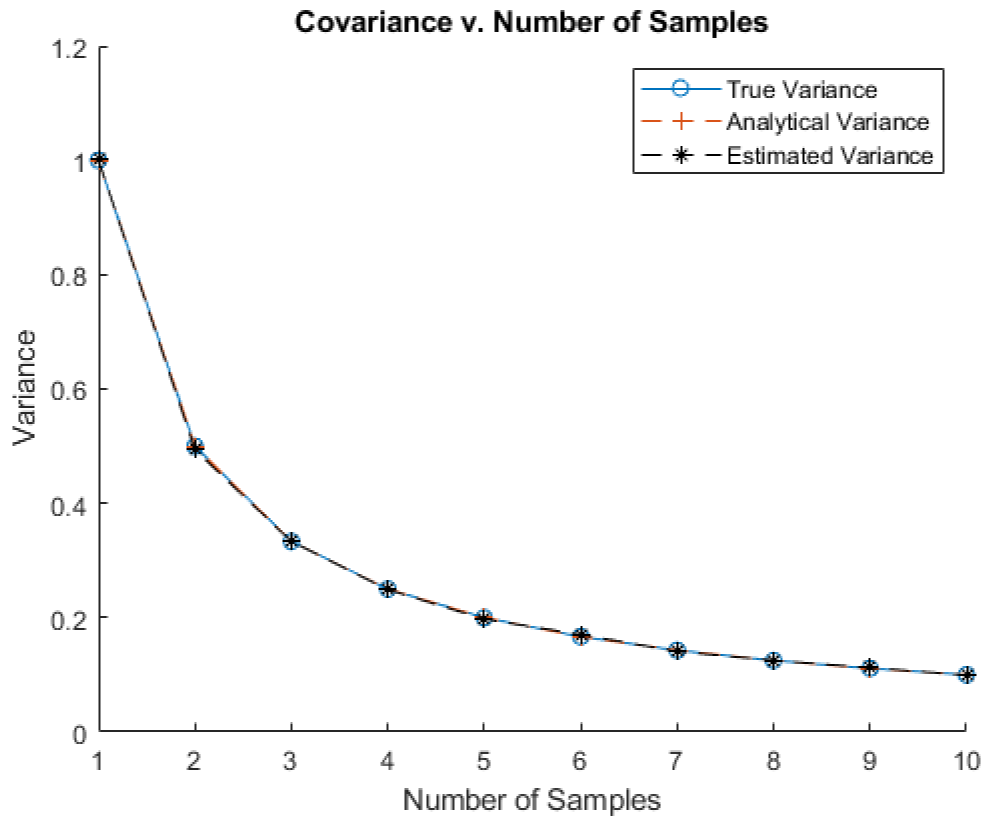
GPS Fundamentals - HW2

Table of Contents

Walter Livingston

QUESTION 1

```
clear; close all; clc;
a = 3;                          % True State
N_max = 10;                     % Number of Samples
M = 1e4;                        % Number of Monte Carlos
y_var = 1;                      % Measurement Variance

P = zeros(N_max,1);             % True Covariance
P_an = zeros(N_max,1);          % Analytical Covariance
P_hat = zeros(N_max,1);         % Monte Carlo Covariance
% Monte Carlo
for N = 1:N_max
    H = ones(N,1);              % Geometry Matrix
    P(N) = y_var/(H'*H);        % True Variance
    P_an(N) = y_var*(1/N);      % Analytical Covariance
    a_hat = zeros(M,1);         % State Estimates
    for i = 1:M
        y = H*a + sqrt(y_var)*randn(N,1);       % Noisy Measurements
        a_hat(i) = LS(y, H, a_hat(i));          % Least Squares Estimate
    end

    P_hat(N) = var(a_hat);      % Sample Variance
end

figure();
hold("on");
title("Covariance v. Number of Samples");
plot(P, '-o');
plot(P_an, '--+');
plot(P_hat, '--*k');
xlabel("Number of Samples");
ylabel("Variance");
legend("True Variance", "Analytical Variance", "Estimated Variance");
```

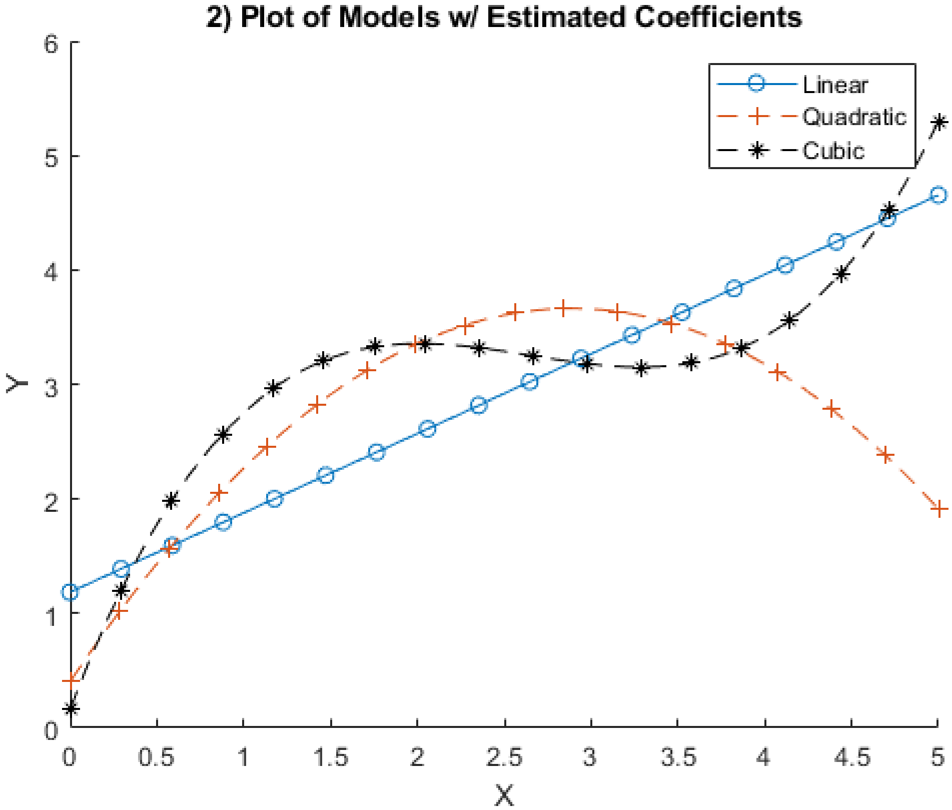


QUESTION 2

```
clear;
x = 0:4;                            % X-values
y = [0.181 2.680 3.467 3.101 3.437]';           % Y-Values
y_var = 0.4^2;                      % Variance on Y

Hab = [ones(length(x),1) x'];       % Linear Geometry Matrix
ab = LS(y, Hab, [0 0]');            % Linear Coefficient Estimates
Pab = y_var*inv(Hab'*Hab);          % Linear Covariance Matrix
fprintf('2a) Linear Fit Coefficients: [%0.4g %0.4g]\n', ab);

Habc = [Hab (x.^2)'];               % Quadratic Geometry Matrix
abc = LS(y, Habc, [0 0 0]');        % Quadratic Coefficient Estimates
Pabc = y_var*inv(Habc'*Habc);       % Quadratic Covariance Matrix
fprintf('2b) Quadratic Fit Coefficients: [%0.4g %0.4g %0.4g]\n', abc);

Habcd = [Habc (x.^3)'];             % Cubic Geometry Matrix
abcd = LS(y, Habcd, [0 0 0 0]');                % Cubic Coefficient Matrix
Pabcd = y_var*inv(Habcd'*Habcd);                % Cubic Covariance Matrix
fprintf('2c) Cubic Fit Coefficients: [%0.4g %0.4g %0.4g %0.4g]\n', abcd);

fprintf('Estimation error for a:\n');
fprintf('\t2a) Linear: %0.4g\n', sqrt(Pab(1,1)));
fprintf('\t2b) Quadratic: %0.4g\n', sqrt(Pabc(1,1)));
```

```
fprintf('\t2c) Cubic: %0.4g\n', sqrt(Pabcd(1,1)));
fprintf(['2d) The coefficent estimates are not consistent between the \n' ...
    '\tthe different models.  This is because it takes different \n' ...
    '\tcoefficients to get different order polynomials to pass through \n'
...
    '\tthe points defined by x and y.\n'])
fprintf(['2d) The estimation error from the linear model is the most \n' ...
    '\taccurate estimation error for a, as that is the system where\n' ...
    '\tthe states are the most overdefined.\n\n']);

range = [0 5];                  % Plotting Range
figure();
hold("on");
title("2) Plot of Models w/ Estimated Coefficients");
fplot(@(x) ab(1) + ab(2).*x, range, '-o');
fplot(@(x) abc(1) + abc(2).*x + abc(3).*x.^2, range, '--+');
fplot(@(x) abcd(1) + abcd(2).*x + abcd(3).*x.^2 + abcd(4).*x.^3, range, '--
*k');
xlabel("X");
ylabel("Y");
legend('Linear', 'Quadratic', 'Cubic');
```

```
2a) Linear Fit Coefficients: [1.187 0.6933]
2b) Quadratic Fit Coefficients: [0.4039 2.259 -0.3914]
2c) Cubic Fit Coefficients: [0.1625 3.989 -1.598 0.2012]
Estimation error for a:
    2a) Linear: 0.3098
    2b) Quadratic: 0.3764
    2c) Cubic: 0.3971
2d) The coefficent estimates are not consistent between the
    the different models.  This is because it takes different
    coefficients to get different order polynomials to pass through
    the points defined by x and y.
2d) The estimation error from the linear model is the most
    accurate estimation error for a, as that is the system where
    the states are the most overdefined.
```



QUESTION 3

```
clear;
a = [0 10 0 10]';               % A-values
b = [0 0 10 10]';               % B-values
r2 = [25 65 45 85]';            % Range-Squared Values [m]
r_var = 0.5^2;                  % Variance on Range Measurements

H_func = @(s) [2*(s(1)-a) 2*(s(2)-b)];          % Jacobian (Geometry Matrix)
s = [0 0]';                     % Initial State Estimate
ds = 1e3;                       % Initial State Delta
% Least Squares
while ds > 1e-4
    r2_hat = (s(1) - a).^2 + (s(2) - b).^2;     % Range Estimate
    H = H_func(s);              % Geometry Matrix
    ds = pinv(H)*(r2 - r2_hat); % Delta State Estimate
    s = s + ds;                 % New State Estimate
end

P = r_var*inv(H'*H);            % Covariance Matrix for Position Estimate
err = sqrt(diag(P));            % Standard Deviation of Position Estimate

M = 1e4;                        % Number of Monte Carlos
s_hat = zeros(2,M);             % Vector of Initial State Estimates
% Monte Carlo
```

```
for i = 1:M
    r2 = [25 65 45 85]' + sqrt(r_var)*randn(4,1);   % Initialize noisy range
measurements
    s = [0 0]';                   % Initialize Current State Estimate
    ds = 1e3;                     % Initial State Delta
    % Least Squares
    while ds > 1e-4
        r2_hat = (s(1) - a).^2 + (s(2) - b).^2;     % Range Estimate
        H = H_func(s);            % Geometry Matrix
        ds = pinv(H)*(r2 - r2_hat);                 % Delta State Estimate
        s = s + ds;               % New State Estimate
    end
    s_hat(:,i) = s;               % Save Current Monte Carlo State
end
err_hat = std(s_hat')';           % Sample Standard Deviation

fprintf(['3b) The expected error (1-sigma) on the estimate: ' ...
    '[%0.3g %0.3g]\n'], err);
fprintf('3c) The position solution: [%0.2g %0.2g]\n', s);
fprintf(['3d) The estimated error (1-sigma) on the estimate: ' ...
    '[%0.3g %0.3g]\n\n'], err_hat);

3b) The expected error (1-sigma) on the estimate: [0.0233 0.0246]
3c) The position solution: [3 4]
3d) The estimated error (1-sigma) on the estimate: [0.0231 0.0247]
```

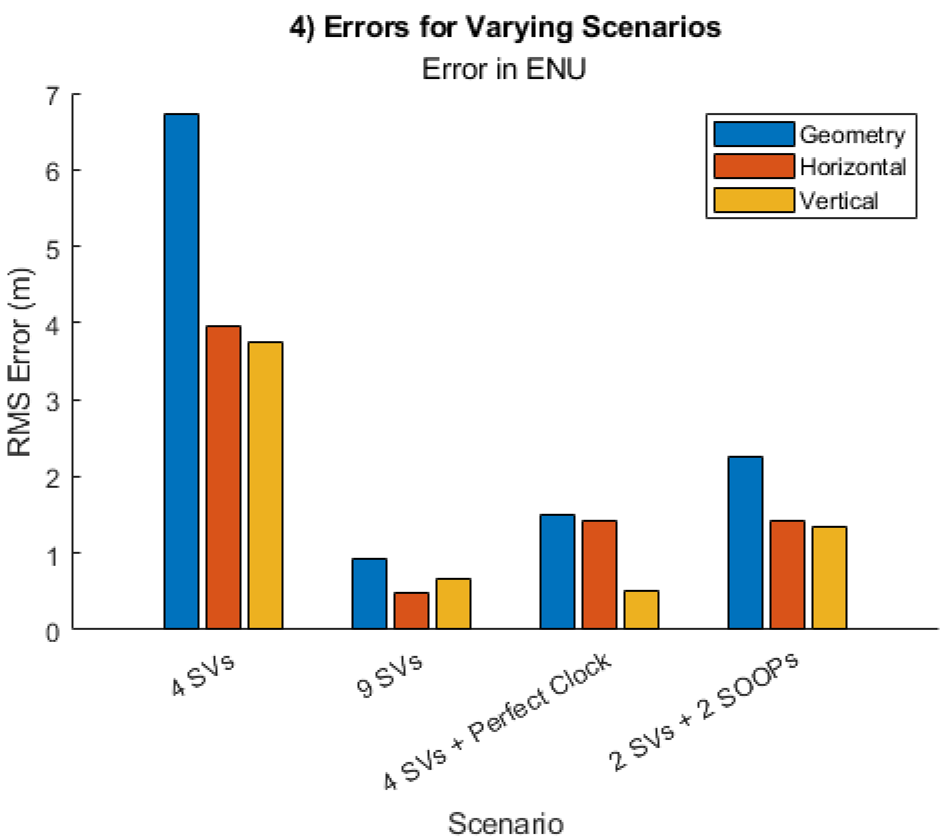
QUESTION 4

```
clear;
filename = 'HW2_data.txt';        % Data File
T = readlines(filename);          % Read in Data File
Xs = zeros(length(T)-4, 3);       % Satellite Positions
rho = zeros(length(T)-4, 1);      % Pseudoranges
r_var = 0.5^2;                    % Receiver Variance
% Read in Satellite Positions and Pseudoranges
for i = 4:length(T)
    data = strsplit(T(i));        % Split String
    Xs(i-3,1) = str2double(data(2));                % SV X-position
    Xs(i-3,2) = str2double(data(3));                % SV Y-position
    Xs(i-3,3) = str2double(data(4));                % SV Z-position
    rho(i-3) = str2double(data(5));                 % Pseudorange
end

X0 = [0 0 0 0];                   % Initial User Position Estimate (Center of
the Earth)
[Xu_4, H_4, idx_4] = GPS_LS(rho(1:4), Xs(1:4,:), X0);       % LS w/ 4 SVs
fprintf('4a) Converged in %d iterations w/ 4 SVs.\n', idx_4);
[Xu_9, H_9, idx_9] = GPS_LS(rho(1:9), Xs(1:9,:), X0);       % LS w/ 9 SVs
fprintf('4b) Converged in %d iterations w/ 9 SVs.\n', idx_9);
[Xu_cl, H_cl, idx_cl] = GPS_LS(rho(1:4), Xs(1:4,:), ...
    [X0(1:3),Xu_9(4)]);           % LS w/ perfect clock
fprintf(['4c) Converged in %d iterations w/ 9 SVs & a' ...
```

```
    'perfect clock.\n'], idx_cl);
% [Xu_SOOP, H_SOOP] = GPS_LS(rho(1:9), Xs(1:9,:), ...
%   X0);       % LS w/ 2 SOOPs & 2 SVs
fprintf('4d) Did not converge w/ 2 SOOPs and 2 SVs due to poor geometry\n');
X_guess = [423000 -5362000 3417000 0];                % Better Position
Estimate
[Xu_SOOP, H_SOOP, idx_SOOP] = GPS_LS([rho(1:2);rho(10:11)], ...
    [Xs(1:2,:);Xs(10:11,:)], X_guess);                % LS w/ 2 SOOPs & 2 SVs
fprintf(['4e) Converged in %d iterations w/ 2 SVs & 2 SOOPs given ' ...
    'a better initial position estimate.\n\n'], idx_SOOP);

lla0 = ecef2lla(Xu_9(1:3));
% This function converts to ENU
[DOP4, err4] = GPS_STATS(r_var, H_4, lla0);
[DOP9, err9] = GPS_STATS(r_var, H_9, lla0);
[DOPcl, errCl] = GPS_STATS(r_var, H_cl, lla0);
[DOP_SOOP, errSOOP] = GPS_STATS(r_var, H_SOOP, lla0);
x = ["4 SVs" "9 SVs" "4 SVs + Perfect Clock" "2 SVs + 2 SOOPs"];
y = [err4.G err9.G errCl.G errSOOP.G;
     err4.H err9.H errCl.H errSOOP.H;
     err4.V err9.V errCl.V errSOOP.V];

fprintf('All positions are in ECEF\n');
fprintf(['4a) The position and bias solution w/ 4 SVs: ' ...
    '[%0.6g %0.6g %0.6g %0.6g] m\n'], Xu_4);
fprintf(['4b) The position and bias solution w/ 9 SVs: ' ...
    '[%0.6g %0.6g %0.6g %0.6g] m\n'], Xu_9);
fprintf(['4c) The position and bias solution w/ 4 SVs & a perfect clock: '
...
    '[%0.6g %0.6g %0.6g %0.6g] m\n'], [Xu_cl Xu_9(4)]);
fprintf(['4e) The position and bias solution w/ 2 SVs & 2 SOOPs: ' ...
    '[%0.6g %0.6g %0.6g %0.6g] m\n\n'], Xu_SOOP);

figure();
hold("on");
title('4) Errors for Varying Scenarios');
subtitle("Error in ENU");
bar(x,y);
xlabel('Scenario');
ylabel('RMS Error (m)');
legend('Geometry', 'Horizontal', 'Vertical');
```

```
4a) Converged in 6 iterations w/ 4 SVs.
4b) Converged in 5 iterations w/ 9 SVs.
4c) Converged in 5 iterations w/ 9 SVs & aperfect clock.
4d) Did not converge w/ 2 SOOPs and 2 SVs due to poor geometry
4e) Converged in 15 iterations w/ 2 SVs & 2 SOOPs given a better initial
position estimate.

All positions are in ECEF
4a) The position and bias solution w/ 4 SVs: [423327 -5.36166e+06
3.41729e+06 100] m
4b) The position and bias solution w/ 9 SVs: [423327 -5.36166e+06
3.41729e+06 100] m
```

```
4c) The position and bias solution w/ 4 SVs & a perfect clock: [423327
-5.36166e+06 3.41729e+06 100] m
4e) The position and bias solution w/ 2 SVs & 2 SOOPs: [423327 -5.36166e+06
3.41729e+06 100] m
```

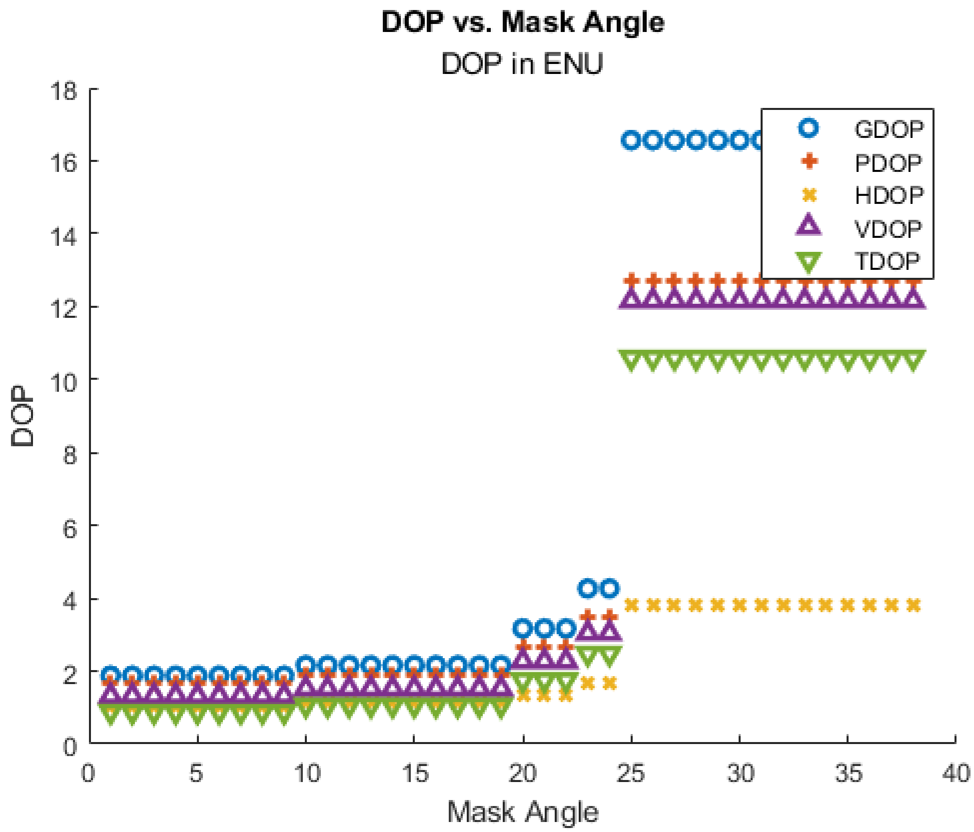


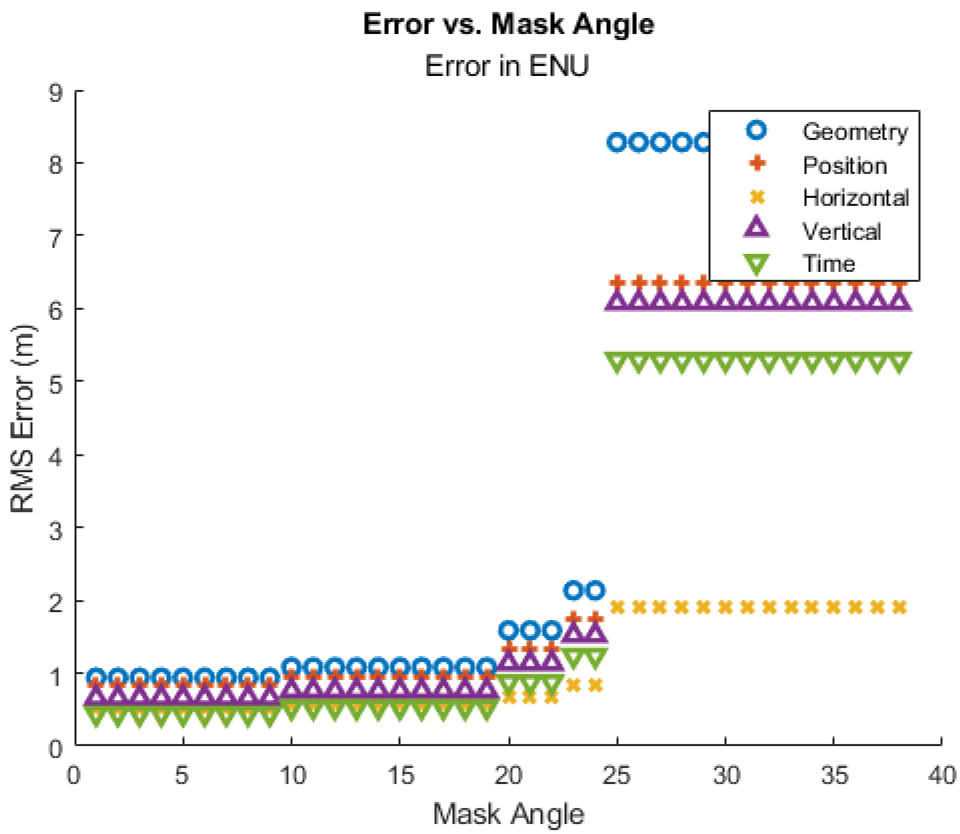
QUESTION 5

```
c = physconst("lightspeed");
A1 = 5e-9;
I_func = @(theta) A1 * (1 + 16*(0.53 - theta/180)^3);

rho9 = rho(1:9);
Xs9 = Xs(1:9,:);

[E, N, U] = ecef2enu(Xs9(:,1), Xs9(:,2), Xs9(:,3), ...
    lla0(1), lla0(2), lla0(3), wgs84Ellipsoid('meter'));
h = vecnorm([E, N]');
ang = atan2d(U, h');

i = 1;
a_range = 1:90;
for a = a_range
    Xs9_filt = Xs9(ang > a, :);
    rho_filt = rho(ang > a);
    if length(rho_filt) > 4
```

```
        dx = 1e3*ones(4,1);
        state(i,:) = [0 0 0 0];
        while norm(dx) > 1e-4
            r = vecnorm((Xs9_filt - state(i,1:3)), 2, 2);
            U = (Xs9_filt - state(i,1:3))./r;
            H = [-U ones(length(U),1)];
            rho_hat = r + state(i,4) + c*I_func(i);
            dx = pinv(H)*(rho_filt - rho_hat);
            state(i,:) = state(i,:) + dx';
        end
        % This function converts to ENU
        [DOP(i), err(i)] = GPS_STATS(r_var, H, lla0);
        i = i + 1;
    else
        break;
    end
end

figure();
hold("on");
title("DOP vs. Mask Angle");
subtitle("DOP in ENU");
plot(vertcat(DOP.G), 'o', LineWidth = 2);
plot(vertcat(DOP.P), '+', LineWidth = 2);
plot(vertcat(DOP.H), 'x', LineWidth = 2);
plot(vertcat(DOP.V), '^', LineWidth = 2);
plot(vertcat(DOP.T), 'v', LineWidth = 2);
xlabel('Mask Angle');
ylabel('DOP');
legend('GDOP', 'PDOP', 'HDOP', 'VDOP', 'TDOP');

figure();
hold("on");
title("Error vs. Mask Angle");
subtitle("Error in ENU");
plot(vertcat(err.G), 'o', LineWidth = 2);
plot(vertcat(err.P), '+', LineWidth = 2);
plot(vertcat(err.H), 'x', LineWidth = 2);
plot(vertcat(err.V), '^', LineWidth = 2);
plot(vertcat(err.T), 'v', LineWidth = 2);
xlabel('Mask Angle');
ylabel('RMS Error (m)');
legend({'Geometry', 'Position', 'Horizontal', 'Vertical', 'Time'});

fprintf(['5) In this particular scenario, the mask angle can be rather\n' ...
    '\thigh without the position error going up significantly. This is \n'
...
    '\tprobably due to the fact that at least 5 satellites are above 30 \n'
...
    '\t deg from the horizon. In a scenario where more of the \n' ...
    '\tSVs were on the horizon, you would see the error rise dramatically
\n' ...
    '\tat lower mask angles.\n']);
```

5) In this particular scenario, the mask angle can be rather high without the position error going up significantly. This is probably due to the fact that at least 5 satellites are above 30 deg from the horizon. In a scenario where more of the SVs were on the horizon, you would see the error rise dramatically at lower mask angles.





FUNCTIONS

```
function [s] = LS(y, H, s0)
    s = s0;
    ds = 1e3*ones(length(s0),1);
    while norm(ds) > 1e-4
        y_hat = H*s;
        ds = pinv(H)*(y - y_hat);
        s = s + ds;
    end
end

function [Xu, H, idx] = GPS_LS(rho, Xs, Xu)
    dx = 1e3*ones(4,1);
    flag = false;
    if Xu(4) ~= 0
        b = Xu(4);
        Xu = [0 0 0];
        rho_hat_func = @(r, Xu) r + b;
        H_func = @(U) -U;
    else
        rho_hat_func = @(r, Xu) r + Xu(4);
        H_func = @(U) [-U ones(length(U),1)];
    end
    idx = 0;
```

```
    while norm(dx) > 1e-4
        r = vecnorm((Xs - Xu(1:3)), 2, 2);
        U = (Xs - Xu(1:3))./r;
        rho_hat = rho_hat_func(r, Xu);
        H = H_func(U);
        dx = pinv(H)*(rho - rho_hat);
        Xu = Xu + dx';
        idx = idx + 1;
    end
end

function [DOP, error] = GPS_STATS(sigma2, H, lla0)
    P = sigma2*inv(H'*H);
    C = ECEF_ENU(lla0(1), lla0(2));
    P(1:3, 1:3) = C*P(1:3, 1:3)*C';
    DOP.G = sqrt(sum(diag(P)./sigma2));
    DOP.P = sqrt(sum(diag(P(1:3,1:3))./sigma2));
    DOP.H = sqrt(sum(diag(P(1:2,1:2))./sigma2));
    DOP.V = sqrt(sum(diag(P(3,3))./sigma2));
    error.G = sqrt(sigma2)*DOP.G;
    error.P = sqrt(sigma2)*DOP.P;
    error.H = sqrt(sigma2)*DOP.H;
    error.V = sqrt(sigma2)*DOP.V;
    if size(H) > 3
        DOP.T = sqrt(sum(diag(P(4,4))./sigma2));
        error.T = sqrt(sigma2)*DOP.T;
    end
end

function [C] = ENU_ECEF(lat, lon)
    C = [-sind(lon), -cosd(lon)*sind(lat), cosd(lon)*cosd(lat);
          cosd(lon), -sind(lon)*sind(lat), sind(lon)*cosd(lat);
                  0,             cosd(lat),            sind(lat)];
end

function [C] = ECEF_ENU(lat, lon)
    C = ENU_ECEF(lat, lon)';
end
```

Published with MATLAB® R2023b