Zadanie 2

Autor: Adrian Fabisiewicz

Numer indeksu: 328935

Numer lotu: 4

Cel ćwiczenia

Celem ćwiczenia jest transformacja współrzędnych elipsoidalnych lecącego samolotu do układu lokalnego. Współrzędne te należało przeliczyć najpierw do geocentrycznych współrzędnych ortokartezjańskich, a ostatecznie do układu współrzędnych horyzontalnych, względem znanego położenia lotniska. W zadaniu należało również wyznaczyć moment, w którym samolot zniknie poniżej horyzontu. Po wykonaniu obliczeń należało wykonać odpowiednie wizualizacje, przedstawiające trasę samolotu oraz inne, wybrane dane.

Dane do zadania

W zadaniu wykorzystane zostały dane wybranego lotu, zawarte w pliku csv i pobrane z portalu flightradar24.com. Plik zawierał między innymi takie dane, jak: szerokość geograficzna φ, długość geograficzna λ, a także wysokość *h* samolotu w odniesieniu do poziomu jednego z lotnisk w danej jednostce czasu. Pierwszy rekord pliku z danymi pozwolił odnaleźć współrzędne rozpatrywanego lotniska.

Kolejność wykonywania obliczeń

Wczytanie pliku

Pierwszym krokiem było wczytanie pliku *lot4.csv*, w którym znajdowały się dane analizowanego przeze mnie lotu. Pozwoliła to zrobić dostarczona wraz z instrukcją do zadania funkcja *read_flightradar*, przyjmująca jako argument plik w formacie csv, a zwracająca kolumny z uporządkowanymi danymi.

In [ ]:
```
import numpy as np

def read_flightradar(file):
    '''
    Parameters
    ----------
    file : .csv file - format as downloaded from fligthradar24
        DESCRIPTION.
    Returns
    -------
    all_data : numpy array
```

```
        columns are:
            0 - Timestamp - ?
            1 - year
            2 - month
            3 - day
            4 - hour
            5 - minute
            6 - second
            7 - Latitude [degrees]
            8 - Longitude [degrees]
            9 - Altitude [feet]
            10 - Speed [?]
            11 - Direction [?]
    '''
    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(Alti
                                np.array(Speed), np.array(Direction)))
    return all_data
```

In [ ]:
```
file = 'lot4.csv'
dane = read_flightradar(file)
```

Wybranie potrzebnych danych oraz przygotowanie ich

Następnym krokiem było wybranie interesujących mnie kolumn z wczytanego pliku, a więc początkowo tych zawierających φ, λ oraz h. Należało również zamienić jednostkę wysokości ze stóp na metry. Przyjęliśmy, że wszystkie wysokości w pliku odniesione są do lotniska początkowego. Aby przeliczyć wysokości do wysokości elipsoidalnej, należało do wszystkich wysokości dodać wysokość normalną oraz odstęp elipsoidy od quasigeoidy, razem 135.4m.

In [ ]:
```
flh = dane[:,[7,8,9]]
flh[:,-1] = flh[:,-1]*0.3048 + 135.40
```

Następnie zapisałem pierwszy rekord, zawierający współrzędne lotniska oraz usunąłem rekordy, w których samolot stał w miejscu.

In [ ]:
```
flh_lotnisko = flh[0,:]
flh = flh[66:,:]
```

Przeliczenie współrzędnych φ, λ, h lotniska do współrzędnych ortokartezjańskich X, Y, Z

Do przeliczenia współrzędnych, wykorzystałem utworzoną funkcję *flh2xyz*, przyjmującą jako argumenty φ oraz λ w radianach oraz h w metrach oraz zwracającą tablicę z wyliczonymi współrzędnymi X, Y, Z.

In [ ]:
```
a = 6378137 # wielka półoś elipsoidy GRS80 w metrach
e2 = 0.00669438002290 # kwadrat pierwszego mimośrodu dla elipsoidy GRS80

def flh2xyz(phi, lamb, h):
    N = a/np.sqrt(1-e2*np.sin(phi)**2)
    x = (N + h)*np.cos(phi)*np.cos(lamb)
    y = (N + h)*np.cos(phi)*np.sin(lamb)
    z = (N*(1-e2)+h)*np.sin(phi)
    return [x, y, z]
```

Najpierw obliczyłem współrzędne X, Y, Z lotniska, uprzednio zamieniając współrzędne φ oraz λ ze stopni na radiany.

In [ ]:
```
xyz_lotnisko = flh2xyz(np.deg2rad(flh_lotnisko[0]), np.deg2rad(flh_lotnisko[1]), flh_l
```

Pętla - główne obliczenia

Najpierw zdefiniowałem macierz obrotu między układem współrzędnych geocentrycznych i lokalnych, potrzebną do kolejnych obliczeń.

In [ ]:
```
def Rneu(phi, lamb):
    R = np.array([[-np.sin(phi)*np.cos(lamb), -np.sin(lamb), np.cos(phi)*np.cos(lamb)]
                  [-np.sin(phi)*np.sin(lamb), np.cos(lamb), np.cos(phi)*np.sin(lamb)
                  [np.cos(phi), 0, np.sin(phi)]])
    return R
```

Wykorzystałem macierz do obliczenia wektora neu dla lotniska.

In [ ]:
```
R = Rneu(np.deg2rad(flh_lotnisko[0]), np.deg2rad(flh_lotnisko[1]))
```

W pętli kolejno zamieniałem wszystkie współrzędne φ, λ, h do współrzędnych X, Y, Z dla każdego rekordu. Następnie w pętli obliczany był wektor samolot-lotnisko xsl oraz zamieniany na współrzędne lokalne neu. Dalej liczony był azymut, długość wektora oraz wysokość horyzontalna. W ostatniej części pętli poszczególne dane z rekordów były zapisywane do tablic.

In [ ]:
```
for fi, lam, h in flh:
    xyz = flh2xyz(np.deg2rad(fi), np.deg2rad(lam), h)

    xsl = np.array(xyz) - xyz_lotnisko
    neu = R.T @ xsl

    az = np.rad2deg(np.arctan2(neu[1], neu[0]))
    s = np.sqrt(neu[0]**2 + neu[1]**2 + neu[2]**2) / 1000
    hrzalt = np.arcsin(neu[2]/(1000*s))

    hrzaltitudes.append(hrzalt)
    azimuths.append(az)
    distances.append(s)
    altitudes.append(h)
    locations.append([fi, lam])
```

Znalezienie punktu, w którym wysokość horyzontalna staje się ujemna

Częścią zadania było również wyznaczenie momentu, w którym samolot znika poniżej horyzontu. Poniższa pętla przeszła przez wszystkie obliczone wartości i zapisała indeks pierwszej ujemnej wartości wysokości.

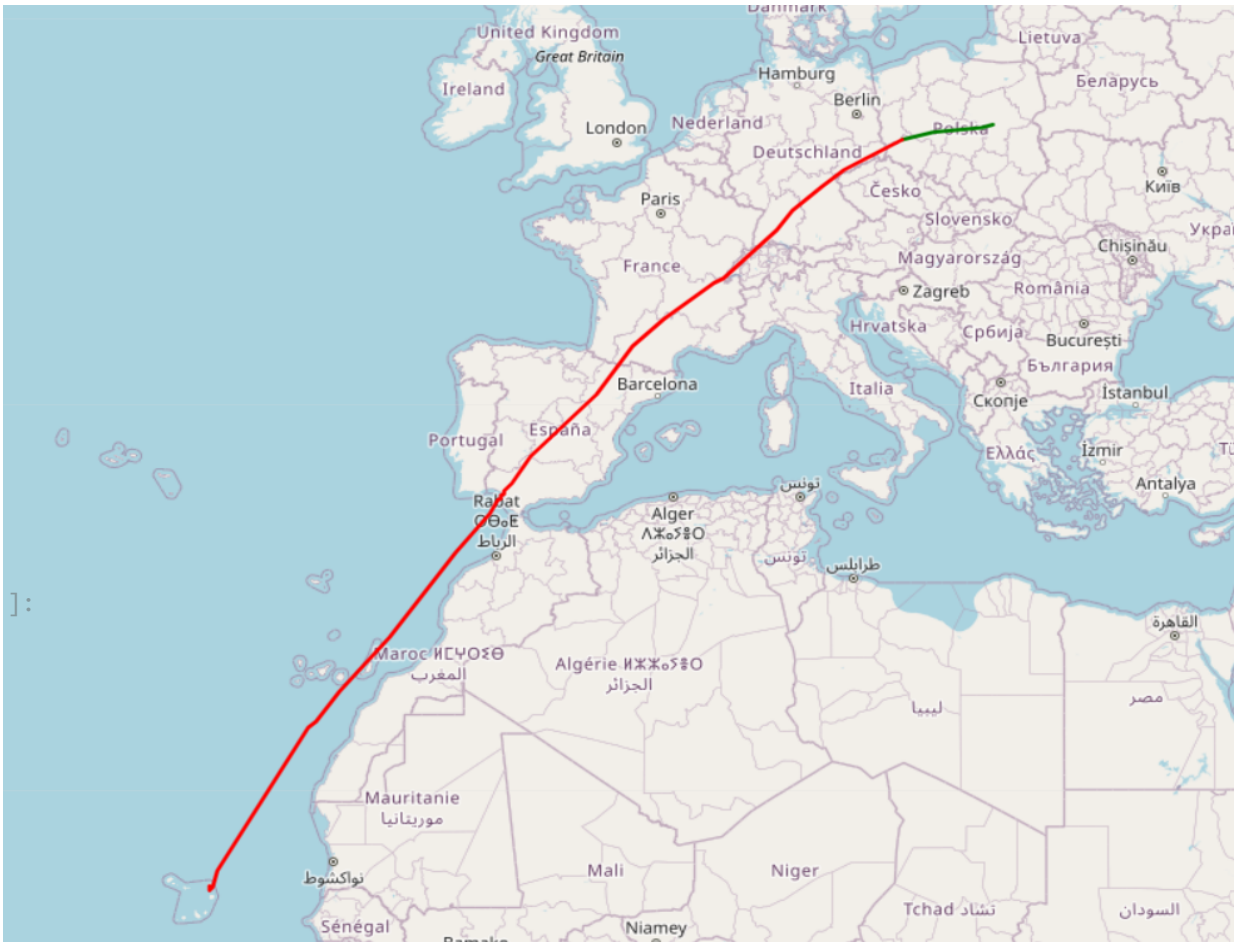
In [ ]:
```
breakpoint = None

for i in range(len(hrzaltitudes)):
    if hrzaltitudes[i] < 0:
        breakpoint = i
        break
```

Wizualizacje

Mapa lotu, przedstawiająca trasę przelotu z lotniska A do B

```
import folium

map = folium.Map(location=[52.164318, 20.981787], zoom_start=6)

folium.PolyLine(locations[:i], color='green', weight=2.5, opacity=1).add_to(map)
folium.PolyLine(locations[i-1:], color='red', weight=2.5, opacity=1).add_to(map)

map.save('mapa.html')
```

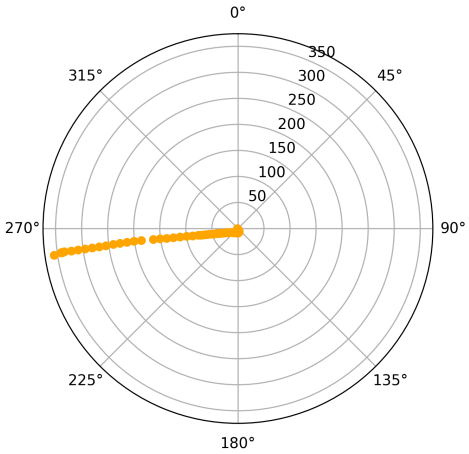
In [ ]:



Zielony kolor oznacza, że samolot jest widoczny na horyzoncie z lotniska początkowego.

Wykres skyplot - przedstawienie położenia samolotu w układzie lotniska początkowego do momentu zniknięcia na horyzoncie

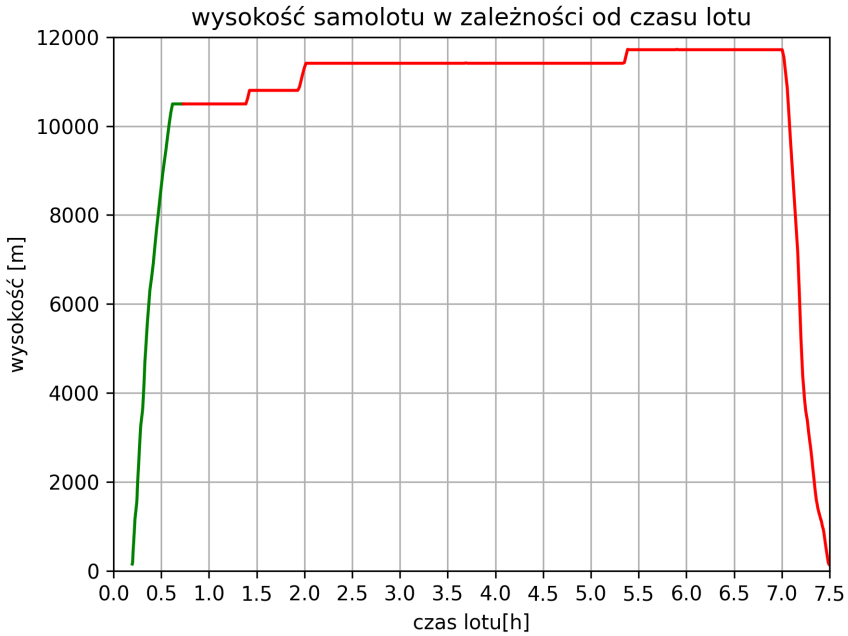
```
import matplotlib.pyplot as plt

plt.polar(np.deg2rad(azimuths[:i]), distances[:i], 'o', color='orange', markersize=5)
```



Wykres przedstawia azymuty w stopniach oraz odległość od lotniska początkowego w kilometrach.

Wykres zmian wysokości lotu samolotu w zależności od czasu

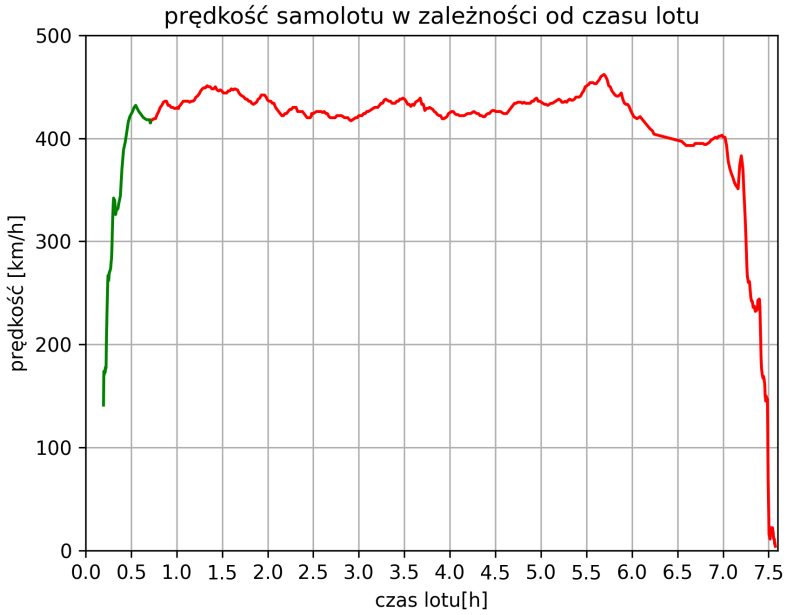
In [ ]:
```
start_time = dane[65,0]
times = []

for i in dane[:,0]:
    times.append((i - start_time)/3600)

times = times[66:]

plt.figure()
plt.plot(times[:i], altitudes[:i], color='green')
plt.plot(times[i-1:], altitudes[i-1:], color='red')
plt.ylim(0, 12000)
plt.xlim(0, 7.5)
plt.xlabel('czas lotu[h]')
plt.ylabel('wysokość [m]')
plt.title('wysokość samolotu w zależności od czasu lotu')
plt.xticks(np.arange(0, 8, 0.5))
plt.grid()
plt.show()
```



Wykres zmian prędkości lotu samolotu w zależności od czasu

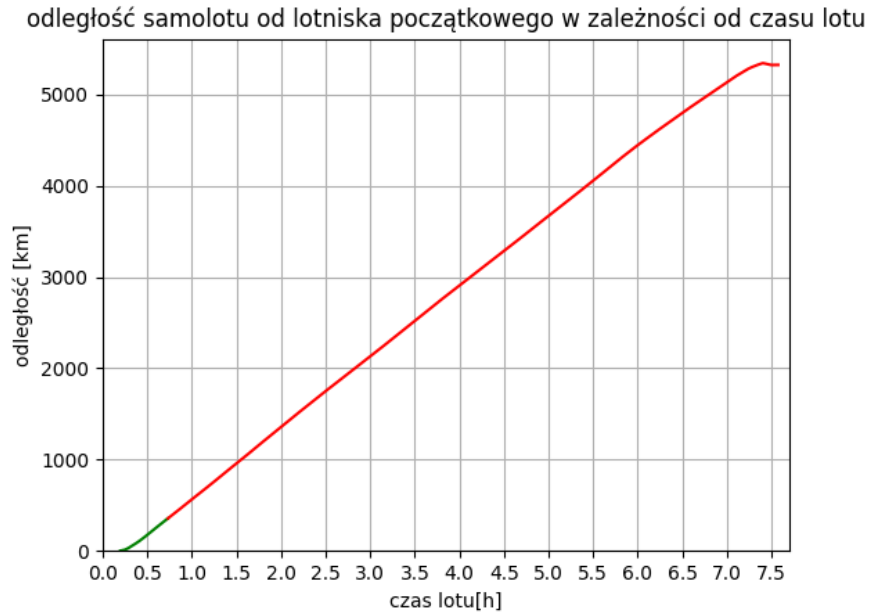
In [ ]:

```
start_time = dane[0,0]
times = []

for i in dane[:,0]:
    times.append((i - start_time)/3600)

spd = dane[:,-2]
speeds = []

for i in spd:
    speeds.append(i)

plt.figure()
i += 66
plt.plot(times[:i], speeds[:i], color='green')
plt.plot(times[i-1:], speeds[i-1:], color='red')
plt.ylim(0, 500)
plt.xlim(0, 7.6)
plt.xlabel('czas lotu[h]')
plt.ylabel('prędkość [km/h]')
plt.title('prędkość samolotu w zależności od czasu lotu')
plt.xticks(np.arange(0, 8, 0.5))
plt.grid()
plt.show()
```



Wykres zmian odległości samolotu od lotniska w zależności od czasu

In [ ]:
```
plt.figure()
plt.plot(times[:i], distances[:i], color='green')
plt.plot(times[i-1:], distances[i-1:], color='red')
plt.ylim(0)
plt.xlim(0, 7.7)
plt.xlabel('czas lotu[h]')
plt.ylabel('odległość [km]')
plt.title('odległość samolotu od lotniska początkowego w zależności od czasu lotu')
plt.xticks(np.arange(0, 8, 0.5))
plt.grid()
plt.show()
```



Wnioski

- Rozpatrywany lot odbywał się na trasie Warszawa-Espargos(Republika Zielonego Przylądka)
- Samolot wystartował z Lotniska Chopina w Warszawie, a wylądował na lotnisku Amílcar Cabral nieopodal Espargos.
- Samolot przestał być widoczny na horyzoncie z poziomu lotniska startowego nieopodal Głogowa, około 45 minut po wystartowaniu.
- Samolot w szczytowym okresie osiągnął wysokość niemal 12km nad ziemią.
- Samolot rozpoczął obniżać swoją prędkość około pół godziny przez wylądowaniem
- Samolot ustabilizował swoją prędkość po około pół godziny lotu, na poziomie 420-450 km/h
- Odległość, jaką przebył samolot to około 5300 km. Lot trwał około 7.5h.

Kod źródłowy

In [ ]:

```
import numpy as np
from read_flightradar import read_flightradar
import folium
import matplotlib.pyplot as plt

a = 6378137 # wielka półoś elipsoidy GRS80 w metrach
e2 = 0.00669438002290 # kwadrat pierwszego mimośrodu dla elipsoidy GRS80

def Rneu(phi, lamb):
    R = np.array([[-np.sin(phi)*np.cos(lamb), -np.sin(lamb), np.cos(phi)*np.cos(lamb)]
                  [-np.sin(phi)*np.sin(lamb), np.cos(lamb), np.cos(phi)*np.sin(lamb)
                  [np.cos(phi), 0, np.sin(phi)]])
    return R

def flh2xyz(phi, lamb, h):
    N = a/np.sqrt(1-e2*np.sin(phi)**2)
    x = (N + h)*np.cos(phi)*np.cos(lamb)
    y = (N + h)*np.cos(phi)*np.sin(lamb)
    z = (N*(1-e2)+h)*np.sin(phi)
    return [x, y, z]

file = 'lot4.csv'
dane = read_flightradar(file)

flh = dane[:,[7,8,9]]
flh[:,-1] = flh[:,-1]*0.3048 + 135.40

flh_lotnisko = flh[0,:]
flh = flh[66:,:]

xyz_lotnisko = flh2xyz(np.deg2rad(flh_lotnisko[0]), np.deg2rad(flh_lotnisko[1]), flh_l

R = Rneu(np.deg2rad(flh_lotnisko[0]), np.deg2rad(flh_lotnisko[1]))

azimuths = []
altitudes = []
distances = []
hrzaltitudes = []
locations = []

start_time = dane[0,0]
times = []

for i in dane[:,0]:
    times.append((i - start_time)/3600)

times = times[66:]

breakpoint = None

speeds = []
spd = dane[:,-2]
for i in spd:
    speeds.append(i)
```

```
speeds = speeds[66:]

for fi, lam, h in flh:
    xyz = flh2xyz(np.deg2rad(fi), np.deg2rad(lam), h)

    xsl = np.array(xyz) - xyz_lotnisko
    neu = R.T @ xsl

    az = np.rad2deg(np.arctan2(neu[1], neu[0]))
    s = np.sqrt(neu[0]**2 + neu[1]**2 + neu[2]**2) / 1000
    hrzalt = np.arcsin(neu[2]/(1000*s))

    hrzaltitudes.append(hrzalt)
    azimuths.append(az)
    distances.append(s)
    altitudes.append(h)
    locations.append([fi, lam])

for i in range(len(hrzaltitudes)):
    if hrzaltitudes[i] < 0:
        breakpoint = i
        break

# mapa lotu
map = folium.Map(location=[52.164318, 20.981787], zoom_start=6)
folium.PolyLine(locations[:i], color='green', weight=2.5, opacity=1).add_to(map)
folium.PolyLine(locations[i-1:], color='red', weight=2.5, opacity=1).add_to(map)
map.save('mapa.html')


# wykres skyplot
fig = plt.figure()
ax = fig.add_subplot(111, projection='polar')
ax.set_theta_direction(-1)
ax.set_theta_offset(np.pi/2)
ax.plot(np.deg2rad(azimuths[:i]), distances[:i], 'o', color='orange', markersize=5)
plt.savefig('skyplot.png', dpi=300)

# wykres wysokości samolotu
plt.figure()
plt.plot(times[:i], altitudes[:i], color='green')
plt.plot(times[i-1:], altitudes[i-1:], color='red')
plt.ylim(0, 12000)
plt.xlim(0, 7.5)
plt.xlabel('czas lotu[h]')
plt.ylabel('wysokość [m]')
plt.title('wysokość samolotu w zależności od czasu lotu')
plt.xticks(np.arange(0, 8, 0.5))
plt.grid()
plt.savefig('wswzoc.png', dpi=300)

# wykres prędkości samolotu
plt.figure()
plt.plot(times[:i], speeds[:i], color='green')
plt.plot(times[i-1:], speeds[i-1:], color='red')
plt.ylim(0, 500)
plt.xlim(0, 7.6)
plt.xlabel('czas lotu[h]')
```

```
plt.ylabel('prędkość [km/h]')
plt.title('prędkość samolotu w zależności od czasu lotu')
plt.xticks(np.arange(0, 8, 0.5))
plt.grid()
plt.savefig('pswzoc.png', dpi=300)

# wykres odległości samolotu od lotniska początkowego
plt.figure()
plt.plot(times[:i], distances[:i], color='green')
plt.plot(times[i-1:], distances[i-1:], color='red')
plt.ylim(0)
plt.xlim(0, 7.7)
plt.xlabel('czas lotu[h]')
plt.ylabel('odległość [km]')
plt.title('odległość samolotu od lotniska początkowego w zależności od czasu lotu')
plt.xticks(np.arange(0, 8, 0.5))
plt.grid()
plt.savefig('wzowzoc.png', dpi=300)

plt.show()
```