GPS User position estimation using ephemeris data from satellites

Ishaan Sharma
Electrical and Computer Engineering Department
ID: 40053363
Concordia University
Avionics Navigation Systems ENGR 6461

***Abstract*—In this report ephemeris data received from satellites by global positioning system receivers is utilized to find out the user position. User's latitude and longitude is pinpointed on Google maps using GPS pseudo-range equation. Linearized GPS pseudo-range equation is solved by weighted least square. The radio signal propagation errors through ionosphere and troposphere are calculated and compensated in the calculation. Satellite clock bias and user clock bias are also included in the calculation model. Error analysis is carried out by constructing the error ellipses.**

1. Introduction

Global positioning system to find users position is an integral part of our most used devices like mobile phones, laptops, car navigation systems and numerous apps. These coordinates are generated by gps receivers which decode data received from satellites available at that particular instant. GPS positioning is based on trilateration, which is the method of determining position by measuring distances to points at known coordinates. At a minimum, trilateration requires 3 ranges to 3 known points. But in GPS 4 pseudo-ranges to satellites are utilized to find out four unknowns. GPS is owned by the US government and operated by the US air force. Other countries have also implemented their own GPS like systems. Examples are GLONASS by Russia, Galileo by the European Union and BeiDou by China.

The report comprises of the assumptions while calculating the gps position, flowchart of the algorithm for coding, explanation of individual components of the code followed by results and error analysis.

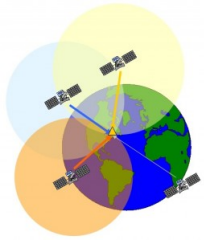


Figure 1. Trilateration using 4 satellites [1]

2. Assumptions

1) Earth's shape is modeled by an ellipsoid which means that geometrically, the equatorial radius is longer than the polar axis by about 23 km. The direction of gravity does not point to the center of the earth. This model is used to convert the geocentric coordinates of user[X,Y,Z] to the geodetic model and locate the latitude, longitude and height on the Google maps.

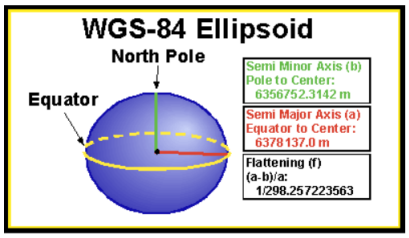


Figure 2. Earth ellipsoid model [2]

2) In order to get the user position from this algorithm at any instant 4 or more satellites must be visible to the user. The satellite constellation is designed in such a way.
3) Least square solution converges after only 4 or 5 iterations. Which means that the value of "dx" (Difference between the true and calculated position) and "db" (Difference in actual and calculated user clock bias) is reduced to e-3 in these iterations when the corrected pseudo range, user clock bias and errors are fed back in each iteration.
4) The values of constants for GPS equations like c (Speed of light), μ (Earth's gravitational constant), $\dot{\theta}$ (Earth's rotation rate), F (rel cor term constant). The initial value of τ = 0.075 sec. This value is updated in each iteration of algorithm.

3. Flowchart for the matlab code

Here the flowchart of the matlab main code is explained and in further sections each component of the code is explained specifying how each component is calculated.

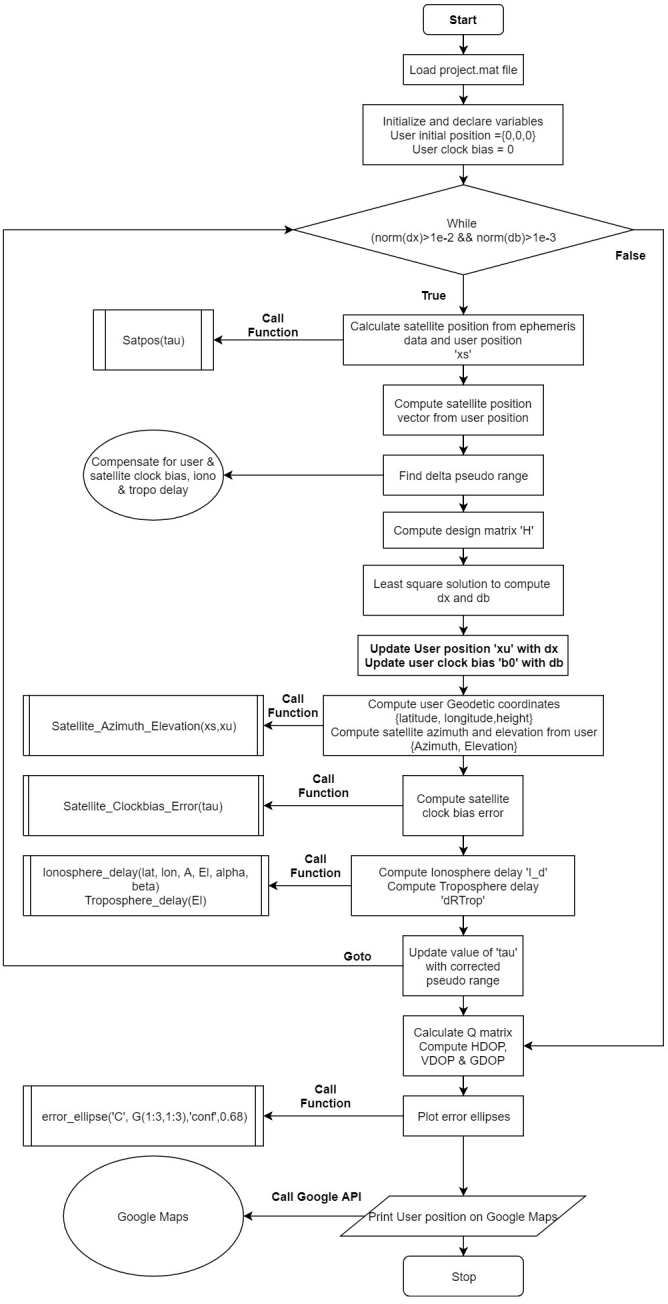


Figure 3. Flowchart for matlab code

4. Components of the code

4.1. Ephemeris data from satellites

Project_data· mat file consists of ephemeris data from 6 satellites. It also includes values of pseudorange measured by receiver as "pr". Iono error constants as "iono". Theses values are extracted into respective variables.

4.2. Initial position of user and user clock bias

For starting the algorithm initial position of user is geocentric origin of the ECEF coordinate system i.e [0,0,0]. User clock bias is also assumed to be [0].

4.3. Satellite position from Ephemeris data

Calculating Satellite position from ephemeris is given in [3] i.e IS-GPS-200D. Following images taken from [3] tell about the parameters involved in finding coordinates and also the equations for calculation. Output x^k, y^k, z^k are the ECEF coordinates. "Satpos.m" function does the calculation. The input for this function is τ which is updated in each iteration.

TABLE I
COMPONENTS OF EPHEMERIS DATA

Name	Description	Units
M_0	Mean anomaly at reference time	Semicircle
Δn	Mean motion difference from computed value	Semicircle/s
e	Eccentricity	Dimensionless
$\sqrt{a}$	Square root of semimajor axis	$m^{1/2}$
Ω_0	Longitude of ascending node of orbit plane at weekly epoch	Semicircle
i_0	Inclination angle at reference time	Semicircle
ω	Argument of perigee	Semicircle
$\dot{\Omega}$	Rate of right ascension	Semicircle/s
$IDOT$	Rate of inclination angle	Semicircle/s
C_{uc}	Amplitude of cosine harmonic correction term to the argument of latitude	Rad
C_{us}	Amplitude of sine harmonic correction term to the argument of latitude	Rad
C_{rc}	Amplitude of cosine harmonic correction term to the orbit radius	m
C_{rs}	Amplitude of sine harmonic correction term to the orbit radius	m
C_{ic}	Amplitude of cosine harmonic correction term to the angle of inclination	Rad
C_{is}	Amplitude of sine harmonic correction term to the angle of inclination	Rad
t_{0e}	Ephemeris reference time	s
$IODE$	Issue of data, ephemeris	Dimensionless

Figure 4.

TABLE 2
ALGORITHM FOR COMPUTING SATELLITE POSITION

Equation	Description
$\mu = 3.986008\times10^{14} m^3/s^2$	WGS 84 value of earth's universal gravitational parameter
$\dot{\Omega}_e = 7.292115167\times10^{-5} rad/s$	WGS 84 value of earth's rotation rate
$a = (\sqrt{a})^2$	Semimajor axis
$t_{n+1} = t - t_{0e}$	Time from ephemeris reference epoch
$f_n = \tan^{-1}\left\{\frac{\sqrt{(1-e^2}\sin E_n/(1-e\cos E_n)}{(\cos E_n - e)/(1-e\cos E_n)}\right\}$	True anomaly
$E_n = \cos^{-1}(\frac{e+\cos f_n}{1+e\cos f_n})$	Eccentric anomaly from cosine
$\phi_n = f_n + \omega$	Argument of latitude
$\delta\mu_n = C_{uC}\cos 2\phi_n + C_{uS}\sin 2\phi_n$	Second-harmonic correction to argument of latitude
$\delta r_n = C_{rC}\cos 2\phi_n + C_{rS}\sin 2\phi_n$	Second-harmonic correction to radius
$\delta i_n = C_{iC}\cos 2\phi_n + C_{iS}\sin 2\phi_n$	Second-harmonic correction to inclination
$\mu_n = \phi_n + \delta\mu_n$	Corrected argument of latitude
$r_n = a(1-e\cos E_n) + \delta r_n$	Corrected radius
$i_n = i_0 + \delta i_n + (IDOT)t_n$	Corrected inclination
$x_n' = r_n\cos\mu_n$	X coordinate in orbit plane
$y_n' = r_n\sin\mu_n$	Y coordinate in orbit plane
$\Omega_n = \Omega_0 + (\dot{\Omega} + \dot{\Omega}_e)t_n - \dot{\Omega}_e t_{0e}$	Corrected longitude of ascending node
$x_n = x_n'\cos\Omega_n - y_n'\cos i_n\sin\Omega_n$	ECEF X coordinate
$y_n = x_n'\sin\Omega_n + y_n'\cos i_n\sin\Omega_n$	ECEF Y coordinate
$z_n = x_n'\sin i_n$	ECEF Z coordinate

Figure 5.

4.4. Compute satellite position vector from user

Satellite position obtained in the previous step is utilized to find the position vector from user to satellite.

4.5. Calculate the $\delta\rho$

The equation used to calculate the $\delta\rho$ is mentioned:

$$\delta\rho = \rho_{measured} - \rho_{corrected} \quad (1)$$

$$\rho_{measured} = pr$$

$$\rho_{corrected} = \sqrt{(x^k - x_u)^2 + (y^k - y_u)^2 + (z^k - z_u)^2} + (b_0 - c * (\delta T^{st}_{clk})) + c * I_d + \delta RTrop$$

where

$xu = x_u, y_u, z_u =$ User's coordinates in ECEF system
$xs = x^k, y^k, z^k =$ Satellite's coodinates in ECEF system
$b_0 =$ User's clock bias
$c =$ Speed of light
$\delta T^{st}_{clk} =$ Satellite's clock bias error
$I_d =$ Ionosphere delay
$\delta RTrop =$ Troposphere delay

4.6. Compute design matrix H

$$H = [-\text{Satellites position unit vectorones(1,6)}]_{6x4} \quad (2)$$

4.7. Least square solution

$$\delta r = (H^T \cdot H)^{-1} \cdot H^T \cdot \delta\rho \quad (3)$$

4.8. Update user position and user clock bias error

$$xu = xu + \delta x$$
$$b_0 = b0 + \delta b$$

where

$$\delta x = \delta r(1:3)$$
$$\delta b = \delta r(4)$$

4.9. Calculate satellite Azimuth and Elevation from user position

This is calculates by Satellite_Azimuth_Elevation function. The inputs to this function are user and satellite position in ECEF coordinates i.e xu & xs. Output of this function is satellite Azimuth "A", Elevation "E" in radians and users latitude "lat", longitude "lon" in degrees and height "h" in meters. This is done in three steps

1) Convert users ECEF coordinates to geodetic [lat, lon, h]
2) Convert the satellites position vector coordinate from user position into ENU coordinate system [E,N,U]
3) Calculate Azimuth and Elevation

$$A = \text{atan2(E,N)} \quad (4)$$
$$E = \text{asin(U/norm[E,N,U])} \quad (5)$$

4.10. Calculate clock bias error of satellites

The function used to calculate this is Satellite_Clockbias_Error. Input to this function is "τ" and output is δT^{st}_{clk} in seconds. The equation utilized to calculate is

$$\delta T^{st}_{clk} = af0 + af1 \cdot T + af2 \cdot (T)^2 + F \cdot e \cdot sqrtA \cdot sin(Ek) - Tgd \quad (6)$$

The values of constants is taken from project_data· mat file. The value of input "τ" is updated in each iteration

4.11. Calculate Ionosphere delay

This is done by the Ionosphere_delay function. There are 6 inputs to this function; user's latitude (lat), longitude (lon) in degrees and satellite azimuth (A) and elevation (E) in radians, alpha, beta values from iono matrix. The output of this function is I_d in seconds. Klobuchar's Algorithm [4] is utilized to calculate delay.

4.12. Calculate Troposphere delay

The Troposphere_delay function calculates it. The input to this function is the elevation (E) in radians of the satellite from user position. The output of this function is the δ RTrop in meters. the equation used to calculate is:

$$\delta RTrop = 2.312/sin(\sqrt{E \cdot E + 1.904e-3}) + 0.084/sin(\sqrt{E \cdot E + 0.6854e-3}) \quad (7)$$

4.13. Update value of τ for next iteration

$$\tau_{new} = \tau_{old} + \delta\rho/c \quad (8)$$

For each iteration a new value of τ is calculated which is passed to all the depending functions.

5. Results

1) The iteration runs for 5 times to reduce the value of δx and δb to the required threshold.
2) The final coordinates of the receiver in ECEF system are :

$$x = 3.8952 * 1.0e+06$$
$$y = 0.3147 * 1.0e+06$$
$$z = 5.0241 * 1.0e+06$$

and in geodetic system i.e [lat,lon,h] are:

$$lat = 52.3093\,°N$$
$$lon = 4.6197\,°E$$
$$h = 197.7866\,m$$

3) These coordinates are plotted on Google maps as shown below by a red point:

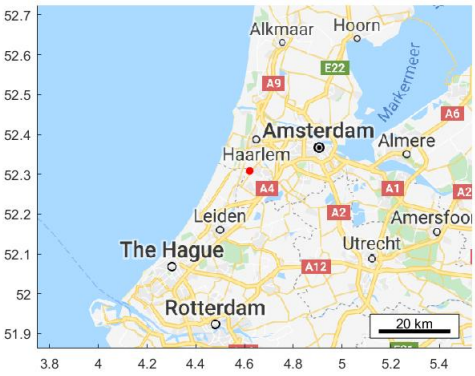


Figure 6. Final location on map

6. Error Analysis

1) Covariance matrix is calculated as follows:

$$G = (H^T \cdot H)^{-1} \tag{9}$$

$$\text{G} = \begin{Bmatrix} 10.3175 & 2.1986 & 4.7947 & 8.1054 \\ 2.1986 & 0.9658 & 1.1574 & 1.8529 \\ 4.7947 & 1.1574 & 3.7528 & 4.3210 \\ 8.1054 & 1.8529 & 4.3210 & 6.7478 \end{Bmatrix}$$

2) The values of DOP are also acceptable as calculated using formulas below:

$$HDOP = \sqrt{G(1,1) + G(2,2)} = 3.3591$$
$$VDOP = \sqrt{G(3,3)} = 1.9372$$
$$TDOP = \sqrt{G(4,4)} = 2.5977$$
$$PDOP = \sqrt{G(1,1) + G(2,2) + G(3,3)} = 3.8776$$
$$GDOP = \sqrt{G(1,1) + G(2,2) + G(3,3) + G(4,4)} = 4.6673$$

3) 1-sigma (68.3%)Error ellipses are plotted using the error_ellipse [5] function. Plots are given below:

7. Conclusion

The ephemeris data received from satellites was utilized to calculate the satellite position, satellite clock bias error, ionosphere delay, troposphere delay and then using pseudorange equation and least square solution user coordinates and user clock bias was calculated. GPS user location was ploted on Google maps using Google API. Error analysis was successfully carried out using dilution of precision and error ellipses. The values of DOP were also acceptable. The MATLAB code has been added in the appendix.

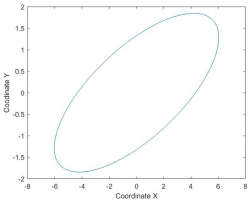


Figure 7. X-Y coordinate variance

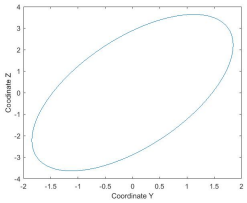


Figure 8. Y-Z coordinate variance

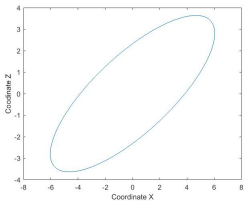


Figure 9. X-Z coordinate variance

References

[1] https://www.sxbluegps.com/technology/gps-the-error-budget/

[2] https://www.unavco.org/education/resources/tutorials-and-handouts/tutorials/elevation-and-geoid.html

[3] https://www.navcen.uscg.gov/pdf/IS-GPS-200D.pdf

[4] https://gssc.esa.int/navipedia/index.php/Klobuchar_Ionospheric_Model

[5] https://www.mathworks.com/matlabcentral/fileexchange/4705-error_ellipse

APPENDIX

Contents

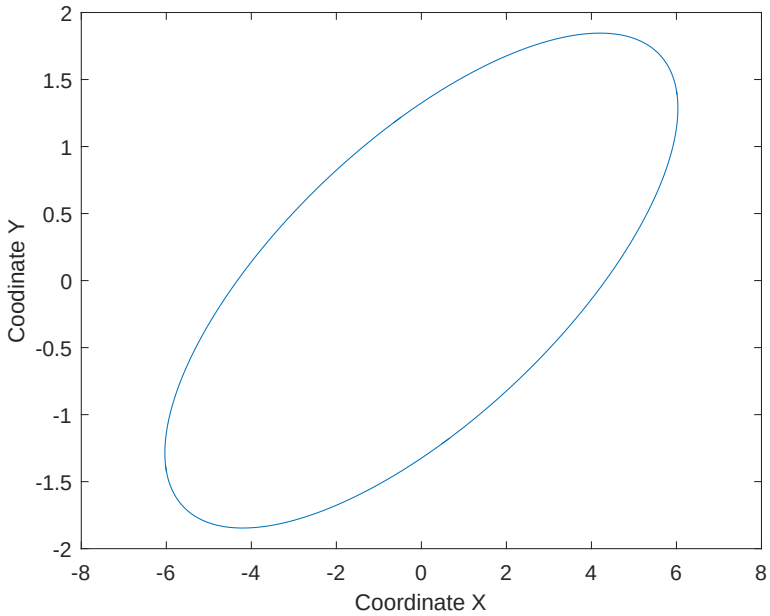
1 User_position_by_LeastSq Main function

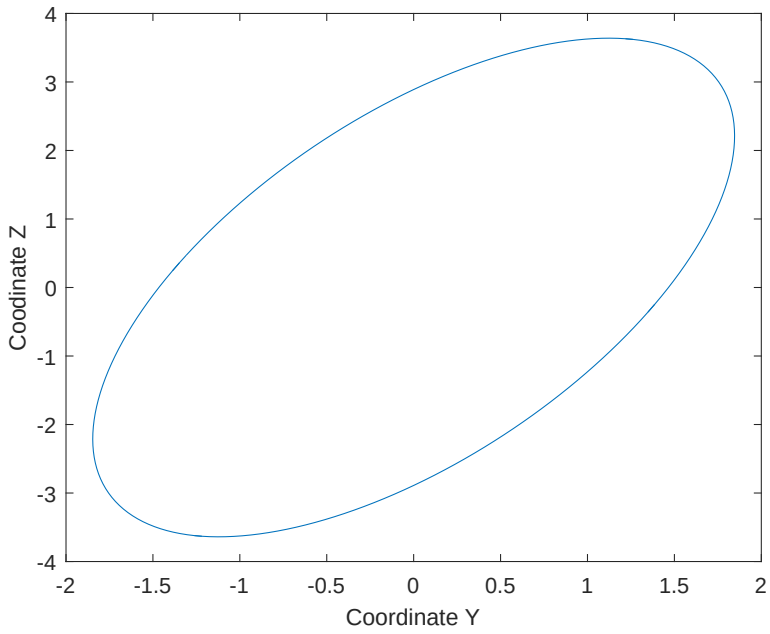
```
clc;
clear all;
close all;
load project_data;
sqrtA = eph(7,:);
toe = eph(8,:);
dn = eph(2,:)*pi;
m0 = eph(3,:)*pi;
e = eph(5,:);
w = eph(14,:)*pi;
Cuc = eph(4,:);
Cus = eph(6,:);
Crs = eph(1,:);
Cis = eph(11,:);
Crc = eph(13,:);
Cic = eph(9,:);
i0 = eph(12,:)*pi;
idot = eph(16,:)*pi;
omg0 = eph(10,:)*pi;
odot = eph(15,:)*pi;
Tgd = eph(17,:);
af0 = eph(21,:);
af1 = eph(20,:);
af2 = eph(19,:);
```

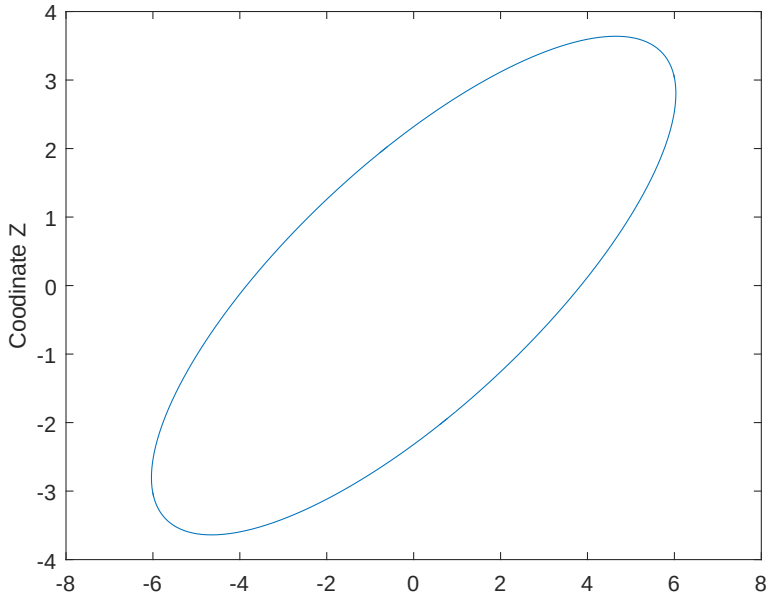
```
toc = eph(18,:);

trcv = iono(1,1); % tow

alpha = iono(2:5);
beta = iono(6:9);

mu =  3.986005e14;
omega_earth = 7.2921151467e-5; %(rad/sec)
F = -4.442807633e-10;
c = 299792458; % speed of light
xu = [0;0;0];%initial user position
bu = 0; % initial user clock bias
b0 =0;
pr_ = pr;
itr_num =0;
dx =100;
db=100;
tau = 0.075*ones(1,6); % Initial value of tau
I_d = zeros(1,6);
dRTrop = zeros(1,6);
dTclk = zeros(1,6);
itr=0;


while (norm(dx)>1e-1 && norm(db) > 1e-3)   % position of each satellite
xs = Satpos(tau);

% Find delta pseudorange
for i = 1 : 6
satellite_position_vector_from_user(:,i) = xs(:,i)-xu;
end
e = satellite_position_vector_from_user'; % for 6 satellites

for i = 1 : 6
delta_p(i) = pr_(i) - (norm(e(i,:)) + (b0 - c*dTclk(i)) + c*I_d(i) + dRTrop(i));
end
%Find H matrix
for i = 1 : 6
for j= 1: 3
satellite_position_unit_vectors(i,j) = e(i,j)/norm(e(i,:));
end
end

H = [-satellite_position_unit_vectors ones(6,1)];



%%Least square solution
```

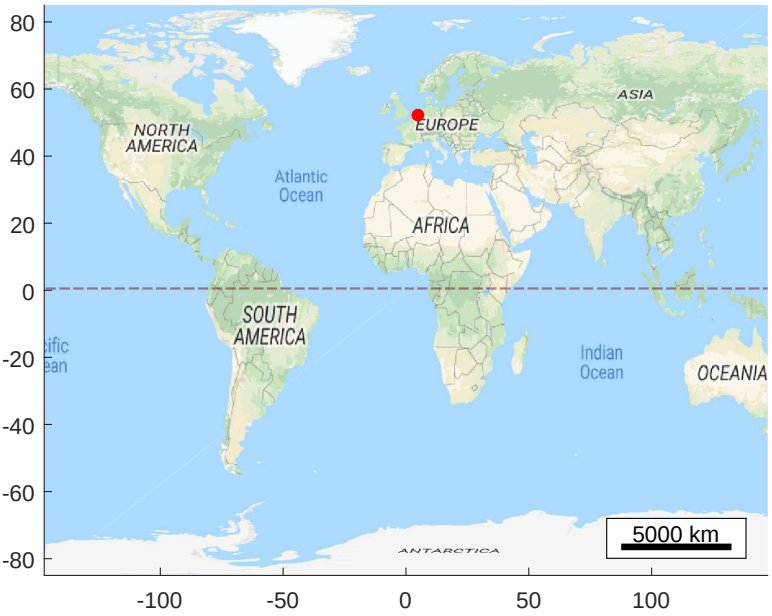
```
delta_r = inv(H'*H)*H'*delta_p';
dx = delta_r(1:3);
db = delta_r(4);

%%Update user position with the delta_x and user clock bias
xu = xu + dx;
b0 = b0 + db;

% User latitude,longitude, height i.e lat, lon, h
% Satellite Azimuth and Elevation from user
[A,El,lat,lon] = Satellite_Azimuth_Elevation(xs,xu);

%Satellite clock bias error for 6 satellites
dTclk = Satellite_Clockbias_Error(tau);

% Ionosphere and Troposphere delays
% I_d is in seconds
% dRtrop is in meters
for i =1:6
I_d(i) = Ionosphere_delay(lat, lon, A(i), El(i), alpha, beta);
dRTrop(i) = Troposphere_delay(El(i));
end

% Pseudorange after compensating for satellite clock bias error
% Update tau for each satellite
for i = 1 : 6
tau(i) = tau(i)+ delta_p(i)/c;
end
itr=itr+1;

end
G = inv(H'*H);
HDOP = sqrt(G(1,1) + G(2,2)); % Horizontal Dilution of precision
VDOP = sqrt(G(3,3)); % Vertical Dilution of precision
TDOP = sqrt(G(4,4)); % Time Dilution of precision
PDOP = sqrt (G(1,1) + G(2,2) + G(3,3));
GDOP = sqrt(G(1,1)+G(2,2)+G(3,3)+G(4,4)); % Geometrical Dilution of precision

% 1-sigma error ellipses
error_ellipse('C', G(1:3,1:3),'conf',0.683);

xu ;  % User poistion
dx ; % Deviation in position
itr; % Number of iterations
b0 ;  % User Clock bias
wgs84 = wgs84Ellipsoid('meters');
[lat,lon,h] = ecef2geodetic(wgs84,xu(1),xu(2),xu(3)); % User latitude and longitude

figure
```

```
plot_google_map('APIKey','AIzaSyA8i39YHp23B_ayfm9Yg8bunfAqlJQRDRE');
plot(lon, lat, '.r', 'MarkerSize', 20);
plot_google_map('MapScale', 1);
```









2 Satpos function

```
% Get data from eph matrix
function [xs] = Satpos(tau)

load project_data;
sqrtA = eph(7,:);
toe = eph(8,:);
dn = eph(2,:)*pi;
m0 = eph(3,:)*pi;
e = eph(5,:);
w = eph(14,:)*pi;
Cuc = eph(4,:);
Cus = eph(6,:);
Crs = eph(1,:);
```

```
Cis = eph(11,:);
Crc = eph(13,:);
Cic = eph(9,:);
i0 = eph(12,:)*pi;
idot = eph(16,:)*pi;
omg0 = eph(10,:)*pi;
odot = eph(15,:)*pi;
Tgd = eph(17,:);
af0 = eph(21,:);
af1 = eph(20,:);
af2 = eph(19,:);
toc = eph(18,:);

trcv = iono(1,1); % tow



% Find satellite coordinates in ECEF frame

%variables for satellite position
x_s = zeros(1,6);
y_s = zeros(1,6);
z_s = zeros(1,6);
%variables for satellite position after rotaion in time tau.
x_r = zeros(1,6);
y_r = zeros(1,6);
z_r = zeros(1,6);
xs = zeros(3,6);

Ek_old = 0;
mu =  3.986005e14;
omega_earth = 7.2921151467e-5; %(rad/sec)
F = -4.442807633e-10;

% Find coordinates for 6 satellites
for i=1:6

t= trcv - tau(i); % Transmission time

A = sqrtA(1,i)^2;
n0 = sqrt(mu/(A^3)); % computed mean motion
tk = t - toe(1,i);

% account for beginning of end of week crossover
if (tk > 302400)
tk = tk-604800;
end
if (tk < -302400)
tk = tk+604800;
```

```
end
% apply mean motion correction
n = n0 + dn(1,i);

% Mean anomaly
Mk = m0(1,i) + n*tk;

% solve for eccentric anomaly
Ek = Mk; %initial value of Ek = Mk
while abs(Ek-Ek_old)< 1e-08
Ek_old = Ek;
Ek = Mk + e(1,i)*sin(Ek_old);
end

% True anomaly:
nu = atan2((sqrt(1-(e(1,i)^2))*sin(Ek))/(1-(e(1,i)*cos(Ek))),
(cos(Ek)-e(1,i))/(1-(e(1,i)*cos(Ek))));
Ek = acos((e(1,i)  + cos(nu))/(1+e(1,i)*cos(nu)));

Phi = nu + w(1,i);
du = 0;
dr = 0;
di = 0;

% compute harmonic corrections
du = Cus(1,i)*sin(2*Phi) + Cuc(1,i)*cos(2*Phi);
dr = Crs(1,i)*sin(2*Phi) + Crc(1,i)*cos(2*Phi);
di = Cis(1,i)*sin(2*Phi) + Cic(1,i)*cos(2*Phi);

%Corrected argument of latitude
uk = Phi + du;

% Corrected radius
rk = A*(1-e(1,i)*cos(Ek)) + dr;

% inclination angle at reference time
ik = i0(1,i) + idot(1,i)*tk + di;

% Satellite position in orbital plane
x_p = rk*cos(uk);
y_p = rk*sin(uk);

%Corrected longitute of ascending node
omega = omg0(1,i) + (odot(1,i) - omega_earth)*tk - omega_earth*toe(1,i);

x_s(1,i) = x_p*cos(omega) - y_p*cos(ik)*sin(omega);
y_s(1,i) = x_p*sin(omega) + y_p*cos(ik)*cos(omega);
z_s(1,i) = y_p*sin(ik);
```

```
end
% Rotate the satellite position with ECEF frame in tau time. theta will be
% omega_earth*tau(i)


for i =1:6
theta = omega_earth*tau(i);
xs(:,i) = [cos(theta) sin(theta) 0; -sin(theta) cos(theta) 0; 0 0 1]
*[x_s(1,i);y_s(1,i) ; z_s(1,i)];
end
end
```

3 Satellite_Azimuth_Elevation function

```
%Input : xs = satellites position in ecef coordinates
%        xu = user position in ecef coordinates
%  Output : A = satellite azimuth from user
%           El= satellite elevation from user

function[A,El,lat,lon,h] = Satellite_Azimuth_Elevation(xs,xu)

%Convert ECEF coordinates to Geodetic coordinates (latitude,longitude) in
%degrees & meters
wgs84 = wgs84Ellipsoid('meters');
[lat,lon,h] = ecef2geodetic(wgs84,xu(1),xu(2),xu(3));

for i = 1:6
% Find enu coordinates of position vector from user to satellite
[xEast(i),yNorth(i),zUp(i)] = ecef2enu(xs(1,i)-xu(1),xs(2,i)-xu(2),
xs(3,i)-xu(3),lat,lon,h,wgs84);
A(i) = atan2(xEast(i),yNorth(i));
mag = sqrt(xEast(i)^2+yNorth(i)^2+zUp(i)^2); %in radians
El(i) = asin(zUp(i)/mag); % in radians
end
end
```

4 Satellite_Clockbias_Error function

```
function [dTclk] = Satellite_Clockbias_Error(tau)

load project_data;
sqrtA = eph(7,:);
toe = eph(8,:);
dn = eph(2,:)*pi;
m0 = eph(3,:)*pi;
e = eph(5,:);
w = eph(14,:)*pi;
```

```
Cuc = eph(4,:);
Cus = eph(6,:);
Crs = eph(1,:);
Cis = eph(11,:);
Crc = eph(13,:);
Cic = eph(9,:);
i0 = eph(12,:)*pi;
idot = eph(16,:)*pi;
omg0 = eph(10,:)*pi;
odot = eph(15,:)*pi;
Tgd = eph(17,:);
af0 = eph(21,:);
af1 = eph(20,:);
af2 = eph(19,:);
toc = eph(18,:);

trcv = iono(1,1)-tau; % tow - tau
mu =  3.986005e14;
F = -4.442807633e-10;

% Satellite clock bias error matrix
dTclk = zeros(1,6);
Ek_old =0;
for i= 1:6

A = sqrtA(1,i)^2;
n0 = sqrt(mu/A^3); % computed mean motion

T = trcv(i)-toc(1,i);
% account for beginning of end of week crossover
if (T > 302400)
T = T-604800;
end
if (T < -302400)
T = T+604800;
end
% apply mean motion correction
n = n0 + dn(1,i);

% Mean anomaly
Mk = m0(1,i) + n*T;

% solve for eccentric anomaly
Ek = Mk; %initial value of Ek = Mk
while abs(Ek-Ek_old)< 1e-08
Ek_old = Ek;
Ek = Mk + e(1,i)*sin(Ek);
end
```

```
dTclk(1,i) = af0(1,i) + af1(1,i)*(T) + af2(1,i)*(T)^2 +
F*e(1,i)*sqrtA(1,i)*sin(Ek)- Tgd(1,i);

end
end
```

5 Ionosphere_delay function

```
% Klobuchar Algorithm
%Input = user approximate geodetic latitude ,
%        longitude , elevation angle and azimuth  of the observed satellite
%        and the coefficients broadcasted in the GPS satellite navigation
%        message
%Output = Ionosphere_timedelay in seconds

function[I_d] = Ionosphere_delay(lat, lon, A, E, alpha, beta)

E = E/pi; % converted to semicircles
lat = lat/180; %converted to semicircles
lon = lon/180; %converted to semicircles

% Calculate the earth-centred angle (elevation  in semicircles)
psi = 0.0137/(E + 0.11) - 0.022;

% Compute the latitude of the Ionospheric Pierce Point
Phi_i = lat + psi*cos(A); % in semicircles
if Phi_i >= 0.416
Phi_i = 0.416;
elseif Phi_i < -0.416
Phi_i = -0.416;
end

%Compute the longitude of the IPP
Lambda_i = lon + psi*sin(A)/cos(Phi_i*pi); % in semicircles

%Find the geomagnetic latitude of the IPP
Phi_m = Phi_i + 0.064*cos((Lambda_i - 1.617)*pi);

%Find the local time at the IPP
t_gps = mod(247080,86400);
t = 43200*Lambda_i + t_gps;
if t < 0
t = t + 86400;
elseif t >= 86400
t= t - 86400;
end

%Compute the amplitude of ionospheric delay
```

```
A_i = alpha(1) + alpha(2)*Phi_m + alpha(3)*(Phi_m^2) + alpha(4)*(Phi_m^3);

if A_i <0
A_i = 0;
end

%Compute the period of ionospheric delay

Per = beta(1) + beta(2)*Phi_m + beta(3)*(Phi_m^2)+ beta(4)*(Phi_m^3);

if Per < 72000
Per = 72000;
end

%Compute the phase of ionospheric delay
X_i = 2*pi*(t-50400)/Per;

% Compute the slant factor (elevation  in semicircles)
F = 1.0 + 16.0*(0.53 - E)^3;

%Compute the ionospheric time delay
if abs(X_i)<= 1.57
I_d = (5e-9 + A_i * (1 - (X_i^2/2) + (X_i^4/24)))*F;
else
I_d = 5e-9 * F;
end

end
```

6 Error_ellipse function

```
function h=error_ellipse(varargin)
% ERROR_ELLIPSE - plot an error ellipse, or ellipsoid, defining confidence
% region
%    ERROR_ELLIPSE(C22) - Given a 2x2 covariance matrix, plot the
%    associated error ellipse, at the origin. It returns a graphics handle
%    of the ellipse that was drawn.
%
%    ERROR_ELLIPSE(C33) - Given a 3x3 covariance matrix, plot the
%    associated error ellipsoid, at the origin, as well as its projections
%    onto the three axes. Returns a vector of 4 graphics handles, for the
%    three ellipses (in the X-Y, Y-Z, and Z-X planes, respectively) and for
%    the ellipsoid.
%
%    ERROR_ELLIPSE(C,MU) - Plot the ellipse, or ellipsoid, centered at MU,
%    a vector whose length should match that of C (which is 2x2 or 3x3).
%
%    ERROR_ELLIPSE(...,'Property1',Value1,'Name2',Value2,...) sets the
```

```
%    values of specified properties, including:
%      'C' - Alternate method of specifying the covariance matrix 'mu' -
%      Alternate method of specifying the ellipse (-oid) center 'conf' - A
%      value betwen 0 and 1 specifying the confidence interval.
%        the default is 0.5 which is the 50% error ellipse.
%      'scale' - Allow the plot the be scaled to difference units. 'style'
%      - A plotting style used to format ellipses. 'clip' - specifies a
%      clipping radius. Portions of the ellipse, -oid,
%        outside the radius will not be shown.
%
%    NOTES: C must be positive definite for this function to work properly.

default_properties = struct(...
'C', [], ... % The covaraince matrix (required)
'mu', [], ... % Center of ellipse (optional)
'conf', 0.5, ... % Percent confidence/100
'scale', 1, ... % Scale factor, e.g. 1e-3 to plot m as km
'style', '', ...  % Plot style
'clip', inf); % Clipping radius

if length(varargin) >= 1 & isnumeric(varargin{1})
default_properties.C = varargin{1};
varargin(1) = [];
end

if length(varargin) >= 1 & isnumeric(varargin{1})
default_properties.mu = varargin{1};
varargin(1) = [];
end

if length(varargin) >= 1 & isnumeric(varargin{1})
default_properties.conf = varargin{1};
varargin(1) = [];
end

if length(varargin) >= 1 & isnumeric(varargin{1})
default_properties.scale = varargin{1};
varargin(1) = [];
end

if length(varargin) >= 1 & ~ischar(varargin{1})
error('Invalid parameter/value pair arguments.')
end

prop = getopt(default_properties, varargin{:});
C = prop.C;

if isempty(prop.mu)
mu = zeros(length(C),1);
```

```
else
mu = prop.mu;
end

conf = prop.conf;
scale = prop.scale;
style = prop.style;

if conf <= 0 | conf >= 1
error('conf parameter must be in range 0 to 1, exclusive')
end

[r,c] = size(C);
if r ~= c | (r ~= 2 & r ~= 3)
error(['Don''t know what to do with ',num2str(r),'x',num2str(c),
' matrix'])
end

x0=mu(1);
y0=mu(2);

% Compute quantile for the desired percentile
k = sqrt(qchisq(conf,r)); % r is the number of dimensions (degrees of freedom)

hold_state = get(gca,'nextplot');

if r==3 & c==3
z0=mu(3);

% Make the matrix has positive eigenvalues - else it's not a valid
% covariance matrix!
if any(eig(C) <=0)
error('The covariance matrix must be positive definite
(it has non-positive eigenvalues)')
end

% C is 3x3; extract the 2x2 matricies, and plot the associated error
% ellipses. They are drawn in space, around the ellipsoid; it may be
% preferable to draw them on the axes.
Cxy = C(1:2,1:2);
Cyz = C(2:3,2:3);
Czx = C([3 1],[3 1]);


[x,y,z] = getpoints(Cxy,prop.clip);
h1=plot(x0+k*x,y0+k*y,prop.style);
st = [x0+k*x, y0+k*y];

xlabel('Coordinate X');
```

```
ylabel('Coodinate Y');
figure;
[y,z,x] = getpoints(Cyz,prop.clip);
h2=plot(y0+k*y,z0+k*z,prop.style);

xlabel('Coordinate Y');
ylabel('Coodinate Z');
figure;
[z,x,y] = getpoints(Czx,prop.clip);
h3=plot(x0+k*x,z0+k*z,prop.style);
xlabel('Coordinate X');
ylabel('Coodinate Z');

[eigvec,eigval] = eig(C);

%   [X,Y,Z] = ellipsoid(0,0,0,1,1,1); XYZ =
%   [X(:),Y(:),Z(:)]*sqrt(eigval)*eigvec';
%
%   X(:) = scale*(k*XYZ(:,1)+x0); Y(:) = scale*(k*XYZ(:,2)+y0); Z(:) =
%   scale*(k*XYZ(:,3)+z0); h4=surf(X,Y,Z); colormap gray alpha(0.3)
%   camlight if nargout
%     h=[h1 h2 h3 h4];
%   end
% elseif r==2 & c==2
%   % Make the matrix has positive eigenvalues - else it's not a valid
%   covariance matrix! if any(eig(C) <=0)
%     error('The covariance matrix must be positive definite (it has
%     non-positive eigenvalues)')
%   end
%
%   [x,y,z] = getpoints(C,prop.clip);
%   h1=plot(scale*(x0+k*x),scale*(y0+k*y),prop.style);
%   set(h1,'zdata',z+1) if nargout
%     h=h1;
%   end
% else
%   error('C (covaraince matrix) must be specified as a 2x2 or 3x3
%   matrix)')
end
% %axis equal
%
% set(gca,'nextplot',hold_state);

%---------------------------------------------------------------
% getpoints - Generate x and y points that define an ellipse, given a 2x2
%   covariance matrix, C. z, if requested, is all zeros with same shape as
%   x and y.
function [x,y,z,r] = getpoints(C,clipping_radius)
```

```
n=100; % Number of points around ellipse
p=0:pi/n:2*pi; % angles around a circle

[eigvec,eigval] = eig(C); % Compute eigen-stuff
xy = [cos(p'),sin(p')] * sqrt(eigval) * eigvec'; % Transformation
x = xy(:,1);
y = xy(:,2);
z = zeros(size(x));

% Clip data to a bounding radius
if nargin >= 2
r = sqrt(sum(xy.^2,2)); % Euclidian distance (distance from center)
x(r > clipping_radius) = nan;
y(r > clipping_radius) = nan;
z(r > clipping_radius) = nan;
end

%---------------------------------------------------------------
function x=qchisq(P,n)
% QCHISQ(P,N) - quantile of the chi-square distribution.
if nargin<2
n=1;
end

s0 = P==0;
s1 = P==1;
s = P>0 & P<1;
x = 0.5*ones(size(P));
x(s0) = -inf;
x(s1) = inf;
x(~(s0|s1|s))=nan;

for ii=1:14
dx = -(pchisq(x(s),n)-P(s))./dchisq(x(s),n);
x(s) = x(s)+dx;
if all(abs(dx) < 1e-6)
break;
end
end

%---------------------------------------------------------------
function F=pchisq(x,n)
% PCHISQ(X,N) - Probability function of the chi-square distribution.
if nargin<2
n=1;
end
F=zeros(size(x));

if rem(n,2) == 0
```

```
s = x>0;
k = 0;
for jj = 0:n/2-1;
k = k + (x(s)/2).^jj/factorial(jj);
end
F(s) = 1-exp(-x(s)/2).*k;
else
for ii=1:numel(x)
if x(ii) > 0
F(ii) = quadl(@dchisq,0,x(ii),1e-6,0,n);
else
F(ii) = 0;
end
end
end

%-------------------------------------------------------------
function f=dchisq(x,n)
% DCHISQ(X,N) - Density function of the chi-square distribution.
if nargin<2
n=1;
end
f=zeros(size(x));
s = x>=0;
f(s) = x(s).^(n/2-1).*exp(-x(s)/2)./(2^(n/2)*gamma(n/2));

%-------------------------------------------------------------
function properties = getopt(properties,varargin)
%GETOPT - Process paired optional arguments as
%'prop1',val1,'prop2',val2,...
%
%   getopt(properties,varargin) returns a modified properties structure,
%   given an initial properties structure, and a list of paired arguments.
%   Each argumnet pair should be of the form property_name,val where
%   property_name is the name of one of the field in properties, and val is
%   the value to be assigned to that structure field.
%
%   No validation of the values is performed.
%
% EXAMPLE:
%   properties =
%   struct('zoom',1.0,'aspect',1.0,'gamma',1.0,'file',[],'bg',[]);
%   properties = getopt(properties,'aspect',0.76,'file','mydata.dat')
% would return:
%   properties =
%         zoom: 1
%       aspect: 0.7600
%        gamma: 1
%         file: 'mydata.dat'
```

```
%           bg: []
%
% Typical usage in a function:
%   properties = getopt(properties,varargin{:})

% Process the properties (optional input arguments)
prop_names = fieldnames(properties);
TargetField = [];
for ii=1:length(varargin)
arg = varargin{ii};
if isempty(TargetField)
if ~ischar(arg)
error('Propery names must be character strings');
end
f = find(strcmp(prop_names, arg));
if length(f) == 0
error('%s ',['invalid property ''',arg,'''; must be one of:']
,prop_names{:});
end
TargetField = arg;
else
% properties.(TargetField) = arg; % Ver 6.5 and later only
properties = setfield(properties, TargetField, arg); % Ver 6.1 friendly
TargetField = '';
end
end
if ~isempty(TargetField)
error('Property names and values must be specified in pairs.');
end
```