = infect_timelength + DeltaT;
    end

    Q = [C, Sdie, Idie, T1];
end

% Update Theta:
LHS_sum = 0;
V_sum = 0;
T_sum = 0;

for j = 1 : I(length(I))
    V_j = ppval(V_poly, infect_timelength(j));
    T_j = ppval(T_poly, infect_timelength(j));

    LHS_sum = LHS_sum + V_j * T_j;
```

```
    V_sum = V_sum + V_j;
    T_sum = T_sum + T_j;
end
V_ave = V_sum / I(length(I));
T_ave = T_sum / I(length(I));

Theta(t_index + 1) = LHS_sum / (V_ave * T_ave);

t_index  = t_index + 1;
t_array(t_index) = t_array(t_index - 1) + DeltaT;

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Debugged!
function array = delete(index, array)

% Remove that the 'index_th' entry in the array
if length(array) == 1
    array = [];
else
    if index > 1 && index < length(array)
        array = [array(1 : index - 1), array(index + 1 : length(array))];
    else
        if index == 1
            array = array(index + 1 : length(array));
        elseif index == length(array)
            array = array(1 : length(array) - 1);
        end
    end
end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Debugged!
function C = update_C(S, I, C, C0)

for i = 1 : I(length(I))
    u4 = rand;
    C(i) = -log(u4) / (C0 * S(length(S)));
end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Debugged!
function regenerate_Q(Lambda, mu, C0)

global Sdie Idie Q C S I

% The waiting time for 1 susceptible to enter the population:
u1 = rand;
T1 = -log(u1) / Lambda;

% Create an array, 'Sdie', to store the waiting time for each susceptible
% to die:
for i = 1 : S(length(S))
    u2 = rand;
    Sdie(i) = -log(u2) / mu;
```

```
end

% Create an array, 'Idie', to store the waiting time for each infectious
% to die:
for i = 1 : I(length(I))
    u3 = rand;
    Idie(i) = -log(u3) / mu;
end

% Create an array, 'C', to record the waiting time for each infectious
% individual to interact with a susceptible person:
for i = 1 : I(length(I))
    u4 = rand;
    C(i) = -log(u4) / (C0 * S(length(S)));
end

% Create the event queues, 'Q':
Q = [C, Sdie, Idie, T1];

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Debugged!
% Deterministic Within-Host Subsystem:
function [t, T, Tstar, V] = Determ_WH(k, c, p, mu_c, delta_c, Lambda, T0, Tstar0, V0,
t_end)

% I perform the log transformation to the W-H system:
y0 = [log(T0); log(Tstar0); log(V0)];
tspan = [0, t_end];
opts = odeset('RelTol',1e-8,'AbsTol',1e-8);
[t, y] = ode15s(@(t,y) ode_sys(t, y, k, c, p, mu_c, delta_c, Lambda), tspan, y0,
opts);
T_Log = y(: , 1); T = exp(T_Log);
Tstar_Log = y(: , 2); Tstar = exp(Tstar_Log);
V_Log = y(: , 3); V = exp(V_Log);


%plot_WH(T, Tstar, V, t); % NOote that Tstar(t) and V(t) almost overlap with each
other.
%xlabel('t'); ylabel('cell population');
%title('Numerical Simulation (Deterministic) for Within-Host Subsystem');

end



function dydt = ode_sys(t, y, k, c, p, mu_c, delta_c, Lambda)

    dydt = [Lambda * exp(-y(1)) - k * exp(y(3)) - mu_c;
            k * exp(y(1) + y(3) - y(2)) - mu_c - delta_c;
            -c + p * exp(y(2) - y(3))];
end


function plot_WH(T, Tstar, V, t)
    plot(t, T, t, Tstar, t, V, 'LineWidth', 1.5);
    legend('T(t)', 'T^{*}(t)', 'V(t)');
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Debugged!
function [t, T, Tstar, V] = Determ_WH_driver()

% The magnitute of parameters is drawn from  the research paper:
% 'The Mechanisms for Within-Host Influenza Virus Control Affect
% Model-Based Assessment and Prediction of Antiviral Treatment'
T0 = 10 ^ (8); %8
Tstar0 = 1;
V0 = 10 ^ (4);

p = 10 ^ (1);
c = 10 ^ (0);
k = 10 ^ (-7);
mu_c = 10 ^ (-1);
delta_c = 10 ^ (1);
Lambda = 1.1 * (mu_c * (mu_c + delta_c) * c) / (k * p);

R_0 = T0 * k * p / (c * (mu_c + delta_c)); % Basic reproductive number

t_end = 10 ^ (8); % One can see that the W-H subsystem timescale is
                  % much shorter than that of B-H subsystem
[t, T, Tstar, V] = Determ_WH(k, c, p, mu_c, delta_c, Lambda, T0, Tstar0, V0, t_end);

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Debugged!
% Constant Within-Host Subsystem:
function [T_SS, Tstar_SS, V_SS] = const_WH()

% This function returns the non-virus-free steady state,
% '(T_SS, Tstar_SS, V_SS)', of the within-host subsytem.
%
% One can run 'Determ_WH_driver.m' to check that under these values, our
% Within-Host subsystem arrives at a non-virus free equilibrium.

% Parameter Initialisation:
p = 10 ^ (1);
c = 10 ^ (0);
k = 10 ^ (-7);
mu_c = 10 ^ (-1);
delta_c = 10 ^ (1);
Lambda = 1.1 * (mu_c * (mu_c + delta_c) * c) / (k * p);

% Compute the Steady States:
T_SS = (mu_c + delta_c) * c / (p * k);
%Tstar_SS = ((Lambda * p * k) / (c * (mu_c + delta_c)) - mu_c) * c / (p * k);
%V_SS = ((Lambda * p * k) / (c * (mu_c + delta_c)) - mu_c) / k;
V_SS = (Lambda - mu_c * T_SS) / (k * T_SS);
Tstar_SS = c * V_SS / p;


end
```