THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

DEPARTMENT OF LAND SURVEYING AND GEO-INFORMATICS
土地測量及地理資訊學系

The Hong Kong Polytechnic University

Department of Land Surveying and Geo-informatics

LSGI3322 Satellite Positioning Systems

GPS Positioning Project

(Intermediate 2 – Calculating GPS Satellite Positions)

Subject Lecturer: Prof. George Liu

Student Name: Waterdragen

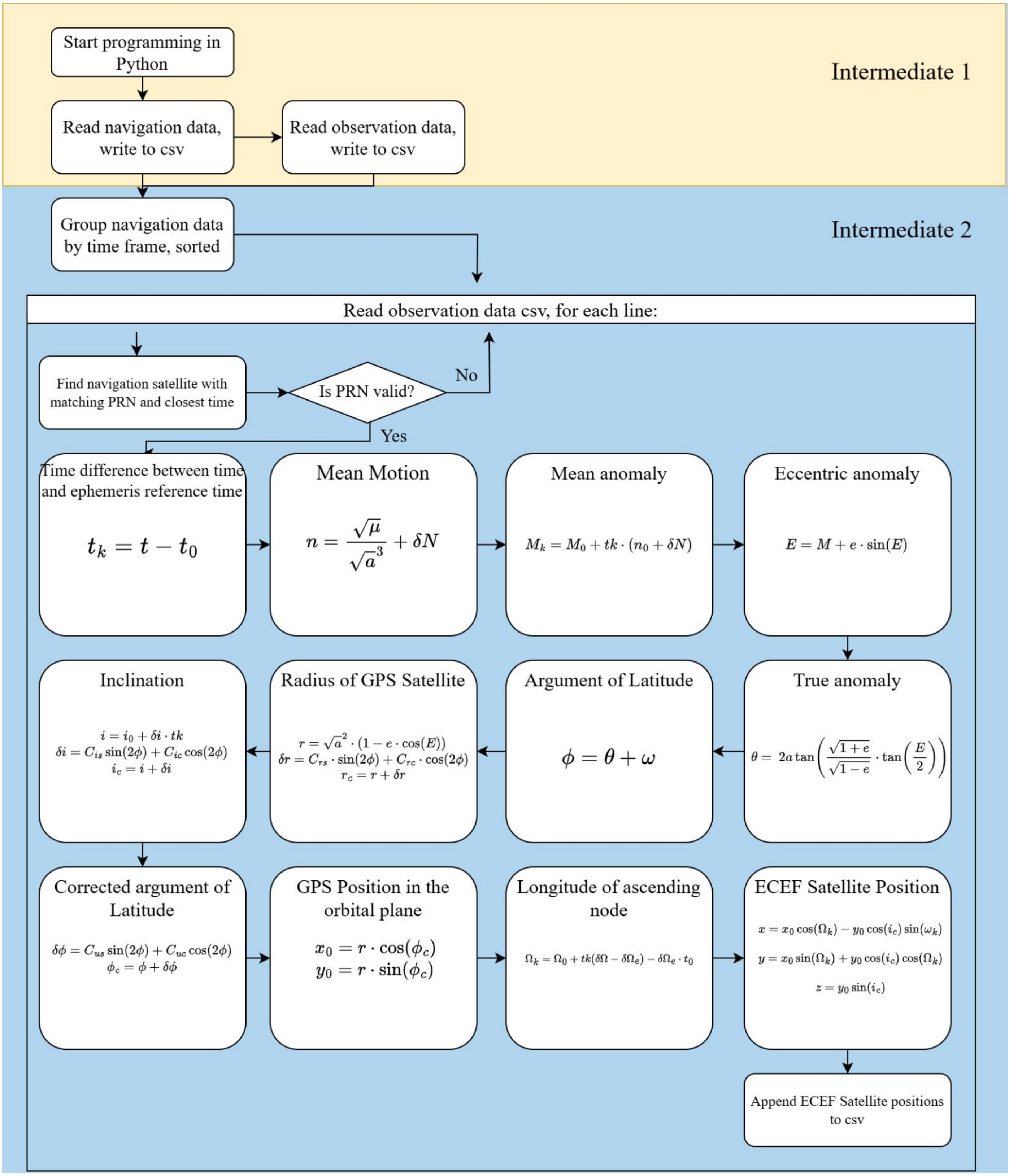
Student ID: 24[redacted]D

Year 3 student of Land Surveying and Geo-informatics

Task

In this exercise I have further developed the Python program to calculate the GPS satellite positions. The below flowchart shows the steps to calculate the GPS satellite positions using various formulas and methods.

Workflow



Modifications to Intermediate 1

1. Calculation of date number and time number for observation data

I have converted the date into number of days since January 0, 0000 in the proleptic ISO calendar to be compatible with the future calculations with the time.

```python
def date_num(dt: datetime) -> int:
    return dt.toordinal() + 366
```

Python's `toordinal` function returns number of days since January 1, 0001. To convert to our desired date number, we add 366 (number of days in year 0000).

```python
def time_num(dt: datetime) -> int:
    return int(timedelta(days=dt.isoweekday() % 7,
                         hours=dt.hour,
                         minutes=dt.minute,
                         seconds=dt.second).total_seconds())
```

Time number is the number of seconds since the start of the Sunday. Python's `isoweekday` function returns 1 for Monday and 7 for Sunday, we can use the modulo 7 to get our desired weekday.

This number can also be seen in the t0 parameter of the navigation data, therefore it is crucial to calculate the time number.

2. Separate into different modules

`main.py`	main script for running the Python program
`core.py`	data structures for `NavData` and `ObsData`
`consts.py`	define constants
`helper_fn.py`	helper functions related to RINEX file formats
`math_fn.py`	helper functions related to math
`util.py`	miscellaneous helper functions related to data type manipulation (group by duplicates, list iterator, find first)

Detailed workflow and corresponding Python code

<table>
<tr><th>Workflow</th><th>Formula / Constant</th><th>Python Code</th></tr>
<tr><td>Define the necessary constants for the upcoming formulas for orbit calculation.</td><td>$c = 299792458\ m/s$
$g = 9.80665\ m/s^2$
$G\ = 6.6725 * 10^{11}\ m^3/kg\ s^2$
$M = 5.972 * 10^{24}\ kg$
$\mu = 3.986004418 * 10^{14}\ m^3/s^2$
$\delta\Omega = 7.2921151467 * 10^{-5}\ rad/s$</td><td><pre><code>class Consts:
    c = 299792458   # Speed of light (m/s)
    g = 9.80665   # Acceleration of earth's gravity (m/s^2)
    G = 6.67259e-11   # Universal gravitational constant (m^3/kg s^2)
    M = 5.972e24   # Mass of the Earth (kg)
    mu = 3.986004418e14 # Standard gravitational parameter (μ = G * M) (m^3/s^2)
    omega_e = 7.2921151467e-5 # Earth rotation rate (rad/s)

    half_week = 3.5 * 60 * 60 * 24</code></pre></td></tr>
<tr><td>Time difference between time and ephemeris reference time, then normalized within [-3.5 day, +3.5 day] range</td><td>$t_k = t - t_0$</td><td><pre><code>def time_diff_k(t: float, t0: float) -> float:
    tk = t - t0
    if tk > Consts.half_week:
        tk -= 2 * Consts.half_week
    elif tk < - Consts.half_week:
        tk += 2 * Consts.half_week
    return tk</code></pre></td></tr>
</table>

Mean motion (n)	$n = \frac{\sqrt{u}}{(\sqrt{a})^3} + \delta n$	`def mean_motion(sqrt_a: float, delta_n: float) -> float:`
		`    # n = sqrt(mu / a^3)`
		`    #   = sqrt(mu) / sqrt(a^3)`
		`    #   = sqrt(mu) / sqrt(a) ^ 3`
		`    # n = n0 + Δn`
		`    return sqrt(Consts.mu) / sqrt_a ** 3 + delta_n`

Mean anomaly (M)	$M_k = M_0 + t_k(n_0 + \delta n)$	`def mean_anomaly(m0: float, n: float, tk: float) -> float:`
		`    # Mk = m0 + tk * (n0 + Δn)`
		`    return m0 + tk * n`

Eccentric anomaly (E)	$E_k = M_k + e \sin(E_k)$	`def ecc_anomaly(m: float, ecc: float) -> float:`
		`    # E = M + e * sin(E)`
		`    # firstly, assume e * sin(M) is very small ~= 0`
		`    # E = M`
		`    # then we can work bottom-up by substituting E to get the new E`
		`    iterations = 30`
		`    e = m`
		`    for _ in range(iterations):`
		`        e = m + ecc * sin(e)`
		
		`    return e`

True anomaly (θ)	$\theta_k = 2 * atan\left(\frac{sqrt(1+e)}{sqrt(1-e)} tan\left(\frac{E}{2}\right)\right)$	`def true_anomaly(E: float, ecc: float) -> float:`
		`    # θ = 2 * atan(sqrt(1 + e) / sqrt(1 - e) * tan(E / 2))`
		`    return 2 * atan(sqrt(1 + ecc) / sqrt(1 - ecc) * tan(E / 2))`

Argument of latitude	$\varphi_k = \theta_k + \omega$	`def argument_of_latitude(theta: float, omega: float) -> float:` `    return theta + omega`

Orbit radius of the satellite position (r)	$r = a(1 - e\cos E_k) + \delta r_k$
	$\delta r_k = C_{rs}\sin(2\varphi_k) + C_{ic}\cos(2\varphi_k)$
	$r_c = r_k + \delta r_k$
		`def orbit_radius(sqrt_a, ecc, E, crs, crc, phi) -> float:`
		`    r = sqrt_a * sqrt_a * (1 - ecc * cos(E))`
		`    delta_r = crs * sin(2 * phi) + crc * cos(2 * phi)`
		`    return r + delta_r`

Inclination (i)	$i_k = i_0 + \delta i + \delta i_k * t_k$
	$\delta i_k = C_{is}\sin(2\varphi_k) + C_{ic}\cos(2\varphi_k)$
	$i_c = i_k + \delta i_k$
		`def inclination(i0, d_i, tk, cis, cic, phi) -> float:`
		`    i = i0 + d_i * tk`
		`    delta_i = cis * sin(2 * phi) + cic * cos(2 * phi)`
		`    return i + delta_i`

Corrected Argument of latitude	$\delta\varphi_k = C_{us}\sin(2\varphi_k) + C_{uc}\cos(2\varphi_k)$
	$\varphi_c = \varphi_k + \delta\varphi_k$
		`def argument_of_latitude_corrected(cus, cuc, phi) -> float:`
		`    delta_phi = cus * sin(2 * phi) + cuc * cos(2 * phi)`
		`    return phi + delta_phi`

GPS Position in orbital plane	$x_0 = r_c \cos(\varphi_c)$
	$y_0 = r_c \sin(\varphi_c)$
		`def gps_position_orbital_plane(r, phi_c) -> tuple[float, float]:`
		`    x0 = r * cos(phi_c)`
		`    y0 = r * sin(phi_c)`
		`    return x0, y0`

Longitude of ascending node	$\Omega_k = \Omega_0 + t_k(\delta\Omega - \delta\Omega_e) - \delta\Omega_e * t_0$	`def longitude_of_ascending_node(omega_0, d_omega, tk, t0) -> float:` `    return omega_0 + (d_omega - Consts.omega_e) * tk - Consts.omega_e * t0`

Earth-centered Earth-fixed (ECEF) frame in orbital terrestrial coordinate system	$x = x_0\cos(\Omega_k) - y_0\cos(i_c)\sin(\Omega_k)$
	$y = x_0\sin(\Omega_k) + y_0\cos(i_c)\cos(\Omega_k)$
	$z = y_0\sin(i_c)$
		`def gps_position_ecef(x0, y0, omega_k, i) -> tuple[float, float, float]:`
		`    x = x0 * cos(omega_k) - y0 * cos(i) * sin(omega_k)`
		`    y = x0 * sin(omega_k) + y0 * cos(i) * cos(omega_k)`
		`    z = y0 * sin(i)`
		`    return x, y, z`

How to run

run `python main.py` in the terminal in the same working directory as the python files. This program requires at least Python 3.7 to run. Ensure that site0900.01n and site0900.01o are also in the working directory.

Results

	A	B	C
1	14424202.6864955	-7926149.83615083	20899008.74447
2	-21087800.7950015	2701056.93683356	15950307.3614751
3	-10602198.5187299	-11147561.243784	21703593.5336796
4	-17125478.1117153	-5915613.1561656	19165903.7623515
5	-19402676.9369459	-17281900.3458851	5640209.19263988
6	-5866493.97649074	-24508057.7876192	8195592.51563319
7	8342380.42363017	-21840710.9099328	11927125.1284361
8	-9770641.3029566	11920631.4268672	21732846.3166791
9	-17257273.5049653	-18385903.2063066	9118524.93318397
10	-13181178.2839277	13390840.1138291	18698219.0205599
11	14491874.9144357	-7884921.53544072	20868401.4861283
12	-21140253.6343006	2667630.53345263	15889092.4521754
13	-10526949.7479321	-11181206.1679044	21721126.7405768
14	-17059965.4784597	-5951416.46161952	19212541.1734001
15	-19380047.5241923	-17276877.7921674	5732014.75958518
16	-5856793.18853853	-24538713.0726568	8108032.99079218
17	8350216.92681111	-21794062.2143702	12007930.0952034
18	-9841227.68844057	11878824.9835394	21723309.4774161
19	-17226978.88321	-18372658.5083951	9206166.87389758
20	-13189593.7675939	13319221.3462447	18743499.483605
21	14559520.377833	-7843876.77217286	20837396.565421
22	-21192549.6469639	2634395.08900019	15827573.966142
23	-10451698.3653181	-11215020.8092436	21738246.0159147
24	-16994358.021139	-5987418.73351678	19258802.870182
25	-19357118.6489109	-17271707.8197262	5823710.89780176
26	-5847140.95054525	-24569055.0157445	8020317.11230355
27	8358132.46446373	-21747119.8825004	12088496.7758304
28	-9911857.14972214	11837159.0482648	21713360.5999631
29	-17196387.9814384	-18359293.1093709	9293637.50098215
30	-13198132.0796353	13247446.9267169	18788418.9415909
31	14627137.3229507	-7803016.01921415	20805994.5951353
32	-21244687.3230142	2601350.39899855	15765753.1338829
33	-10376446.1534322	-11249004.6579586	21754950.9657517
34	-16928657.5615554	-6023619.84506323	19304687.928476
35	-19333889.9916796	-17266391.8086376	5915295.85508939
36	-5837536.0207886	-24599082.9288251	7932446.55910247
37	8366128.21073461	-21699884.9978681	12168823.5873727
38	-9982528.05891449	11795634.3847151	21702999.8510496
39	-17165500.7475319	-18345808.5435893	9380935.2136605
40	-13206793.776755	13175518.435179	18832976.5331901
41	14694723.9944979	-7762339.73904651	20774196.1955315
42	-21296665.1559876	2568496.24939963	15703631.1912116
43	-10301194.8930131	-11283157.1937577	21771241.2038811
44	-16862865.9225666	-6060019.65852866	19350195.431482
45	-19310361.2434005	-17260931.1424267	6006767.88132391

site0900-satpos

CSV successfully written to site0900_satpos.csv. There are a total of 23209 rows in this csv. The A, B, C columns correspond to the X, Y, Z values of the satellites in ECEF coordinate system, respectively.

```
PS C:\[redacted]\Desktop\Satellite Positioning> python main.py
Successfully wrote navigation data to site0900-01n.csv
FirstObsTime(year=2001, month=3, day=31, hour=0, minute=0, second=0.0)
Successfully wrote observation data to site0900-01o.csv
Successfully wrote GPS satellite position data to site0900-satpos.csv
PS C:\[redacted]\Desktop\Satellite Positioning>
```

Console output

Appendix – Source code excerpt for orbit calculation

```python
def calculate_gps_position(nav_data: NavData, obs_data: ObsData) -> tuple[float, float, float]:
    # Time from ephemeris reference epoch (tk)
    t = obs_data.time_num - math_fn.transmission_sec(obs_data.c1)
    tk = math_fn.time_diff_k(t, nav_data.t0)

    # Mean motion (n)
    n = math_fn.mean_motion(nav_data.sqrt_a, nav_data.delta_n)

    # Mean anomaly (M)
    M = math_fn.mean_anomaly(nav_data.m0, n, tk)

    # Eccentric anomaly (E)
    E = math_fn.ecc_anomaly(M, nav_data.ecc)

    # True anomaly (θ)
    theta = math_fn.true_anomaly(E, nav_data.ecc)

    # Argument of latitude (φ)
    phi = math_fn.argument_of_latitude(theta, nav_data.omega)

    # Orbit radius of GPS satellite position (r)
    r = math_fn.orbit_radius(nav_data.sqrt_a, nav_data.ecc, E,
                             nav_data.crs, nav_data.crc, phi)

    # Corrected GPS orbit inclination (i)
    i = math_fn.inclination(nav_data.i0, nav_data.d_i, tk,
                            nav_data.cis, nav_data.cic, phi)

    # Corrected argument of latitude
    phi_c = math_fn.argument_of_latitude_corrected(nav_data.cus, nav_data.cuc, phi)

    # GPS Positions in the orbital plane (φ)
    x0, y0 = math_fn.gps_position_orbital_plane(r, phi_c)
```

```python
    # Longitude of ascending node (Ω_k)
    omega_k = math_fn.longitude_of_ascending_node(nav_data.omega_0, nav_data.d_omega, tk, nav_data.t0)

    # GPS Positions in ECEF
    x, y, z = math_fn.gps_position_ecef(x0, y0, omega_k, i)

    return x, y, z
```

Calculation workflow function

```python
def process_gps_position(obs_file):
    nav_data_list = read_nav()

    # The navigation file is always sorted by time, record the ranges of timeframes with slices
    grouped_nav_data_list, nav_slice_list = group_by_duplicates(iter(nav_data_list),
                                                                lambda nav1, nav2: nav1.time() == nav2.time())
    nav_time_gen = (nav_data.time() for nav_data in grouped_nav_data_list)
    # Maps time to slice of original list
    time_map = [(t, s) for t, s in zip(nav_time_gen, nav_slice_list)]

    sat_pos_csv_file = open(SAT_POS_CSV_FILE, 'w')

    for obs_row in obs_file:
        obs_data = ObsData.from_csv_row(obs_row)
        obs_time = obs_data.time()
        # Sort time-slice map by closest time first
        time_map.sort(key=lambda item: abs(item[0] - obs_time))
        match_nav_data = None
        for _, s in time_map:
            # Linear search within the same time frame
            match_nav_data = find_first(list_iter(nav_data_list, s), lambda nav_data: nav_data.prn == obs_data.prn)
            if match_nav_data is not None:
                break

        # No satellites with matching PRN found
        if match_nav_data is None:
            continue
        # Found a matching satellite with matching PRN, but expired
        if abs(match_nav_data.time() - obs_time) >= timedelta(hours=4):
            continue

        x, y, z = calculate_gps_position(match_nav_data, obs_data)
        sat_pos_csv_file.write(f"{x},{y},{z}\n")

    sat_pos_csv_file.close()
    print(f"Successfully wrote GPS satellite position data to {SAT_POS_CSV_FILE}")
```

The function that processes the GPS Positions

```python
def group_by_duplicates(iterator, equals_fn):
    result_list = []
    result_slices = []
    start_idx = 0
    first_item = next(iterator)  # Get the first item from the iterator
    result_list.append(first_item)  # Always append the first item
    prev_item = first_item  # Set prev_item to the first item

    for i, item in enumerate(iterator, start=1):  # Start enumeration from 1
        if not equals_fn(item, prev_item):
            result_slices.append(slice(start_idx, i))
            start_idx = i
            result_list.append(item)
            prev_item = item

    # Append the last slice for the final group
    result_slices.append(slice(start_idx, len(result_list)))

    return result_list, result_slices

def list_iter(ref_list, slice_obj):
    return (ref_list[index] for index in range(slice_obj.start, slice_obj.stop))

def find_first(iterable, predicate):
    for item in iterable:
        if predicate(item):
            return item
    return None
```

Utility functions used

```python
def main():
    read_nav()
    read_obs()
    with open(OBS_CSV_FILE, newline='') as obs_file:
        obs_file = csv.reader(obs_file)
        next(obs_file)  # exclude header from csv
        process_gps_position(obs_file)

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

Updated main function

References

Calais E. *GPS Geodesy – Lab 5 From GPS ephemerides to ECEF satellite positions.* Retrieved from https://www.geologie.ens.fr/~ecalais/teaching/gps-geodesy/lab_5.pdf

Crocetto N., Gatti M. & Perfetti N. (1997) *The GPS Single Point Positioning: A Data Processing Program for Tutorial Purposes.* ISPRS Commision VI, Working Group 3. 1-5 Retrieved from https://www.isprs.org/proceedings/XXXII/6-W1/131_XXXII-6-W1.pdf

Gurtner W. & Mader G. (1990) *RINEX: The Receiver Independent Exchange Format Version 2.* CSTG GPS Bulletin. Vol. 3. Retrieved from https://www.ngs.noaa.gov/CORS/RINEX-2.txt

Mussio L., Crippa B. & Vettore A. (1997) *International Society for Photogrammetry and Remote Sensing* ISPRS 32:1-11. Retrieved from https://www.researchgate.net/profile/Gabriella-Caroti/publication/260934448_Kinematic_GNSS_SurveyExperiences_to_be_Transferred/links/58f759d50f7e9be34b3426db/Kinematic-GNSS-SurveyExperiences-to-be-Transferred.pdf

Xu, G. & Xu, Y., *GPS: Theory, Algorithm and Applications*. Berlin, Heidelberg: Springer Berlin, 2016. Retrieved from https://link.springer.com/book/10.1007/978-3-662-50367-6

Zhao, S., Cui, X., & Lu, X., *Single point positioning using full and fractional pseudo range measurements*

from GPS and BDS, Survey review - Directorate of Overseas Surveys, vol. 53, no. 376, pp. 27–34, 2021, Retrieved from https://www.researchgate.net/publication/336977979_Single_point_positioning_using_full_and_fractional_pseudorange_measurements_from_GPS_and_BDS