Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский авиационный институт

(национальный исследовательский университет)»

Институт №4 «Радиоэлектроника, инфокоммуникации и информационная безопасность»

ОТЧЁТ

на тему:

«Исследование точности различных форматов представления ионосферных данных»

по дисциплине:

«УИРС»

Вариант 3

Проверил:

Подкорытов А.Н.

Выполнил:

Студент группы М4О-503С-20

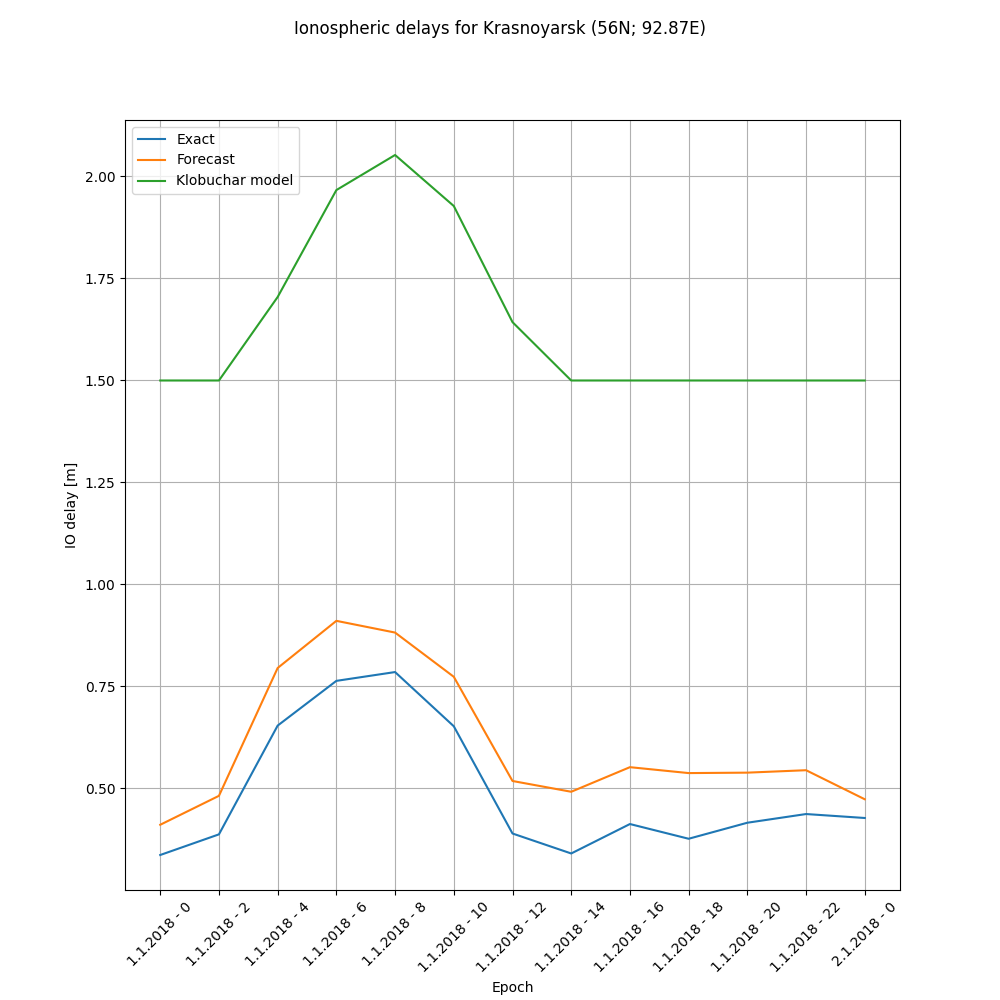
Лукашук А.Е.

Москва, 2024

**Задание**

1. Определить координаты местонахождения потребителя (Красноярск).
2. С использованием прогнозного файла ионосферных данных igrg0010.18i определить четыре точки ионосферной сетки, образующие прямоугольник, проекция которого на земную поверхность покрывает точку местоположения потребителя.
3. Вычислить зависимость вертикальной ионосферной задержки (vertical ionospheric delay) в метрах от времени для суточного интервала (всего 12 значений на сутки) для точки расположения потребителя. Для этого:
   1. Сначала для каждого требуемого момента времени следует вычислить значение вертикальной ионосферной задержки (в тех же единицах, что и задержки в файлах с ионосферными данными, т.е. в TECU, Total Electronic Content Unit) для точки расположения потребителя. Соответствующий алгоритм интерполяции описан в пункте И.2.3 ИКД СДКМ.
   2. Пересчитать полученное для каждого требуемого момента времени значение вертикальной ионосферной задержки из единиц TECU в метры.
4. Повторить вычисления п.2-п.3 для точного файла ионосферных данных igsg0010.18i.
5. С использованием модели Клобучара (Klobuchar Ionospheric model) вычислить значение вертикальной ионосферной задержки в метрах для точки расположения потребителя.
6. Построить на одном графике зависимости вертикальной ионосферной задержки для точки расположения потребителя от времени на суточном интервале для трёх используемых вариантов расчёта (прогнозные ионосферные данные, точные ионосферные данные, модель Клобучара).

**Результаты**



**Программный код**

import re

import numpy as np

from math import floor

import matplotlib.pyplot as plt

# Config

CITY\_NAME = "Krasnoyarsk"

LATITUDE = '56N'

LONGITUDE = '92.87E'

ELEVATION\_ANGLE = 90

AZIMUTH = 0

# Constants

c = 2.99792458e8

deg2semi = 1./180.

semi2rad = np.pi;

deg2rad = np.pi/180.

SECONDS\_IN\_WEEK = 604800

TECU2meters = (40.308193 / 1575.42e6 \*\* 2) \* 1e16

def find\_cords(filename: str, pos\_lat: float, pos\_long: float) -> tuple:

'''

finding nearest IGP position based on latitude and longitude of IPP

:param filename: file name of TEC delay file

:param pos\_lat: latitude of IPP

:param pos\_long: longitude of IPP

:return:

'''

lat\_north = 90.

lon\_west = -180.

lat\_south = -lat\_north

lon\_east = lon\_west

with open(filename, 'r') as file:

for line in file:

if 'LAT/LON1/LON2/DLON/H' in line:

LAT, LON1, LON2, DLON, H = tuple(float(match) for match in re.findall(r'-?\d+\.\d+', line.strip()))

if LAT - pos\_lat > 0 and lat\_north > LAT: lat\_north = LAT

elif LAT - pos\_lat < 0 and lat\_south < LAT: lat\_south = LAT

for long in range(int(LON1), int(LON2) + 1, int(DLON)):

if long - pos\_long < 0 and np.fabs(pos\_long - lon\_west) > np.fabs(pos\_long - long): lon\_west = long

elif long - pos\_long > 0 and np.fabs(lon\_east - pos\_long) > np.fabs(pos\_long - long): lon\_east = long

return lat\_north, lat\_south, float(lon\_west), float(lon\_east)

def find\_TEC\_delays(filename: str, latitude: float, longitude: float) -> dict:

"""

collecting all the TEC delays in time period

:param filename: filename of TEC delays file

:param latitude: latitude of IGP

:param longitude: longitude of IGP

:return: dictionary of TEC delays in format {epoch: TEC delay value}

"""

TEC\_delays = dict()

epoch = 0

line\_number = 0

with open(filename, 'r') as file:

lines = file.readlines()

for line in lines:

if 'EPOCH OF CURRENT MAP' in line:

epoch = tuple(int(match) for match in re.findall(r'\d+', line.strip()))

if 'START OF RMS MAP' in line: break

if 'LAT/LON1/LON2/DLON/H' in line:

LAT, LON1, LON2, DLON, H = tuple(float(match) for match in re.findall(r'-?\d+\.\d+', line.strip()))

if LAT == latitude:

delays\_on\_lat = [match for match in re.findall(r'\d+',''.join(lines[line\_number + 1 : line\_number + 6]))]

if epoch not in TEC\_delays.keys(): TEC\_delays[epoch] = int(delays\_on\_lat[int(np.fabs(longitude - LON1) / DLON)])

line\_number += 1

return TEC\_delays

def io\_delay(lat: float, long: float, delays: list, points: tuple) -> float:

"""

Calculation of delay interpolation (algorithm in И2.1 of the SDCM ICD)

:param lat: Latitude of IPP

:param long: Longitude of IPP

:param delays: An array containing delays in the following order: [Delay on NE, Delay on NW, Delay on SW, Delay on SE]

:param points: Contains longitudes to the west and east of the IPP and Latitudes to the north and south of the IPP

:return: IO delay (in meters) at IPP

"""

phi\_pp = lat

lambda\_pp = long

tau\_v = delays

lambda\_1, lambda\_2, phi\_1, phi\_2 = points

x\_pp = (lambda\_pp - lambda\_1) / (lambda\_2 - lambda\_1)

y\_pp = (phi\_pp - phi\_1) / (phi\_2 - phi\_1)

W = [x\_pp \* y\_pp,

(1 - x\_pp) \* y\_pp,

(1 - x\_pp) \* (1 - y\_pp),

x\_pp \* (1 - y\_pp)

]

tau\_vpp = 0

for k in range(3):

tau\_vpp = tau\_vpp + W[k] \* tau\_v[k] / 10.

return tau\_vpp \* TECU2meters

def klobuchar(latitude, longitude, elev, azim, tow, alpha, beta):

fi = float(latitude)

lamb = float(longitude)

a = azim \* deg2rad

e = elev \* deg2semi

psi = 0.0137 / (e + 0.11) - 0.022

lat\_i = fi \* deg2semi + psi \* np.cos(a)

if lat\_i > 0.416: lat\_i = 0.416

elif lat\_i < -0.416: lat\_i = -0.416

long\_i = lamb \* deg2semi + (psi \* np.sin(a) / np.cos(lat\_i \* semi2rad))

lat\_m = lat\_i + 0.064 \* np.cos((long\_i - 1.617) \* semi2rad)

t = 4.32e4 \* long\_i + tow

t = t % 86400.

if t > 86400.: t = t - 86400.

if t < 0: t = t + 86400

sF = 1. + 16. \* (0.53-e)\*\*3

PER = beta[0] + beta[1] \* lat\_m + beta[2] \* lat\_m \*\* 2 + beta[3] \* lat\_m \*\* 3

if PER < 72000: PER = 72000.

x = 2. \* np.pi \* (t - 50400.) / PER

AMP = alpha[0] + alpha[1] \* lat\_m + alpha[2] \* lat\_m \*\* 2 + alpha[3] \* lat\_m \*\* 3

if AMP < 0.: AMP = 0.

if np.fabs(x) > 1.57: dIon = sF \* (5.e-9)

else: dIon = sF \* (5.e-9 + AMP \* (1. - x\*x/2. + x\*x\*x\*x/24.))

return c \* dIon

def io\_delays\_by\_epoch(filename, LAT, LONG) -> dict:

"""

calculating interpolated IO delay by epoch at IPP

:param filename: filename of TEC delays file

:param LAT: latitude of IPP

:param LONG: longitude of IPP

:return: dictionary of delays in format {epoch: IO delay value}

"""

lat\_north, lat\_south, lon\_west, lon\_east = find\_cords(filename, float(LAT), float(LONG))

delays\_on\_NW = find\_TEC\_delays(filename, lat\_north, lon\_west)

delays\_on\_NE = find\_TEC\_delays(filename, lat\_north, lon\_east)

delays\_on\_SW = find\_TEC\_delays(filename, lat\_south, lon\_west)

delays\_on\_SE = find\_TEC\_delays(filename, lat\_south, lon\_east)

io\_delays = dict()

for epoch in delays\_on\_NW.keys():

delays = []

delays.append(delays\_on\_NE[epoch])

delays.append(delays\_on\_NW[epoch])

delays.append(delays\_on\_SW[epoch])

delays.append(delays\_on\_SE[epoch])

io\_delays[epoch] = io\_delay(float(LAT), float(LONG), delays, (lon\_west, lon\_east, lat\_south, lat\_north))

return io\_delays

def time\_of\_week(date\_time: tuple) -> tuple:

'''

This function converts calendar date/time to GPS week/time.

:param date\_time: tuple in the format (year, month, day, hours, minutes, seconds)

:return: gps\_week - gps\_seconds - integer seconds elapsed in gps\_week.

'''

year, month, day, hours, mins, sec = date\_time

if month <= 2:

y = year - 1

m = month + 12

if month > 2:

y = year

m = month

JD = floor(365.25 \* y) + floor(30.6001 \* (m + 1)) + day + ((hours + mins / 60 + sec / 3600) / 24) + 1720981.5

gps\_week = floor((JD - 2444244.5) / 7)

gps\_seconds = round(((((JD - 2444244.5) / 7) - gps\_week) \* SECONDS\_IN\_WEEK) / 0.5) \* 0.5

return gps\_seconds

def get\_ion\_corrections(filename: str):

with open(filename, 'r') as file:

for line in file:

line = line.replace('D', 'e')

if "ION ALPHA" in line:

ion\_alpha = tuple(float(match) for match in re.findall(r'-?\d+\.\d+e-\d+', line.strip()))

elif "ION BETA" in line:

ion\_beta = tuple(float(match) for match in re.findall(r'-?\d+\.\d+e\+\d+', line.strip()))

return ion\_alpha, ion\_beta

if \_\_name\_\_ == '\_\_main\_\_':

LAT = LATITUDE[:-1] if LATITUDE[-1] == 'N' else '-' + LATITUDE[:-1]

LONG = LONGITUDE[:-1] if LONGITUDE[-1] == 'E' else '-' + LONGITUDE[:-1]

exact\_delays = io\_delays\_by\_epoch('data/igsg0010.18i', LAT, LONG)

forecast\_delays = io\_delays\_by\_epoch('data/igrg0010.18i', LAT, LONG)

ion\_alpha, ion\_beta = get\_ion\_corrections('data/brdc0010.18n')

X = []

exact\_plot = []

forecast\_plot = []

klobuchar\_plot = []

for epoch in exact\_delays.keys():

epoch\_str = f"{epoch[2]}.{epoch[1]}.{epoch[0]} - {epoch[3]}"

X.append(epoch\_str)

exact\_plot.append(exact\_delays[epoch])

forecast\_plot.append(forecast\_delays[epoch])

tow = time\_of\_week(epoch)

klobuchar\_plot.append(klobuchar(LAT, LONG, ELEVATION\_ANGLE, AZIMUTH, tow, ion\_alpha, ion\_beta))

plt.figure(figsize=(10, 10))

plt.plot(X, exact\_plot, label='Exact')

plt.plot(X, forecast\_plot, label='Forecast')

plt.plot(X, klobuchar\_plot, label="Klobuchar model")

plt.xlabel('Epoch')

plt.xticks(rotation=45)

plt.ylabel('IO delay [m]')

plt.legend(loc="upper left")

plt.grid()

plt.suptitle(f"Ionospheric delays for {CITY\_NAME} ({LATITUDE}; {LONGITUDE})")

plt.show()