

Computation of Receiver Position from Code Pseudoranges

Kevin Mead

April 19, 2014

1 Introduction

The task is to compute the coordinates of a GPS receiver at given time using P1-code pseudoranges. My specific observation epoch is: 2004-02-02, 1 h 14 min 00 sec. The calculations were done with observation files and navigation files in RINEX format. The computation of this position is difficult because the satellite and receiver clocks are not synchronised, the satellite and receiver are both moving with varying velocities, and propagation times. There are many more problems that need to be accounted for like ionospheric and tropospheric corrections, but we have neglected them in this lab because we want to gain a general understanding before we take on the harder tasks

2 Methodology

Matlab was used to solve the problem and used the GPS single point positioning algorithm.[1] The steps from this algorithm are documented in the Matlab code, Section 5. The pseudoranges and navigation messages are used to calculate a more precise position of the receiver. The shift in position is a measure of the time for the signal to travel from satellite to receiver. This time traveled by the signal multiplied by the speed of light is the pseudo range. All the satellites were used in this calculation in order to get the best possible approximation of the receiver coordinates. Since we will be using least-squares method to solve for the parameters, the more independent equations we have, the better the approximation will be. First, the signal propagation time was calculated using,

$$\Delta t_A^s = P_A^s(\tilde{t}_A)/c$$

which was then used to calculate the nominal transmission time,

$$\tilde{t}^s = \tilde{t}_A - P_A^s(\tilde{t}_A)/c$$

where \tilde{t}_A is the nominal time of signal reception by the receiver. For this lab report, $\tilde{t}_A = 90840$ seconds, or 2 days, 1 hour, and 14 minutes. The weeks in the year was omitted because the difference between epoch times was less than a day and therefore would have cancelled out. Next the satellite clock correction, δt_{L1}^s , was computed using equations,

$$\begin{aligned}\delta t_{L1}^s &= \Delta t_{SV} - T_{GD} \\ \Delta t_{SV} &= a_{f0} + a_{f1}(\tilde{t}^s - t_{oc}) + a_{f2}(\tilde{t}^s - t_{oc})^2 + \Delta t_r\end{aligned}$$

but the relativistic correction, $\Delta t_r = Fe\sqrt{A}\sin(E_k)$ was neglected until the eccentric anomaly and satellite coordinates were computed. Next, the system time of signal transmission, t^s was

calculated using the satellite clock correction. The eccentric anomaly was calculated using Kepler's equation,

$$E_k = M_k + e \sin(E_k)$$

and the solution was iterated until convergence. The relativistic correction is now calculated and t^s is corrected in order to compute the satellite coordinates at that time given by the equations,

$$\begin{aligned} X^s &= x'_k \cos \Omega_k - y' \cos i \sin \Omega_k \\ Y^s &= x'_k \sin \Omega_k + y' \cos i \cos \Omega_k \\ Z^s &= y' \sin i_k \end{aligned}$$

Now that the satellite coordinates have been calculated, the satellite clock correction is corrected again to include the relativistic correction, Δt_r . Now the approximate distance of the satellite is calculated using equation,

3 Results

The results for the receiver position are given in Table 1. The results agree completely with the reference material given because both calculations used all of the 11 satellites available.

	Epoch 2004 2 2 1 14 0.0		
	Final	Approximate	σ
X	3104225.071 m	3104219.453	1.330 m
Y	998384.754 m	998383.982	1.101 m
Z	5463300.077 m	5463290.508	2.566 m
T	-0.0005198825 s		4.7543792389e - 09 s

Table 1: Coordinates of the receiver after several iterations of steps 11-16 until the convergence condition was met. They agree completely with SPPResults and are very close to the approximate positions in the observation file.

4 Analysis and Discussion

- Are the results reasonable? Compare the results with your expectations.

Yes the results are reasonable. The final position of the receiver is very close to the approximate position and this is shown in Table 1. We also get very consistent results if we compute the same calculations for different epochs.

- Can we draw any conclusion/implications from the results?

We can conclude that the approximate coordinates are very close to the true receiver position, even though we did not factor in ionospheric and tropospheric effects.

- Are results reliable and accurate?

I believe the results are reliable and accurate because the σ value is very small for all x,y,z and all 11 satellites were taken into account. But the least squares method can give poor results if there are any outliers or the errors aren't Gaussian distributed.[2] Looking at

other epochs in the SPP Results, we can see that they also give approximately the same position for the receiver.

- Would it have been more appropriate to use another method? Does the method need to be further developed?

I think it is only necessary to further develop this method if you needed a higher degree of accuracy. This method could be further developed to take into account the ionospheric and tropospheric effects and factor in all the given epochs for the satellites. I think it could be more appropriate to use the difference method is best used when you have more satellites and tropospheric and ionospheric effects are cancelled out in the equations. This means our final solution should be more accurate using the difference method.

References

- [1] M. Horemuž, “Gps single point positioning algorithm,” Royal Institute of Technology, Tech. Rep., 2014.
- [2] A. L. Garcia, *Numerical Methods for Physics (2Nd Edition)*, 2nd ed., A. Reeves and P. F. Corey, Eds. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1999.

5 Main Code

Contents

- Computation of receiver’s position
- Import observer file
- Import navigation file
- Match satellite numbers with available satellites
- Main loop steps 1-14
- 17. Repeat steps 11 -16 until the solution has converged.
- 17. Convergence condition
- Lab 2

Computation of receiver’s position

```
clear all;
c = 299792458; % speed of light (m/s)
mu = 3.986005e14; % universal gravitational parameter (m/s)^3
omega_e_dot = 7.2921151467e-5; % earth rotation rate (rad/s)
F = -4.442807633e-10; % s/m^1/2
time = [2,1,14,0]; % days, hours, minutes, seconds
junk = num2cell(time);
[nday,nhours,nminutes,nseconds] = junk{:};
clear junk;
```

Import observer file

```
lov033b = importObserverFileAsString('0lov033b.04o', 1, 5629);
% Import P1 numbers and satellite numbers
% match your time with observer time
```

```
[rowInObs,nOfRows] = findTimeInObsFunction( lov033b,time );
% Import P1 numbers from the matched time above
p1_numbers = importObsP1numbers('0lov033b.04o', rowInObs+1,rowInObs+nOfRows*2);
% Import satellite numbers from matched time
satelliteNumbers = importObsSatelliteNumbers('0lov033b.04o', rowInObs,rowInObs);
[XA0,YA0,ZA0] = sampleFunction(lov033b); % Record Approximate Position
```

Import navigation file

```
navfiles = importNavigationFiles('0lov033b.04n');
```

Match satellite numbers with available satellites

```
satNumMatch = navfiles(1:8:96,1); % Order of satellite numbers import
sortedSatelliteNumbers = sortrows([satelliteNumbers',p1_numbers],1);
```

Main loop steps 1-14

Calculates variables needed for correction iterations

```
count = 1;
for i = 1:length(satNumMatch)
```

Steps 1-14 done inside

```
    if cell2mat(satNumMatch(i))==sortedSatelliteNumbers(count,1)
        [ Lmat(count,:), ...
          Amat(count,:),...
          rho(count,:),...
          Xs(count,:),Ys(count,:),Zs(count,:),...
          P1(count,:),...
          dtsL1_with_dtr(count,:),...
          tAtoS(count,:)]...
        = satLandP( i,sortedSatelliteNumbers(count,2),navfiles,...
XA0,YA0,ZA0,nday,nhours,nminutes,nseconds);
        count = count + 1;
    else
        fprintf('No data for Satellite%d\n',cell2mat(satNumMatch(i)))
    end
```

```
end
```

17. Repeat steps 11 -16 until the solution has converged.

The solution has converged if the condition is fulfilled

```
for i = 1:10
```

```

changeX = (Amat'*Amat)\(Amat'*Lmat); % eq. (21)
v(:,i) = -Amat*changeX + Lmat; % eq. (17)
newXYZ = [XA0,YA0,ZA0] + changeX(1:3)'; % eq. (22) est. coordinates
newxyzcell = num2cell(newXYZ);
[XA0,YA0,ZA0] = newxyzcell{:};
clear newxyzcell;
rho = sqrt(... % recompute rho
    (Xs - XA0 + omega_e_dot * YA0 * tAtoS).^2 + ... % x^2
    (Ys - YA0 - omega_e_dot * XA0 * tAtoS).^2 + ... % y^2
    (Zs - ZA0).^2 ... % z^2
);
Amat = [-(Xs - XA0)./rho,... % recompute matrix A
    -(Ys-YA0)./rho,...
    -(Zs-ZA0)./rho,...
    rho./rho];
Lmat = P1 - rho + c*dtsL1_with_dtr; % recompute matrix L

```

17. Convergence condition

```

if i>1 % check for convergence condition
    condition = abs(v(:,end)'+v(:,end)-v(:,end-1)'+v(:,end-1)));
% condition must be 1e-5 or lower
    if condition < 1e-4

        fprintf('Convergence condition met = %d\n',condition);

```

Lab 2

Find sigma and Q

```

    Q = inv((Amat'*Amat));
    sigma_0 = sqrt(v(:,end)'+v(:,end)/(length(Amat)-length(Q)));
    sigma_x = sigma_0*sqrt(Q(1,1));
    sigma_y = sigma_0*sqrt(Q(2,2));
    sigma_z = sigma_0*sqrt(Q(3,3));
    sigma_t = sigma_0*sqrt(Q(4,4))/c;
    fprintf('X = %7.3f, mX = %7.3f\n',XA0,sigma_x);
    fprintf('Y = %7.3f, mX = %7.3f\n',YA0,sigma_y);
    fprintf('Z = %7.3f, mX = %7.3f\n',ZA0,sigma_z);
    fprintf('T = %0.10f, mt = %0.10d\n',-changeX(4)'/c,sigma_t);
    return

end

end

end

```

6 Matlab Function: satLandP.m

Contents

- Constants
- Importing numbers from table 3 for a satellite
- 1. Compute signal propagation time by (13)
- 2. Compute signal transmission time by (14)
- 3. Compute satellite clock correction dtsL1
- 4. Compute ts using the correction from the step 3.
- 5. Compute eccentric anomaly (Table 2)
- 6. Compute dtr by (26) and ts by (15).
- 7. Compute satellite coordinates Xs, Ys, Zs, for time ts
- 8. Compute satellite clock correction dtsL1 by (24) - (27)
- 9. Compute tropospheric correction T_A_to_s (tA)
- 10. Compute ionospheric correction I_A_to_s (tA)
- 11. Compute approximate distance rho_A0_to_s (tA) by (11).
- 12. Repeat steps 1 - 11 for all measured satellites.
- 13. Compute elements of vector L (19).
- 14. Compute elements of matrix A (20); a_x_to_s , a_y_to_s , a_z_to_s by (12)

```
function [ Lmatrix, Amatrix, rho_f, Xs_f, Ys_f, Zs_f, P1_f, dtsL1_dtr_f, tAtoS_f] ...  
    = satLandP(  
satelliteNumberOrder, P1_f, navfiles, XA0, YA0, ZA0, ndaysf, nhoursf, nminutesf, nsec  
onds)
```

Constants

```
c = 299792458; % speed of light (m/s)  
mu = 3.986005e14; % universal gravitational parameter (m/s)^3  
omega_e_dot = 7.2921151467e-5; % earth rotation rate (rad/s)  
F = -4.442807633e-10; % s/m^1/2
```

Importing numbers from table 3 for a satellite

```
i = 1:8:112;  
sat = navfiles(i(satelliteNumberOrder):i(satelliteNumberOrder)+8,:);  
sat = transpose(cell2mat(sat));  
sat = num2cell(sat);  
% Imports numbers to all variables  
[~, af0, af1, af2, ...  
~, crs, change_n, m0, ...  
cuc, ec, cus, sqrtA, ...  
toe, cic, omega0, cis, ...  
i0, crc, w, omegadot, ...  
idot, ~, ~, ~, ...  
~, ~, tgd, ~, ...  
~] ...  
=sat{:};
```

1. Compute signal propagation time by (13)

```
tA_nom = seconds_in_week(ndaysf,nhoursf,nminutesf,nsecondsf); % 2 days, 1 hour,  
% 14 minutes My Time  
tAtoS_f = P1_f/c; % signal propagation time
```

2. Compute signal transmission time by (14)

```
tS_nom = tA_nom - P1_f/c;
```

3. Compute satellite clock correction dtsL1

by (24) and (25), neglect dtr

```
t_oc = toe; % I believe this is true, but not sure  
change_tsv_f = af0 + af1*(tS_nom-t_oc)+af2*(tS_nom-t_oc)^2; % (25)  
dtsL1 = change_tsv_f - tgd; % (24)
```

4. Compute ts using the correction from the step 3.

```
ts_f = tS_nom - dtsL1;
```

5. Compute eccentric anomaly (Table 2)

```
ek = mk + ec*sin(ek)  
  
A = sqrt(A^2);  
n0 = sqrt(mu/A^3); % Computed mean motion  
n = n0 + change_n;  
tk = ts_f - toe;  
tk = fixTk(tk); % if,then for table 2 of tk  
mk = m0 + n*tk;  
Ek = keplersEquation(mk,ec);
```

6. Compute dtr by (26) and ts by (15).

```
change_tr = F*ec*sqrt(A*sin(Ek)); % (26)  
ts_with_dtr = ts_f - change_tr;
```

7. Compute satellite coordinates Xs, Ys, Zs, for time ts

- Table 2 Calculate rk

```
vk = atan2((sqrt(1-ec^2)*sin(Ek)/(1-ec*cos(Ek))),...  
((cos(Ek)-ec)/(1-ec*cos(Ek))));  
Phik = vk + w;  
drk = crs*sin(2*Phik) + crc*cos(2*Phik);  
rk = A*(1-ec*cos(Ek)) + drk; % Corrected radius  
% Calculate uk  
duk = cus*sin(2*Phik) + cuc*cos(2*Phik);
```

```

uk = Phik + duk;
% Calculate ik
dik = cis*sin(2*Phik) + cic*cos(2*Phik);
ik = i0 + dik + idot*tk;
% Calculate omega's
omegak = omega0 + (omegadot-omega_e_dot)*tk - omega_e_dot*toe;
% Calculate xkp and ykp
xkp = rk*cos(uk);
ykp = rk*sin(uk);
% Calculate xk,yk,zk -> Xs, Ys, Zs for time ts
xk = xkp*cos(omegak) - ykp*cos(ik)*sin(omegak);
yk = xkp*sin(omegak) + ykp*cos(ik)*cos(omegak);
zk = ykp*sin(ik);
Xs_f = xk;
Ys_f = yk;
Zs_f = zk;

```

8. Compute satellite clock correction dtsL1 by (24) - (27)

```
dtsL1_dtr_f = change_tsv_f + change_tr - tgd; % (24)
```

9. Compute tropospheric correction T_A_to_s (tA)

10. Compute ionospheric correction I_A_to_s (tA)

11. Compute approximate distance rho_A0_to_s (tA) by (11).

```

rho_f = sqrt(...
    (Xs_f - XA0 + omega_e_dot * YA0 * tAtoS_f)^2 + ... % x^2
    (Ys_f - YA0 - omega_e_dot * XA0 * tAtoS_f)^2 + ... % y^2
    (Zs_f - ZA0)^2    ... % z^2
);
% dtA = 0;
% rho_A_to_s = P1 + c*dtsL1_with_dtr - c*dtA; % (8) dtA = 0

```

12. Repeat steps 1 - 11 for all measured satellites.

13. Compute elements of vector L (19).

```
Lmatrix = P1_f - rho_f + c*dtsL1_dtr_f;
```

14. Compute elements of matrix A (20); a_x_to_s , a_y_to_s , a_z_to_s by (12)

```
Amatrix = 1/rho_f*[-(Xs_f - XA0),-(Ys_f - YA0),-(Zs_f - ZA0),rho_f];
```

end

Functions

%% Function used to fix tk value

```
function tk = fixTk( tk )
```



```

if tk > 302400
    tk = tk - 604800;
elseif tk < -302400
    tk = tk + 604800;
else
    tk = tk;
end

%% Function used to calculate keplers equation
function Ek = keplersEquation( mk,ec )
E0(1) = mk;
n=7;
for i = 1:n
    E0(i+1) = mk + ec*sin(E0(i));
    eps = abs(E0(i)-E0(i+1));
end
Ek = E0(end);
end

%% Function that converts days,hours,min,seconds to total seconds

function total_seconds = seconds_in_week( days, hours, min, sec )
total_seconds = (days-1)*24*3600 + hours*3600 + min*60 + sec;
end

```