SPRAWOZDANIE

Geometria elipsoidy/ układ neu
ćwiczenie 2

Izabella Kaim 319193

1. Cel ćwiczenia

Celem ćwiczenia było przedstawienie trasy lotu samolotu na podstawie danych pobranych z portalu *flightradar24.com*. Należało przeliczyć współrzędne geodezyjne najpierw do układu współrzędnych ortokartezjańskich, a następnie do układu współrzędnych horyzontalnych, względem znanego położenia lotniska startowego. Należało również wyznaczyć moment, w którym samolot zniknie poniżej horyzontu. Wyniki należało zwizualizować na różnych wykresach.

2. Dane

Nazwa pliku z danymi: W61539_2dde28b4.csv

Początek trasy: Warszawa 52.179287, 20.962482
Koniec trasy: Reykjavik 63.992825, -22.622004

Czas podróży: ok. 4h 21m

3. Wykonanie

Wczytanie pliku csv za pomocą funkcji read_flightradar()

```
def read_flightradar(file):
    with open(file, 'r') as f:
        i = 0
        size= []
        Timestamp = []; date = []; UTC = []; Latitude = []; Longitude = [];
        Altitude = []; Speed = []; Direction = []
        for linia in f:
            if linia[0:1]!='T':
                splited_line = linia.split(',')
                size.append(len(splited_line))
                i+=1
                Timestamp.append(int(splited_line[0]))
                full_date = splited_line[1].split('T')
                date.append(list(map(int,full_date[0].split('-'))))
                UTC.append(list(map(int, full_date[1].split('Z')[0].split(':'))))
                Callsign = splited_line[2]
                Latitude.append(float(splited_line[3].split('"')[1]))
                Longitude.append(float(splited_line[4].split('"')[0]))
                Altitude.append(float(splited_line[5]))
                Speed.append(float(splited_line[6]))
                Direction.append(float(splited_line[7]))
    all_data = np.column_stack((np.array(Timestamp), np.array(date), np.array(UTC),
                                np.array(Latitude), np.array(Longitude), np.array(Altitude),
                                np.array(Speed), np.array(Direction)))

    return all_data, i
```

Funkcja przeliczająca współrzędne φ, λ, h na współrzędne w układzie ortokartezjańskim X, Y, Z

```
def geo2xyz(lat, lon, h):
    a = 6378137
    e2 = 0.00669438002290
    N = a / (m.sqrt(1 - e2*np.sin(lat)*np.sin(lat)))
    x = (N+h)*np.cos(lat)*np.cos(lon)
    y = (N+h)*np.cos(lat)*np.sin(lon)
    z = (N*(1-e2)+h)*np.sin(lat)
    xyz = [x, y, z]
    return np.array(xyz)
```

Przetworzenie współrzędnych samolotu i lotniska na osobne tablice array

```
def na_array(Warszawa, data):
    f_w = np.deg2rad(Warszawa[7])
    l_w = np.deg2rad(Warszawa[8])
    h_w = Warszawa[9] * 0.3048 + 135.4

    wspWarszawa = np.array(geo2xyz(f_w, l_w, h_w))

    wspSamolot = np.array([geo2xyz(dms2rad(deg2dms(data[i][7])),
                                   dms2rad(deg2dms(data[i][8])),
                                   data[i][9]*0.3048 + hel) for i in range(len(data))])

    return wspWarszawa, wspSamolot
```

Funkcja obliczająca wektor samolot-lotnisko we współrzędnych ortokartezjańskich

```
def wektorsamolotu(wspSamolot, wspWarszawa):
    wektor = wspSamolot - wspWarszawa
    return wektor
```

Funkcja przeliczająca współrzędne do współrzędnych lokalnych NEU

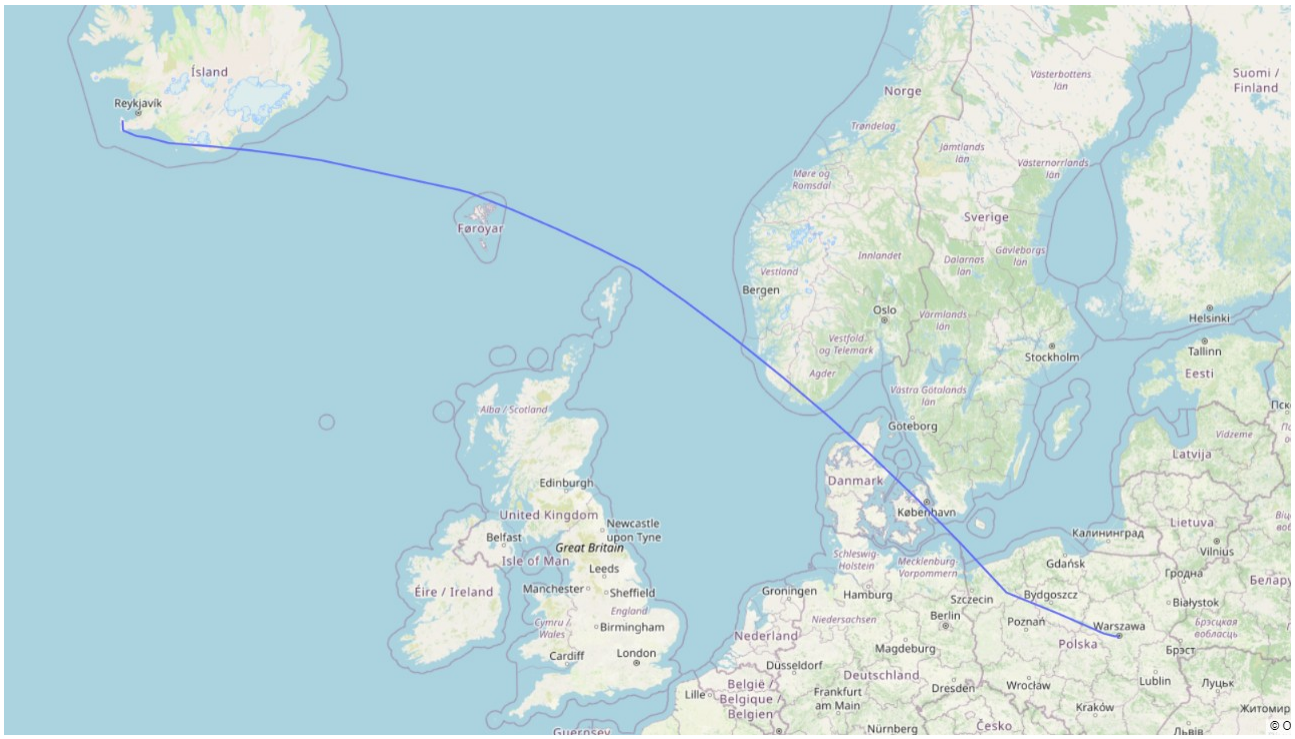
```
def NEU(f_w, l_w):
    n = np.array([
        -m.sin(f_w)*m.cos(l_w),
        -m.sin(f_w)*m.sin(l_w),
        m.cos(f_w)])
    e = np.array([
        -m.sin(f_w),
        m.cos(f_w),
        0 ])
    u = np.array([
        m.cos(f_w)*m.cos(l_w),
        m.cos(f_w)*m.sin(l_w),
        m.sin(f_w)])

    return [n,e,u]
```

Funkcje pośrednie:

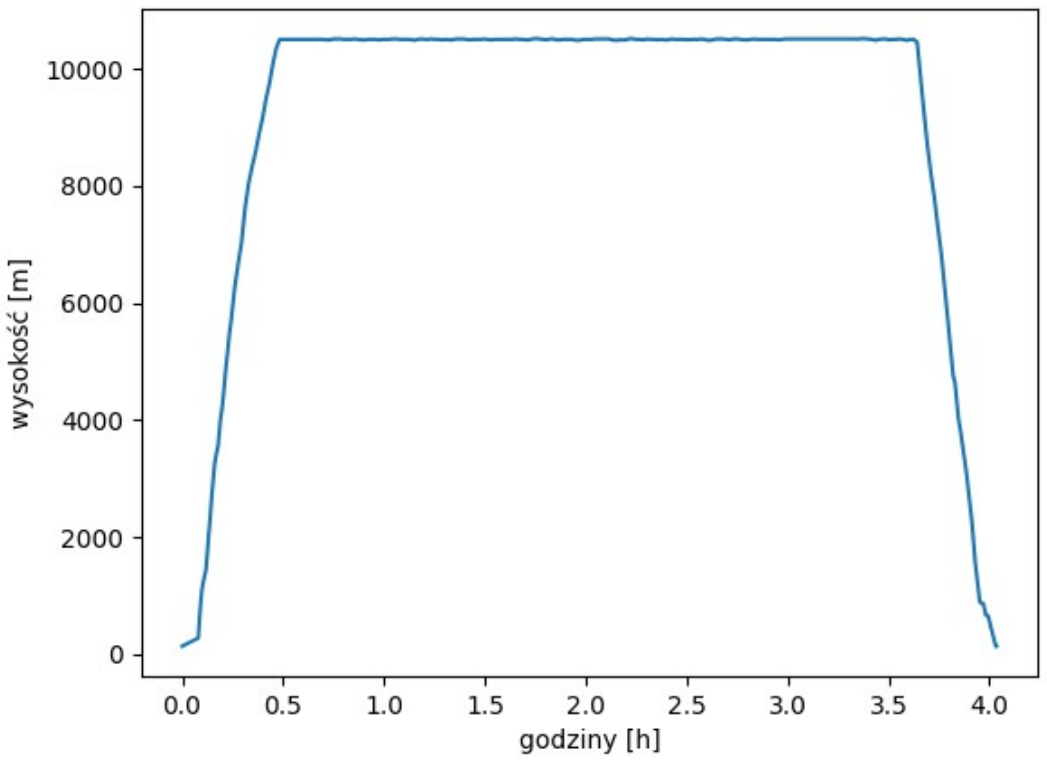
```
def RTneu(n,e,u):
    Rneu = np.column_stack((n,e,u))
    RTneu = Rneu.transpose()
    return RTneu

def xslneu(RTneu, wektor):
    xslneu = []
    for i in range(len(wektor)):
        xslneu.append(RTneu @ wektor[i])
    xn = []
    xe = []
    xu = []
    for i in range(len(xslneu)):
        xn.append(np.array(xslneu[i][0]))
        xe.append(np.array(xslneu[i][1]))
        xu.append(np.array(xslneu[i][2]))
    return [xn, xe, xu]

def dl_wektora(xn, xe, xu):
    s = np.sqrt(np.square(xn) + np.square(xe) + np.square(xu))
    return s

def azymut(xe, xn):
    Az = np.array(np.arctan2(xe,xn))
    Az[Az<0] = Az[Az<0] + 2 * m.pi
    return Az

def wys(xn, xe, xu):
    h = np.array(np.arcsin(xu/dl_wektora(xn, xe, xu)))
    return h
```

4. Wyniki

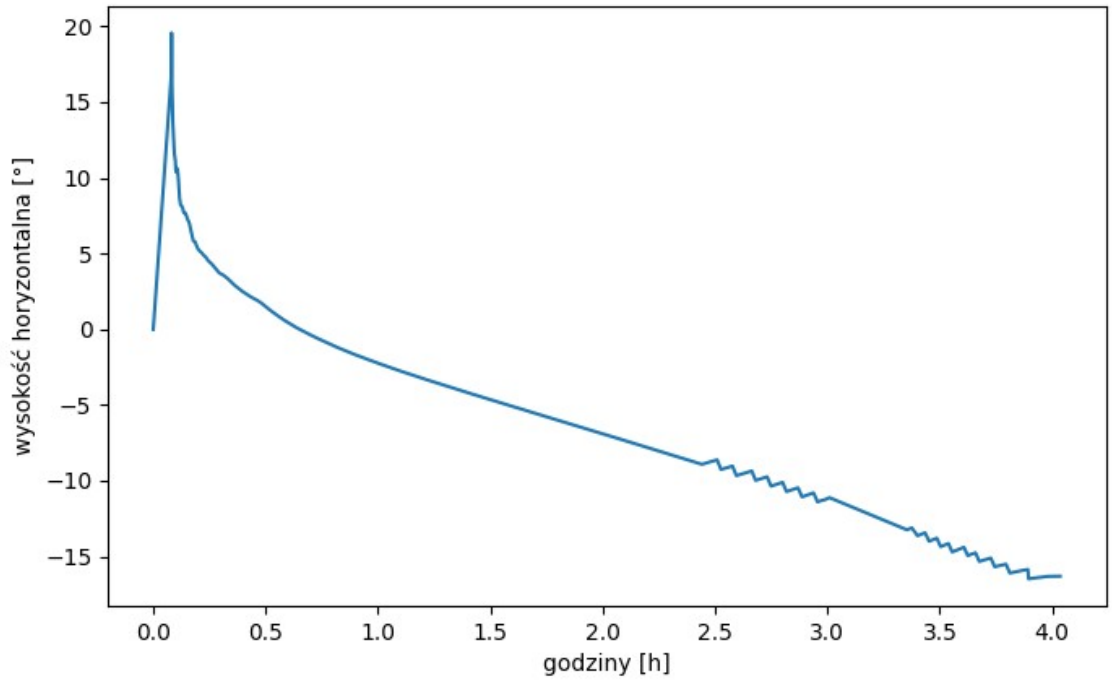
Trasa samolotu Warszawa - Reykjavik



Wykres zależności wysokości samolotu od czasu

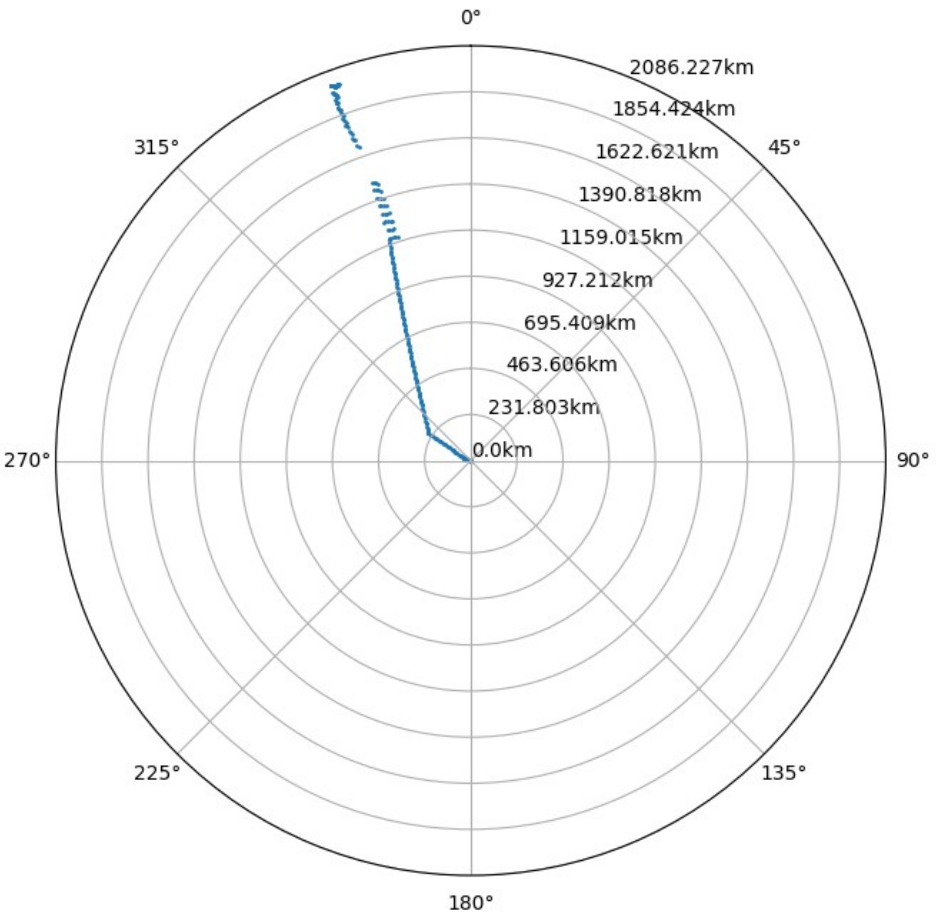


Wykres zależności wysokości horyzontalnej od czasu



Samolot zniknie za linią horyzontu na współrzędnych:
φ = 53°29'43.8"
λ = 15°52'30.3"
po około 40 minutach lotu.

Wykres skyplot – kierunek i trasa samolotu



Wykres zależności odległości samolot-lotnisko od czasu