МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

Факультет радіоелектроніки, комп'ютерних систем та інфокомунікацій

**Лабораторна робота № 2 `**

з «Обробка даних засобами Python»

(назва дисципліни)

тема: «Визначення параметрів польоту літального апарату з трекових даними засобами мови програмування Python»

Виконав: студент 5 курсу групи № 555вМн

напряму підготовки (спеціальності)

123 Системне програмування

(шифр і назва напряму підготовки (спеціальності))

Логачов М.Г.

(прізвище й ініціали студента)

Прийняв: д.т.н., доцент

Дергачов К.Ю.

(посада, науковий ступінь, прізвище й ініціали)

Національна шкала: \_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_

Харків – 2023

Лабораторна робота №1

**Тема**: Визначення параметрів польоту літального апарату з трекових даними засобами мови програмування Python.

**Мета роботи :** Вироблення навичок роботи з навігаційними даними, визначення характеристик польоту літального апарату.

**Хід роботи**

1. У викладача отриманий трек польоту літального апарату згідно варіанту і виконати наступні дії:

2. Використовуючи отриманий трек, розробити програмний проект у середовищі PyChar.

3. Результати розрахунків представити у вигляді таблиці 1.1 і на графіках (Траєкторія польоту літального апарату, Залежність швидкості польоту від часу, Залежність висоти польоту від часу, пройдений шлях від часу)

4. Результати представити у вигляді графіків залежності:

* швидкості польоту від час;
* висоти польоту від часу;
* пройденого шляху від часу;
* траєкторії руху літака в координатах (широта, довгота, висота);

4. На підставі отриманих результатів зробити висновки, щодо режимів польоту літака та використаних засобів мови програмування Python

Отриманий трек польоту літального апарту згідно варінту (№7).

0GPGGA,061344.00,4945.6853,N,03642.2266,E,1,04,12.6,951.6,M,16.1,M,,\*53

0GPGGA,061345.00,4945.7258,N,03642.2776,E,1,05,8.3,959.0,M,16.1,M,,\*67

0GPGGA,061346.00,4945.7560,N,03642.3328,E,1,05,8.3,958.5,M,16.1,M,,\*62

0GPGGA,061347.00,4945.7908,N,03642.3850,E,1,05,9.5,960.0,M,16.1,M,,\*6C

0GPGGA,061348.00,4945.8282,N,03642.4336,E,1,05,9.5,965.3,M,16.1,M,,\*6F

0GPGGA,061349.00,4945.8639,N,03642.4823,E,1,05,9.5,969.6,M,16.1,M,,\*6C

0GPGGA,061350.00,4945.9015,N,03642.5302,E,1,06,7.0,973.2,M,16.1,M,,\*63

0GPGGA,061351.00,4945.9399,N,03642.5734,E,1,06,7.0,980.1,M,16.1,M,,\*6B

0GPGGA,061352.00,4945.9786,N,03642.6148,E,1,06,7.0,986.5,M,16.1,M,,\*6E

0GPGGA,061353.00,4946.0176,N,03642.6559,E,1,05,9.5,993.2,M,16.1,M,,\*63

0GPGGA,061354.00,4946.0614,N,03642.6919,E,1,06,7.4,1000.8,M,16.1,M,,\*5B

0GPGGA,061355.00,4946.1058,N,03642.7287,E,1,06,7.4,1007.9,M,16.1,M,,\*5E

0GPGGA,061356.00,4946.1500,N,03642.7623,E,1,05,9.5,1013.2,M,16.1,M,,\*5D

0GPGGA,061357.00,4946.1951,N,03642.7935,E,1,05,9.5,1019.4,M,16.1,M,,\*50

0GPGGA,061358.00,4946.2407,N,03642.8259,E,1,05,9.5,1022.4,M,16.1,M,,\*54

0GPGGA,061359.00,4946.2849,N,03642.8593,E,1,05,9.4,1019.8,M,16.1,M,,\*57

0GPGGA,061400.00,4946.3299,N,03642.8874,E,1,05,9.4,1021.7,M,16.1,M,,\*5A

0GPGGA,061401.00,4946.3769,N,03642.9130,E,1,05,6.8,1025.4,M,16.1,M,,\*5D

0GPGGA,061402.00,4946.4242,N,03642.9372,E,1,05,6.8,1028.6,M,16.1,M,,\*5E

0GPGGA,061403.00,4946.4706,N,03642.9627,E,1,04,13.2,1027.8,M,16.1,M,,\*61

0GPGGA,061404.00,4946.5190,N,03642.9844,E,1,04,13.2,1030.6,M,16.1,M,,\*6D

Приклад парсингу отриманих даних

0GPGGA,061344.00,4945.6853,N,03642.2266,E,1,04,12.6,951.6,M,16.1,M,,\*53

*0GPGGA* – це команда на запись повідомлення  
*061344.00*– час реєстрації даних 06 годин, 13 хвилин 44.00 секунд  
*4945.6853*– географічна широта 49 градусів 45.6853 хвилин

*N* – признак широти «північна» (Nord)  
*03642.2266*– географічна довгота, 036 градусів, 42.2266 хвилин

*E* - признак довготи «східна» (East)  
*951.6*– висота

*M* – одиниця виміру висоти – метри  
\**53* – контрольна сума повідомлення у 16-річній системі за модулем 16.

В ході виконання лабораторної роботи було створено програмний код за допомогою мови програмування Python. Використовиуючи бібліотеки *NumPy  
matplotlib, mplcursors, Geopy, pyproj, geographiclib* були побудовані графіки залежностей швидкості польоту від часу, висоти польоту від часу, пройденого шляху від часу та траєкторії руху літака в координата.

Необхідні залежності представлені у файлі *requirements.txt* додаток 1.

Для роботи з цим файлом необхідно в робочій деректорії виконати команди за добомогою терміналу GitBash.

* ця команда дозволить встановити віртуальне оточення в робочій дерикторії

source venv/Scripts/activate

* ця команда вказує на файл з необхідними залежностями

pip freeze > requirements.txt

* безпосереднє встановлення потрібних бібліотек

pip install -r requirements.txt

За допомогою програмного обчислення були отримані значення залежності швидкості польоту від часу, залежності висоти польоту від часу, пройденого шляху від часу. Використовуючи бібліотеки *Pandas* та *Python-Docx* була сформована таблиця залежностей параметрів польоту від часу.

Таблиця 1. Залежність параметрів польоту від часу

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | Latitude | Longitude | Distance | Time | Velocity |
| 1 | 49.456853 | 36.422266 | 0.00 | 06:13:44 | 0.0 |
| 2 | 49.457258 | 36.422776 | 58.74 | 06:13:45 | 58.74 |
| 3 | 49.45756 | 36.423328 | 110.99 | 06:13:46 | 52.25 |
| 4 | 49.457908 | 36.42385 | 165.14 | 06:13:47 | 54.15 |
| 5 | 49.458282 | 36.424336 | 219.91 | 06:13:48 | 54.77 |
| 6 | 49.458639 | 36.424823 | 273.22 | 06:13:49 | 53.31 |
| 7 | 49.459015 | 36.425302 | 327.70 | 06:13:50 | 54.48 |
| 8 | 49.459399 | 36.425734 | 381.11 | 06:13:51 | 53.41 |
| 9 | 49.459786 | 36.426148 | 433.97 | 06:13:52 | 52.86 |
| 10 | 49.460176 | 36.426559 | 487.02 | 06:13:53 | 53.05 |
| 11 | 49.460614 | 36.426919 | 542.80 | 06:13:54 | 55.78 |
| 12 | 49.461058 | 36.427287 | 599.37 | 06:13:55 | 56.57 |
| 13 | 49.4615 | 36.427623 | 654.49 | 06:13:56 | 55.12 |
| Index | **Latitude** | **Longitude** | **Distance** | **Time** | **Velocity** |
| 14 | 49.461951 | 36.427935 | 709.86 | 06:13:57 | 55.37 |
| 15 | 49.462407 | 36.428259 | 765.83 | 06:13:58 | 55.97 |
| 16 | 49.462849 | 36.428593 | 820.69 | 06:13:59 | 54.86 |
| 17 | 49.463299 | 36.428874 | 874.76 | 06:14:00 | 54.07 |
| 18 | 49.463769 | 36.42913 | 930.35 | 06:14:01 | 55.59 |
| 19 | 49.464242 | 36.429372 | 985.90 | 06:14:02 | 55.55 |
| 20 | 49.464706 | 36.429627 | 1040.72 | 06:14:03 | 54.82 |
| 21 | 49.46519 | 36.429844 | 1096.87 | 06:14:04 | 56.15 |

Знайдені значення для сумарної довжини маршруту польоту, середньої швидкості польоту, мінімальної та максимальної швидкості польоту, мінімальної та максимальної висоти польоту, загальний час усього польоту представлені в таблиці 2.

Таблиця 2. Результати обчисленнь

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Parameter | Value | Units |
| 1 | Total distance | 1096.87 | m |
| 2 | Average velocity | 197.424 | km/h |
| 3 | Maximum altitude | 1030.6 | m |
| 4 | Minimum altitude | 951.6 | m |
| 5 | Maximum velocity | 211.464 | km/h |
| 6 | Minimum velocity | 188.1 | km/h |
| 7 | Total flight time | 20.0 | s |

Рисунок 1. Графік залежності швидкості польоту від часу

Chart, line chart

Description automatically generated

Рисунок 2. Графік залежності висоти польоту від часу

Chart, line chart

Description automatically generated

Рисунок 3. Графік залежності пройденого шляху від часу

Chart, line chart

Description automatically generated

Рисунок 4. Графік траєкторії руху літака

Chart, surface chart

Description automatically generated

Рисунок 5. Графік траєкторії руху літакаChart, line chart

Description automatically generated

Висновки

За допомогою бібліотеки Matplotlib були побудовані необхідні графіки залежностей параметрів польоту від часу.

Можливість побудови тривімірного графіку польоту дає змогу оцінити траекторію руху в певний часовий проміжок часу або впевній частині записаного треку польтоу літака.

Використання Pandas розширює можливості представлення розрахунків у вигляді ріноманітних таблиць.

Додаток 1

[Посилання на Git репозиторій з вихідним кодом](https://github.com/lohachovKhai/geodata_processing/tree/main/Lab_2)

Модуль main.py

import model

# Open and read a file with flight data

dataFlight = []

flyTrek = open('fly\_trek.csv', 'r')

for line in flyTrek.readlines():

s = list(map(str, line.split(',')))

dataFlight.append(s)

flightDetails = model.DataParser(dataFlight)

# Dependency charts

flightDetails.generateChart("time-altitude")

flightDetails.generateChart("time-speed")

flightDetails.generateChart("time-distance")

flightDetails.generateChart("latitude - longitude - altitude")

# Required tables

flightDetails.generateTable("Table\_1")

flightDetails.generateTable("Table\_2")

Модуль distanceCalculator.py

import numpy as np

from geopy import distance

from pyproj import Geod

from geographiclib.geodesic import Geodesic

# This module was created to calculate the distance considering Earth's form

def getDistanceGeopy(point1, point2):

distance\_2d = distance.distance(point1[:2], point2[:2]).m

distance\_3d = np.sqrt(distance\_2d \*\* 2 + (point1[2] - point2[2]) \*\* 2)

return np.around([distance\_3d], decimals=2, out=None)[0]

def getDistanceGeod(point1, point2):

'''Set Earth params'''

g = Geod(ellips='WGS84')

azimuth1, azimuth2, distance\_2d = g.inv(point1[1], point1[0], point2[1], point2[0])

distance\_3d = np.hypot(distance\_2d, point2[2] - point1[2])

return np.around([distance\_3d], decimals=2, out=None)[0]

def getDistanceGeodesic(point1, point2):

geod = Geodesic.WGS84

g = geod.Inverse(point1[0], point1[1], point2[0], point2[1])

distance\_3d = np.hypot(g['s12'], point2[2] - point1[2])

return np.around([distance\_3d], decimals=2, out=None)[0]

Модуль chartBuilder.py

import matplotlib.pyplot as plt

import matplotlib.dates as mdates

import mplcursors

# Matplot function for generating charts/plots

def chartBuilder(xData, yData, xlabel, ylabel, title, colorPlot):

def textRun(annotation):

syt = annotation.replace('x', xlabel)

syt = syt.replace('y', ylabel)

return syt

plt.style.use('seaborn-v0\_8')

myFmt = mdates.DateFormatter('%H:%M:%S')

fig, ax = plt.subplots()

faceColor = None

fig.set(facecolor=faceColor)

ax.plot(xData, yData, color=colorPlot)

ax.set\_xlabel(xlabel, fontsize=20)

ax.set\_ylabel(ylabel, fontsize=20)

plt.title(title, fontsize=20)

dots = ax.scatter(xData, yData, color=colorPlot)

ax.xaxis.set\_major\_formatter(myFmt)

crs = mplcursors.cursor(dots, hover=True)

crs.connect("add", lambda sel: sel.annotation.set\_text(textRun(sel.annotation.get\_text())))

plt.show()

def chartBuilder3D(xData, yData, zData, xlabel, ylabel, zLabel, title):

fig = plt.figure()

fig.set(facecolor='grey')

ax = fig.add\_subplot(projection='3d')

plt.title(title)

ax.plot3D(xData, yData, zData, 'black', label="")

ax.scatter3D(xData, yData, zData, label="")

ax.set\_xlabel(xlabel, fontsize=15)

ax.set\_ylabel(ylabel, fontsize=15)

ax.set\_zlabel(zLabel, fontsize=15)

ax.legend(loc=2)

plt.show()

# Function for splitting dependencies charts/plots

def createChart(value, dependency):

match dependency:

case "time-speed":

chartBuilder(value.\_timeForChart, value.\_velocity, 'time', \

'velocity', '', 'black')

case "time-altitude":

chartBuilder(value.\_timeHigh, value.\_arrHight, 'time', \

'altitude (m)', '', 'navy')

case "time-distance":

chartBuilder(value.\_timeForChart, value.\_chartDistance, 'time', \

'distance (m)', '', 'crimson')

case "latitude - longitude - altitude":

chartBuilder3D(value.\_latitude, value.\_longtitude, value.\_arrHight, \

'Latitude', 'Longitude', 'Altitude (m)', '')

Модуль model.py

import numpy as np

import distanceCalculator

import chartBuilder

from datetime import datetime

import tablePrinter

class DataParser:

def \_\_init\_\_(self, arrData, distanceMethod="Geopy"):

self.\_arrData = arrData

self.executeFlight(self.\_arrData, distanceMethod)

def executeFlight(self, arrData, distanceMethod="Geopy"):

sumDis = 0

maxSpeed = 0

minSpeed = 5000

minFlight = float(arrData[0][9])

maxFlight = float(arrData[0][9])

arrDis = [0]

arrSpeed = [0]

arrHight = [arrData[0][9]]

latitude = [float(arrData[0][2])/100]

longitude = [float(arrData[0][4])/100]

for i in range(1, len(arrData)):

point1 = [float(arrData[i - 1][2]) / 100, float(arrData[i - 1][4]) / 100, float(arrData[i - 1][9])]

point2 = [float(arrData[i][2]) / 100, float(arrData[i][4]) / 100, float(arrData[i][9])]

dis2point = None

match distanceMethod:

case "Geopy":

dis2point = distanceCalculator.getDistanceGeopy(point1, point2)

case "Geod":

dis2point = distanceCalculator.getDistanceGeod(point1, point2)

case "Geodesic":

dis2point = distanceCalculator.getDistanceGeodesic(point1, point2)

sumDis += dis2point

arrDis.append(sumDis)

arrSpeed.append(float(dis2point))

arrHight.append(float(arrData[i][9]))

maxSpeed = max(dis2point, maxSpeed)

minSpeed = min(dis2point, minSpeed)

minFlight = min(float(arrData[i][9]), minFlight)

maxFlight = max(float(arrData[i][9]), maxFlight)

latitude.append(float(arrData[i][2])/100)

longitude.append(float(arrData[i][4])/100)

latitude[i] = np.around([float(latitude[i])], decimals=6, out=None)[0]

longitude[i] = np.around([float(longitude[i])], decimals=6, out=None)[0]

# Transform to numpy array

arrSpeed = np.array(arrSpeed)

arrSpeed = arrSpeed.astype(np.float64)

arrHight = np.array(arrHight)

arrHight = arrHight.astype(np.float64)

arrDis = np.array(arrDis)

arrDis = arrDis.astype(np.float64)

longitude = np.array(longitude)

longitude = longitude.astype(np.float64)

latitude = np.array(latitude)

latitude = latitude.astype(np.float64)

timeHigh = []

for i in range(0, len(arrData)):

strD = arrData[i][1][:-3]

datetime\_object = datetime.strptime(strD, '%H%M%S')

timeHigh.append(datetime\_object)

# total seconds

resultTime = timeHigh[len(timeHigh) - 1] - timeHigh[0]

# Separate arrays for better representing dependencies charts for \*speed from time\* and \*distance from time\*

timeForChart = []

for i in range(1, len(timeHigh)):

timeForChart.append(timeHigh[i])

velocity = []

for i in range(1, len(arrSpeed)):

velocity.append(arrSpeed[i])

chartDistance = []

for i in range(1, len(arrDis)):

chartDistance.append(arrDis[i])

# data request

self.\_maxSpeed = maxSpeed

self.\_minSpeed = minSpeed

self.\_minFlight = minFlight

self.\_maxFlight = maxFlight

self.\_sumDis = sumDis

self.\_averSpeed = sumDis / (len(arrDis) - 1)

self.\_timeFlight = resultTime.total\_seconds()

# array for charts

self.\_chartDistance = chartDistance

self.\_velocity = velocity

self.\_timeForChart = timeForChart

self.\_timeHigh = timeHigh

self.\_arrSpeed = arrSpeed

self.\_arrHight = arrHight

self.\_arrDis = arrDis

self.\_longtitude = longitude

self.\_latitude = latitude

def getMinAltitude(self):

return np.around([self.\_minFlight], decimals=2, out=None)[0]

def getMaxAltitude(self):

return np.around([self.\_maxFlight], decimals=2, out=None)[0]

def getTotalDistance(self):

return np.around([self.\_sumDis], decimals=2, out=None)[0]

def getTimeFlight(self):

return np.around([self.\_timeFlight], decimals=2, out=None)[0]

def getAverSpeed(self):

return np.around([self.\_averSpeed], decimals=2, out=None)[0]

def getMaxSpeed(self):

return np.around([self.\_maxSpeed], decimals=2, out=None)[0]

def getMinSpeed(self):

return np.around([self.\_minSpeed], decimals=2, out=None)[0]

def generateChart(self, dependency):

chartBuilder.createChart(self, dependency)

def generateTable(self, type):

tablePrinter.tableSeparator(self, type)

Модуль tablePrinter.py

import numpy as np

import pandas as pd

import docx as docx

def genearateTable1(lat, lon, dist, time, vel, fileName):

doc = docx.Document()

valSize = np.arange(1, np.array(lat).size + 1)

table\_data = {'Index': valSize, 'Latitude': lat, 'Longitude': lon, 'Distance': dist, 'Time': time, 'Velocity': vel}

df = pd.DataFrame(data=table\_data)

df['Time'] = pd.to\_datetime(df.Time).dt.strftime('%H:%M:%S')

# Initialise the table

t = doc.add\_table(rows=(df.shape[0] + 1), cols=df.shape[1])

t.style = 'Light Shading Accent 1'

# Add the column headings

for j in range(df.shape[1]):

t.cell(0, j).text = df.columns[j]

# Add the body of the data frame

for i in range(df.shape[0]):

for j in range(df.shape[1]):

cell = df.iat[i, j]

t.cell(i + 1, j).text = str(cell)

# Format Distance cells

for i in range(df.shape[0]):

cell = df.iat[i, 3]

t.cell(i + 1, 3).text = str("%.2f" % cell)

doc.save(fileName + '.docx')

def genearateTable2(totalDistance, averSpeed, maxFlight, minFlight, maxSpeed, minSpeed, timeFlight, fileName):

doc = docx.Document()

parameters = ['Total distance', 'Average velocity', 'Maximum altitude', 'Minimum altitude',

'Maximum velocity', 'Minimum velocity', 'Total flight time']

values = [totalDistance, velocityConverter(averSpeed),

maxFlight, minFlight,

velocityConverter(maxSpeed), velocityConverter(minSpeed),

timeFlight]

units = ['m', 'km/h', 'm', 'm', 'km/h', 'km/h', 's']

index = np.arange(1, np.array(values).size + 1)

table\_data = {'Index': index, 'Parameter': parameters, 'Value': values, 'Units': units}

df = pd.DataFrame(data=table\_data)

# Initialise the table

t = doc.add\_table(rows=(df.shape[0] + 1), cols=df.shape[1])

t.style = 'Light Shading Accent 1'

# Add the column headings

for j in range(df.shape[1]):

t.cell(0, j).text = df.columns[j]

# Add the body of the data frame

for i in range(df.shape[0]):

for j in range(df.shape[1]):

cell = df.iat[i, j]

t.cell(i + 1, j).text = str(cell)

cell = df.iat[1, 2]

t.cell(2, 2).text = str("%.3f" % cell)

doc.save(fileName + '.docx')

def velocityConverter(value):

value = value / 1000 \* 3600

return value

def tableSeparator(v, type):

match type:

case "Table\_1":

genearateTable1(v.\_latitude, v.\_longtitude, v.\_arrDis, v.\_timeHigh, v.\_arrSpeed, type)

case "Table\_2":

genearateTable2(v.getTotalDistance(), v.getAverSpeed(), v.getMaxAltitude(),

v.getMinAltitude(), v.\_maxSpeed, v.\_minSpeed, v.\_timeFlight, type)

Файл requirements.txt

numpy

matplotlib

mplcursors

geopy

pyproj

geographiclib

pandas

python-docx