Table of Contents

```
% Test script for satellite dynamics with nadir-pointing camera
% This script simulates a satellite in circular orbit and visualizes:
% 1. 3D animated orbit trajectory
% 2. Camera pointing direction
% 3. RA and Dec vs time

clear; close all; clc;
```

Parameters

```
params.mu = 3.986004418e14;     % Earth gravitational parameter [m^3/s^2]
params.R_e = 6378137;           % Earth radius [m]
params.mass = 500;              % Satellite mass [kg]
params.J2 = 1.08263e-3;         % J2 coefficient
params.I_CB = params.mass * eye(3); % Symmetric moment of inertia [kg*m^2]

% Control gains (not used for free orbit, but needed for function)
params.Kp_pos = 0;
params.Kd_pos = 0;
params.Kp_att = 0;
params.Kd_att = 0;
```

Initial Conditions

```
% Altitude
altitude = 200e3;  % 200 km
r_orbit = params.R_e + altitude;

% Circular orbit velocity
v_circ = sqrt(params.mu / r_orbit);

% Orbital period
T_orbit = 2 * pi * r_orbit / v_circ;
fprintf('Orbital period: %.2f minutes\n', T_orbit/60);

% Random orbital plane orientation
% Generate random inclination and RAAN (Right Ascension of Ascending Node)
```

```
inclination = rand() * pi;  % 0 to 180 degrees
RAAN = rand() * 2 * pi;     % 0 to 360 degrees
true_anomaly = rand() * 2 * pi;  % Random position in orbit

fprintf('Orbital inclination: %.2f degrees\n', rad2deg(inclination));
fprintf('RAAN: %.2f degrees\n', rad2deg(RAAN));

% Construct rotation matrices
R_RAAN = [cos(RAAN), -sin(RAAN), 0;
          sin(RAAN),  cos(RAAN), 0;
          0,          0,         1];

R_inc = [1, 0,            0;
         0, cos(inclination), -sin(inclination);
         0, sin(inclination),  cos(inclination)];

R_ta = [cos(true_anomaly), -sin(true_anomaly), 0;
        sin(true_anomaly),  cos(true_anomaly), 0;
        0,                  0,                 1];

% Position in orbital plane (perifocal frame)
r_perifocal = [r_orbit; 0; 0];
v_perifocal = [0; v_circ; 0];

% Transform to ECI
R_perifocal_to_ECI = R_RAAN * R_inc * R_ta;
r_eci = R_perifocal_to_ECI * r_perifocal;
v_eci = R_perifocal_to_ECI * v_perifocal;

% Initial attitude: nadir-pointing
% For nadir-pointing:
% - Body Z-axis points away from Earth (radial out)
% - Body X-axis in velocity direction
% - Body Y-axis completes right-handed system (normal to orbit plane)

% Define body axes in ECI frame
z_body_eci = r_eci / norm(r_eci);  % Radial direction (away from Earth)
y_body_eci = cross(r_eci, v_eci);
y_body_eci = y_body_eci / norm(y_body_eci);  % Orbit normal
x_body_eci = cross(y_body_eci, z_body_eci);  % Velocity direction

% Rotation matrix from ECI to Body
R_ECI_to_Body = [x_body_eci'; y_body_eci'; z_body_eci'];

% Convert rotation matrix to quaternion (scalar-last convention)
q0 = dcm_to_quaternion(R_ECI_to_Body);

% Initial angular velocity for nadir-pointing
% Angular velocity should be along the orbit normal (h = r x v direction)
omega_orbit = v_circ / r_orbit;  % Orbital angular rate [rad/s]

% Compute orbit angular momentum direction in ECI
h_eci = cross(r_eci, v_eci);
h_eci = h_eci / norm(h_eci);  % Unit vector
```

```
% Angular velocity in body frame (along orbit normal direction)
omega0 = -omega_orbit * h_eci;

fprintf('Orbital angular velocity: %.6f rad/s (period: %.2f min)\n', ...
        omega_orbit, 2*pi/omega_orbit/60);
fprintf('Omega in body frame: [%.6f, %.6f, %.6f] rad/s\n', omega0(1),
omega0(2), omega0(3));

% Assemble initial state
X0 = [r_eci; v_eci; q0; omega0];
```

```
Orbital period: 88.49 minutes
Orbital inclination: 122.17 degrees
RAAN: 272.79 degrees
```

Simulation

```
% Simulate for 1.5 orbits
t_sim = 1.5 * T_orbit;
tspan = linspace(0, t_sim, 500);

% Desired state (for dynamics function - we'll use zero control)
desired_state = X0;  % No control, free orbit

% ODE options
options = odeset('RelTol', 1e-8, 'AbsTol', 1e-10);

% Propagate orbit
fprintf('\nPropagating orbit...\n');
[t, X] = ode45(@(t, X) Sat_template(t, X, desired_state, params, ...
                'useJ2', false, 'useAtmDrag', false, 'useControl', false), ...
                tspan, X0, options);

fprintf('Simulation complete!\n');
```

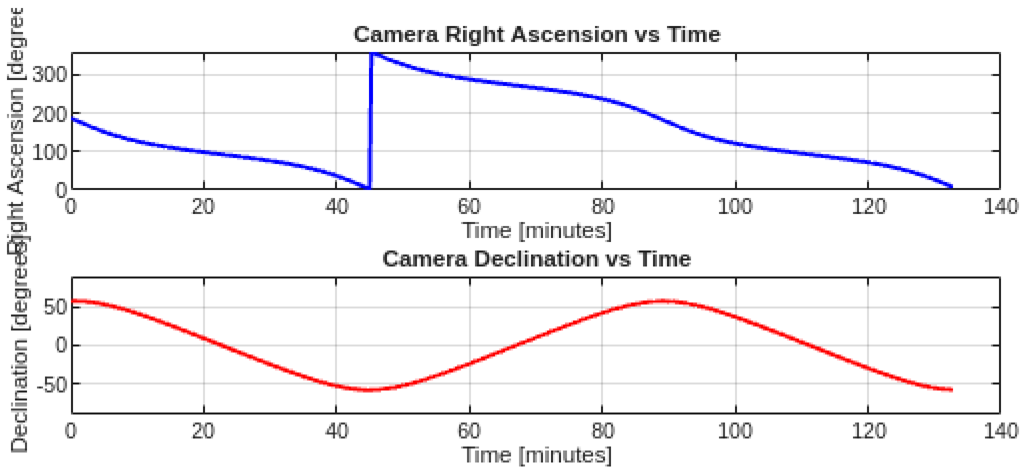
```
Propagating orbit...
Simulation complete!
```

Compute observations (RA, Dec)

```
n_points = length(t);
ra = zeros(n_points, 1);
dec = zeros(n_points, 1);
pointing_vectors = zeros(n_points, 3);

for i = 1:n_points
    obs = state_to_observation(X(i, :)', params);
    ra(i) = obs.ra;
    dec(i) = obs.dec;
    pointing_vectors(i, :) = obs.pointing_eci';
end
```

Plot 1: RA and Dec vs Time

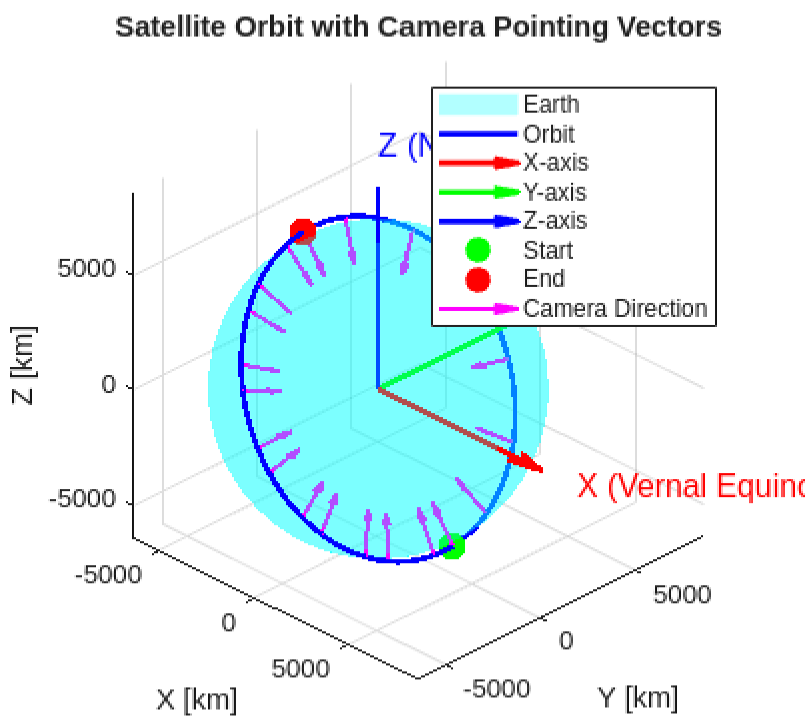
```
figure('Position', [100, 100, 1200, 500]);

subplot(2, 1, 1);
plot(t/60, ra, 'b-', 'LineWidth', 1.5);
xlabel('Time [minutes]');
ylabel('Right Ascension [degrees]');
title('Camera Right Ascension vs Time');
grid on;
ylim([0 360]);

subplot(2, 1, 2);
plot(t/60, dec, 'r-', 'LineWidth', 1.5);
xlabel('Time [minutes]');
ylabel('Declination [degrees]');
title('Camera Declination vs Time');
grid on;
ylim([-90 90]);
```



Plot 2: 3D Orbit Visualization (Static)

```
figure('Position', [100, 100, 800, 800]);

% Extract positions
pos = X(:, 1:3) / 1e3;  % Convert to km

% Plot Earth
[x_earth, y_earth, z_earth] = sphere(50);
x_earth = x_earth * params.R_e / 1e3;
y_earth = y_earth * params.R_e / 1e3;
z_earth = z_earth * params.R_e / 1e3;

surf(x_earth, y_earth, z_earth, 'FaceColor', 'cyan', 'FaceAlpha', 0.3, ...
```

```matlab
     'EdgeColor', 'none');
hold on;

% Plot orbit
plot3(pos(:, 1), pos(:, 2), pos(:, 3), 'b-', 'LineWidth', 2);

% Plot ECI axes
axis_length = params.R_e / 1e3 * 1.5;
quiver3(0, 0, 0, axis_length, 0, 0, 'r', 'LineWidth', 2, 'MaxHeadSize', 0.5);
quiver3(0, 0, 0, 0, axis_length, 0, 'g', 'LineWidth', 2, 'MaxHeadSize', 0.5);
quiver3(0, 0, 0, 0, 0, axis_length, 'b', 'LineWidth', 2, 'MaxHeadSize', 0.5);
text(axis_length*1.1, 0, 0, 'X (Vernal Equinox)', 'Color', 'r', 'FontSize',
12);
text(0, axis_length*1.1, 0, 'Y', 'Color', 'g', 'FontSize', 12);
text(0, 0, axis_length*1.1, 'Z (North Pole)', 'Color', 'b', 'FontSize', 12);

% Plot initial and final positions
plot3(pos(1, 1), pos(1, 2), pos(1, 3), 'go', 'MarkerSize', 10, ...
      'MarkerFaceColor', 'g');
plot3(pos(end, 1), pos(end, 2), pos(end, 3), 'ro', 'MarkerSize', 10, ...
      'MarkerFaceColor', 'r');

% Plot some camera pointing vectors
sample_indices = round(linspace(1, n_points, 20));
scale = 2000;  % km
for i = sample_indices
    p = pos(i, :);
    v = pointing_vectors(i, :) * scale;
    quiver3(p(1), p(2), p(3), v(1), v(2), v(3), 'm', 'LineWidth', 1.5, ...
            'MaxHeadSize', 1);
end

xlabel('X [km]');
ylabel('Y [km]');
zlabel('Z [km]');
title('Satellite Orbit with Camera Pointing Vectors');
legend('Earth', 'Orbit', 'X-axis', 'Y-axis', 'Z-axis', 'Start', 'End', ...
       'Camera Direction', 'Location', 'best');
axis equal;
grid on;
view(45, 30);
```

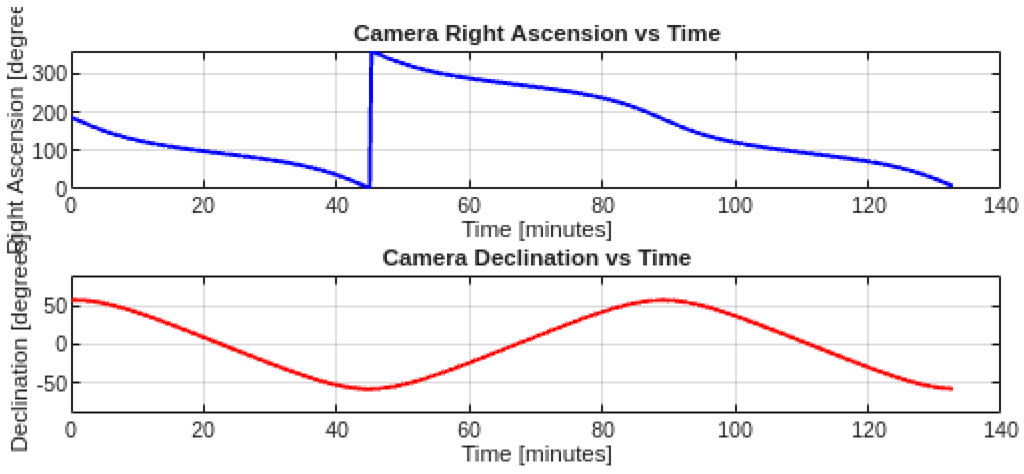


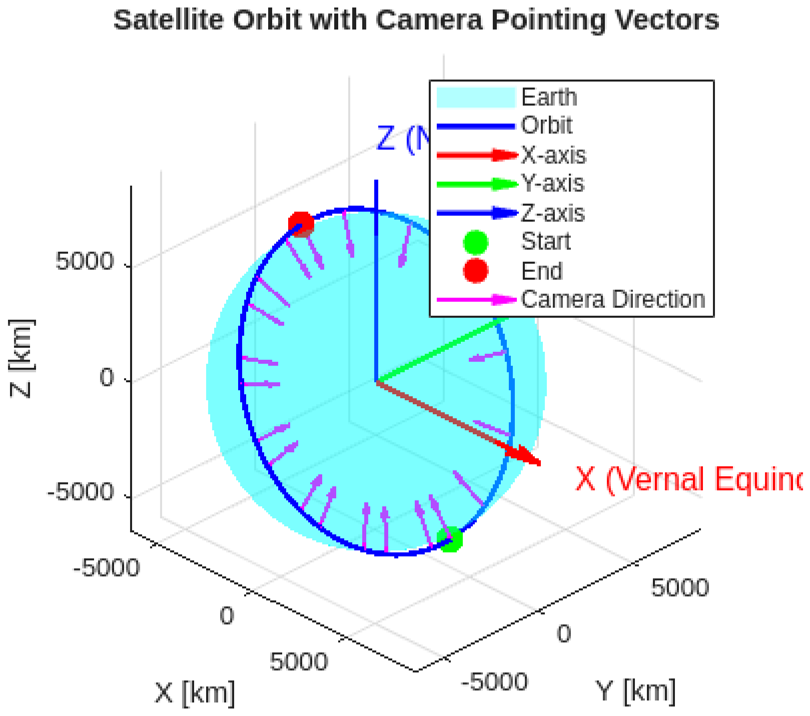
Plot 3: Animated 3D Orbit

```
figure('Position', [100, 100, 800, 800]);

% Initialize plot
[x_earth, y_earth, z_earth] = sphere(50);
x_earth = x_earth * params.R_e / 1e3;
y_earth = y_earth * params.R_e / 1e3;
z_earth = z_earth * params.R_e / 1e3;

h_earth = surf(x_earth, y_earth, z_earth, 'FaceColor', 'cyan', ...
               'FaceAlpha', 0.3, 'EdgeColor', 'none');
hold on;

% ECI axes
axis_length = params.R_e / 1e3 * 1.5;
quiver3(0, 0, 0, axis_length, 0, 0, 'r', 'LineWidth', 2, 'MaxHeadSize', 0.5);
quiver3(0, 0, 0, 0, axis_length, 0, 'g', 'LineWidth', 2, 'MaxHeadSize', 0.5);
quiver3(0, 0, 0, 0, 0, axis_length, 'b', 'LineWidth', 2, 'MaxHeadSize', 0.5);

% Orbit trail
h_trail = plot3(pos(1, 1), pos(1, 2), pos(1, 3), 'b-', 'LineWidth', 1.5);

% Satellite position
h_sat = plot3(pos(1, 1), pos(1, 2), pos(1, 3), 'ro', 'MarkerSize', 12, ...
              'MarkerFaceColor', 'r');

% Camera pointing vector
```

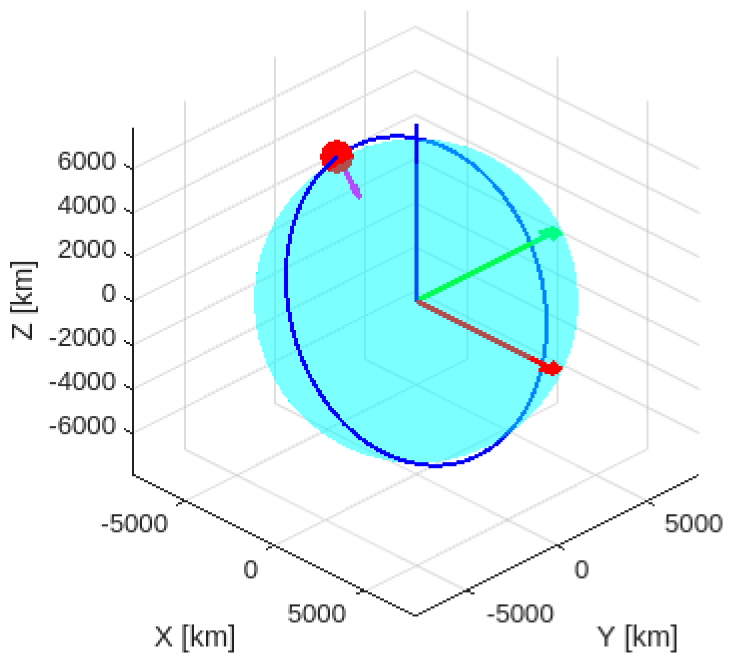
```
scale = 2000;  % km
v_init = pointing_vectors(1, :) * scale;
h_camera = quiver3(pos(1, 1), pos(1, 2), pos(1, 3), ...
                   v_init(1), v_init(2), v_init(3), ...
                   'm', 'LineWidth', 2, 'MaxHeadSize', 1);

xlabel('X [km]');
ylabel('Y [km]');
zlabel('Z [km]');
title('Animated Satellite Orbit - Camera Nadir Pointing');
axis equal;
grid on;
view(45, 30);
xlim([-1.2*r_orbit/1e3, 1.2*r_orbit/1e3]);
ylim([-1.2*r_orbit/1e3, 1.2*r_orbit/1e3]);
zlim([-1.2*r_orbit/1e3, 1.2*r_orbit/1e3]);

% Animation
fprintf('\nAnimating orbit (this may take a moment)...\n');
skip = 5;  % Show every 5th point for smoother animation
for i = 1:skip:n_points
    % Update trail
    set(h_trail, 'XData', pos(1:i, 1), 'YData', pos(1:i, 2), 'ZData',
pos(1:i, 3));

    % Update satellite position
    set(h_sat, 'XData', pos(i, 1), 'YData', pos(i, 2), 'ZData', pos(i, 3));

    % Update camera vector
    p = pos(i, :);
    v = pointing_vectors(i, :) * scale;

    % Delete old quiver and create new one (quiver set doesn't work well)
    delete(h_camera);
    h_camera = quiver3(p(1), p(2), p(3), v(1), v(2), v(3), ...
                       'm', 'LineWidth', 2, 'MaxHeadSize', 1);

    % Update title with time and RA/Dec
    title(sprintf('Time: %.1f min | RA: %.1f° | Dec: %.1f°', ...
                  t(i)/60, ra(i), dec(i)));

    drawnow;
    pause(0.01);
end

fprintf('Animation complete!\n');
```

```
Animating orbit (this may take a moment)...
Animation complete!
```


Camera Right Ascension vs Time
Right Ascension [degree
Time [minutes]
Camera Declination vs Time
Declination [degrees]
Time [minutes]


Satellite Orbit with Camera Pointing Vectors
Earth
Orbit
X-axis
Y-axis
Z-axis
Start
End
Camera Direction
Z (N
X (Vernal Equinc
Z [km]
X [km]
Y [km]

Time: 131.7 min | RA: 15.4° | Dec: -57.2°



Helper Function: DCM to Quaternion

```
function q = dcm_to_quaternion(R)
    % Convert Direction Cosine Matrix to quaternion (scalar-last)
    % Using Shepperd's method

    trace_R = trace(R);

    if trace_R > 0
        s = 0.5 / sqrt(trace_R + 1.0);
        qw = 0.25 / s;
        qx = (R(3,2) - R(2,3)) * s;
        qy = (R(1,3) - R(3,1)) * s;
        qz = (R(2,1) - R(1,2)) * s;
    elseif (R(1,1) > R(2,2)) && (R(1,1) > R(3,3))
        s = 2.0 * sqrt(1.0 + R(1,1) - R(2,2) - R(3,3));
        qw = (R(3,2) - R(2,3)) / s;
        qx = 0.25 * s;
        qy = (R(1,2) + R(2,1)) / s;
        qz = (R(1,3) + R(3,1)) / s;
    elseif R(2,2) > R(3,3)
        s = 2.0 * sqrt(1.0 + R(2,2) - R(1,1) - R(3,3));
        qw = (R(1,3) - R(3,1)) / s;
        qx = (R(1,2) + R(2,1)) / s;
        qy = 0.25 * s;
        qz = (R(2,3) + R(3,2)) / s;
    else
        s = 2.0 * sqrt(1.0 + R(3,3) - R(1,1) - R(2,2));
        qw = (R(2,1) - R(1,2)) / s;
        qx = (R(1,3) + R(3,1)) / s;
        qy = (R(2,3) + R(3,2)) / s;
        qz = 0.25 * s;
    end

    % Scalar-last convention
    q = [qx; qy; qz; qw];
    q = q / norm(q);
end
```

```
Orbital angular velocity: 0.001183 rad/s (period: 88.49 min)
Omega in body frame: [0.001000, 0.000049, 0.000630] rad/s
```

Published with MATLAB® R2025b