Dane do zadania: “lot 24”

1. Cel ćwiczenia

Celem ćwiczenia jest poznanie i zrozumienia sposobu przeliczania współrzędnych geodezyjnych samolotu na współrzędne lokalne oraz stworzenie wizualizacji związanych z lotem samolotu.

2. Wstęp teoretyczny

Współrzędne geodezyjne(sferyczne) oraz ortokartezjańskie są metodami opisywania położenia punktów w przestrzeni trójwymiarowej. Powierzchnią odniesienia do współrzędnych geodezyjnych jest elipsoida obrotowa. Współrzędne geodezyjne opisują przestrzeń za pomocą trzech parametrów: długości geograficznej, szerokości geograficznej, a także odległość od środka sfery(w naszym przypadku Ziemi). Natomiast współrzędne ortokartezjańskie opisują położenie punktów za pomocą trzech wzajemnie prostopadłych osi: x,y i z.

3. Opis przebiegu ćwiczenia

Na początku zamieniamy podane w pliku flight radar wysokości ze stóp na metry, mnożąc kolejne wartości razy stałą 0.3048. Przeliczamy wysokość na elipsoidalną, dodając wysokość normalną równą 104 m oraz odstęp elipsoidy od quasigeoidy równy 31,4 m(w celu uproszczenia obliczeń).

Obliczamy promień przekroju Ziemi w kierunku wertykału I. Korzystamy ze wzoru:

$N = \frac{a}{\sqrt{1-e^2 sin^2 \varphi}}$. Występujące w tym wzorze a i e^2 to

to wielka półoś i kwadrat pierwszego mimośrodu dla elipsoidy GRS80:

a = 6 378 137 m

e^2= 0.006 694 380 022 90.

Następnie, znając wartość N przeliczamy współrzędne geodezyjne φ, λ, h na ortokartezjańskie X, Y, Z. Wykorzystujemy do tego następujące wzory:

$X = (N + h)cos\varphi cos\lambda$

$Y = (N + h)cos\varphi sin\lambda$

$Z = [N(1 - e^2) + h]sin\varphi.$

Obliczamy wektor samolot - lotnisko X_l^s we współrzędnych geocentrycznych, odejmując współrzędne lotniska od współrzędnych samolotu.

Wykorzystując poszczególne operacje na macierzach obliczamy położenie tego wektora we współrzędnych lokalnych. Wprowadzamy wektor normalny do elipsoidy (u):

Wektor normalny (u) jest definiowany jako [$cos\varphi cos\lambda$, $cos\varphi sin\lambda$, $sin\varphi$], gdzie φ i λ to współrzędne punktu lotniska (szerokość i długość geograficzna).

Obliczamy wektorów kierunkowych $\vec{n}$ i $\vec{e}$:

Obliczamy $\vec{n}$ jako [-$sin\varphi$ * $cos\lambda$, [-$sin\varphi$ * $sin\lambda$, $cos\varphi$] i $\vec{e}$ jako [$\frac{1}{cos\varphi}$), -$sin\varphi$, $cos\lambda$].

Tworzymy macierzy obrotu R_{neu}. Macierz obrotu między układem współrzędnych geocentrycznych a lokalnymi jest tworzona jako: R_{neu} = [n, e, u]

Przy pomocy macierzy obrotu zmieniamy wektor X^s_l do układu lokalnego:

$$X^s_{l_{neu}} = R_{neu}^T * X^s_l$$

wektor $X^s_{l_{neu}}$ w układzie współrzędnych lokalnych opisuje położenie samolotu względem lotniska.

4. Wykresy i wnioski

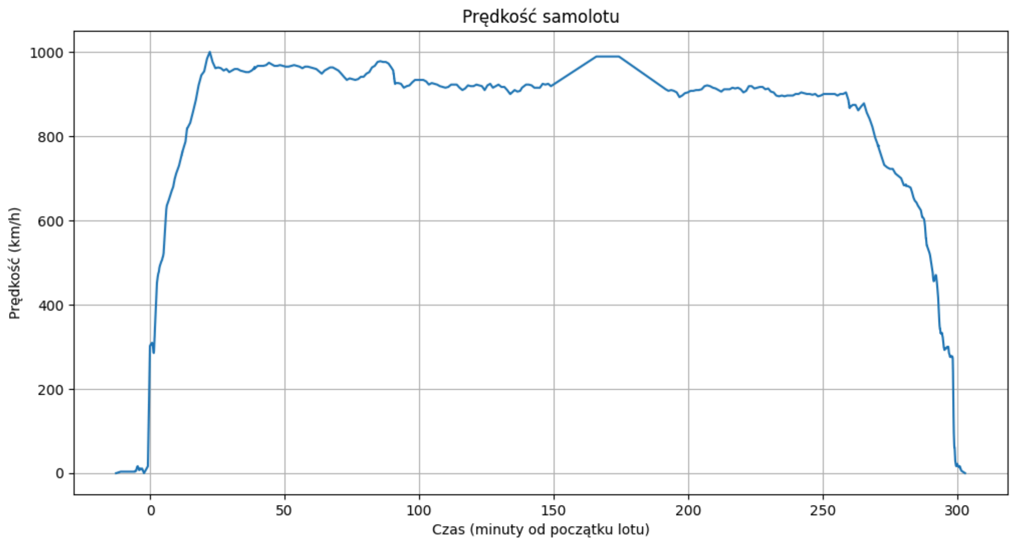
4.1 Mapa przedstawiająca trasę samolotu.

Linia czerwona - trasa samolotu
Linia niebieska - widoczność samolotu nad horyzontem
Linia pomarańczowa - linia geodezyjna (krzywa stanowiąca najkrótszą drogę pomiędzy dwoma punktami)

Wnioski:

- Samolot wystartował z lotniska “Okęcie” w Warszawie i wylądował na wschodnim wybrzeżu Kataru w porcie lotniczym “Hamad”.
- Podczas lotu przekroczył granice państw: Słowacji, Węgier, Rumunii, Bułgarii, Turcji, Iraku, Iranu, Kataru, co świadczy o międzynarodowym charakterze trasy.
- Przebieg lotu obejmował przelot nad różnorodnymi obszarami geograficznymi, w tym obszarami górskimi np. Karpatami, równinami np. Niziną Węgierską, obszarami pustymi w Iraku i Iranie, a także nad dużymi akwenami wodnymi Morzem Czarnym i Zatoką Perską.
- Trasa lotu z Warszawy do Kataru nie była najkrótszą możliwą trasą, gdyż nie pokrywała się bezpośrednio z linią geodezyjną

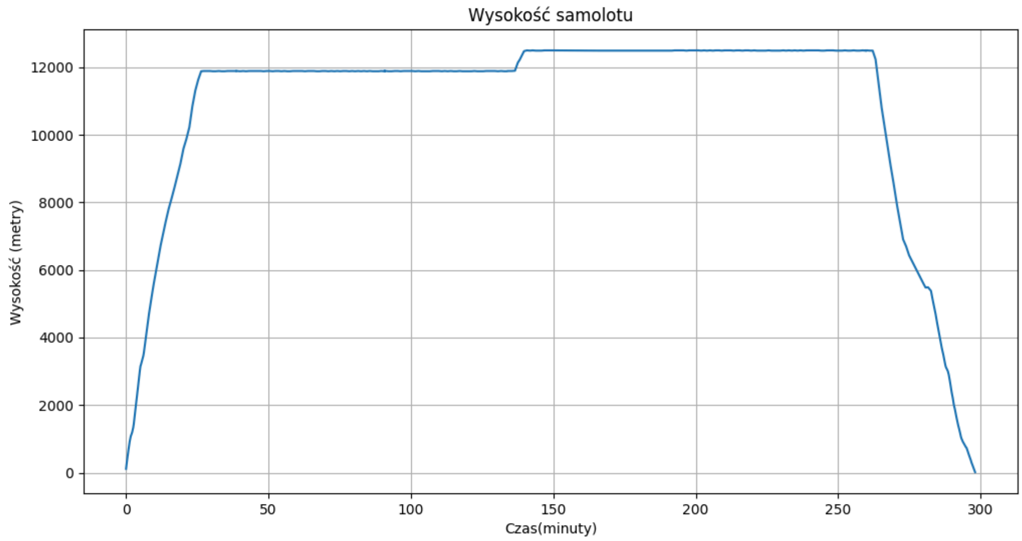
4.2 Wykres zależności prędkości samolotu od czasu



Wnioski:

- Początkowy gwałtowny wzrost prędkości po opuszczeniu lotniska sugeruje, że samolot przechodzi przez fazę startu, podczas której przyspiesza, aby osiągnąć bezpieczną prędkość do wznoszenia.
- W większej części trwania lotu przedział prędkości jest stabilny i wynosi od 900 km/h do 1000 km/h. Jest to tak zwana faza krążenia.
- Maksymalną prędkość 1000 km/h osiąga w 25 minucie od wyruszenia z lotniska.
- W ostatnich 30 min lotu prędkość samolotu gwałtownie spada. Jest to spowodowane tym, że samolot redukuje prędkość w celu bezpiecznego i kontrolowanego lądowania.

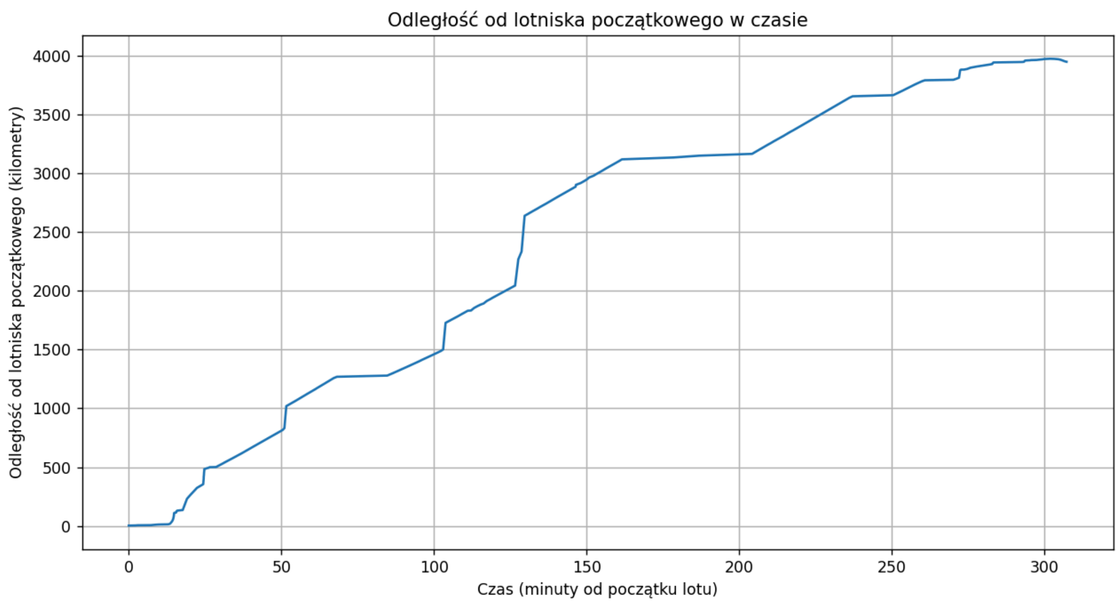
4.3 Wykres zależności wysokości samolotu od czasu.



Wnioski:

- Na początku lotu obserwujemy stały wzrost wysokości samolotu przez 30 min, co oznacza, że samolot znajduje się w fazie startu.
- Przez ponad 2 h wysokość samolotu jest stała i wynosi 11,9 km, a przez kolejne 2 h zwiększa się do 12,5 km. Zmiana wysokości może wynikać z uwarunkowań atmosferycznych oraz geograficznych.
- Podczas ostatnich 40 minut lotu wysokość samolotu się zmniejsza, co jest spowodowane fazą lądowania samolotu.

4.4 Wykres zależności odległości od lotniska początkowego od czasu.



Wnioski:

- Początkowy szybszy wzrost odległości od lotniska wynika z fazy startu, samolot przyspiesza do bezpiecznej prędkości.
- Zmienność odległości na wykresie wynika z różnych prędkości i zmian kierunku lotu samolotu.
- W ostatnich minutach odległość stabilizuje się, co świadczy o tym, że samolot zwalnia i przygotowuje się do lądowania.

5. Kod

```
from read_flightradar import read_flightradar
import numpy as np
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
from cartopy.io.img_tiles import OSM
from geopy.distance import geodesic
from datetime import datetime

plik = 'dane/lot24.csv'
dane = read_flightradar(plik)

wspolrzedne = dane[:, [7, 8, 9]]
lot = np.where(wspolrzedne[:, -1] > 0)[0]
wspolrzedne[:, -1] = wspolrzedne[:, -1] * 0.3048 + 135.4
wspolrzedne_lot = wspolrzedne[lot, :]
wspolrzedne_lotniska = wspolrzedne[lot[0] - 1, :]

#przeliczenie wspolrzednych lotniska i samolotu do ortokartezjanskich
def blh2xyz(phi, lam, h):
    phi_rad = np.deg2rad(phi)
    lam_rad = np.deg2rad(lam)
    a = 6378137
    e2 = 0.00669438002290
    N = a / (np.sqrt(1 - e2 * np.sin(phi_rad) * np.sin(phi_rad)))

    X = (N + h) * np.cos(phi_rad) * np.cos(lam_rad)
    Y = (N + h) * np.cos(phi_rad) * np.sin(lam_rad)
    Z = (N * (1 - e2) + h) * np.sin(phi_rad)
    return np.array([X, Y, Z]) #w radianach

xyz_lotniska = blh2xyz(np.deg2rad(wspolrzedne_lotniska[0]),
                       np.deg2rad(wspolrzedne_lotniska[1]), np.deg2rad(wspolrzedne_lotniska[2]))

def Rneu(phi, lam):
    phi_rad = np.deg2rad(phi)
    lam_rad = np.deg2rad(lam)
    macierz = np.array([[-np.sin(phi_rad) * np.cos(lam_rad), -np.sin(lam_rad), np.cos(phi_rad) * np.cos(lam_rad)],
                        [-np.sin(phi_rad) * np.sin(lam_rad), np.cos(lam_rad), np.cos(phi_rad) * np.sin(lam_rad)],
                        [np.cos(phi_rad), 0, np.sin(phi_rad)]])
    return macierz

R = Rneu(wspolrzedne_lotniska[0], wspolrzedne_lotniska[1])
def generate_speed_plot(data):
    time = data[:, 0]
    ground_speed_knots = data[:, 10]
    ground_speed_kmph = ground_speed_knots * 1.852

    # Przeliczenie czasu na minuty od początku lotu
    start_time = time[lot[0]]
    elapsed_time_minutes = (time - start_time) / 60

    # Tworzenie wykres prędkości w czasie
    plt.figure(figsize=(12, 6))
    plt.plot(elapsed_time_minutes, ground_speed_kmph)
    plt.title("Prędkość samolotu")
    plt.xlabel("Czas (minuty od początku lotu)")
    plt.ylabel("Prędkość (km/h)")
    plt.grid(True)
    plt.show()

def calculate_distance(lat1, lon1, lat2, lon2):
    point1 = (lat1, lon1)
    point2 = (lat2, lon2)
    return geodesic(point1, point2).meters

def odleglos_od_lotniska_od_czasu(data):
    start_point = (wspolrzedne_lotniska[0], wspolrzedne_lotniska[1])
    distances = calculate_distances(wspolrzedne_lot, start_point)
    # Przekształć timestamp na obiekt datetime
    dt_object = datetime.fromtimestamp(Timestamp[0])
    # Dostosowanie formatu czasu
    formatted_time = dt_object.strftime("%Y-%m-%d %H:%M:%S")
    # Tworzenie wykresu
    plt.figure(figsize=(12, 6))
    plt.plot(czas_trwania_lotu, distances)
    plt.title("Odległość od lotniska początkowego w czasie")
    plt.xlabel("Czas (minuty od początku lotu)")
    plt.ylabel("Odległość od lotniska początkowego (kilometry)")
    plt.grid(True)
    plt.show()
```

```
def generate_altitude_plot(data):
    altitudes = []
    time = data[:, 0]
    start_time = time[lot[0]]
    for i in lot:
        altitude_feet = data[i, 9]
        altitude_kilometers = altitude_feet * 0.3048
        elapsed_time_minutes = (time[i] - start_time) / 60
        altitudes.append((elapsed_time_minutes, altitude_kilometers))
    time_minutes, altitude_meters = zip(*altitudes)
    plt.figure(figsize=(12, 6))
    plt.plot(time_minutes, altitude_meters)
    plt.title("Wysokość samolotu")
    plt.xlabel("Czas(minuty)")
    plt.ylabel("Wysokość (metry)")
    plt.grid(True)
    plt.show()

def calculate_distances(wspolrzedne_lot, lotnisko):
    distances = [geodesic(lotnisko, (lat, lon)).kilometers for lat, lon, alt in wspolrzedne_lot]
    return distances

azymuty = []
długosci_geograficzne = []
szerokosci_geograficzne = []
```

```
for flh in wspolrzedne_lot:
    xyz_samolotu = blh2xyz(np.deg2rad(flh[0]), np.deg2rad(flh[1]), np.deg2rad(flh[2]))
    wektor_samolot_lotnisko = np.array([xyz_lotniska[0], xyz_lotniska[1], xyz_lotniska[2]])
    neu = R.T.dot(xyz_samolotu-wektor_samolot_lotnisko)
    az = np.arctan2(neu[1], neu[0])
    if az < 0:
        az += 2*np.pi
    azymuty.append(az)
    długosci_geograficzne.append(flh[1])
    szerokosci_geograficzne.append(flh[0])
request = OSM()
print(azymuty)

fig = plt.figure(figsize=(10, 5))
ax = plt.axes(projection=request.crs)
extent = [10, 80, 10, 50]
ax.set_extent(extent)
ax.add_image(request, 5)
start_point =(wspolrzedne_lotniska[0], wspolrzedne_lotniska[1])
end_point = (25.273396,51.614727)
num_points = 100
lats = np.linspace(start_point[0], end_point[0], num_points)
longs = np.linspace(start_point[1], end_point[1], num_points)
line_coordinates = list(zip(lats, longs))
#tworzenie linii geodezyjnej
ax.plot(longs, lats, transform=ccrs.PlateCarree(), color='orange', linewidth=2)

colors = []

for  flh,az in zip(wspolrzedne_lot, azymuty):
    if az <= np.pi:
        color = 'red'
    else:
        color = 'blue'

    colors.append(color)


ax.plot(długosci_geograficzne, szerokosci_geograficzne, color='red',
         transform=ccrs.PlateCarree(), linewidth=2, zorder=1)
blue_points = [idx for idx, color in enumerate(colors) if color == 'blue']
#tworzenie linii przedstawiającej widoczność samolotu nad horyzontem
ax.scatter(np.array(długosci_geograficzne)[blue_points],
            np.array(szerokosci_geograficzne)[blue_points], c='blue', transform=ccrs.PlateCarree(), s=2, zorder=2)
initial_lat = wspolrzedne_lotniska[0]
initial_lon = wspolrzedne_lotniska[1]
#inicjalizacja współrzędnych początkowych i końcowych
initial_coordinates = (initial_lat, initial_lon)
final_lat = dane[-1, 3]
final_lon = dane[-1, 4]
final_coordinates = (final_lat, final_lon)
#obliczenie dystansu między lotniskami
distance_between_airports = geodesic(initial_coordinates, final_coordinates).meters
Timestamp = dane[:, 0]
distances = calculate_distances(wspolrzedne_lot, start_point)
czas_trwania_lotu = (Timestamp - Timestamp[0]) / 60
czas_trwania_lotu = czas_trwania_lotu[:len(distances)]
last_blue_point_index = blue_points[-1]
last_blue_point_coordinates = wspolrzedne_lot[last_blue_point_index]
last_blue_point_time = Timestamp[last_blue_point_index]

#wyswietlenie wykresów
generate_speed_plot(dane)
generate_altitude_plot(dane)
odleglos_od_lotniska_od_czasu(dane)

plt.show()
```