SPRAWOZDANIE Z ĆWICZENIA 2:

Transformacja współrzędnych elipsoidalnych samolotu do układu lokalnego

Dane nr 15

Maja Kret
325693
Wydział Geodezji i Kartografii
Politechnika Warszawska

Warszawa, 15 listopada 2023

Spis treści

1 Cel ćwiczenia **2**

2 Wstęp teoretyczny **2**

3 Dane do ćwiczenia **2**

3.1 Dane lotu 2

3.2 Dane elipsoidy 3

4 Przebieg ćwiczenia **3**

5 Wyniki ćwiczenia **4**

5.1 Mapa lotu 4

5.2 Wykresy Skyplot dla lotniska WAW 5

5.3 Wykresy Skyplot dla lotniska OSL 6

5.4 Wykresy w funkcji czasu 7

6 Kod programu **9**

1 Cel ćwiczenia

Celem ćwiczenia jest transformacja współrzędnych elipsoidalnych lecącego samolotu do układu lokalnego. Następnie należy zwizualizować trasę lotu na mapach i wykresach współrzędnych.

2 Wstęp teoretyczny

Współrzędne elipsoidalne są powszechnie stosowane w geodezji, reprezentując położenie punktu na powierzchni elipsoidy odniesienia za pomocą szerokości (φ) i długości (λ) geograficznej oraz wysokości (h) nad powierzchnią elipsoidy. W kontekście lotnictwa, te współrzędne pozwalają na precyzyjne określenie lokalizacji samolotu w danym momencie lotu. Przekształcenie tych danych do układu lokalnego, związanego z konkretnym punktem na Ziemi - w tym przypadku lotniskiem - umożliwia lepsze zrozumienie względnej pozycji samolotu w odniesieniu do lokalizacji lotniska, a co za tym idzie, stwierdzenie, czy samolot znajduje się w zasięgu widzenia. W przypadku, gdy kąt elewacji jest ujemny, samolot znajduje się poniżej horyzontu i nie jest widoczny z lotniska.

3 Dane do ćwiczenia

3.1 Dane lotu

Danymi do ćwiczenia jest plik `lot15.csv` zawierający dane o locie. Analizowany był lot o numerze LOT4YH - LO483 z Warszawy do Oslo. Każdy wiersz pliku zawiera następujące dane:

Indeks	Nazwa	Jednostka
0	Znacznik czasu	
1	rok	
2	miesiąc	
3	dzień	
4	godzina	
5	minuta	
6	sekunda	
7	Szerokość geograficzna [φ]	°
8	Długość geograficzna [λ]	°
9	Wysokość [h]	stopy
10	Prędkość	kts
11	Kierunek	°

Tabela 1: Dane samolotu

3.2 Dane elipsoidy

Kod źródłowy 1: Dane elipsoidy

```
h_norm_waw = 104
h_norm_osl = 208
undulacja = 31.4
a = 6378137.0
e2 = 0.00669438002290
```

gdzie h_norm to wysokość normalna dla danego lotniska, a a i $e2$ to parametry elipsoidy GRS80

4 Przebieg ćwiczenia

1. **Wczytanie danych z pliku:** Dane dotyczące badanego lotu zostały wczytane z pliku `lot15.csv` za pomocą funkcji `read_flightradar`.

2. **Selekcja danych lotu:** Wydobyto dane, zebrane podczas, gdy samolot znajdował się ponad poziomem Ziemi - miał dodatnią wysokość. Interesują nas współrzędne samolotu, jego prędkość i wysokość oraz dokładna godzina, o której te parametry zostały zarejestrowane. Następnie wyselekcjonowano współrzędne lotnisk odlotu i przylotu. Do wysokości dodano też wysokość normalną dla danego lotniska oraz undulację. Undulację dla Oslo pominięto. Prędkość przeliczono z węzłów na $\frac{km}{h}$. A wysokość samolotu z stóp na metry.

3. **Przeliczanie współrzędnych lotniska i samolotu do współrzędnych ortokartezjańskich:** Funkcja `orto` z kodu źródłowego 3 przyjmuje jako argumenty szerokość $[\varphi]$ i długość $[\lambda]$ geograficzną w radianach oraz wysokość $[h]$ w metrach. Następnie przelicza i zwraca współrzędne ortokartezjańskie $[X, Y, Z]$.

4. **Transformacja wektora współrzędnych samolotu do współrzędnych lokalnych:** Transformacja odbywa się za pomocą macierzy obrotu R tworzonej w funkcji `macierz_obrotu` przyjmującej jako argumenty szerokość $[\varphi]$ i długość $[\lambda]$ geograficzną w radianach. Funkcja znajduje się w kodzie źródłowym 3.

5. **Obliczenia azymutu i elewacji:** W pętli opisanej w kodzie źródłowym 5, dla każdej z pozycji samolotu obliczono azymut i elewację względem lotnisk oraz odległość od lotniska. Dane zapisano w tablicach `data_waw` i `data_osl` wraz z czasem.

6. **Wizualizacje:** Do przygotowania wykresów wykorzystano bibliotekę `matplotlib`, a do map lotu - `plotly` oraz `cartopy`. Wykonano mapy trasy samolotu 1 i 2 oraz wykresy 3 - 6 pokazujące znikanie i pojawianie się samolotu nad horyzontem. Następnie opracowano wykresy 7 - 10: wysokości, prędkości, kąta elewacji i odległości samolotu od lotniska w funkcji czasu.

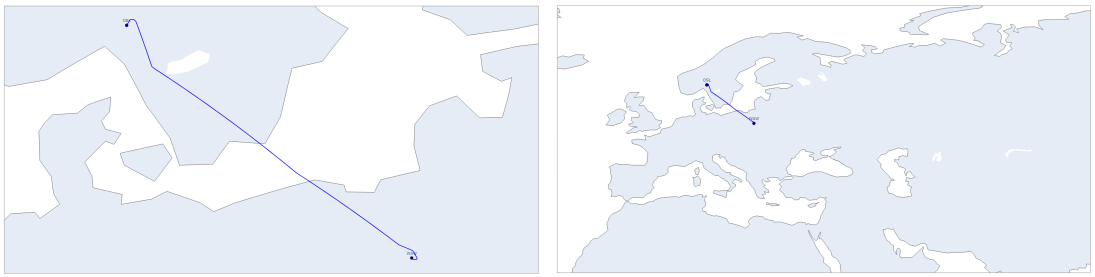
7. **Opracowanie wyników** Na podstawie obliczonych wartości stwierdzono w jakich odległościach od lotniska samolot był widoczny nad horyzontem, a także w jakich godzinach znikał i się pojawiał. Zebrano także informacje o maksymalnej wysokości, prędkości.

5 Wyniki ćwiczenia

5.1 Mapa lotu

Wykres 1: Linia lotu

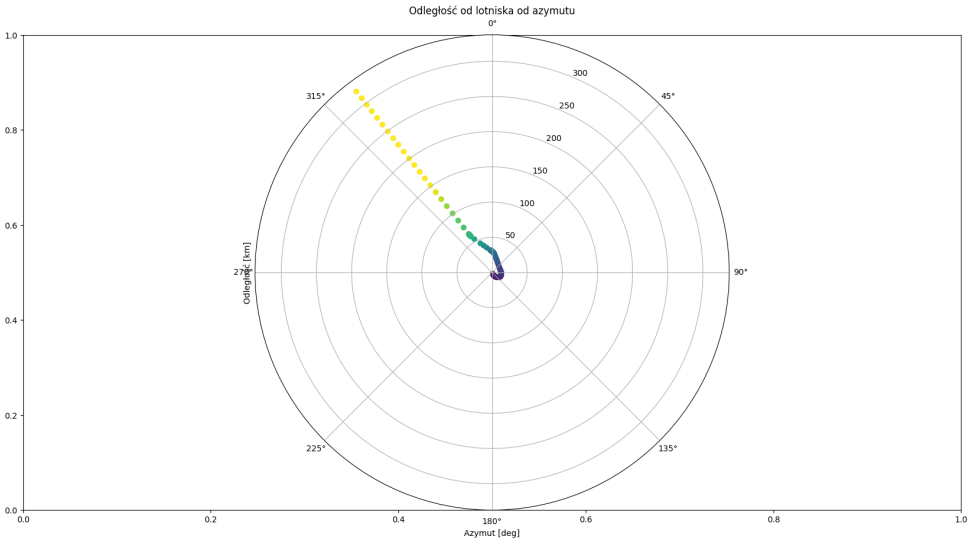
Mapa 1 przedstawia linię geodezyjną pomiędzy lotniskami przylotu i odlotu będąca odwzorowaniem idealnego przebiegu trasy. Trasa lotu ma długość 1076.99 km, a czas przelotu wynosi 1h 45min.



Wykres 2: Mapa lotu

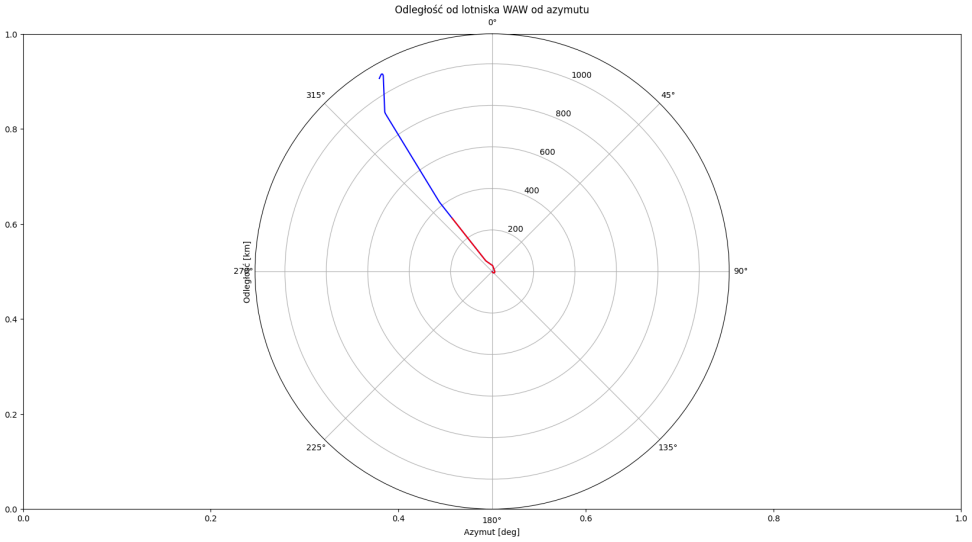
Mapa nr 2 przedstawia trasę samolotu stworzoną z kolejnych zarejestrowanych lokalizacji samolotu.

5.2 Wykresy Skyplot dla lotniska WAW



Wykres 3: Wykres odległości samolotu od azymutu z lotniska WAW

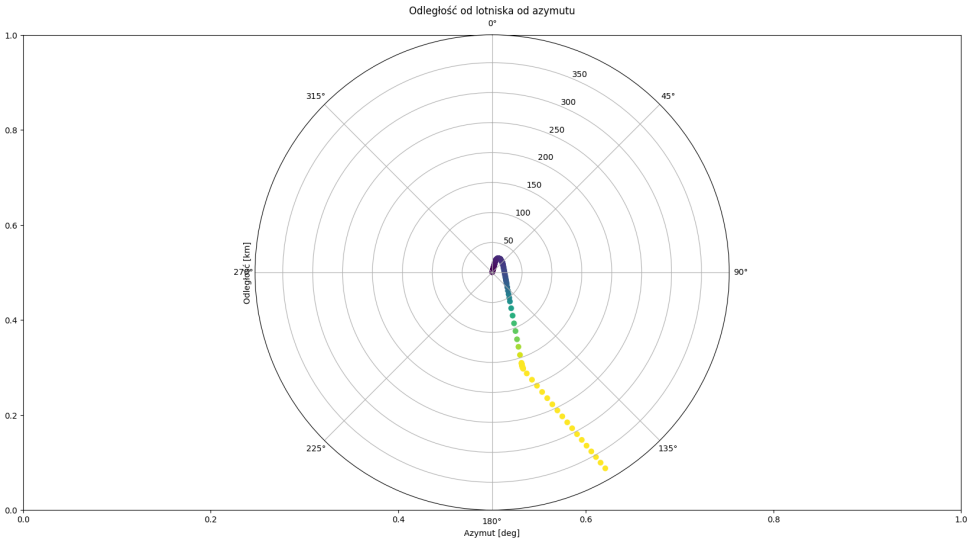
Wykres 3 przedstawia kolejne odległości samolotu w funkcji azymutu aż do momentu zniknięcia poniżej horyzontu o godzinie 15:48 w odległości 321.52 km od lotniska. W tym momencie kąt elewacji samolotu schodzi poniżej 0°.



Wykres 4: Wykres odległości samolotu od azymutu z lotniska WAW w postaci linii

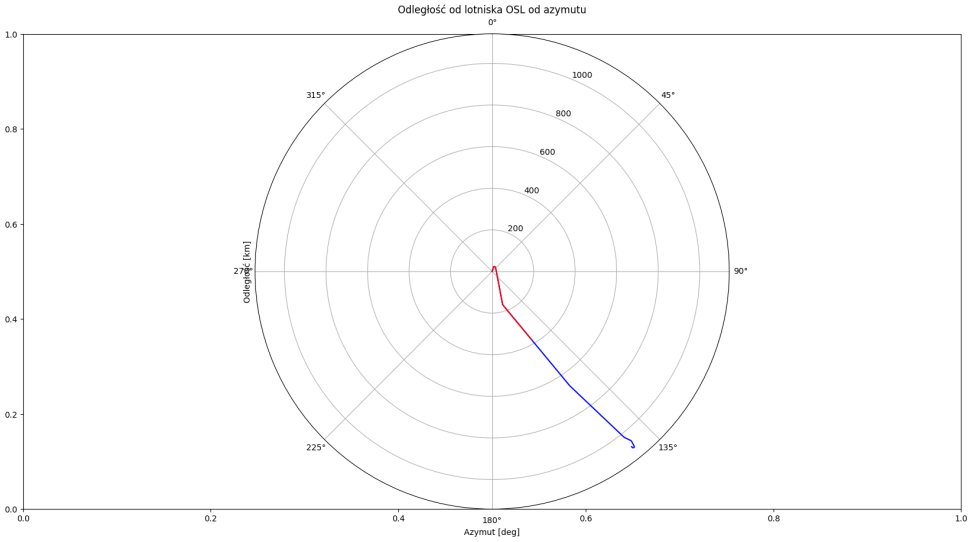
Na wykresie 4 również zaznaczona jest zależność odległości samolotu od azymutu z lotniska Warszawie. Zawarte są tu dane z całego lotu, a interesują nas są te zaznaczone na czerwono - od momentu startu do zniknięcia samolotu za horyzontem.

5.3 Wykresy Skyplot dla lotniska OSL



Wykres 5: Wykres odległości samolotu od azymutu z lotniska OSL

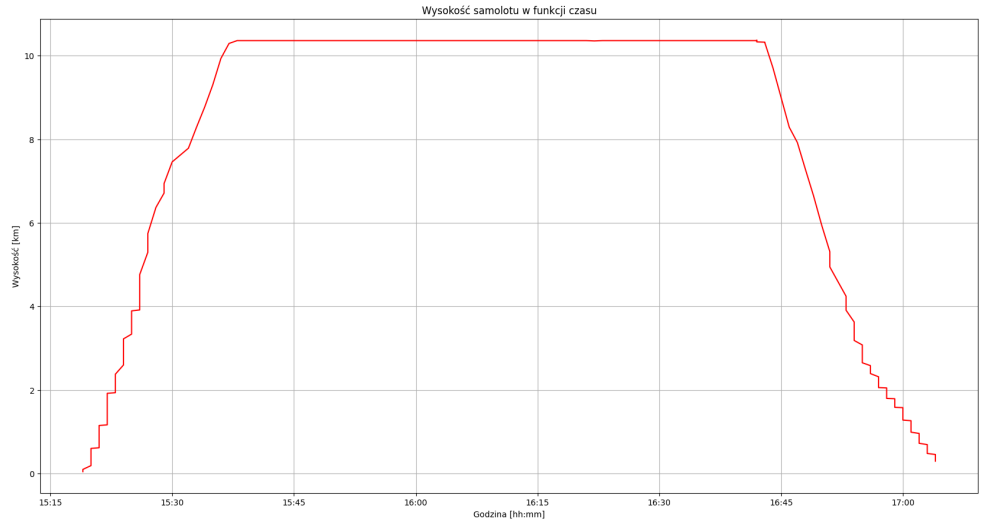
Wykres 5 pokazuje odległości samolotu od lotniska OSL w funkcji azymutu, w momencie gdy samolot znajduje się powyżej horyzontu. Pojawienie się samolotu następuje o godzinie 16:24 w odległości 365.43 km od lotniska. Jest to pierwsza zarejestrowana odległość od lotniska, gdzie wartość kąta elewacji jest dodatnia.



Wykres 6: Wykres Skyplot odległości samolotu od azymutu z lotniska OSL w postaci linii

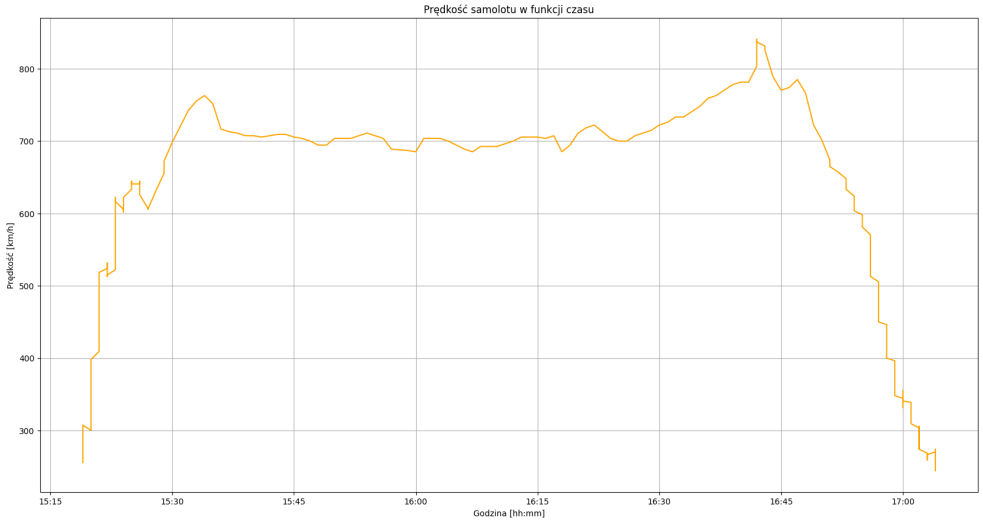
Powyższy wykres 6 przedstawia te same dane w postaci ciągłej linii reprezentującej całą trasę samolotu. Wyselekcjonowano tu na czerwono momenty, gdy samolot był widoczny z Oslo.

5.4 Wykresy w funkcji czasu



Wykres 7: Wykres zależności wysokości samolotu od czasu

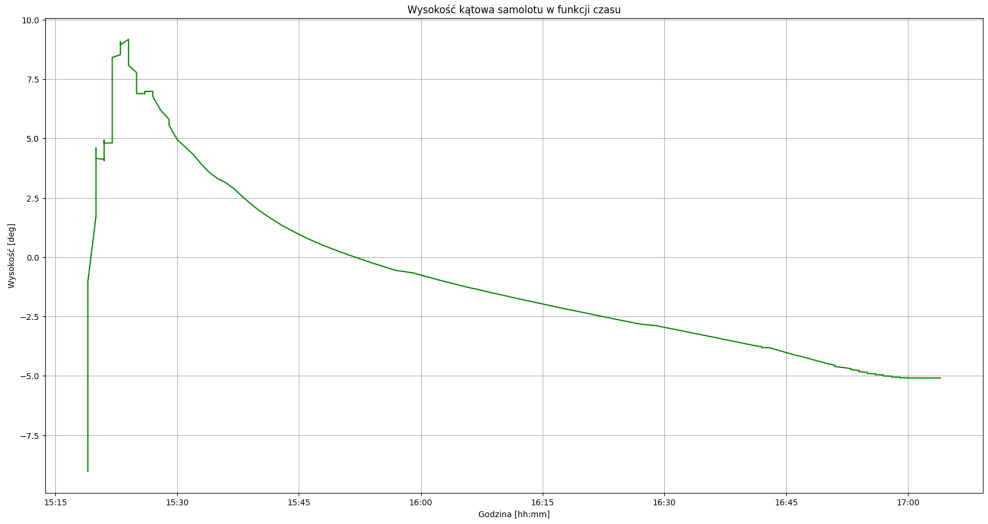
Na wykresie 7 zobrazowano wysokości samolotu od chwili startu o 15:19 do lądowania o 17:04. Nie zawarto tu danych o samolocie podczas gdy przebywał na ziemi i wysokość wynosiła 0 m. Po osiągnięciu maksymalnej wysokości 10.370 km, samolot pozostaje na tym poziomie większość lotu, aż do momentu hamowania przed lądowaniem.



Wykres 8: Wykres zależności prędkości samolotu od czasu

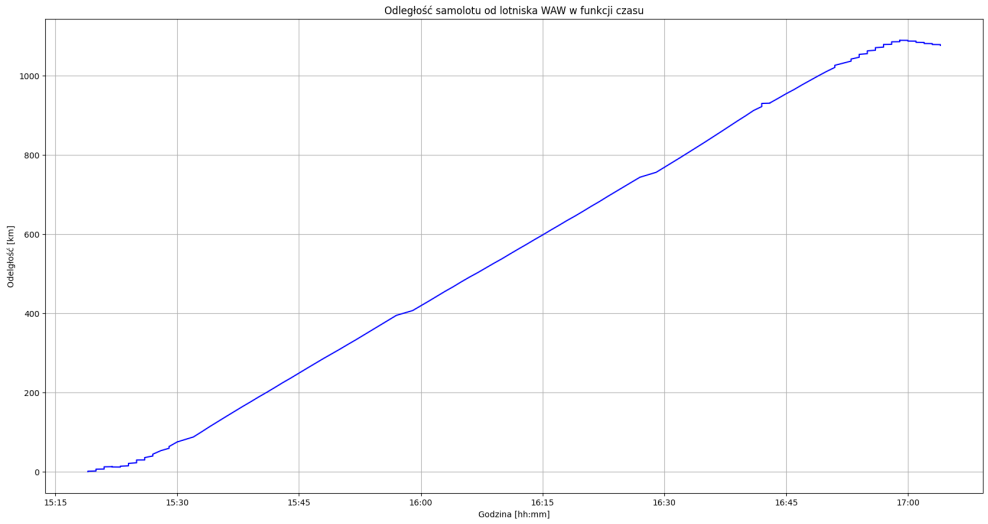
Wykres 8 zawiera zarejestrowane prędkości samolotu. Podczas wznoszenia się samolotu, w przeciągu 15 min, prędkość samolotu rośnie z $300\frac{km}{h}$ - podczas startu aż do $750\frac{km}{h}$ - w momencie osiągnięcia wysokości ponad 10 km. Następnie prędkość spada do $700\frac{km}{h}$ i utrzymuje się na wysokim poziomie do 20 minut przed lądowaniem. Wraz z zmniejszeniem prędkości maleje też wysokość samolotu. Największa zarejestrowana

prędkość wynosi aż $840.808 \frac{km}{h}$, natomiast średnia z całego lotu to 615.135 $\frac{km}{h}$.



Wykres 9: Wykres zależności wysokości kątowej samolotu od czasu z lotniska WAW

Kąt elewacji, którego wartości przedstawiono na wykresie 9 maleje wraz z czasem. Można doczytać się przybliżonej godziny zniknięcia samolotu poniżej horyzontu gdy wykres osiąga wartość 0°.



Wykres 10: Wykres zależności odległości samolotu od lotniska WAW od czasu

Wykres 10 przedstawia zmianę odległość samolotu od lotniska odlotu. Jest ona stale rosnąca w czasie.

6 Kod programu

Kod źródłowy 2: Funkcja wczytywania danych udostępniona przez prowadzącego

```
def read_flightradar(file):
    with open(file, 'r') as f:
        i = 0
        size= []
        Timestamp = []; date = []; UTC = []; Latitude = []; Longitude = [];
        Altitude = []; Speed = []; Direction = []; datetime_date = []
        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
```

Kod źródłowy 3: Funkcje transformacji współrzędnych

```
# Funkcja przeliczająca współrzędne geograficzne na współrzędne ortogonalne
def orto(p, l, h):
    N = a / (np.sqrt(1 - e2 * np.sin(l) * np.sin(l)))
    X = (N + h) * np.cos(p) * np.cos(l)
    Y = (N + h) * np.cos(p) * np.sin(l)
    Z = (N * (1 - e2) + h) * np.sin(p)
    return([X, Y, Z])

# Funkcja obliczająca macierz obrotu
def macierz_obrotu(p, l):
    macierz = np.array([[-np.sin(p) * np.cos(l), -np.sin(l), np.cos(p) * np.cos(l)],
                        [-np.sin(p) * np.sin(l), np.cos(l), np.cos(p) * np.sin(l)],
                        [np.cos(p), 0, np.sin(p)]])
    return macierz
```

Kod źródłowy 4: Selekcja danych

```
# Współrzędne
wspolrzedne = dane[:, 7:10]
lot = np.where(wspolrzedne[:, -1]>0)[0]
wspolrzedne[:, -1] = wspolrzedne[:, -1] * 0.3048
wspolrzedne_lot = wspolrzedne[lot, :]
wspol_waw = wspolrzedne[lot[0]-1, :]
wspol_osl = wspolrzedne[lot[-1]+1, :]

# Czas
time_hms = dane[:, 4:7]
time_hms_lot = time_hms[lot, :]
time = time_hms[:, 0] * 60 + time_hms[:, 1] + time_hms[:, 2] / 60
time_lot = time[lot]
time_lot = time_lot - time_lot[0]

# Start i lądowanie
take_off = datetime.datetime(int(dane[lot[0], 1]), int(dane[lot[0], 2]), int(dane[lot
    [0], 3]), int(dane[lot[0], 4]), int(dane[lot[0], 5]), int(dane[lot[0], 6]))
landing = datetime.datetime(int(dane[lot[-1], 1]), int(dane[lot[-1], 2]), int(dane[lot
    [-1], 3]), int(dane[lot[-1], 4]), int(dane[lot[-1], 5]), int(dane[lot[-1], 6]))
flight_time = landing - take_off

# Oś czasu dla lotu
datetime_time_lot = [(take_off + datetime.timedelta(minutes = i)).strftime('%H:%M') for
     i in time_lot]
datetime_time_lot = np.array(datetime_time_lot)
datetime_time_lot = pd.to_datetime(datetime_time_lot)

# Prędkość
speed = dane[:, 10] * 1.852
speed_lot = speed[lot]

# Współrzędne lotniska WAW
phi_waw = np.deg2rad(wspol_waw[0])
lambda_waw = np.deg2rad(wspol_waw[1])
h_waw = wspol_waw[2] + h_norm_waw + undulacja

xyz_waw = orto(phi_waw, lambda_waw, h_waw)
R_waw = macierz_obrotu(phi_waw, lambda_waw)

# Wspolrzedne lotniska OSL
phi_osl = np.deg2rad(wspol_osl[0])
lambda_osl = np.deg2rad(wspol_osl[1])
h_osl = wspol_osl[2] + h_norm_osl

xyz_osl = orto(phi_osl, lambda_osl, h_osl)
R_osl = macierz_obrotu(phi_osl, lambda_osl)
```

Kod źródłowy 5: Transformacja współrzędnych

```
# Azymut
azimuths = []
azimuths_o = []
azimuths_waw = []
azimuths_osl = []
# Odległość od lotniska
distance = []
distance_o = []
distance_waw = []
distance_osl = []
# Wysokość
height = []
height_waw = []
height_osl = []
# Czas
time_waw = []
time_osl = []

# Warszawa
for flh in wspolrzedne_lot:
    xyz_samolotu = orto(np.deg2rad(flh[0]), np.deg2rad(flh[1]), flh[2])
    wektor_samolot_lotnisko = np.array([xyz_samolotu[0] - xyz_waw[0], xyz_samolotu[1] -
        xyz_waw[1], xyz_samolotu[2] - xyz_waw[2]])
    neu = R_waw.T@wektor_samolot_lotnisko
    h = np.arcsin(np.clip(neu[2] / np.sqrt(neu[0]**2 + neu[1]**2 + neu[2]**2), -1, 1))
    h = np.degrees(h)
    height.append(h)

    az = np.arctan2(neu[1], neu[0])
    azimuths.append(az)

    dist = np.sqrt(neu[0]**2 + neu[1]**2 + neu[2]**2) / 1000
    distance.append(dist)

    if h > 0:
        height_waw.append(h)
        distance_waw.append(dist)
        azimuths_waw.append(az)
        time_waw.append(flh[2])

# Oslo
for flh in wspolrzedne_lot:
    xyz_samolotu = orto(np.deg2rad(flh[0]), np.deg2rad(flh[1]), flh[2])
    wektor_samolot_lotnisko = np.array([xyz_samolotu[0] - xyz_osl[0], xyz_samolotu[1] -
        xyz_osl[1], xyz_samolotu[2] - xyz_osl[2]])
    neu = R_osl.T @ wektor_samolot_lotnisko

    h = np.arcsin(np.clip(neu[2] / np.sqrt(neu[0]**2 + neu[1]**2 + neu[2]**2), -1, 1))
    h = np.degrees(h)

    az = np.arctan2(neu[1], neu[0])
    az = az if az >= 0 else az + 360
    azimuths_o.append(az)

    dist = np.sqrt(neu[0]**2 + neu[1]**2 + neu[2]**2) / 1000
```

```
        distance_o.append(dist)

        if h > 0:
            height_osl.append(h)
            distance_osl.append(dist)
            azimuths_osl.append(az)
            time_osl.append(flh[2])

    # Tablica współrzędnych i czasu
    data_waw = np.column_stack((azimuths_waw, np.array(height_waw), np.array(distance_waw),
         np.array(time_waw)))
    data_osl = np.column_stack((azimuths_osl, np.array(height_osl), np.array(distance_osl),
         np.array(time_osl)))
```

Kod źródłowy 6: Mapy lotu

```
# Linia geodezyjna lotu
fig = plt.figure(figsize=(10, 5))
ax = plt.axes(projection=ccrs.Robinson())
request = cimgt.OSM()
ax.add_image(request, 6)
ax.stock_img()
ax.coastlines()
# ax.plot(wspolrzedne_lot[:,1], wspolrzedne_lot[:,0],transform=ccrs.PlateCarree(),color
    ='b')
ax.plot([wspol_waw[1], wspol_osl[1]], [wspol_waw[0], wspol_osl[0]], transform=ccrs.
    Geodetic(), color='r')
extent = [-5, 35, 50, 65]
ax.set_extent(extent)

# Mapa lokalizacji samolotu
fig = px.scatter_geo(wspolrzedne_lot,
        lat=wspolrzedne_lot[:, 0],
        lon=wspolrzedne_lot[:, 1],
        color = time_lot, size = 1,
        title = 'Zarejestrowane współrzędne samolotu',
        hover_name = time_lot,
        projection="orthographic")
fig.update_geos(fitbounds="locations")
fig.update_layout(width = 1000, height = 1000)
fig.show()

# Trasa lotu
fig = go.Figure(data=go.Scattergeo(
        lon = wspolrzedne_lot[:, 1],
        lat = wspolrzedne_lot[:, 0],
        mode = 'lines',
        line = dict(width = 2, color = 'blue')))
fig.update_geos(fitbounds="locations")
fig.update_layout(title = 'LOT4YH', width = 2000, height = 1000)

# Etykiety lotnisk
fig.add_trace(go.Scattergeo(
        lon = [wspolrzedne_lot[0, 1], wspolrzedne_lot[-1, 1]],
        lat = [wspolrzedne_lot[0, 0], wspolrzedne_lot[-1, 0]],
        mode = 'markers + text',
        marker = dict(size = 10, color = 'navy'),
        text = ['WAW'],
        textposition="top center"))
fig.add_trace(go.Scattergeo(
        lon = [wspolrzedne_lot[-1, 1], wspolrzedne_lot[-1, 1]],
        lat = [wspolrzedne_lot[-1, 0], wspolrzedne_lot[-1, 0]],
        mode = 'markers + text',
        marker = dict(size = 10, color = 'navy'),
        text = ['OSL'],
        textposition="top center"))
fig.show()
```

Kod źródłowy 7: Wykresy Skyplot pokazujące znikanie i pojawianie się samolotu nad horyzontem

```
for data in (data_waw, data_osl):
  # Skyplot z elewacją punkty
  fig, ax = plt.subplots()
  ax = plt.axes(projection = 'polar')
  ax.set_theta_zero_location('N')
  ax.set_theta_direction(-1)
  ax.set_xlabel('Azymut [deg]')
  ax.set_ylabel('Wysokość [deg]')
  sc = ax.scatter(data[:,0], data[:,1], c = data[:,3], cmap = 'viridis')
  ax.set_title('Elewacja od lotniska od azymutu')

  # Skyplot z elewacją w postaci linii geodezyjnej
  fig, ax = plt.subplots()
  ax = plt.axes(projection = 'polar')
  ax.set_theta_zero_location('N')
  ax.set_theta_direction(-1)
  ax.set_xlabel('Azymut [deg]')
  ax.set_ylabel('Wysokość [deg]')
  ax.plot(data[:,0], data[:,1], color = 'blue')
  ax.set_title('Elewacja od lotniska od azymutu')

  # Skyplot z odległością punkty
  fig, ax = plt.subplots()
  ax = plt.axes(projection = 'polar')
  ax.set_theta_zero_location('N')
  ax.set_theta_direction(-1)
  ax.set_xlabel('Azymut [deg]')
  ax.set_ylabel('Odległość [km]')
  #sc = ax.scatter(azimuths, distance, color = 'blue')
  sc = ax.scatter(data[:,0], data[:,2], c = data[:,3], cmap = 'viridis')
  ax.set_title('Odległość od lotniska od azymutu')

# Skyplot z odległością w postaci linii geodezyjnej - WAW
fig, ax = plt.subplots()
ax = plt.axes(projection = 'polar')
ax.set_theta_zero_location('N')
ax.set_theta_direction(-1)
ax.grid(True)
ax.plot(azimuths, distance, color = 'blue')
ax.plot(data_waw[:,0], data_waw[:,2], color = 'red')
ax.set_xlabel('Azymut [deg]')
ax.set_ylabel('Odległość [km]')
ax.set_title('Odległość od lotniska WAW od azymutu')

# Skyplot z odległością w postaci linii geodezyjnej - OSL
fig, ax = plt.subplots()
ax = plt.axes(projection = 'polar')
ax.set_theta_zero_location('N')
ax.set_theta_direction(-1)
ax.grid(True)
ax.plot(azimuths_o, distance_o, color = 'blue')
ax.plot(data_osl[:,0], data_osl[:,2], color = 'red')
ax.set_xlabel('Azymut [deg]')
ax.set_ylabel('Odległość [km]')
ax.set_title('Odległość od lotniska OSL od azymutu')
```

```
plt.show()

print('Odległość w momencie zniknięcia: ', data_waw[-1, 2], 'km')
print('Odległość w momencie pojawienia się: ', data_osl[1, 2], 'km')

print('Czas lotu: ', flight_time)
print('Długość trasy: ', distance[-1], 'km')
print('Wysokość maksymalna: ', np.max(wspolrzedne_lot[:, 2])/1000, 'km')

print('Prędkość średnia: ', distance[-1] / (flight_time.total_seconds()/3600), 'km/h')
print('Prędkość maksymalna: ', np.max(speed_lot), 'km/h')
```

Kod źródłowy 8: Wykresy położenia samolotu od czasu

```
# Wysokośc od czasu
fig, ax = plt.subplots()
ax.plot(datetime_time_lot, wspolrzedne_lot[:, 2]/1000, color = 'red')
ax.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M'))
ax.grid(True)
ax.set_xlabel('Godzina [hh:mm]')
ax.set_ylabel('Wysokość [km]')
ax.set_title('Wysokość samolotu w funkcji czasu')

# Prędkość lotu od czasu
fig, ax = plt.subplots()
ax.plot(datetime_time_lot, speed_lot, color = 'orange')
ax.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M'))
ax.grid(True)
ax.set_xlabel('Godzina [h:min]')
ax.set_ylabel('Prędkość [km/h]')
ax.set_title('Prędkość samolotu w funkcji czasu')

# Elewacja od godziny
fig, ax = plt.subplots()
ax.plot(datetime_time_lot, height, color = 'green')
ax.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M'))
ax.grid(True)
ax.set_xlabel('Godzina [h:min]')
ax.set_ylabel('Wysokość [deg]')
ax.set_title('Wysokość kątowa samolotu w funkcji czasu')

# Odległość od godziny
fig, ax = plt.subplots()
ax.plot(datetime_time_lot, distance, color = 'blue')
ax.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M'))
ax.grid(True)
ax.set_xlabel('Godzina [h:min]')
ax.set_ylabel('Odelgłość [km]')
ax.set_title('Odległość samolotu od lotniska WAW w funkcji czasu')
```

Spis tabel

1 Dane samolotu 2

Spis wykresów

1 Linia lotu 4
2 Mapa lotu 4
3 Wykres odległości samolotu od azymutu z lotniska WAW 5
4 Wykres odległości samolotu od azymutu z lotniska WAW w postaci linii 5
5 Wykres odległości samolotu od azymutu z lotniska OSL 6
6 Wykres Skyplot odległości samolotu od azymutu z lotniska OSL w postaci linii 6
7 Wykres zależności wysokości samolotu od czasu 7
8 Wykres zależności prędkości samolotu od czasu 7
9 Wykres zależności wysokości kątowej samolotu od czasu z lotniska WAW 8
10 Wykres zależności odległości samolotu od lotniska WAW od czasu 8

Spis kodów źródłowych

1 Dane elipsoidy 3
2 Funkcja wczytywania danych udostępniona przez prowadzącego 9
3 Funkcje transformacji współrzędnych 9
4 Selekcja danych 10
5 Transformacja współrzędnych 11
6 Mapy lotu 13
7 Wykresy Skyplot pokazujące znikanie i pojawianie się samolotu nad horyzontem 14
8 Wykresy położenia samolotu od czasu 15