NTNU
Kunnskap for en bedre verden

TBA4565 - Module GPS

GPS Absolute (Point) positioning with code pseudorange & Accurate Relative Positioning with Carrier Phases

Tobias Andresen

29.11.2025

Table of Contents

1 Summary of Project 1 Results

Before presenting the results, note that all computations correspond to the observation epoch

$$T = 558000 \text{ s},$$

and all satellite coordinates are expressed in the Earth-Centered Earth-Fixed (ECEF) WGS84 system.

1.1 Satellite Coordinates at Transmission Time

Satellite coordinates are computed from the broadcast ephemerides and evaluated at the *transmission time*:

$$t_s = T - \frac{P}{c} + dt,$$

which gives the true satellite position at the moment the signal left the satellite.

SV	X (m)	Y (m)	Z (m)
G08	24658770.407	4652595.974	9213667.379
G10	-1629222.309	17219689.226	20239421.461
G21	15692829.631	1628012.373	21977344.510
G24	-15097557.727	4371547.933	21018541.140
G17	-6364757.833	-18393511.726	18452877.605
G03	23098433.073	-12669412.758	2685881.086
G14	5287122.116	-18484903.254	18275415.197

Table 1: Corrected satellite coordinates at transmission time.

1.2 Satellite Coordinates Without Corrections and Differences

The nine correction terms in the broadcast ephemerides (harmonic corrections to argument of latitude, radius, and inclination) account for perturbations from Earth's oblateness, luni-solar gravity, and solar radiation pressure. Removing these terms produces a purely Keplerian orbit approximation, which leads to significant coordinate differences.

Typical magnitudes range from ~ 200 to 700 m, demonstrating that satellite perturbation modelling is essential for accuracy.

SV	X (m)	Y (m)	Z (m)
G08	24658457.953	4653140.685	9213816.590
G10	-1629417.678	17219673.320	20239695.025
G21	15692874.173	1628216.760	21977577.030
G24	-15097730.806	4371614.604	21018617.712
G17	-6365127.807	-18393387.844	18453001.198
G03	23097867.054	-12669731.239	2686194.158
G14	5287372.096	-18484765.190	18275662.908

Table 2: Uncorrected satellite coordinates.

SV	ΔX (m)	ΔY (m)	ΔZ (m)	$\|\Delta\mathbf{X}\|$ (m)
G08	312.454	-544.711	-149.211	645.447
G10	195.369	15.905	-273.563	336.540
G21	-44.542	-204.387	-232.520	312.768
G24	173.079	-66.671	-76.572	200.660
G17	369.974	-123.882	-123.593	409.271
G03	566.020	318.481	-313.072	720.987
G14	-249.980	-138.063	-247.712	378.037

Table 3: Differences between corrected and uncorrected coordinates.

1.3 Approximate Receiver Coordinates

The approximate receiver geodetic coordinates used for linearization are:

$$(\varphi_0, \lambda_0, h_0) = (63.2^\circ,\ 10.2^\circ,\ 100\ \text{m}).$$

Transforming these to ECEF (WGS84):

$$\mathbf{x}_0 = \begin{bmatrix} 2837931.501 \\ 510624.472 \\ 5670153.346 \end{bmatrix}\ \text{m}.$$

1.4 Observation Equation and Linearization

In this step we develop the linearized observation equations used in the least–squares adjustment. The goal is to relate the measured pseudorange observations to corrections in the receiver coordinates and receiver clock bias. All expressions are formulated for each satellite j.

The undifferenced pseudorange observation equation is:

$$P^j = \rho^j + c\,dt^j - c\,dT + d^j_{\text{ion}} + d^j_{\text{trop}},$$

where:

- P^j: measured pseudorange to satellite j,
- ρ^j: geometric range between satellite j (at transmission time) and receiver,
- dt^j: satellite clock error,
- dT: receiver clock error (common for all satellites),
- d^j_{ion}, d^j_{trop}: ionospheric and tropospheric delays.

1.4.1 Linearization

The geometric range is linearized around the approximate receiver coordinates:

$$\rho^j = \sqrt{(X^j - X)^2 + (Y^j - Y)^2 + (Z^j - Z)^2}.$$

Expanding this by Taylor linearization about the approximate point (X_0, Y_0, Z_0) yields:

$$\rho^j \approx \rho^j_0 - \frac{X^j - X_0}{\rho^j_0}\Delta X - \frac{Y^j - Y_0}{\rho^j_0}\Delta Y - \frac{Z^j - Z_0}{\rho^j_0}\Delta Z.$$

We define the direction cosines as:

$$a_X^j = -\frac{X^j - X_0}{\rho_0^j}, \qquad a_Y^j = -\frac{Y^j - Y_0}{\rho_0^j}, \qquad a_Z^j = -\frac{Z^j - Z_0}{\rho_0^j}.$$

We also define the corrected observation:

$$\Delta\ell^j = P^j - \rho_0^j - c\,dt^j - d_{\text{ion}}^j - d_{\text{trop}}^j.$$

Substituting and collecting terms gives the linearized observation equation:

$$\Delta\ell^j = a_X^j\,\Delta X + a_Y^j\,\Delta Y + a_Z^j\,\Delta Z - c\,\Delta T.$$

1.4.2 Matrix Form

All satellite observations are stacked to form the vector equation:

$$\Delta\mathbf{L} = \mathbf{A}\,\Delta\mathbf{X}.$$

The vectors and matrices are:

$$\Delta\mathbf{L} = \begin{bmatrix} \Delta\ell^1 \\ \Delta\ell^2 \\ \vdots \\ \Delta\ell^7 \end{bmatrix}, \qquad \mathbf{A} = \begin{bmatrix} a_X^1 & a_Y^1 & a_Z^1 & -c \\ a_X^2 & a_Y^2 & a_Z^2 & -c \\ \vdots & \vdots & \vdots & \vdots \\ a_X^7 & a_Y^7 & a_Z^7 & -c \end{bmatrix}, \qquad \Delta\mathbf{X} = \begin{bmatrix} \Delta X \\ \Delta Y \\ \Delta Z \\ \Delta T \end{bmatrix}.$$

This relates the observation residuals directly to corrections in the receiver state vector.

1.4.3 Least-Squares Solution

Solving the overdetermined system using the least–squares criterion gives:

$$\Delta\mathbf{X} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\Delta\mathbf{L}.$$

The iteration converges to the final Cartesian coordinates:

$$\mathbf{x} = \begin{bmatrix} 2814985.362 \\ 516910.388 \\ 5680955.795 \end{bmatrix}\ \text{m}.$$

1.5 PDOP

The geometric strength of the satellite configuration is evaluated using the covariance matrix:

$$\mathbf{Q_X} = (\mathbf{A}^T\mathbf{A})^{-1}.$$

The position dilution of precision is:

$$PDOP = \sqrt{q_{XX} + q_{YY} + q_{ZZ}},$$

where q_{XX}, q_{YY}, q_{ZZ} are the first three of the four diagonal elements of $\mathbf{Q_X}$.

The resulting value for this satellite geometry is:

$$PDOP = 1.9998.$$

1.6 Final Geodetic Coordinates

The final ECEF coordinates are transformed back to geodetic coordinates using the iterative inverse WGS84 transformation. The final receiver position is:

Parameter	Value
Latitude	63.415472°
Longitude	10.405196°
Height	115.054 m

Table 4: Final receiver geodetic position.

1.7 Receiver Clock Bias

The estimated receiver clock offset from the adjustment is:

$$\Delta T = -1.054 \times 10^{-8} \text{ s.}$$

The corresponding range error is:

$$c\,\Delta T \approx -3.16 \text{ m},$$

meaning the receiver clock is late by approximately 3 m of signal travel time.

2 Summary of Project 2 Results

In this project we determine the position of a rover receiver B relative to a known base receiver A using double-difference carrier phase observations on L1 at two epochs. All coordinates are expressed in the Earth-Centered, Earth-Fixed (ECEF) WGS84 system unless otherwise stated.

2.1 Base and Approximate Rover Coordinates

The geodetic coordinates (latitude, longitude, ellipsoidal height) of the base station A and the approximate geodetic coordinates of the rover station B are given as input to the program. These are:

Station	Latitude (°)	Longitude (°)	Height (m)
Base A (given)	-32.00388500	115.89480200	23.983
Rover B (approx.)	-31.90000000	115.75000000	50.000

Table 5: Given geodetic coordinates of base and approximate rover position (WGS84).

These geodetic coordinates are then transformed to Cartesian ECEF coordinates using the WGS84 ellipsoid. The resulting Cartesian coordinates (from the Python implementation) are:

Station	X (m)	Y (m)	Z (m)
Base A (known)	-2364337.651	4870285.650	-3360809.439
Rover B (approx.)	-2354679.618	4881756.102	-3351049.072

Table 6: Base and approximate rover Cartesian coordinates (ECEF, WGS84).

These values are used as fixed coordinates for A and as linearization point for the unknown rover station B in the subsequent least-squares adjustment.

2.2 Float Solution: Model, Linearization and Ambiguities

The relative positioning is carried out using carrier phase observations between the two receivers (A and B) and a set of GPS satellites at two epochs t_1 and t_2. We form *double differences* (DD) using one reference satellite i and four other satellites j, k, l, m.

2.2.1 Phase Observation and Differences

The undifferenced carrier phase observation (in meters) to satellite α at receiver R $\in \{A, B\}$ can be written as

$$\phi_R^j(t) = \rho_R^j(t) + c\left(dt^j(t) - dT_R(t)\right) + d_{\text{ion},R}^j(t) + d_{\text{trop},R}^j(t) + \lambda N_R^j,$$

where ρ_R^j is the geometric range, dt^j the satellite clock error, dT_R the receiver clock error, d_{ion} and d_{trop} the ionospheric and tropospheric delays, λ the L1 wavelength and N_R^j the (integer) phase ambiguity.

A *single difference* between receivers A and B for satellite j is

$$\phi_{AB}^j(t) = \phi_B^j(t) - \phi_A^j(t),$$

which removes most satellite-related errors. A *double difference* between satellites i and j is then

$$\Delta\ell_{AB}^{ij}(t) = \phi_{AB}^j(t) - \phi_{AB}^i(t) = \phi_B^j(t) - \phi_B^i(t) - \phi_A^j(t) + \phi_A^i(t).$$

In this combination, the receiver clock errors cancel and many effects are further reduced.

The corresponding double-difference ambiguity is denoted N_{AB}^{ij}, and the DD phase model simplifies to

$$\Delta\ell_{AB}^{ij}(t) = \rho_{AB}^{ij}(t) + \lambda N_{AB}^{ij} + \varepsilon_{AB}^{ij}(t),$$

where $\rho_{AB}^{ij}(t)$ is the double-difference geometric range and $\varepsilon_{AB}^{ij}(t)$ represents residual unmodelled errors and noise.

2.2.2 Geometric Range and Linearization

The geometric range between receiver B and satellite j at time t is

$$\rho_B^j(t) = \sqrt{\left(X^j(t) - X_B\right)^2 + \left(Y^j(t) - Y_B\right)^2 + \left(Z^j(t) - Z_B\right)^2}.$$

We linearize this around the approximate rover coordinates (X_{B0}, Y_{B0}, Z_{B0}):

$$\rho_B^j(t) \approx \rho_{B0}^j(t) + \frac{\partial \rho_B^j}{\partial X_B}\bigg|_0 \Delta X_B + \frac{\partial \rho_B^j}{\partial Y_B}\bigg|_0 \Delta Y_B + \frac{\partial \rho_B^j}{\partial Z_B}\bigg|_0 \Delta Z_B.$$

The partial derivatives yield the familiar direction cosines, and for a DD combination between satellites i and j we obtain

$$a_{X_B}^{ij}(t) = -\frac{X^j(t) - X_{B0}}{\rho_{B0}^j(t)} + \frac{X^i(t) - X_{B0}}{\rho_{B0}^i(t)},$$

with analogous expressions for $a_{Y_B}^{ij}(t)$ and $a_{Z_B}^{ij}(t)$. Using these coefficients, the linearized DD observation equation becomes

$$\Delta\ell_{AB}^{ij}(t) = a_{X_B}^{ij}(t)\,\Delta X_B + a_{Y_B}^{ij}(t)\,\Delta Y_B + a_{Z_B}^{ij}(t)\,\Delta Z_B + \lambda N_{AB}^{ij}.$$

2.2.3 Observation Vector $\Delta\mathbf{L}$ and Unknown Vector $\Delta\mathbf{X}$

With one reference satellite i and four other satellites j, k, l, m observed at two epochs t_1 and t_2, we obtain eight DD observations. The observation vector is

$$\Delta\mathbf{L} = \begin{bmatrix} \Delta\ell_{AB}^{ij}(t_1) \\ \Delta\ell_{AB}^{ik}(t_1) \\ \Delta\ell_{AB}^{il}(t_1) \\ \Delta\ell_{AB}^{im}(t_1) \\ \Delta\ell_{AB}^{ij}(t_2) \\ \Delta\ell_{AB}^{ik}(t_2) \\ \Delta\ell_{AB}^{il}(t_2) \\ \Delta\ell_{AB}^{im}(t_2) \end{bmatrix}.$$

The unknown vector consists of corrections to the rover coordinates and the four double-difference ambiguities (in cycles):

$$\Delta\mathbf{X} = \begin{bmatrix} \Delta X_B \\ \Delta Y_B \\ \Delta Z_B \\ N_{AB}^{ij} \\ N_{AB}^{ik} \\ N_{AB}^{il} \\ N_{AB}^{im} \end{bmatrix}.$$

2.2.4 Design Matrix $\mathbf{A}$

Stacking all eight linearized DD equations yields the matrix form

$$\Delta\mathbf{L} = \mathbf{A}\,\Delta\mathbf{X},$$

with the design matrix

$$\mathbf{A} = \begin{bmatrix} a_{X_B}^{ij}(t_1) & a_{Y_B}^{ij}(t_1) & a_{Z_B}^{ij}(t_1) & \lambda & 0 & 0 & 0 \\ a_{X_B}^{ik}(t_1) & a_{Y_B}^{ik}(t_1) & a_{Z_B}^{ik}(t_1) & 0 & \lambda & 0 & 0 \\ a_{X_B}^{il}(t_1) & a_{Y_B}^{il}(t_1) & a_{Z_B}^{il}(t_1) & 0 & 0 & \lambda & 0 \\ a_{X_B}^{im}(t_1) & a_{Y_B}^{im}(t_1) & a_{Z_B}^{im}(t_1) & 0 & 0 & 0 & \lambda \\ a_{X_B}^{ij}(t_2) & a_{Y_B}^{ij}(t_2) & a_{Z_B}^{ij}(t_2) & \lambda & 0 & 0 & 0 \\ a_{X_B}^{ik}(t_2) & a_{Y_B}^{ik}(t_2) & a_{Z_B}^{ik}(t_2) & 0 & \lambda & 0 & 0 \\ a_{X_B}^{il}(t_2) & a_{Y_B}^{il}(t_2) & a_{Z_B}^{il}(t_2) & 0 & 0 & \lambda & 0 \\ a_{X_B}^{im}(t_2) & a_{Y_B}^{im}(t_2) & a_{Z_B}^{im}(t_2) & 0 & 0 & 0 & \lambda \end{bmatrix}.$$

2.2.5 Least-Squares Float Solution and Covariance

The float solution is obtained from the weighted least-squares estimator:

$$\Delta\mathbf{X} = (\mathbf{A}^T\mathbf{P}\mathbf{A})^{-1}\mathbf{A}^T\mathbf{P}\,\Delta\mathbf{L},$$

where $\mathbf{P}$ is the weight matrix of the double-difference observations.

The Python implementation yields the following final rover coordinates (float solution) in Cartesian form:

$$\mathbf{x}_B^{\text{float}} = \begin{bmatrix} -2364335.656 \\ 4870281.513 \\ -3360816.471 \end{bmatrix} \text{ m.}$$

The estimated double-difference phase ambiguities (float values, in cycles) are:

$$\mathbf{N}_{AB}^{\text{float}} = \begin{bmatrix} N_{AB}^{ij} \\ N_{AB}^{ik} \\ N_{AB}^{il} \\ N_{AB}^{im} \end{bmatrix} = \begin{bmatrix} 4.9503 \\ 12.0162 \\ 25.0747 \\ 12.0879 \end{bmatrix} \text{ cycles.}$$

The full covariance matrix of the unknown vector $\Delta\mathbf{X}$ from the float solution is reported by the code as $\mathbf{C_X}^{\text{float}}$. To fit on the page, we factor out a common power of ten and reduce the number of decimals:

$$\mathbf{C_X}^{\text{float}} = 10^{-3} \begin{bmatrix} 2.032 & -0.041 & 0.449 & 11.666 & 2.516 & 2.642 & -0.129 \\ -0.041 & 10.289 & 0.366 & 44.405 & 28.473 & 48.977 & 20.666 \\ 0.449 & 0.366 & 0.674 & 2.554 & -0.110 & 3.610 & -0.002 \\ 11.666 & 44.405 & 2.554 & 266.923 & 143.927 & 224.972 & 91.754 \\ 2.516 & 28.473 & -0.110 & 143.927 & 88.767 & 136.136 & 60.031 \\ 2.642 & 48.977 & 3.610 & 224.972 & 136.136 & 241.229 & 97.394 \\ -0.129 & 20.666 & -0.002 & 91.754 & 60.031 & 97.394 & 43.832 \end{bmatrix},$$

where the upper 3×3 block corresponds to the rover coordinates and the remaining rows and columns to the ambiguities.

The least-squares adjustment yields the residual vector $\mathbf{v}$ (in cycles) as:

$$\mathbf{v} = \begin{bmatrix} 0.0017 \\ -0.0058 \\ -0.0032 \\ -0.0023 \\ -0.0017 \\ 0.0058 \\ 0.0032 \\ 0.0023 \end{bmatrix},$$

with sum of squared residuals:

$$SSR = 1.3521.$$

These results form the basis for the ambiguity fixing and refined positioning in the next step.

2.3 Ambiguity Resolution and Fixed Solutions

The float solution provides real-valued estimates of the double-difference ambiguities $N_{AB}^{ij}, N_{AB}^{ik}, N_{AB}^{il}, N_{AB}^{im}$. To exploit the full precision of carrier phase observations, these ambiguities should be fixed to correct integer values. In this project, three scenarios are investigated:

a) Use the real (float) ambiguity values directly.

b) Round each real ambiguity to its nearest integer.

c) Use the standard deviations of the ambiguities to define a search space and perform an integer search, selecting the best candidate based on the sum of squared residuals.

2.3.1 Scenario (a): Real Ambiguities

In this case, the real-valued ambiguities from the float solution are treated as if they were correct:

$$\mathbf{N}_{AB}^{(a)} = \begin{bmatrix} N_{AB}^{ij} \\ N_{AB}^{ik} \\ N_{AB}^{il} \\ N_{AB}^{im} \end{bmatrix} = \begin{bmatrix} 4.9503 \\ 12.0162 \\ 25.0747 \\ 12.0879 \end{bmatrix} \text{ cycles.}$$

Re-computing the rover position with these ambiguities fixed gives:

$$\mathbf{x}_B^{(a)} = \begin{bmatrix} -2364335.656 \\ 4870281.513 \\ -3360816.471 \end{bmatrix} \text{ m},$$

which is numerically identical (up to rounding) to the original float solution.

The reported covariance matrix for the rover coordinates in this case is:

$$\mathbf{C_X}^{(a)} = 10^{-5} \begin{bmatrix} 7.639 & -6.359 & 5.188 \\ -6.359 & 9.980 & -4.333 \\ 5.188 & -4.333 & 6.883 \end{bmatrix} \text{ m}^2.$$

2.3.2 Scenario (b): Rounded Integer Ambiguities

In this scenario, each float ambiguity is rounded to the nearest integer:

$$\mathbf{N}_{AB}^{(b)} = \begin{bmatrix} N_{AB}^{ij} \\ N_{AB}^{ik} \\ N_{AB}^{il} \\ N_{AB}^{im} \end{bmatrix} = \begin{bmatrix} 5 \\ 12 \\ 25 \\ 12 \end{bmatrix} \text{ cycles.}$$

The resulting fixed-position solution for the rover is:

$$\mathbf{x}_B^{(b)} = \begin{bmatrix} -2364335.628 \\ 4870281.490 \\ -3360816.466 \end{bmatrix} \text{ m.}$$

The corresponding covariance matrix is:

$$\mathbf{C_X}^{(b)} = 10^{-5} \begin{bmatrix} 7.639 & -6.359 & 5.188 \\ -6.359 & 9.980 & -4.333 \\ 5.188 & -4.333 & 6.883 \end{bmatrix} \text{ m}^2,$$

which is practically identical to $\mathbf{C_X}^{(a)}$.

2.3.3 Scenario (c): Integer Search Using Ambiguity Standard Deviations

To guide the integer search, we use the standard deviations of the float ambiguities (in cycles):

$$\boldsymbol{\sigma}_N = \begin{bmatrix} 0.51664623 \\ 0.29793792 \\ 0.49115036 \\ 0.20936087 \end{bmatrix} \text{ cycles.}$$

Based on these values, search ranges around the float ambiguities are defined. We look at the intervals of 3 standard deviations in either direction, and include the integers just outside the interval. This way we perform a thorough check of all possible anomaly integers. In the Python output this corresponds to:

$$N_{AB}^{ij} \in \{3,4,5,6,7\}, \quad N_{AB}^{ik} \in \{11,12,13\}, \quad N_{AB}^{il} \in \{23,24,25,26,27\}, \quad N_{AB}^{im} \in \{11,12,13\}.$$

For each integer candidate vector $\mathbf{N}_{AB}^{(c)} = \left(N_{AB}^{ij}, N_{AB}^{ik}, N_{AB}^{il}, N_{AB}^{im}\right)^T$, a least-squares adjustment is run and the sum of squared residuals (SSR) is computed. The candidate with the minimum SSR is chosen as the best solution; the second best candidate is also recorded for comparison.

The best fixed ambiguities found by the search are:

$$\mathbf{N}_{AB}^{\text{best}} = \begin{bmatrix} 5 \\ 12 \\ 25 \\ 12 \end{bmatrix} \text{ cycles,}$$

which coincide with the rounded integer values from scenario (b).

The corresponding rover coordinates (best solution) are:

$$\mathbf{x}_{B}^{\text{best}} = \begin{bmatrix} -2364335.628 \\ 4870281.490 \\ -3360816.466 \end{bmatrix} \text{ m,}$$

with covariance matrix:

$$\mathbf{C_X}^{\text{best}} = 10^{-5} \begin{bmatrix} 7.639 & -6.359 & 5.188 \\ -6.359 & 9.980 & -4.333 \\ 5.188 & -4.333 & 6.883 \end{bmatrix} \text{ m}^2.$$

The second-best candidate ambiguities and rover coordinates are:

$$\mathbf{N}_{AB}^{\text{2nd}} = \begin{bmatrix} 6 \\ 13 \\ 27 \\ 13 \end{bmatrix} \text{ cycles,} \qquad \mathbf{x}_{B}^{\text{2nd}} = \begin{bmatrix} -2364335.590 \\ 4870281.474 \\ -3360816.436 \end{bmatrix} \text{ m.}$$

The ratio between the SSR of the second-best and best solutions is:

$$\frac{SSR_{\text{2nd}}}{SSR_{\text{best}}} = 7.9078,$$

which is significantly larger than the commonly used threshold of 3. This indicates that the best integer ambiguity set is very likely to be correct.

2.3.4 Comparison of Scenarios and Final Choice

Table 7 summarizes the three scenarios in terms of ambiguities and rover coordinates. Because the integer search in scenario (c) selects the same integer vector as simple rounding in scenario (b), and the SSR-ratio test is strongly satisfied, we adopt the best search solution (identical to scenario (b)) as the final rover coordinates.

Scenario	N_{AB}^{ij}	N_{AB}^{ik}	N_{AB}^{il}	N_{AB}^{im}	X (m)	Y (m)	Z (m)
(a) Float	4.9503	12.0162	25.0747	12.0879	-2364335.656	4870281.513	-3360816.471
(b) Rounded	5.0000	12.0000	25.0000	12.0000	-2364335.628	4870281.490	-3360816.466
(c) Search best	5.0000	12.0000	25.0000	12.0000	-2364335.628	4870281.490	-3360816.466

Table 7: Comparison of ambiguity scenarios and corresponding rover Cartesian coordinates.

2.4 Final Rover Coordinates in Geodetic Form

In the last step, the various Cartesian rover solutions are transformed to geodetic coordinates (latitude, longitude, ellipsoidal height) using the iterative inverse WGS84 transformation, as in Project 1.

The Python script reports the following geodetic coordinates (lat, lon, height) with high precision:

Solution type	Latitude (°)	Longitude (°)	Height (m)
Float solution (step 2)	-32.00396038	115.89480214	23.81502303
Real ambiguities fixed (3a)	-32.00396038	115.89480214	23.81502302
Rounded ambiguities (3b)	-32.00396050	115.89480197	23.78448149
Best integer search (3c, best)	-32.00396050	115.89480197	23.78448149
2nd best candidate	-32.00395824	115.89480144	24.19566760

Table 8: Rover coordinates in geodetic form for different ambiguity solutions.

The best solution from the integer search (scenario (c)) coincides with the rounded ambiguity solution and is chosen as the final rover position:

$$\varphi_B = -32.00396050^\circ, \quad \lambda_B = 115.89480197^\circ, \quad h_B = 23.784 \text{ m}.$$

Given the small standard deviations in the Cartesian domain and the large SSR ratio between the best and second-best ambiguity candidates, this solution can be regarded as reliable at the centimeter level for relative positioning.

Appendix

A Project 1 Python code

```python
import georinex as gr
import warnings
import pandas as pd
import numpy as np
warnings.filterwarnings("ignore", category=FutureWarning, module="georinex")

from data_project1_absolute_code.variables import T

"""
General constants
"""
c = 299792458
GM = 3.986005e14
omega_e = 7.2921151467e-5
pi = 3.1415926535898


"""
Functions
"""

def R1(x): #rotation matrix around x axis
    return np.array([[1, 0, 0],
                     [0, np.cos(x), -np.sin(x)],
                     [0, np.sin(x), np.cos(x)]])

def R3(x): #rotation matrix around z axis
    return np.array([[np.cos(x), -np.sin(x), 0],
                     [np.sin(x), np.cos(x), 0],
                     [0, 0, 1]])

def satellite_coordinates(sat, T, correction=True):

    deltaN = sat["DeltaN"] if correction else 0
    Idot = sat["IDOT"] if correction else 0
    OmegaDot = sat["OmegaDot"] if correction else 0
    Cuc = sat["Cuc"] if correction else 0
    Cus = sat["Cus"] if correction else 0
    Crc = sat["Crc"] if correction else 0
    Crs = sat["Crs"] if correction else 0
    Cic = sat["Cic"] if correction else 0
    Cis = sat["Cis"] if correction else 0

    # print(sat)
    t_s = T - sat["P"]/c + sat["dt"]

    t_k = t_s - sat["Toe"]
    if t_k >  302400: t_k -= 604800
    if t_k < -302400: t_k += 604800

    M_k = sat["M0"] + (GM**0.5 / sat["sqrtA"]**3 + deltaN) * t_k

    E_k = M_k
    for _ in range(3):
        E_k = E_k + (M_k - E_k + sat["Eccentricity"] * np.sin(E_k)) / (1 -
        ↪   sat["Eccentricity"] * np.cos(E_k))

    f_k = 2 * np.arctan(np.sqrt((1 + sat["Eccentricity"]) / (1 - sat["Eccentricity"])) *
    ↪   np.tan(E_k / 2))
```

```python

    u_k = sat["omega"] + f_k + Cuc * np.cos(2 * (sat["omega"] + f_k)) + Cus * np.sin(2 * (sat["omega"] + f_k))

    r_k = sat["sqrtA"]**2 * (1 - sat["Eccentricity"] * np.cos(E_k)) + Crc * np.cos(2 * (sat["omega"] + f_k)) + Crs * np.sin(2 * (sat["omega"] + f_k))

    i_k = sat["Io"] + Idot * t_k + Cic * np.cos(2 * (sat["omega"] + f_k)) + Cis * np.sin(2 * (sat["omega"] + f_k))

    lambda_k = sat["Omega0"] + (OmegaDot - omega_e) * t_k - omega_e * sat["Toe"]

    coords = R3(lambda_k) @ R1(i_k) @ R3(u_k) @ np.array([r_k, 0, 0]) # not negative rotation parameters because Y is flipped negative

    return coords


def geodetic_to_cartesian(geodetic_coords):
    """
    Convert geodetic coordinates (latitude, longitude, height) to ECEF Cartesian coordinates (X, Y, Z).
    Latitude and longitude are in degrees, height is in meters.
    Returns a numpy array [X, Y, Z] in meters.
    """
    phi, lam, h = np.deg2rad(geodetic_coords[0]), np.deg2rad(geodetic_coords[1]), geodetic_coords[2]
    a = 6378137.0 # WGS-84
    b = 6356752.3142

    N = a**2 / np.sqrt(a**2 * np.cos(phi)**2 + b**2 * np.sin(phi)**2)
    x = (N + h) * np.cos(phi) * np.cos(lam)
    y = (N + h) * np.cos(phi) * np.sin(lam)
    z = (b**2 / a**2 * N + h) * np.sin(phi)
    return np.array([x, y, z])


def A_matrix(satellites, approx_receiver_cartesian):
    A = []
    for sat in satellites:
        rho_i = np.linalg.norm(sat["coords"] - approx_receiver_cartesian[:3])
        row = [
            -(sat["coords"][0] - approx_receiver_cartesian[0]) / rho_i,
            -(sat["coords"][1] - approx_receiver_cartesian[1]) / rho_i,
            -(sat["coords"][2] - approx_receiver_cartesian[2]) / rho_i,
            -c
        ]
        A.append(row)
    return np.array(A)

def delta_L_vector(satellites, approx_receiver_cartesian):
    L = []
    for sat in satellites:
        rho_i = np.linalg.norm(sat["coords"] - approx_receiver_cartesian)
        L.append(sat["P"] - rho_i - c * sat["dt"] - sat["dion"] - sat["dtrop"])
    return np.array(L).reshape(-1, 1)


def cartesian_to_geodetic(cartesian_coords):
    """
    Convert ECEF Cartesian coordinates (X, Y, Z) to geodetic coordinates (latitude, longitude, height).
    X, Y, Z are in meters.
```

```
    Returns a numpy array [latitude, longitude, height] where latitude and longitude are in degrees and height is in meters.
    """
    x, y, z = cartesian_coords
    a = 6378137.0 # WGS-84
    b = 6356752.3142
    e2 = 1 - (b**2 / a**2)
    lam = np.arctan2(y, x)
    p = np.sqrt(x**2 + y**2)
    phi_0 = np.arctan2(z, p * (1 - e2))
    for i in range(10):
        N_0 = a**2 / np.sqrt(a**2 * np.cos(phi_0)**2 + b**2 * np.sin(phi_0)**2)
        h = p / np.cos(phi_0) - N_0
        phi = np.arctan2(z, p * (1 - e2 * (N_0 / (N_0 + h))))
        if abs(phi - phi_0) < 1e-12: break
        if i == 9:
            print("Max iterations reached")
            break
        phi_0 = phi
    return np.array([np.rad2deg(phi), np.rad2deg(lam), h])


"""
Steps
"""

def main():
    ephimerides = gr.load("data_project1_absolute_code/ephimerides.nav").to_dataframe()
    obs = pd.read_csv("data_project1_absolute_code/observations.csv")

    satellites = [
        pd.concat([ephimerides.xs("G08", level="sv").iloc[0], obs[obs["sv"]=="G08"].iloc[0]]),
        pd.concat([ephimerides.xs("G10", level="sv").iloc[0], obs[obs["sv"]=="G10"].iloc[0]]),
        pd.concat([ephimerides.xs("G21", level="sv").iloc[0], obs[obs["sv"]=="G21"].iloc[0]]),
        pd.concat([ephimerides.xs("G24", level="sv").iloc[0], obs[obs["sv"]=="G24"].iloc[0]]),
        pd.concat([ephimerides.xs("G17", level="sv").iloc[1], obs[obs["sv"]=="G17"].iloc[0]]),
        pd.concat([ephimerides.xs("G03", level="sv").iloc[1], obs[obs["sv"]=="G03"].iloc[0]]),
        pd.concat([ephimerides.xs("G14", level="sv").iloc[0], obs[obs["sv"]=="G14"].iloc[0]]),
    ]

    for sat in satellites:
        coords = satellite_coordinates(sat, T, correction=True)
        sat["coords"] = coords

    print("\nCoordinates (ECEF) at transmission time:")
    [print(sat["coords"]) for sat in satellites]


    """
    Step 2:
    """

    for sat in satellites:
        coords_uncorrected = satellite_coordinates(sat, T, correction=False)
        sat["coords_uncorrected"] = coords_uncorrected

    print("\nUncorrected coordinates (ECEF) at transmission time:")
```

```
[print(sat["coords_uncorrected"]) for sat in satellites]

for sat in satellites:
    diff = sat["coords"] - sat["coords_uncorrected"]
    sat["diff"] = diff
    sat["diff_magnitude"] = np.linalg.norm(diff)

print("\nDifference between corrected and uncorrected coordinates:")
[print(sat["diff"], "magnitude:", sat["diff_magnitude"]) for sat in satellites]


"""
Step 3:
"""

approx_receiver_geodetic = np.array([63.2, 10.2, 100]) # lat, lon, height in degrees
↪   and meters

approx_receiver_cartesian = geodetic_to_cartesian(approx_receiver_geodetic)
print("\nApprox receiver coordinates in Cartesian:")
print(approx_receiver_cartesian)


"""
Step 4:
"""

receiver_cartesian = approx_receiver_cartesian.copy()
receiver_clock_bias = 0
for i in range(10):
    A = A_matrix(satellites, receiver_cartesian)
    delta_L = delta_L_vector(satellites, receiver_cartesian)
    delta_X = np.linalg.inv(A.T @ A) @ A.T @ delta_L
    receiver_cartesian += delta_X[:3].flatten()
    receiver_clock_bias = delta_X[3, 0]
    if np.linalg.norm(delta_X[:3]) < 1e-6: break
    if i == 9:
        print("Max iterations reached")
        break
    print(f"Iteration {i+1}:", receiver_cartesian)

print("\nFinal receiver coordinates in Cartesian:", receiver_cartesian)


"""
Step 5:
"""

Q_x = np.linalg.inv(A.T @ A)

print("\nCovariance matrix Q_x:\n", Q_x)

PDOP = np.sqrt(Q_x[0,0] + Q_x[1,1] + Q_x[2,2])
print("PDOP:", PDOP)


"""
Step 6:
"""

receiver_geodetic = cartesian_to_geodetic(receiver_cartesian)
print("\nFinal receiver coordinates in Geodetic (lat, lon, height):")
print(receiver_geodetic)
```

```python

    """
    Step 7:
    """

    print("\nReceiver clock bias (in seconds):", receiver_clock_bias)

if __name__ == "__main__":
    main()
```

B Project 2 Python code

```python
import pandas as pd
import numpy as np
from project1_absolute_code import geodetic_to_cartesian, cartesian_to_geodetic

from data_project2_relative_phase.variables import T1, T2, base_known_llh, rover_approx_llh, P_matrix

"""
General constants
"""
c = 299792458
frequency_L1 = 1575.42e6
wavelength_L1 = c / frequency_L1


"""
Functions
"""

def A_matrix(satellites_rover, approx_rover_xyz, fixed_ambiguities=False):
    A = []
    for T in (T1, T2):
        sat_rover_i = satellites_rover[T].iloc[0]
        rho_B_i = np.linalg.norm(sat_rover_i[["X", "Y", "Z"]].values - approx_rover_xyz[:3])
        for j in range(1, len(satellites_rover[T])):
            sat_rover = satellites_rover[T].iloc[j]
            rho_B_j = np.linalg.norm(sat_rover[["X", "Y", "Z"]].values - approx_rover_xyz[:3])

            if not fixed_ambiguities:
                row = np.zeros(3 + len(satellites_rover[T]) -1)
                row[3 + j -1] = wavelength_L1
            else:
                row = np.zeros(3)

            for x, X in enumerate(["X","Y","Z"]):
                row[x] = ( -( (sat_rover[X] - approx_rover_xyz[x]) / rho_B_j ) + ( (sat_rover_i[X] - approx_rover_xyz[x]) / rho_B_i ) )
            A.append(row)
    return np.array(A)

def delta_L_vector(satellites_base, satellites_rover, approx_base_xyz, approx_rover_xyz):
    L = []
    for T in (T1, T2):
        sat_base_i = satellites_base[T].iloc[0]
        sat_rover_i = satellites_rover[T].iloc[0]
        rho_A_i = np.linalg.norm(sat_base_i[["X", "Y", "Z"]].values - approx_base_xyz)
        rho_B_i = np.linalg.norm(sat_rover_i[["X", "Y", "Z"]].values - approx_rover_xyz)
        for j in range(1, len(satellites_rover[T])):
            sat_base = satellites_base[T].iloc[j]
            sat_rover = satellites_rover[T].iloc[j]
            rho_A_j = np.linalg.norm(sat_base[["X", "Y", "Z"]].values - approx_base_xyz)
            rho_B_j = np.linalg.norm(sat_rover[["X", "Y", "Z"]].values - approx_rover_xyz)

            Phi_AB_ij = (sat_rover["L1"] - sat_rover_i["L1"] - sat_base["L1"] + sat_base_i["L1"]) * wavelength_L1
            L.append(Phi_AB_ij - rho_B_j + rho_B_i + rho_A_j - rho_A_i)
    return np.array(L).reshape(-1, 1)

def estimate_position_float(satellites_base, satellites_rover, base_known_xyz, rover_approx_xyz):
```

```python
    rover_xyz = rover_approx_xyz.copy()
    for i in range(10):
        A = A_matrix(satellites_rover, rover_xyz)
        delta_L = delta_L_vector(satellites_base, satellites_rover, base_known_xyz, rover_xyz)
        delta_X = np.linalg.inv(A.T @ P_matrix @ A) @ A.T @ P_matrix @ delta_L
        rover_xyz += delta_X[:3].flatten()
        if np.linalg.norm(delta_X[:3]) < 1e-6: break
        if i == 9:
            print("Max iterations reached")
            break
    phase_ambiguities = delta_X[3:].flatten()
    C_X = np.linalg.inv(A.T @ P_matrix @ A)
    v = delta_L - A @ delta_X
    return rover_xyz, phase_ambiguities, C_X, v

def estimate_position_fixed(satellites_base, satellites_rover, base_known_xyz, rover_approx_xyz, fixed_ambiguities):
    rover_xyz_fixed = rover_approx_xyz.copy()
    for i in range(10):
        A = A_matrix(satellites_rover, rover_xyz_fixed, fixed_ambiguities=True)
        delta_L = delta_L_vector(satellites_base, satellites_rover, base_known_xyz, rover_xyz_fixed)
        for j in range(len(fixed_ambiguities)): # Subtract the fixed ambiguities contribution
            delta_L[j] -= fixed_ambiguities[j] * wavelength_L1
            delta_L[j + len(fixed_ambiguities)] -= fixed_ambiguities[j] * wavelength_L1
        delta_X = np.linalg.inv(A.T @ P_matrix @ A) @ A.T @ P_matrix @ delta_L
        rover_xyz_fixed += delta_X.flatten()
        if np.linalg.norm(delta_X) < 1e-6: break
        if i == 9:
            print("Max iterations reached")
            break
    C_X = np.linalg.inv(A.T @ P_matrix @ A)
    v = delta_L - A @ delta_X
    return rover_xyz_fixed, C_X, v


def main():
    satellites_base  = {T1: pd.read_csv("data_project2_relative_phase/base_T1.csv"),
                        T2: pd.read_csv("data_project2_relative_phase/base_T2.csv")}
    satellites_rover = {T1: pd.read_csv("data_project2_relative_phase/rover_T1.csv"),
                        T2: pd.read_csv("data_project2_relative_phase/rover_T2.csv")}

    """
    Step 1: Transform receiver to Cartesian coordinates
    """
    print("Step 1:")

    base_known_xyz = geodetic_to_cartesian(base_known_llh)
    rover_approx_xyz = geodetic_to_cartesian(rover_approx_llh)
    print("Base position (XYZ):", base_known_xyz)
    print("Rover approximate position (XYZ):", rover_approx_xyz)


    """
    Step 2: Observation equation, design matrix and delta L vector. Estimate rover position with Double difference.
    """
    print("Step 2:")

    rover_xyz_float, phase_ambiguities, C_X_float, v = estimate_position_float(satellites_base, satellites_rover, base_known_xyz, rover_approx_xyz)
```

```
    print("Final rover coordinates in Cartesian:", rover_xyz_float)
    print("Estimated phase ambiguities (in cycles):", phase_ambiguities)
    print("Covariance matrix C_X:", C_X_float)
    print("Residuals vector v:", v.flatten())
    ssr = v.T @ P_matrix @ v
    print("Sum of squared residuals (SSR):", ssr)


    """
    Step 3: Fixing the ambiguities and re-estimating the rover position
    """
    print("Step 3a:")

    fixed_ambiguities_real = phase_ambiguities.copy() # Real ambiguities
    print("Fixed ambiguities (in cycles) (real)    :", fixed_ambiguities_real)
    rover_xyz_fixed_real, C_X, v = estimate_position_fixed(satellites_base,
    ↪  satellites_rover, base_known_xyz, rover_approx_xyz, fixed_ambiguities_real)
    print("Rover:", rover_xyz_fixed_real, "C_X:", C_X)


    print("Step 3b:")

    fixed_ambiguities_rounded = np.round(phase_ambiguities) # Round to nearest integer
    print("Fixed ambiguities (in cycles) (rounded):", fixed_ambiguities_rounded)
    rover_xyz_fixed_rounded, C_X, v = estimate_position_fixed(satellites_base,
    ↪  satellites_rover, base_known_xyz, rover_approx_xyz, fixed_ambiguities_rounded)
    print("Rover:", rover_xyz_fixed_rounded, "C_X:", C_X)

    print("Step 3c:")

    ambiguities_std = np.sqrt(np.diag(C_X_float)[3:])
    print("Ambiguity standard deviations (in cycles):", ambiguities_std)
    ambiguities_min = phase_ambiguities - 3 * ambiguities_std
    ambiguities_max = phase_ambiguities + 3 * ambiguities_std
    fixed_ambiguities_min = np.floor(ambiguities_min)
    fixed_ambiguities_max = np.ceil(ambiguities_max)

    def product_ranges(ranges):
        if not ranges:
            yield ()
            return
        first, *rest = ranges
        for value in first:
            for prod in product_ranges(rest):
                yield (value, ) + prod

    ranges = [
        range(int(min_i), int(max_i) + 1)
        for min_i, max_i in zip(fixed_ambiguities_min, fixed_ambiguities_max)
    ]
    print("Searching over ranges for fixed ambiguities:")
    for r in ranges:
        print(r, end=" ")
    print()

    ssr_best = float('inf')
    combination_best = None
    rover_xyz_best, C_X_best, v_best = None, None, None
    ssr_2best = float('inf')
    combination_2best = None
    rover_xyz_2best, C_X_2best, v_2best = None, None, None
    for fixed_ambiguities_combo in product_ranges(ranges):
        rover_xyz, C_X, v = estimate_position_fixed(satellites_base, satellites_rover,
        ↪  base_known_xyz, rover_approx_xyz, fixed_ambiguities_combo)
```

```
            ssr = (v.T @ P_matrix @ v).item()
            if ssr < ssr_best:
                ssr_2best = ssr_best
                combination_2best = combination_best
                rover_xyz_2best, C_X_2best, v_2best = rover_xyz_best, C_X_best, v_best
                ssr_best = ssr
                combination_best = fixed_ambiguities_combo
                rover_xyz_best, C_X_best, v_best = rover_xyz, C_X, v
            elif ssr < ssr_2best:
                ssr_2best = ssr
                combination_2best = fixed_ambiguities_combo
                rover_xyz_2best, C_X_2best, v_2best = rover_xyz, C_X, v


    """
    Step 4: Convert final rover position to Geodetic coordinates
    """
    print("Step 4:")

    rover_llh_float = cartesian_to_geodetic(rover_xyz_float)
    print("Rover coordinates float solution            (lat, lon, height):",
    ↪   rover_llh_float)

    rover_llh_fixed_real = cartesian_to_geodetic(rover_xyz_fixed_real)
    print("Rover coordinates real fixed ambiguities    (lat, lon, height):",
    ↪   rover_llh_fixed_real)

    rover_llh_fixed_rounded = cartesian_to_geodetic(rover_xyz_fixed_rounded)
    print("Rover coordinates rounded fixed ambiguities (lat, lon, height):",
    ↪   rover_llh_fixed_rounded)

    rover_llh_best = cartesian_to_geodetic(rover_xyz_best)
    print("Best rover coordinates in Geodetic          (lat, lon, height):",
    ↪   rover_llh_best)
    print("Best fixed ambiguities (in cycles)          :", combination_best)

    rover_llh_2best = cartesian_to_geodetic(rover_xyz_2best)
    print("2nd Best rover coordinates in Geodetic      (lat, lon, height):",
    ↪   rover_llh_2best)
    print("2nd Best fixed ambiguities (in cycles)      :", combination_2best)

    print("ssr ratio 2nd best / best:", ssr_2best / ssr_best)
    print("Best solution likely correct if ratio > 3:", ssr_2best / ssr_best > 3)


if __name__ == "__main__":
    main()
```

C GitHub Repository

The full source code for this project, including all Python scripts and data files, is available at the following GitHub repository:

`https://github.com/tiltobias/TBA4565-GPS`