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 9

Проверил:

Подкорытов А.Н.

Выполнила:

Студент группы М4О-503С-20

Тихонова М.А.

2024 г.

Задание

1. Определить координаты местонахождения потребителя (Томск).

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:

a. 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.

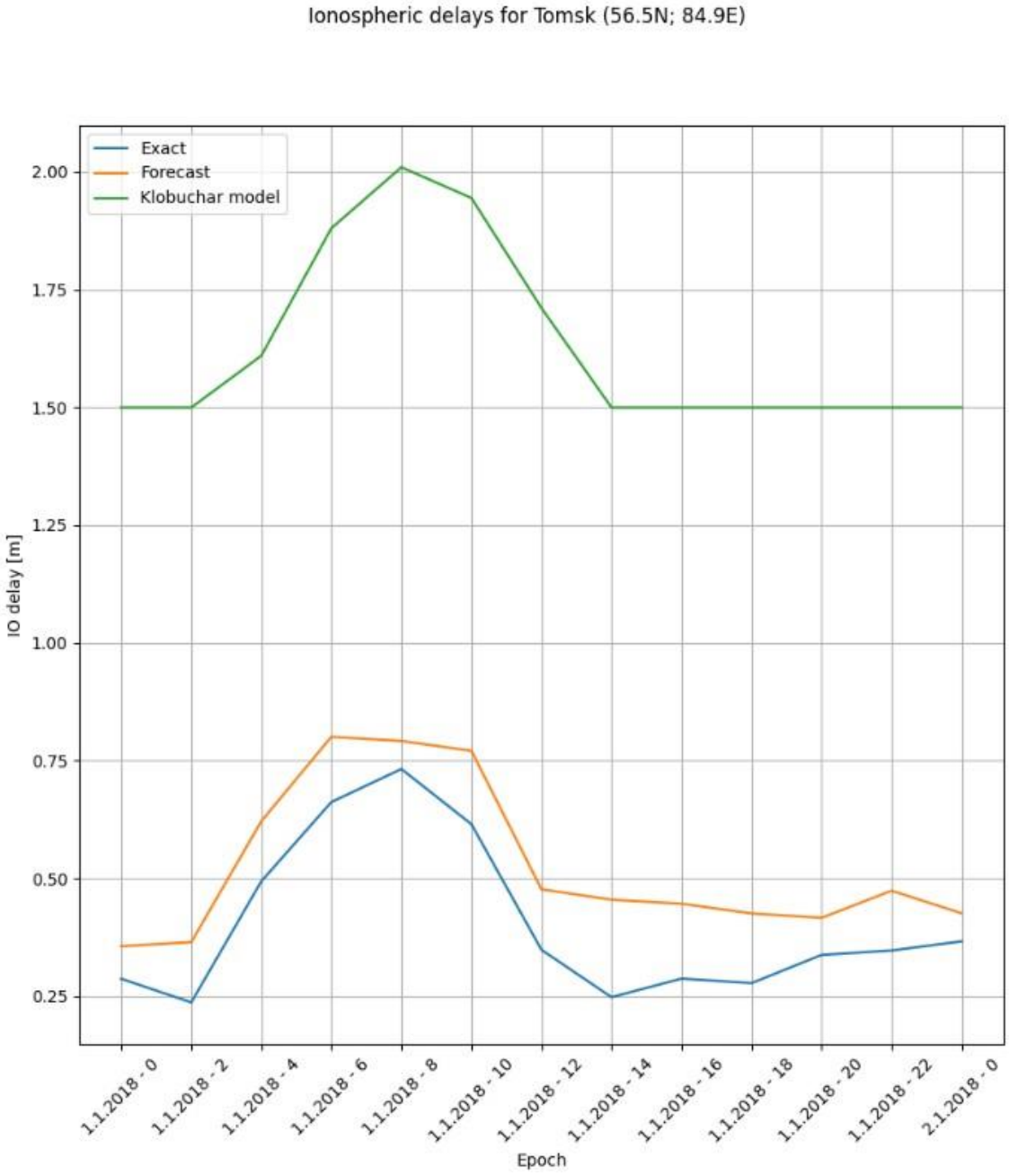
b. Пересчитать полученное для каждого требуемого момента времени значение вертикальной ионосферной задержки из единиц TECU в метры.

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


Ionospheric delays for Tomsk (56.5N; 84.9E)
Exact
Forecast
Klobuchar model
IO delay [m]
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
1.1.2018 - 0
1.1.2018 - 2
1.1.2018 - 4
1.1.2018 - 6
1.1.2018 - 8
1.1.2018 - 10
1.1.2018 - 12
1.1.2018 - 14
1.1.2018 - 16
1.1.2018 - 18
1.1.2018 - 20
1.1.2018 - 22
2.1.2018 - 0
Epoch

Программный код

```
import re
import numpy as np
from math import floor
import matplotlib.pyplot as plt

# Config
CITY_NAME = "Tomsk"
LATITUDE = '56.5N'
LONGITUDE = '84.9E'

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()
```