GNSS Homework 2

Khushaldas Badhan

Runway 30 Location

a) Geodetic latitude, longitude, and altitude above the WGS 84 ellipsoid

The answer is calculated using a python code provided in Appendix A. The code uses Python library Pyproj to convert the Earth-Centered Earth-Fixed (ECEF) WGS84 coordinates of the starting point of a runway to geodetic coordinates, which include latitude, longitude, and altitude.

1) For this the WGS84 ECEF coordinate system and the WGS84 geodetic coordinates system are defined using CRS (coordinate reference system) class in the library. To define these coordinates systems EPSG codes for the ECEF WGS84 and geodetic WGS84 systems are used. These codes for WGS84 are 4978 and 4326 for ECEF and geodetic systems respectively. These are found using the reference [1].
2) Then the transformer is created and applied using the transform function that converts the ECEF into geodetic coordinates of the WGS84 system. The transform function can convert one type of coordinate system into another in the Pyproj library [2].

The result for the location of start of point of Runway 30 in Geodetic WGS84 are:

1) Latitude: 37.45837643292903°
2) Longitude: -122.11233899586935°
3) Altitude: -31.455705164931715 m

b) Latitude and longitude of the rwy30Start and rwy30End position in degrees, arc-minutes, and arc-seconds.

For this, we find coordinates of runway end point in WGS84 geodetic coordinates system using the code in (a). Then we use decimal to degrees, minutes, and seconds convertor as provided in Appendix B to convert the decimal angles into degrees, minutes, and seconds. The results are given below:

rwy30Start Latitude: 37 degrees, 27 arc-minutes, 30.1552 arc-seconds

rwy30Start Longitude: -122 degrees, 6 arc-minutes, 44.4204 arc-seconds

rwy30End Latitude: 37 degrees, 27 arc-minutes, 49.5447 arc-seconds rwy30End Longitude: -122 degrees, 7 arc-minutes, 3.6497 arc-seconds

c) Height of the rwy30Start position above the geoid

The Altitude of the rwy30Start is -31.455705164931715 m, and the geoid is 33 m below the ellipsoid. Therefore, the height of the runway above the geoid =

33 - -31.455705164931715 = -31.455705164931715 m.

d) ENU coordinates of the rwy30End

The answer is calculated using a Python code provided in Appendix C. The code uses the Pyproj library to convert the ECEF WGS84 coordinates of the runway endpoints to ENU coordinates relative to the runway start.

1) ECEF and geodetic WGS84 coordinate systems are defined using the CRS class with EPSG codes 4978 and 4326.

2) A transformer is created for conversion between these systems.

3) An ECEF to ENU conversion function is defined, considering the curvature and rotation of the Earth.

4) This function is used to calculate the ENU coordinates of the runway end relative to the runway start.

The result for this is as below:

East: -472.5482654582989 m, North: 597.7817327427318 m, Up: 0.005550572422743947 m.

Time conversion from local time given in yr-mo-day hr:min:sec

a) To UTC

The python code the convert local time to UTC JD (Julien Date) and MJD (Modified Julien Date) is provided in Appendix D. Here datetime library is used to parse the provided date and time. Then adjustments are made for the fact that the Julian calendar starts the year on March 1, hence the checks and adjustments to the year and month at the beginning. The variables A and B are part of the algorithm that corrects for leap years and the transition from the Julian to the Gregorian calendar. After calculating the whole-day component of the Julian Date (JD), the fractional day represented by the hour, minute, and second is added to JD.

For the MJD, 2400000.5 is subtracted from JD.

b) To GPST

The python code the convert local time to GPST is provided in Appendix E. Here datetime library is used to parse the provided date and time. Then GPS time start is defined. The difference between local time date and GPS time date is calculated using the datetime library. The total seconds are calculated for this difference. For the GPS week calculated by dividing by (7*24*3600).

c) TAI (in leap seconds)

The python code the convert local time to TAI is provided in Appendix F. Here the number of leap seconds are 37. And the local time is parsed using datetime library. The using the datetime library, the leap seconds are added to the local time to TAI.

References

[1] spatialreference.org, “https://spatialreference.org/ref/epsg/,” [Online]. [Accessed October 2023].

[2] J. Whitaker. [Online]. Available: https://pyproj4.github.io/pyproj/stable/api/transformer. [Accessed October 2023].

Appendix A.

```python
from pyproj import Transformer, CRS

# Define the WGS 84 ECEF CRS
crs_ecef = CRS.from_epsg(4978)  # WGS 84 ECEF

# Define the WGS 84 Geodetic CRS
crs_geodetic = CRS.from_epsg(4326)  # WGS 84

# Create a transformer to convert ECEF coordinates to geodetic coordinates
transformer_ecef_to_geodetic = Transformer.from_crs(crs_ecef, crs_geodetic)

# Given ECEF coordinates for the start of the runway
rwy30Start = [-2694685.473, -4293642.366, 3857878.924]

# Perform the transformation from ECEF to geodetic coordinates
lat_start,  lon_start,  alt_start  =  transformer_ecef_to_geodetic.transform(*rwy30Start,
direction='FORWARD')

# Print the results
print(f"Latitude: {lat_start} degrees")
print(f"Longitude: {lon_start} degrees")
print(f"Altitude: {alt_start} meters")

```

Appendix B.

```python
def decimal_to_dms(decimal_degrees):
    degrees = int(decimal_degrees)
    minutes_full = abs(decimal_degrees - degrees) * 60
    minutes = int(minutes_full)
    seconds = (minutes_full - minutes) * 60
    return degrees, minutes, seconds

print("lat ", decimal_to_dms(lat)) # lat = lattitude
print("lon ", decimal_to_dms(lon)) # lon = longitude
```

Appendix C.

```python
import numpy as np
from pyproj import Transformer, CRS

transformer_ecef_to_geodetic = Transformer.from_crs(CRS.from_epsg(4978), CRS.from_epsg(4326))

# Function to convert ECEF coordinates to ENU coordinates
def ecef_to_enu(x, y, z, lat_ref, lon_ref, alt_ref):
    # Convert reference point to radians
    lat_ref_rad = np.radians(lat_ref)
    lon_ref_rad = np.radians(lon_ref)

    # Reference ECEF coordinates
    x0, y0, z0 = transformer_ecef_to_geodetic.transform(lat_ref, lon_ref, alt_ref, direction='INVERSE')

    # ECEF vector
    dx = x - x0
    dy = y - y0
    dz = z - z0

    # Trigonometric shortcuts
    slat = np.sin(lat_ref_rad)
    clat = np.cos(lat_ref_rad)
    slon = np.sin(lon_ref_rad)
    clon = np.cos(lon_ref_rad)

    # ENU transformation matrix
    t = np.array([[-slon, clon, 0], [-slat*clon, -slat*slon, clat], [clat*clon, clat*slon, slat]])

    # Perform the transformation
    east, north, up = t @ np.array([dx, dy, dz])
    return east, north, up

# Define ECEF coordinates for rwy30Start and rwy30End
rwy30Start = [-2694685.473, -4293642.366, 3857878.924]
rwy30End = [-2694892.460, -4293083.225, 3858353.437]

# Get geodetic coordinates for rwy30Start
lat_start, lon_start, alt_start = transformer_ecef_to_geodetic.transform(*rwy30Start)

# Calculate ENU coordinates of rwy30End relative to rwy30Start
easting, northing, upping = ecef_to_enu(rwy30End[0], rwy30End[1], rwy30End[2], lat_start, lon_start, alt_start)

print(f"Easting: {easting} meters, Northing: {northing} meters, Upping: {upping} meters")

```

Appendix D.

```python
from datetime import datetime

# Function to convert a local time to UTC Julian Date (JD) and Modified Julian Date (MJD)
def local_time_to_utc_jd_mjd(local_time_str):
    # Convert the local time string to a datetime object
    local_time = datetime.strptime(local_time_str, '%Y-%m-%d %H:%M:%S')

    # Extract year, month, day, hour, minute, and second
    year = local_time.year
    month = local_time.month
    day = local_time.day
    hour = local_time.hour
    minute = local_time.minute
    second = local_time.second

    # Calculate Julian Date (JD) using the formula
    if month <= 2:
        year -= 1
        month += 12
    A = year // 100
    B = 2 - A + A // 4
    jd = int(365.25 * (year + 4716)) + int(30.6001 * (month + 1)) + day + B - 1524.5
    jd += (hour + minute / 60 + second / 3600) / 24

    # Calculate Modified Julian Date (MJD)
    mjd = jd - 2400000.5

    return jd, mjd

# Example usage:
local_time_str = '2023-10-29 00:00:00'
utc_jd, utc_mjd = local_time_to_utc_jd_mjd(local_time_str)
print(f"Julian Date (JD): {utc_jd}")
print(f"Modified Julian Date (MJD): {utc_mjd}")

```

Appendix E.

```python
from datetime import datetime, timedelta

# Define the function to convert local time to GPST (GPS week and seconds of the week)
def local_time_to_gpst(local_time_str):
    # GPS time starts from January 6, 1980 at 00:00 UTC
    gps_start = datetime(1980, 1, 6)

    # Convert the local time string to a datetime object assuming the string is in UTC
    local_time = datetime.strptime(local_time_str, '%Y-%m-%d %H:%M:%S')

    # Calculate the time difference between the local time and the GPS start time
    delta = local_time - gps_start

    # Calculate the total number of seconds from the GPS start time to the local time
    total_seconds = delta.total_seconds()

    # Determine the GPS week by dividing the total seconds by the number of seconds in a week
    gps_week = total_seconds // (7 * 24 * 3600)

    # Calculate the seconds into the week by taking the remainder of the total seconds divided by the number of seconds in a week
    seconds_of_week = total_seconds % (7 * 24 * 3600)

    return int(gps_week), int(seconds_of_week)

# Example usage
local_time_str = '2023-10-29 00:00:00'
gpst_week, gpst_seconds_of_week = local_time_to_gpst(local_time_str)
print(f"GPS Week: {gpst_week}")
print(f"Seconds into the GPS week: {gpst_seconds_of_week}")

```

Appendix F.

```python
from datetime import datetime, timedelta

# Function to convert local time to TAI (International Atomic Time)
def local_time_to_tai(local_time_str, leap_seconds=37):
    # Convert the local time string to a datetime object
    local_time = datetime.strptime(local_time_str, '%Y-%m-%d %H:%M:%S')

    # Add the current number of leap seconds to get TAI
    # As of the last update in April 2023, the total was 37 seconds
    tai_time = local_time + timedelta(seconds=leap_seconds)

    return tai_time

# Example usage
local_time_str = '2023-10-29 00:00:00'
tai_time = local_time_to_tai(local_time_str)

# Convert TAI time to string for display
tai_time_str = tai_time.strftime('%Y-%m-%d %H:%M:%S')
print(f"TAI time: {tai_time_str}")

```