Attached code:clc; clear; close all;

```
%% ---------- Physical/Numerical Parameters ----------
% 1. Mucus layer
Dw = 1e-9; Cw0 = 1e12; L1 = 2e-4; Nx1 = 100; dx1 = L1/(Nx1-1);
Tsim = 5000; dt = 0.01; Nt = round(Tsim/dt);
alpha1 = (Dw*dt/dx1^2)/2;
k_wm = 1e-7;

% 3. Mucosa
Dm = 5e-11; k_m_delete = 0.01; L3 = 1e-4; Nx3 = 100; dx3 = L3/(Nx3-1);
alpha3 = (Dm*dt/dx3^2)/2;
k_mb = 1e-6;

% 4. Blood compartment
Vb = 5e-6; k_b_delete = 0.005; A_mb = 1e-4; k_bg = 1e-5;

% 6. Intestinal mucosa
Dg = 5e-11; Lg = 1e-3; Nxg = 100; dxg = Lg/(Nxg-1);
alphag = (Dg*dt/dxg^2)/2; k_g_delete = 0.001;
A_bg = 1e-4;

% 7. Intestinal lumen
V_in = 1e-6; k_in_delete = 0.002; k_gin = 1e-6; A_in = 1e-4;
r_max = 1e6; K = 1e14; % logistic growth

%% ---------- Initial Fields ----------
u1 = zeros(Nx1, Nt+1); u1(:, 1) = 0;
u3 = zeros(Nx3, Nt+1); u3(:, 1) = 0;
Cb = zeros(1, Nt+1); Cb(1) = 0;
ug = zeros(Nxg, Nt+1); ug(:, 1) = 0;
cin = zeros(1, Nt+1); cin(1) = 0;
r = zeros(1, Nt+1); r(1) = 0;

%% ---------- Assemble Implicit Matrices and Pre-factorize ----------
% Mucus layer
e1 = ones(Nx1, 1);
A1 = diag((1 + 2*alpha1)*e1) - alpha1*diag(e1(2:end), 1) - alpha1*diag(e1(1:end-1), -1);
B1 = diag((1 - 2*alpha1)*e1) + alpha1*diag(e1(2:end), 1) + alpha1*diag(e1(1:end-1), -1);
A1(1, :) = 0; A1(1, 1) = 1; % Left boundary Dirichlet
A1(end, :) = 0; A1(end, end) = 1; % Right boundary Dirichlet
B1(end, :) = 0; B1(end, end) = 1;
[L1, U1] = lu(A1); % Pre-factorization
```

```
% Mucosa
e3 = ones(Nx3, 1);
A3 = diag((1 + 2*alpha3)*e3) - alpha3*diag(e3(2:end), 1) - alpha3*diag(e3(1:end-1), -1);
B3 = diag((1 - 2*alpha3 - k_m_delete*dt)*e3) + alpha3*diag(e3(2:end), 1) + alpha3*diag(e3(1:end-1),
-1);
A3(1, :) = 0; A3(1, 1) = 1; % Left boundary Dirichlet
A3(end, :) = 0; A3(end, end) = 1; % Right boundary Dirichlet
B3(end, :) = 0; B3(end, end) = 1;
[L3, U3] = lu(A3); % Pre-factorization

% Intestinal mucosa
eg = ones(Nxg, 1);
Ag = diag((1 + 2*alphag)*eg) - alphag*diag(eg(2:end), 1) - alphag*diag(eg(1:end-1), -1);
Bg = diag((1 - 2*alphag - k_g_delete*dt)*eg) + alphag*diag(eg(2:end), 1) + alphag*diag(eg(1:end-1),
-1);
Ag(1, :) = 0; Ag(1, 1) = 1; % Left boundary Dirichlet
Ag(end, :) = 0; Ag(end, end) = 1; % Right boundary Dirichlet
Bg(end, :) = 0; Bg(end, end) = 1;
[Lg, Ug] = lu(Ag); % Pre-factorization

%% ---------- Main Time Loop ----------
for n = 1:Nt
% Stop vaccine supply after 3 minutes
Cw = Cw0;
if (n-1)*dt >= 180
Cw = 0;
end
% Step1: Mucus implicit diffusion
rhs1 = B1 * u1(:, n);
rhs1(1) = Cw;
u1(:, n+1) = U1 \ (L1 \ rhs1);

% Step2: Mucus→Mucosa mass transfer

cw = u1(end, n);
cm0 = u3(1, n);
J_wm = k_wm * (cw - cm0);
u1(end, n+1) = max(u1(end-1, n+1) - J_wm * dt / dx1, 0);
% Step3: Mucosa implicit diffusion + immune response
rhs3 = B3 * u3(:, n);
rhs3(1) = cm0 + J_wm * dt / dx3;
u3(:, n+1) = U3 \ (L3 \ rhs3);
u3(1, n+1) = cm0 + J_wm * dt / dx3;
u3(end, n+1) = max(u3(end-1, n+1) - J_wm * dt / dx3, 0);

% Step4: Mucosa→Blood mass transfer
```

```
J_mb = k_mb * (u3(end, n) - Cb(n));
Cb(n+1) = Cb(n) + dt / Vb * (A_mb * J_mb) - k_b_delete * dt * Cb(n);

% Step5: Blood→Intestinal mucosa mass transfer

J_bg = k_bg * (Cb(n) - ug(1, n));
rhs_g = Bg * ug(:, n);
rhs_g(1) = ug(1, n) + J_bg * dt / dxg;
ug(:, n+1) = Ug \ (Lg \ rhs_g);
ug(1, n+1) = ug(1, n) + J_bg * dt / dxg;
ug(end, n+1) = max(ug(end-1, n+1) - J_bg * dt / dxg, 0);

% Step6: Intestinal mucosa→Intestinal lumen mass transfer

J_g_in = k_gin * (ug(end, n) - cin(n));
cin(n+1) = cin(n) + (A_in * J_g_in * dt / V_in) - k_in_delete * dt * cin(n);
cin(n+1) = min(max(cin(n+1), 0), K);
end

%% ---------- Plotting ----------
% Create figure with explicit property setting
fig = figure;
set(fig, 'Name', 'Concentration Profiles', 'Units', 'normalized', 'Position', [0.1, 0.1, 0.8, 0.7]);

% Mucus layer concentration
ax1 = subplot(2, 2, 1);
plot((0:Nt)*dt, u1(end, :), 'LineWidth', 1.5);
set(get(ax1, 'XLabel'), 'String', 'Time (s)');
set(get(ax1, 'YLabel'), 'String', 'Concentration (CFU/m^3)');
set(get(ax1, 'Title'), 'String', 'Mucus Layer Concentration');
set(ax1, 'XGrid', 'on', 'YGrid', 'on');

% Blood concentration
ax2 = subplot(2, 2, 2);
plot((0:Nt)*dt, Cb, 'LineWidth', 1.5);
set(get(ax2, 'XLabel'), 'String', 'Time (s)');
set(get(ax2, 'YLabel'), 'String', 'Concentration (CFU/m^3)');
set(get(ax2, 'Title'), 'String', 'Blood Concentration');
set(ax2, 'XGrid', 'on', 'YGrid', 'on');

% Intestinal mucosa concentration
ax3 = subplot(2, 2, 3);
plot((0:Nt)*dt, ug(end, :), 'LineWidth', 1.5);
set(get(ax3, 'XLabel'), 'String', 'Time (s)');
set(get(ax3, 'YLabel'), 'String', 'Concentration (CFU/m^3)');
set(get(ax3, 'Title'), 'String', 'Intestinal Mucosa Concentration');
set(ax3, 'XGrid', 'on', 'YGrid', 'on');

% Intestinal lumen concentration
ax4 = subplot(2, 2, 4);
plot((0:Nt)*dt, cin, 'LineWidth', 1.5);
set(get(ax4, 'XLabel'), 'String', 'Time (s)');
set(get(ax4, 'YLabel'), 'String', 'Concentration (CFU/m^3)');
set(get(ax4, 'Title'), 'String', 'Intestinal Lumen Concentration');
set(ax4, 'XGrid', 'on', 'YGrid', 'on');

% Set font size for all axes
set([ax1, ax2, ax3, ax4], 'FontSize', 10);

% Add overall title using annotation
annotation(fig, 'textbox', [0.3, 0.95, 0.4, 0.05], 'String', ...
'Concentration Profiles in Different Compartments', ...
'HorizontalAlignment', 'center', 'VerticalAlignment', 'middle', ...
'FontSize', 14, 'FontWeight', 'bold', 'LineStyle', 'none');


function total_AUC = pk_auc_extrapolation(t, C, varargin)

% Use end-logarithmic linear regression to extrapolate and calculate AUC (0→∞)

% Compliance with FDA Bioanalytical Method Validation Guidelines (2018)
%

% import ：

% t-time vector (monotonic increase)
% C-Concentration vector (non-negative)

% optional parameters ：

% 'EndFraction" -The proportion of end data used for regression (default 0.2)
% 'MinPoints' -Minimum regression points (default 3)
% 'RSQCutoff' -acceptable R? threshold (default 0.8)

% output ：

% total_AUC-extrapolated total AUC

% ===== Parameter validation =====
p = inputParser;
addParameter(p, 'EndFraction', 0.2, @(x)x>0 && x<0.5);
addParameter(p, 'MinPoints', 3, @(x)x>=2);
addParameter(p, 'RSQCutoff', 0.8, @(x)x>0 && x<1);
parse(p, varargin{:});
```

```
end_frac = p.Results.EndFraction;
min_points = p.Results.MinPoints;
rsq_cutoff = p.Results.RSQCutoff;

% Basic check
assert(isvector(t) && isvector(C) && numel(t)==numel(C), ...
'Input must be an equal-length vector');
assert(all(diff(t)> 0), 'the time series must be monotonically increasing');
assert(all(C>=0), 'the concentration value must be non-negative');

% Unify as column vectors
t = t(:);
C = C(:);

% ===== Calculate AUC of the simulated area =====
AUC_sim = trapz(t, C);

% ===== Identify end elimination phase =====
% 1. Find the region after the peak
[~, peak_idx] = max(C);
if isempty(peak_idx) || peak_idx == numel(C)
Total_AUC = AUC_sim;% No valid post-peak region
return;
end

elim_phase = C(peak_idx:end);
t_elim = t(peak_idx:end);

% 2. Select end points (at least min_points points)
n_end = max(min_points, round(end_frac*numel(elim_phase)));
if n_end > numel(elim_phase)
nEnd = numel(elim_phase);% Prevents exceeding array boundaries
end
end_points = elim_phase(end-n_end+1:end);
t_end = t_elim(end-n_end+1:end);

% 3. Check validity
if any(end_points <= 0) || numel(end_points) < 2
total_AUC = AUC_sim;
return;
end

% ===== Logistic regression =====
% Ensure vector direction consistency
```

```
t_end = t_end (:);% Forced column vector
endpoints = end_points(:);% Forced column vector

logC = log(end_points);

% Use polyfit for robust regression
p = polyfit(t_end, logC, 1);
y_fit = polyval(p, t_end);


% count R?

residuals = logC - y_fit;
SS_res = sum(residuals.^2);
SS_tot = sum((logC - mean(logC)).^2);
r_squared = 1 - (SS_res/SS_tot);

% Check regression quality
if r_squared < rsq_cutoff
total_AUC = AUC_sim;
return;
end

% ===== Calculate the extrapolation tail =====
lambda_z = -p (1);% elimination rate constant
C_last = end_points(end);
AUC_tail = C_last / lambda_z;

% ===== Calculate total AUC =====
total_AUC = AUC_sim + AUC_tail;
end


clc; clear; close all;
warning('off', 'all');

%% ===== Parameter optimization =====

λ= 532e-9;% Green light wavelength [m]

I0 = 1;% Inlet light intensity
L = 0.5;% propagation distance [m]


μ= 0.015;% dynamic viscosity [Pa·s]

H = 0.05;% Pipeline half height [m]
dp = 120;% pressure difference [Pa]
```

```
Qsca = 0.85;% scattering efficiency
rhop = 100;% particle apparent density [kg/m�0�6]

ρ  = 1100;% true particle density [kg/m�0�6]

d_part = 8e-6;% particle diameter [m]

%% ===== Calculate the grid =====
Nx = 300; Ny = 100; Nz = 50;
dx = L/Nx; dy = 2*H/Ny; dz = 0.01;
[X,Y,Z] = meshgrid(0:dx:L, -H:dy:H, 0:dz:0.1);

%% ===== Pre-allocations =====
I = I0 * ones (Ny, Nx);% Optical intensity field
Cn2 = 8e-15;% refractive index structure constant
turb Ratio = zeros (Nz, 1);% of turbulent regions
z_pos = squeeze (Z(1,1,:));% z direction coordinates

%% ===== Main cycle =====
for iz = 1:Nz
Current section%
vx = generate_flow_field(X(:,:,iz), Y(:,:,iz), mu, H, dp, L);

% laminar decay coefficient
v_perp = abs(vx);
alpha_lam = 0.045 + 3*Qsca*rhop./(rho*d_part) .* (1 + 0.03*v_perp/(mu/0.001));

% Local Reynolds number & Bifurcation Mask
Re_local = 1000 * v_perp * (2*H) / mu;
turb_mask = (Re_local > 4000);
turb_ratio(iz) = sum(turb_mask(:)) / numel(turb_mask);

% Stream phase screen
phi_scr = turbulence_phase_screen(Nx, Ny, L, lambda, Cn2);

Decay calculation%
if any(turb_mask(:))
beta_turb = 1.2.*alpha_lam + 4*pi^2*Cn2*L^(5/3)/lambda^(7/6);
I_turb = I .* exp(-beta_turb*dz) .* turb_mask;
I_lam = I .* exp(-alpha_lam*dz) .* ~turb_mask;
I = I_turb + I_lam;
gamma = abs(mean(exp(1i*phi_scr(:))))^2;
I = I * gamma;
else
```

```
I = I .* exp(-alpha_lam*dz);
end
end

%% ===== Visual =====
figure('Position',[100 100 1400 600])

subplot(2,2,1)
imagesc(X(1,:,1), Y(:,1,1), I)
axis equal tight; colormap hot; colorbar
title ("Export Light Intensity Distribution"); xlabel ("x [m]"); ylabel ("y [m]')")

subplot(2,2,2)
plot(z_pos, squeeze(I(round(Ny/2), round(Nx/2), :)), 'LineWidth',2)
title ('Beam Centerline Attenuation'); xlabel ('z [m]'); ylabel ('Relative Light Intensity'); grid on; ylim
([0 1]);

subplot(2,2,3)
plot(z_pos, turb_ratio, 'LineWidth',2)
title("Stallage-Along Variation of Turbulent Area Proportion"); xlabel("z [m]"); ylabel("Turbulence
Proportion"); grid on; ylim([0 0.4]);

subplot(2,2,4)
imagesc(phi_scr); axis equal; colorbar; colormap jet
title('Turbulent Phase Screen')

%% ===== Light intensity profile =====
figure('Position',[100 100 800 400]); hold on
depths = [1 round(Nz/3) round(2*Nz/3) Nz];
clr = {'b','g','r','k'};
for i = 1:4
idx = depths(i);
plot(X(1,:,1), squeeze(I(round(Ny/2),:,idx)), clr{i}, 'LineWidth',1.5)
end
legend(arrayfun(@(k)sprintf('z=%.2f m',z_pos(k)),depths,'uni',0),'location','northeast')
title ("Different Depth Light Intensity Profiles"); xlabel ("x [m]"); ylabel ("Relative Light Intensity");
grid on

%% ===== Viscosity effect analysis =====
figure('Position',[100 100 800 400])
mu_test = [0.005 0.01 0.015 0.02];
I_end = zeros(size(mu_test));
for j = 1:numel(mu_test)
mu_cur = mu_test(j);
```

```
I_tmp = I0;
for k = 1:10
I_tmp = I_tmp * exp(-0.1*(1 + 0.5/mu_cur));
end
I_end(j) = I_tmp;
end
plot(mu_test, I_end, 'o-', 'LineWidth',2)
title ('Mudiness to the final light intensity')
xlabel ('μ  [Pa·s]'); ylabel ('Export Light Intensity'); grid on


%% ===== Local function =====
function vx = generate_flow_field(X,Y,mu,H,dp,L)
vx_base = dp/(2*mu*L) * (H.^2 - Y.^2);
Re_crit = 4000 * (1 + 0.5*(mu/0.001));
v_avg = mean(vx_base(:));
Re = 1000*v_avg*(2*H)/mu;

if Re > Re_crit
N_vor = max(1, round((Re - Re_crit)/2000));
vx = vx_base;
for k = 1:N_vor
x0 = rand*L; y0 = (rand-0.5)*2*H;
R = sqrt((X-x0).^2 + (Y-y0).^2);
w = 0.15*v_avg*randn;
R0 = 0.015 + 0.01*rand;
vx = vx + w*R.*(R<=R0) + w*R0^2./R.*(R>R0);
end
else
vx = vx_base;
end
end

function phi = turbulence_phase_screen(Nx, Ny, L, lambda, Cn2)
fx = (-Nx/2:Nx/2-1)/L;
fy = (-Ny/2:Ny/2-1)/L;
[FX,FY] = meshgrid(fx,fy);
f = sqrt(FX.^2 + FY.^2);
f(f==0) = 1e-6;
PSD = 0.033*Cn2*f.^(-11/3).*exp(-(f/0.5).^2);
randP = 2*pi*rand(Ny,Nx);
H0 = sqrt(PSD).*exp(1i*randP);
phi = real(ifft2(ifftshift(H0))) * (2*pi/lambda)*(L/Nx);
end
```

```
%% ========== Model solution function (modified) ==========
function Y = solve_model(params, fixed)
% Analytical parameters (11)
Dw = params (1);% viscous layer diffusion coefficient

k_wm = params(2);% Mucin → Mucosal mass transfer coefficient (new variable parameter)

k_m_del = params(3);% mucosal layer degradation rate
Dm = params (4);% Mucosal diffusion coefficient

k_mb = params(5);% Mucosal → Blood mass transfer coefficient

k_b_del = params(6);% blood degradation rate

k BG = params(7);% Blood → Intestinal mucosa mass transfer coefficient

Dg = params (8);% intestinal mucosal diffusion coefficient
k_g_del = params (9);% intestinal mucosal degradation rate
k_in_del = params(10);% intestinal degradation rate

k_g_in = params (11);% Intestinal mucosa → Intestinal lumen mass transfer coefficient

% Analysis of fixed parameters
Cw0 = fixed.Cw0;
L1 = fixed.L1;
Nx1 = fixed.Nx1;
dx1 = L1/(Nx1-1);
dt = fixed.dt;
Nt = fixed.Nt;
L3 = fixed.L3;
Nx3 = fixed.Nx3;
dx3 = L3/(Nx3-1);
Vb = fixed.Vb;
A_mb = fixed.A_mb;
Lg = fixed.Lg;
Nxg = fixed.Nxg;
dxg = Lg/(Nxg-1);
A_bg = fixed.A_bg;
V_in = fixed.V_in;
A_in = fixed.A_in;
% Calculation of stability parameters
alpha1 = (Dw * dt / dx1^2) / 2;
alpha3 = (Dm * dt / dx3^2) / 2;
alpha_g = (Dg * dt / dxg^2) / 2;
Initial field%
u1 = zeros(Nx1, Nt+1); u1(:, 1) = 0;
u3 = zeros(Nx3, Nt+1); u3(:, 1) = 0;
```

```
Cb = zeros(1, Nt+1); Cb(1) = 0;
ug = zeros(Nxg, Nt+1); ug(:, 1) = 0;
cin = zeros(1, Nt+1); cin(1) = 0;
% Assembly implicit matrix (mucous layer)
e1 = ones(Nx1, 1);
A1 = diag((1 + 2*alpha1)*e1) - alpha1*diag(e1(2:end), 1) - alpha1*diag(e1(1:end-1), -1);
B1 = diag((1 - 2*alpha1)*e1) + alpha1*diag(e1(2:end), 1) + alpha1*diag(e1(1:end-1), -1);
A1(1, :) = 0; A1(1, 1) = 1;
A1(end, :) = 0; A1(end, end) = 1;
B1(end, :) = 0; B1(end, end) = 1;
[L1_mat, U1_mat] = lu(A1);
% Assembly implicit matrix (mucosa)
e3 = ones(Nx3, 1);
A3 = diag((1 + 2*alpha3)*e3) - alpha3*diag(e3(2:end), 1) - alpha3*diag(e3(1:end-1), -1);
B3 = diag((1 - 2*alpha3 - k_m_del*dt)*e3) + alpha3*diag(e3(2:end), 1) + alpha3*diag(e3(1:end-1),
-1);
A3(1, :) = 0; A3(1, 1) = 1;
A3(end, :) = 0; A3(end, end) = 1;
B3(end, :) = 0; B3(end, end) = 1;
[L3_mat, U3_mat] = lu(A3);
% Assembly implicit matrix (intestinal mucosa)
eg = ones(Nxg, 1);
Ag = diag((1 + 2*alpha_g)*eg) - alpha_g*diag(eg(2:end), 1) - alpha_g*diag(eg(1:end-1), -1);
Bg = diag((1 - 2*alpha_g - k_g_del*dt)*eg) + alpha_g*diag(eg(2:end), 1) + alpha_g*diag(eg(1:end-1),
-1);
Ag(1, :) = 0; Ag(1, 1) = 1;
Ag(end, :) = 0; Ag(end, end) = 1;
Bg(end, :) = 0; Bg(end, end) = 1;
[Lg_mat, Ug_mat] = lu(Ag);
% main time loop
for n = 1:Nt
% 5 minutes to stop vaccine supply
Cw_val = Cw0;
if (n-1)*dt >= 300
Cw_val = 0;
end
% Step1: Mucin implicit diffusion
rhs1 = B1 * u1(:, n);
rhs1(1) = Cw_val;
u1(:, n+1) = U1_mat \ (L1_mat \ rhs1);
u1 (:, n+1) = max (0, u1 (:, n+1));% to ensure nonnegativity

% Step2: Mucilage → mucosal mass transfer (using variable k_wm)

cw = u1(end, n);
```

```
cm0 = u3(1, n);
J_wm = k_wm * (cw-cm0);% Using the parameter k_wm
u1(end, n+1) = max(u1(end-1, n+1) - J_wm * dt / dx1, 0);
% Step3: Mucosal implicit diffusion + immunity
rhs3 = B3 * u3(:, n);
rhs3(1) = cm0 + J_wm * dt / dx3;
u3(:, n+1) = U3_mat \ (L3_mat \ rhs3);
u3(1, n+1) = cm0 + J_wm * dt / dx3;
u3(end, n+1) = max(u3(end-1, n+1) - J_wm * dt / dx3, 0);
u3 (:, n+1) = max (0, u3 (:, n+1));% Ensure nonnegativity

% Step4: Mucosal → blood mass transfer

J_mb_val = k_mb * (u3(end, n) - Cb(n));
Cb(n+1) = Cb(n) + dt / Vb * (A_mb * J_mb_val) - k_b_del * dt * Cb(n);
Cb(n+1) = max (0, Cb(n+1));% Ensure nonnegativity

% Step5: Blood → Intestinal mucosa mass transfer

J_bg_val = k_bg * (Cb(n) - ug(1, n));
rhs_g = Bg * ug(:, n);
rhs_g(1) = ug(1, n) + J_bg_val * dt / dxg;
ug(:, n+1) = Ug_mat \ (Lg_mat \ rhs_g);
ug(1, n+1) = ug(1, n) + J_bg_val * dt / dxg;
ug(end, n+1) = max(ug(end-1, n+1) - J_bg_val * dt / dxg, 0);
ug(:, n+1) = max (0, ug(:, n+1));% Ensure nonnegativity

% Step6: Intestinal mucosa → intestinal lumen mass transfer (remove the proliferation item)

J_g_in_val = k_g_in * (ug(end, n) - cin(n));
cin(n+1) = cin(n) + (A_in * J_g_in_val * dt / V_in) - k_in_del * dt * cin(n);
cin(n+1) = max(0, cin(n+1));% Ensure nonnegativity
end
% Calculate the objective function: the integral of bacterial concentration in blood over time
t = (0:Nt) * fixed.dt;
if iscolumn(t)
t = t';
end
if iscolumn(Cb)
Cb = Cb';
end
% Call the extrapolation function
Y = pk_auc_extrapolation(t, Cb, ...
"EndFraction", 0.2,...% Using last 20% of data
'MinPoints', 4,...% at least 4 points back
"RSQCutoff", 0.85);% R?> 0.85 to extrapolate
% Ensure that the output is scalar
```

```
if ~isscalar(Y)
Warning('Non-quantitative output: Use average')
Y = mean(Y(:));
end
end
```