```
%function demoDopplerGroundReceiverNewSatData_GeodeticOutput()
    %% ========== 1. 参数与常量定义 ==========
    clear; clc;

    % WGS-84参数
    a = 6378137;                        % 长半轴 [m]
    f = 1/298.257223563;                % 扁率
    e2 = 2*f - f^2;                     % 第一偏心率平方

    % 常量
    c = 3e8;                            % 光速 [m/s]
    lambda = 0.19029367;                 % 载波波长 (例如GPS L1) [m]

    %% ========== 2. 接收机状态设定 ==========
    % 地面接收机地理坐标: lat = 45.75 deg, lon = 126.65 deg, alt = 0 km
    lat_rec_deg = 50.754236;
    lon_rec_deg = 137.615124;
    alt_rec_km  = 50.241;
    lat_rec_deg_rate = 0.00005;
    lon_rec_deg_rate = 0.00004;
    alt_rec_km_rate  = 0.00001;


    lat_rec_deg_guess = 52;
    lon_rec_deg_guess = 138;
    alt_rec_km_guess  = 0;
    lat_rec_deg_rate_guess = 0;
    lon_rec_deg_rate_guess = 0;
    alt_rec_km_rate_guess  = 0;


    % 转换为弧度和米
    lat_rec = deg2rad(lat_rec_deg);
    lon_rec = deg2rad(lon_rec_deg);
    alt_rec = alt_rec_km * 1000;  % [m]

    lat_rec_guess = deg2rad(lat_rec_deg_guess);
    lon_rec_guess = deg2rad(lon_rec_deg_guess);
    alt_rec_guess = alt_rec_km_guess * 1000;  % [m]

    lat_rec_rate = deg2rad(lat_rec_deg_rate);
    lon_rec_rate = deg2rad(lon_rec_deg_rate);
    alt_rec_rate = alt_rec_km_rate * 1000;  % [m]
```

```
    lat_rec_rate_guess = deg2rad(lat_rec_deg_rate_guess);
    lon_rec_rate_guess = deg2rad(lon_rec_deg_rate_guess);
    alt_rec_rate_guess = alt_rec_km_rate_guess * 1000;  % [m]


    % 计算接收机ECEF位置
    r_true = geodetic2ecef(lat_rec, lon_rec, alt_rec, a, e2);
    r_guess = geodetic2ecef(lat_rec_guess, lon_rec_guess,↙
alt_rec_guess, a, e2);
    % 接收机速度：设为零（静止）
    v_true = geodeticRates2ecef(lat_rec, lon_rec, alt_rec,↙
lat_rec_rate, lon_rec_rate, alt_rec_rate, a, e2);
    v_guess = geodeticRates2ecef(lat_rec_guess, lon_rec_guess,↙
alt_rec_guess, lat_rec_rate_guess, lon_rec_rate_guess,↙
alt_rec_rate_guess, a, e2);
    % 初始钟偏 (s)
    deltaR_true = 1e-8;
    % 假设接收机钟偏率 (dδ_R/dt)
    dDeltaRdt_true = 1e-8;  % [s/s]

    % 删除原来的状态向量定义中的deltaR，改为7维
    x_true = [r_true; v_true; dDeltaRdt_true]; % 7维状态向量 [r; v;↙
dDeltaRdt]
    % 初始猜测x0也调整为7维
    x_guess = [r_guess; v_guess; dDeltaRdt_true];  % 初始钟差率猜测
    x0 = x_guess;


    %% ========== 3. 卫星数据输入与转换 ==========
    % 卫星数据：每行：[Lat(deg), Lon(deg), Alt(km), LatRate(deg/s),↙
LonRate(deg/s), AltRate(km/s)]
    satData = [...
        51.979,   93.150,  552.679853,   0.014623,   0.094727,↙
0.002798;  % STARLINK-1008_44714
        43.716,  105.748,  551.083427,   0.035137,   0.067749,↙
0.007329;  % STARLINK-1039_44744
        37.589,  115.247,  549.894428,   0.041150,   0.055722,↙
0.007486;  % STARLINK-1193_45100
        23.995,  120.646,  547.522829,  -0.047565,   0.040975,↙
-0.004785;  % STARLINK-1582_46043
        31.153,  133.213,  548.216734,  -0.044950,   0.047234,↙
-0.006859;  % STARLINK-1292_45394
        37.890,  144.494,  549.371205,  -0.040929,   0.056219,↙
-0.007745;  % STARLINK-1300_45374
```

```
        48.505,  135.964,  346.164370,   0.028053,   0.085285,↙
0.006675;  % STARLINK-1170_45073
        % 52.683,  139.925,  430.119274,  -0.009879,   0.100688,↙
-0.002644
    ]; % STARLINK-1474_45738

    lat_deg = satData(:,1);
    lon_deg = satData(:,2);
    alt_km  = satData(:,3);
    latRate_degPerSec = satData(:,4);
    lonRate_degPerSec = satData(:,5);
    altRate_kmPerSec  = satData(:,6);

    % 单位转换：纬度和经度转换为弧度；alt从 km 转为 m
    lat_rad = deg2rad(lat_deg);
    lon_rad = deg2rad(lon_deg);
    alt_m = alt_km * 1000;
    latRate_radPerSec = deg2rad(latRate_degPerSec);
    lonRate_radPerSec = deg2rad(lonRate_degPerSec);
    altRate_mPerSec   = altRate_kmPerSec * 1000;

    nSat = size(satData,1);
    % 卫星钟偏率：取0 (简化)
    dot_delta_sat = zeros(nSat,1);
    r_sat = zeros(3, nSat);
    v_sat = zeros(3, nSat);
    for j = 1:nSat
        r_sat(:,j) = geodetic2ecef(lat_rad(j), lon_rad(j), alt_m(j),↙
a, e2);
        v_sat(:,j) = geodeticRates2ecef(lat_rad(j), lon_rad(j), alt_m↙
(j), ...
                                         latRate_radPerSec(j),↙
lonRate_radPerSec(j), altRate_mPerSec(j), a, e2);
    end


    %% ========== 4. 根据简化多普勒公式生成观测值 ==========
    % 对于第 j 颗卫星：
    % f_j(x_true) = hat_rho_j'*(v_true - v_sat(:,j)) +↙
c*dDeltaRdt_true - c*dot_delta_sat(j) + lambda*D_j = 0
    % 故：
    % D_j = -[hat_rho_j'*(v_true - v_sat(:,j)) + c*dDeltaRdt_true -↙
c*dot_delta_sat(j)]/lambda
    doppler_meas = zeros(nSat,1);
```

```
    for j = 1:nSat
        d_vec = x_true(1:3) - r_sat(:,j);
        norm_d = norm(d_vec);
        hat_rho = d_vec / norm_d;
        relative_v = x_true(4:6) - v_sat(:,j);
        term = hat_rho' * relative_v + c*x_true(7) - c*dot_delta_sat↙
(j);
        doppler_meas(j) = - term / lambda;
    end

    fprintf('生成的多普勒测量值 (Hz):\n');
    disp(doppler_meas);


    %% ========== 5. 生成多普勒方程与牛顿法迭代求解 ==========

    D = doppler_meas;
    maxIter = 50; tol = 1e-8;
    x = x0;
    [x_est, iter, err_hist] = newtonIteration7(x0, r_sat, v_sat,↙
doppler_meas, dot_delta_sat, c, lambda, maxIter, tol);


    %% ========== 6. 结果转换输出 ==========
    % 将初始猜测和最终估计的接收机位置转换为地理坐标
    [lat_est, lon_est, alt_est] = ecef2geodetic(x_est(1:3), a, e2);


    fprintf('真实位置 (geodetic: lat[deg],lon[deg],alt[km]):\n');
    fprintf('  Lat = %.6f deg, Lon = %.6f deg, Alt = %.3f km\n',↙
lat_rec_deg, lon_rec_deg, alt_rec/1000);
    fprintf('初始猜测位置 (geodetic):\n');
    fprintf('  Lat = %.6f deg, Lon = %.6f deg, Alt = %.3f km\n',↙
lat_rec_deg_guess, lon_rec_deg_guess, alt_rec_guess/1000);
    fprintf('最终估计位置 (geodetic):\n');
    fprintf('  Lat = %.6f deg, Lon = %.6f deg, Alt = %.3f km\n',↙
lat_est, lon_est, alt_est/1000);


    fprintf('真实速度 (ECEF):\n');
    fprintf('  x = %.6f m/s, y = %.6f m/s, z = %.6f m/s\n', x_true(4),↙
x_true(5), x_true(6));
    fprintf('初始猜测速度 (ECEF):\n');
    fprintf('  x = %.6f m/s, y = %.6f m/s, z = %.6f m/s\n', x_guess↙
```

```
(4), x_guess(5), x_guess(6));
    fprintf('最终估计速度 (ECEF):\n');
    fprintf('  x = %.6f m/s, y = %.6f m/s, z = %.6f m/s\n', x_est(4),↙
x_est(5), x_est(6));

    fprintf('最终残差 norm: %e\n', norm(dopplerResidual7(x_est, r_sat,↙
v_sat, doppler_meas, dot_delta_sat, c, lambda)));

    figure;
    plot(err_hist(1:iter),'-o');
    xlabel('迭代次数');
    ylabel('状态增量范数');
    title('牛顿迭代收敛情况');



%% ========== 函数：多普勒残差计算 ==========
function f = dopplerResidual7(x, r_sat, v_sat, D, dot_delta_sat, c,↙
lambda)
    nSat = size(r_sat,2);
    f = zeros(nSat,1);
    r = x(1:3);
    v = x(4:6);
    dDeltaRdt = x(7);
    for j = 1:nSat
        d_vec = r - r_sat(:,j);
        norm_d = norm(d_vec);
        hat_rho = d_vec / norm_d;
        relative_v = v - v_sat(:,j);
        f(j) = hat_rho' * relative_v + c*dDeltaRdt - c*dot_delta_sat↙
(j) + lambda*D(j);
    end
end
%% ========== 函数：雅可比矩阵计算 ==========
% 启用正确的7维雅可比函数
function J = dopplerJacobian7(x, r_sat, v_sat, ~, c, ~)
    nSat = size(r_sat,2);
    J = zeros(nSat,7);
    r = x(1:3);
    v = x(4:6);
    for j = 1:nSat
        d_vec = r - r_sat(:,j);
        norm_d = norm(d_vec);
        hat_rho = d_vec / norm_d;
```

```
        dHat_dr = (eye(3)/norm_d) - (d_vec*d_vec')/(norm_d^3);
        J(j,1:3) = (dHat_dr*(v - v_sat(:,j)))';
        J(j,4:6) = hat_rho';
        J(j,7) = c;
    end
end

%% ========== 函数：牛顿迭代 ==========
function [ x, iters, err_hist] = newtonIteration7(x0, r_sat, v_sat, D,↙
dot_delta_sat, c, lambda, maxIter, tol)
    x = x0;
    err_hist = zeros(maxIter,1);
    for k = 1:maxIter
        f_vec = dopplerResidual7(x, r_sat, v_sat, D, dot_delta_sat, c,↙
lambda);
        if norm(f_vec) < tol
            fprintf('迭代收敛，共迭代 %d 次。\n', k);
            break;
        end
        J = dopplerJacobian7(x, r_sat, v_sat, dot_delta_sat, c,↙
lambda);
        dx = - (J \ f_vec);
        % dx = dx';
        x = x + dx;
        [lat, lon, alt] = ecef2geodetic(x(1:3), 6378137, 1/298.↙
257223563);
        fprintf('  Lat = %.6f deg, Lon = %.6f deg, Alt = %.3f km\n',↙
lat, lon, alt/1000);
        err_hist(k) = norm(dx);
    end
    iters = k;
    err_hist = err_hist(1:iters);
end




%% ========== 辅助函数：地理坐标转ECEF ==========
function r_ecef = geodetic2ecef(lat, lon, alt, a, e2)
    % 将地理坐标 (lat, lon, alt) 转换为ECEF (WGS-84)
    % lat, lon: 弧度制, alt: m
    N = a ./ sqrt(1 - e2*sin(lat).^2);
    x = (N + alt).*cos(lat).*cos(lon);
    y = (N + alt).*cos(lat).*sin(lon);
```

```
    z = ((1 - e2)*N + alt).*sin(lat);
    r_ecef = [x; y; z];
end
%% ========== 辅助函数：ECEF转地理坐标 ==========
function [lat, lon, alt] = ecef2geodetic(r, a, e2)
    % ecef2geodetic: 将ECEF坐标转换为WGS-84地理坐标 (lat, lon, alt)
    % 输入:
    %   r: 3x1向量 [x; y; z] (m)
    %   a: 长半轴 (m)
    %   f: 扁率
    % 输出:
    %   lat, lon: 单位为度, alt: 单位为 m
    x = r(1); y = r(2); z = r(3);
    lon = atan2(y, x);
    p = sqrt(x.^2 + y.^2);
    % 初始纬度估计
    lat = atan2(z, p*(1 - e2));
    lat0 = 0;
    % 迭代求解
    while abs(lat - lat0) > 1e-12
        lat0 = lat;
        N = a / sqrt(1 - e2*sin(lat)^2);
        alt = p / cos(lat) - N;
        lat = atan2(z, p*(1 - e2*(N/(N+alt))));
    end
    % 转为度
    lat = rad2deg(lat);
    lon = rad2deg(lon);
end
%% ========== 辅助函数：地理速度转ECEF速度 ==========
function v_ecef = geodeticRates2ecef(lat, lon, alt, latRate, lonRate,↙
altRate, a, e2)
    % 将地理速率转换为ECEF速度 (m/s)
    N = a/sqrt(1 - e2*sin(lat)^2);
    dN_dlat = a*e2*sin(lat)*cos(lat)/( (1 - e2*sin(lat)^2)^(3/2) );

    dx_dlat = dN_dlat*cos(lat)*cos(lon) - (N+alt)*sin(lat)*cos(lon);
    dx_dlon = -(N+alt)*cos(lat)*sin(lon);
    dx_dalt = cos(lat)*cos(lon);

    dy_dlat = dN_dlat*cos(lat)*sin(lon) - (N+alt)*sin(lat)*sin(lon);
    dy_dlon = (N+alt)*cos(lat)*cos(lon);
    dy_dalt = cos(lat)*sin(lon);
```

```
    dz_dlat = (1-e2)*dN_dlat*sin(lat) + ((1-e2)*N+alt)*cos(lat);
    dz_dlon = 0;
    dz_dalt = sin(lat);

    vx = dx_dlat*latRate + dx_dlon*lonRate + dx_dalt*altRate;
    vy = dy_dlat*latRate + dy_dlon*lonRate + dy_dalt*altRate;
    vz = dz_dlat*latRate + dz_dlon*lonRate + dz_dalt*altRate;
    v_ecef = [vx; vy; vz];
end
```