ФГБОУ ВО

Национальный исследовательский университет «МЭИ» »

Институт радиотехники и электроники

Направление радиоэлектронные системы и комплексы

Кафедра радиотехнических систем

Курсовой проект

по курсу «Аппаратура потребителей СРНС»

Выполнил студент:

Муратов Николай Сергеевич
группа: ЭР-15-17

Проверил:

к.т.н., доцент
Корогодин Илья Владимирович

Москва, 2022 г.

Содержание

2 Моделирование траектории движения **3**

2.1 Цель проекта 3

2.2 Задание 3

2.3 Основная часть 4

2.3.1 Пункт 1 4

2.3.2 Пункт 2 14

2.3.3 Пункт 3 16

2.4 Выводы 17

Этап 2

Моделирование траектории движения

2.1 Цель проекта

Конечная цель всего курсового проекта - получить библиотеку функций на Си++, позволяющую рассчитывать положение спутника GPS по данным с демодулятора его сигнала L1 C/A. На первом этапе реализуем модуль разбора навигационного сообщения до структуры эфемерид, сравним результаты со сторонней программой.

2.2 Задание

Эфемериды - параметры некоторой модели движения спутника. В разных ГНСС эти модели разные, а значит отличается и формат эфемерид, и алгоритмы расчета положения спутника.

Одна из самых простых и удобных моделей - в системе GPS.

Требуется реализовать на языке Matlab или Python функцию расчета положения спутника GPS на заданный момент по шкале времени UTC. В качестве эфемерид использовать данные, полученные на предыдущем этапе.

Построить трехмерные графики множества положений спутника GPS с системным номером, соответствующим номеру студента по списку. Графики в двух вариантах: в СК ECEF WGS84 и соответствующей ей инерциальной СК. Положения должны соответствовать суточному интервалу на дне формирования наблюдений, определенном на предыдущем этапе. Допускается использовать одни и те же эфемериды на весь рассматриваемый интервал.

Вывести значения координат спутника в файл out.txt в системе ECEF WGS 84 в виде строк: Секунда от начала дня | X | Y | Z

Используя оценку местоположения с предыдущего этапа, построить

Sky Plot за указанный временной интервал и сравнить результат с Trimble GNSS Planning Online.

Муратов Николай Сергеевич: спутник №4. Язык: Python.

Оформить отчет по результатам этапа:

- Реализация в Python;
- Таблица использованных эфемерид;
- Трехмерные графики положений спутника в ECEF и ECI (не забудьте подписать оси, изобразите соответствующую Земле сферу в начале СК)
- Расчётный и полученный в GNSS Planing Online SkyView;
- Оформить отчет по этапу и разместить на Github;
- Завести Pull Request.

2.3 Основная часть

2.3.1 Пункт 1

Так как первой развернутой на орбите ГНСС была GPS (NAVSTAR), то и первой ГНСС шкалой времени стала GPS Time (TGPS), которая сейчас отличается от UTC на 18с. Эта величина называется leap second. Эпоха в шкале времени GPS определяется номером недели (GPS Week) и номером секунды в неделе. Начало отсчета этой шкалы приходится на ночь с субботы на воскресенье 6 января 1980 г. в 00:00 ч (UTC). Каждая новая неделя также начинается в ночь с субботы на воскресенье. Номер GPS недели передается в навигационном сообщении в 10-битном поле, по этой причине по прошествии 1024 недель счетчик обнуляется. Этот эффект назван GPS week number rollover и происходит каждые 19,7 лет. Первый сброс недели произошел 21 августа 1999г, второй – 6 апреля 2019г.

GPS представление времени:

Номер недели: номер секунды от начала недели с учетом мкс.

В предыдущем пункте мы нашли номер недели и номер секунды от начала недели:

2197:93600

Переведем в UTC:

14/02/2022 03:04:44

Напишем по алгоритму из ИКД программу на языке Python, рассчитывающую координатызаданного спутника на интервале суток.

Table 30-II. Broadcast Navigation User Equations (sheet 1 of 4)

Element/Equation	Description
$\mu = 3.986005 \times 10^{14}$ meters^3/sec^2	WGS 84 value of the earth's gravitational constant for GPS user
$\dot{\Omega}_e = 7.2921151467 \times 10^{-5}$ rad/sec	WGS 84 value of the earth's rotation rate
$A_0 = A_{REF} + \Delta A$ *	Semi-Major Axis at reference time
$A_k = A_0 + (\dot{A})\, t_k$	Semi-Major Axis
$n_0 = \sqrt{\frac{\mu}{A_0^3}}$	Computed Mean Motion (rad/sec)
$t_k = t - t_{oe}$ **	Time from ephemeris reference time
$\Delta n_A = \Delta n_0 + \frac{1}{2} \Delta \dot{n}_0\, t_k$	Mean motion difference from computed value
$n_A = n_0 + \Delta n_A$	Corrected Mean Motion
$M_k = M_0 + n_A\, t_k$	Mean Anomaly
	Kepler's equation ($M_k = E_k - e \sin E_k$) may be solved for Eccentric Anomaly (E_k) by iteration:
$E_0 = M_k$	– Initial Value (radians)
$E_j = E_{j-1} + \frac{M_k - E_{j-1} + e \sin E_{j-1}}{1 - e \cos E_{j-1}}$	– Refined Value, minimum of three iterations, (j=1,2,3)
$E_k = E_j$	–Final Value (radians)
$\nu_k = 2 \tan^{-1}\left(\sqrt{\frac{1+e}{1-e}} \tan \frac{E_k}{2}\right)$	True Anomaly (unambiguous quadrant)

* $A_{REF} = 26{,}559{,}710$ meters

** **t** is GPS system time at time of transmission, i.e., GPS time corrected for transit time (range/speed of light). Furthermore, t_k shall be the actual total difference between the time **t** and the epoch time t_{oe}, and must account for beginning or end of week crossovers. That is if t_k is greater than 302,400 seconds, subtract 604,800 seconds from t_k. If t_k is less than -302,400 seconds, add 604,800 seconds to t_k.

Table 30-II. Broadcast Navigation User Equations (sheet 2 of 4)

Element/Equation *	Description
$\Phi_k = \nu_k + \omega_n$	Argument of Latitude
$\delta u_k = C_{us\text{-}n}\sin 2\Phi_k + C_{uc\text{-}n}\cos 2\Phi_k$	Argument of Latitude Correction (Second Harmonic Perturbations)
$\delta r_k = C_{rs\text{-}n}\sin 2\Phi_k + C_{rc\text{-}n}\cos 2\Phi_k$	Radial Correction (Second Harmonic Perturbations)
$\delta i_k = C_{is\text{-}n}\sin 2\Phi_k + C_{ic\text{-}n}\cos 2\Phi_k$	Inclination Correction (Second Harmonic Perturbations)
$u_k = \Phi_k + \delta u_k$	Corrected Argument of Latitude
$r_k = A_k(1 - e_n \cos E_k) + \delta r_k$	Corrected Radius
$i_k = i_{o\text{-}n} + (i_{o\text{-}n}\text{-DOT})t_k + \delta i_k$	Corrected Inclination
$x_k' = r_k \cos u_k$ $y_k' = r_k \sin u_k$	Positions in orbital plane
$\dot{\Omega} = \dot{\Omega}_{REF} + \Delta\dot{\Omega}$ ***	Rate of Right Ascension
$\Omega_k = \Omega_{0\text{-}n} + (\dot{\Omega} - \dot{\Omega}_e)\,t_k - \dot{\Omega}_e\,t_{oe}$	Corrected Longitude of Ascending Node
$x_k = x_k' \cos\Omega_k - y_k' \cos i_k \sin\Omega_k$ $y_k = x_k' \sin\Omega_k + y_k' \cos i_k \cos\Omega_k$ $z_k = y_k' \sin i_k$	Earth-fixed coordinates of SV antenna phase center
*** $\dot{\Omega}_{REF} = -2.6 \times 10^{-9}$ semi-circles/second.	

Имеем входные данные:

main.cpp | out.txt | in.txt

```
LNAV Ephemeris (slot = 221473810) =
        Crs    = -14.7812
        Dn     = 1.36379e-09
        M0     = -142.99       [deg]
        Cuc    = -8.08388e-07
        e      = 0.00169565
        Cus    = 1.0509e-05
        sqrtA  = 5153.65
        toe    = 93600
        Cic    = -1.49012e-08
        Omega0 = 11.8269       [deg]
        Cis    = 4.09782e-08
        i0     = 55.0915       [deg]
        Crc    = 178.406
        omega  = -175.597      [deg]
        omeDot = -4.4271e-07       [deg/s]
        iDot   = 2.93244e-08       [deg/s]
        Tgd    = 0         [sec]
        toc    = 93600
        af2    = 0
        af1    = 2.27374e-12
        af0    = -0.00018969
        WN     = 149
        IODC   = 588
        URA    = 0
        Health = 0
        IODE2  = 76
        IODE3  = 76
        codeL2 = 1
        L2P    = 1
```

Рисунок 2.3.1 — Входной файл in.txt

Листинг 2.1 — Программа декодирования подкадров навигационного сообщения GPS

```
# Made on Earth by Murathon
import math
import numpy as np
import matplotlib.pyplot as plt
import pymap3d


class TakeCoordinate():

    def __init__(self, t):
        self.time = t
        self.XYZ = np.zeros((1, 4))
        self.XYZv = np.zeros((1, 4))
        self.WGS_XYZ = np.zeros((1, 4))
        self.WGS_XYZv = np.zeros((1, 4))

    def coordinates(self):
        for i in range(86400):
            print(i)
            self.XYZv[0, 0] = i
            self.WGS_XYZv[0, 0] = i
            t_k = self.time[i] - t_oe

            if t_k > 302400:
                t_k -= 604800
            if t_k < -302400:
                t_k += 604800

            n_0 = math.sqrt(mu / A ** 3)
            n = n_0 + Dn

            E_old = M_0 + n * t_k

            E_old = E_old
            E_new = E_old + e_n * math.sin(E_old)
            i = 1
            while abs(E_new - E_old) > 1e-8:
```

```
                E_old = E_new
                E_new = E_old + e_n * math.sin(E_old)
                i += 1
                if (i > 10):
                    break
            E_k = E_new

            cosNu_k = (math.cos(E_k) - e_n) / (1 - e_n * math.cos(E_k))
            sinNu_k = (math.sqrt(1 - e_n ** 2) * math.sin(E_k)) / (1 - e_n * math.cos(E_k))

            Nu_k = math.atan2(sinNu_k, cosNu_k)
            Phi_k = Nu_k + omega

            delta_u_k = Cus * math.sin(2 * Phi_k) + Cuc * math.cos(2 * Phi_k)
            delta_r_k = Crs * math.sin(2 * Phi_k) + Crc * math.cos(2 * Phi_k)
            delta_i_k = Cis * math.sin(2 * Phi_k) + Cic * math.cos(2 * Phi_k)

            u_k = Phi_k + delta_u_k
            r_k = A * (1 - e_n * math.cos(E_k)) + delta_r_k
            i_k = i_0 + iDot * t_k + delta_i_k

            x_k_shtrih = r_k * math.cos(u_k)
            y_k_shtrih = r_k * math.sin(u_k)

            Omega_k = Omega_0 + (Omega_dot - Omega_e_Dot) * t_k - Omega_e_Dot * t_oe

            x_k = x_k_shtrih * math.cos(Omega_k) - y_k_shtrih * math.cos(i_k) * math.sin(Omega_k)
            y_k = x_k_shtrih * math.sin(Omega_k) + y_k_shtrih * math.cos(i_k) * math.cos(Omega_k)
            z_k = y_k_shtrih * math.sin(i_k)
            self.WGS_XYZv[0, 1] = x_k
            self.WGS_XYZv[0, 2] = y_k
            self.WGS_XYZv[0, 3] = z_k
```

```
70                 self.WGS_XYZ = np.vstack((self.WGS_XYZ, self.WGS_XYZv))
71
72                 X = x_k * math.cos(Omega_k) + y_k * math.sin(Omega_k)
73                 Y = -x_k * math.sin(Omega_k) + y_k * math.cos(Omega_k)
74                 Z = z_k
75
76                 self.XYZv[0, 1] = X
77                 self.XYZv[0, 2] = Y
78                 self.XYZv[0, 3] = Z
79                 self.XYZ = np.vstack((self.XYZ, self.XYZv))
80             np.savetxt("XYZ.txt", self.XYZ)
81             np.savetxt("WGS_XYZ.txt", self.WGS_XYZ)
82
83
84 if __name__ == '__main__':
85
86     gps_pi = 3.1415926535898
87     toRad = gps_pi / 180
88     sec2rad = gps_pi / (3600 * 180)
89     min2rad = gps_pi / (60 * 180)
90     mu = 3.986005e14
91     Omega_e_Dot = 7.2921151467e-5
92     Omega_ref_Dot = -2.6e-9
93
94     # LNAV    Ephemeris(slot=221473810) =
95     Crs = -14.7812
96     Dn = 1.36379e-09 * toRad
97     M_0 = -142.99 * toRad
98     Cuc = -8.08388e-07
99     e_n = 0.00169565
100    Cus = 1.0509e-05
101    A = 5153.65 ** 2
102    t_oe = 93600
103    Cic = -1.49012e-08
104    Omega_0 = 11.8269 * toRad
105    Cis = 4.09782e-08
106    i_0 = 55.0915 * toRad
107    Crc = 178.406
108    omega = -175.597 * toRad
109    Omega_dot = -4.4271e-07 * toRad
```

```
110     iDot = 2.93244e-08 * toRad
111     Tgd = 0
112     toc = 93600
113     af2 = 0
114     af1 = 2.27374e-12
115     af0 = -0.00018969
116     WN = 149
117     IODC = 588
118     URA = 0
119     Health = 0
120     IODE2 = 76
121     IODE3 = 76
122     codeL2 = 1
123     L2P = 1
124     begin_time = 86382
125     end_time = 172782
126     step = 1
127     # Создадим массив времени
128     t = np.arange(begin_time, end_time, step, int)
129     # Найдем координаты на интервале времени
130     moment1 = TakeCoordinate(t)
131     #moment1.coordinates()
132     moment1.XYZ = np.loadtxt("XYZ.txt")
133     moment1.WGS_XYZ = np.loadtxt("WGS_XYZ.txt")
134     # print(moment1.XYZ)
135     # Переведем в радианы координата приемника
136     N_gr = 44
137     N_min = 9
138     N_sec = 36.3261
139     N = N_gr * toRad + N_min * min2rad + N_sec * sec2rad
140
141     E_gr = 39
142     E_min = 00
143     E_sec = 13.0546
144     E = E_gr * toRad + E_min * min2rad + E_sec * sec2rad
145
146     H = 1.247
147
148     # найдем x
149     SKP = np.zeros((1, 3))
```

```
150     SKPF = np.zeros((1, 3))
151     # for i in range(86400):
152     #     Vrem = pymap3d.ecef2enu(moment1.WGS_XYZ[i, 1], moment1.
    WGS_XYZ[i, 2], moment1.WGS_XYZ[i, 3], N, E, H, deg = False)
153     #     SKP[0, 0] = Vrem[0]
154     #     SKP[0, 1] = Vrem[1]
155     #     SKP[0, 2] = Vrem[2]
156     #     SKPF = np.vstack((SKPF, SKP))
157     #     print(i)
158     # np.savetxt("SKPF.txt", SKPF)
159     SKPF = np.loadtxt("SKPF.txt")
160     RFT = np.zeros((1, 3))
161     RFTF = np.zeros((1, 3))
162 
163     for i in range(86400):
164         print(i)
165         if SKPF[i, 2] > 0:
166             RFT[0, 0] = math.sqrt(SKPF[i, 0] ** 2 + SKPF[i, 1] ** 2
    + SKPF[i, 2] ** 2)
167             RFT[0, 1] = math.acos(SKPF[i, 2] / RFT[0, 0])
168             if SKPF[i, 0] > 0:
169                 RFT[0, 2] = -math.atan(SKPF[i, 1] / SKPF[i, 0]) +
    gps_pi / 2
170 
171             elif ((SKPF[i, 0] < 0) and (SKPF[i, 1] > 0)):
172                 RFT[0, 2] = -math.atan(SKPF[i, 1] / SKPF[i, 0]) + 3
    * gps_pi / 2
173 
174             elif ((SKPF[i, 0] < 0) and (SKPF[i, 1] < 0)):
175                 RFT[0, 2] = -math.atan(SKPF[i, 1] / SKPF[i, 0]) -
    gps_pi / 2
176         else:
177             RFT[0, 1] = np.nan
178             RFT[0, 0] = np.nan
179             RFT[0, 2] = np.nan
180         RFTF = np.vstack((RFTF, RFT))
181 
182     plt.subplot(111, polar=True)  # Полярная система координат
183 
184 
```

```
185     plt.plot(RFTF[0:86400, 2], RFTF[0:86400, 1]/toRad, lw=2)
186
187     plt.show()
188     Найдем# xyz приемника
189
190     X_RCV = 6400000 * math.cos(N)*math.cos(E)
191
192     Y_RCV = 6400000 * math.cos(N)*math.sin(E)
193
194     Z_RCV = 6400000*math.sin(N)
195
196
197     fig = plt.figure()
198     ax = fig.add_subplot(111, projection='3d')
199     m = 0
200     for i in range(85):
201         m += 1000
202         ax.scatter(moment1.WGS_XYZ[m, 1], moment1.WGS_XYZ[m, 2],
    moment1.WGS_XYZ[m, 3], s=5)
203     print(X_RCV,Y_RCV,Z_RCV)
204     u = np.linspace(0, 2 * np.pi, 100)
205     v = np.linspace(0, np.pi, 100)
206
207     x = 6400000 * np.outer(np.cos(u), np.sin(v))
208     y = 6400000 * np.outer(np.sin(u), np.sin(v))
209     z = 6400000 * np.outer(np.ones(np.size(u)), np.cos(v))
210
211     ax.plot_surface(x, y, z, rstride=4, cstride=4, color='b')
212     ax.set_xlim3d(-3e7, 3e7)
213     ax.set_ylim3d(-3e7, 3e7)
214     ax.set_zlim3d(-3e7, 3e7)
215     plt.show()
216
217     fig = plt.figure()
218     ax = fig.add_subplot(111, projection='3d')
219     m = 0
220     for i in range(85):
221         m += 1000
222         ax.scatter(moment1.XYZ[m, 1], moment1.XYZ[m, 2], moment1.XYZ
    [m, 3], s=5)
```

```
223
224     u = np.linspace(0, 2 * np.pi, 100)
225     v = np.linspace(0, np.pi, 100)
226
227     x = 6400000 * np.outer(np.cos(u), np.sin(v))
228     y = 6400000 * np.outer(np.sin(u), np.sin(v))
229     z = 6400000 * np.outer(np.ones(np.size(u)), np.cos(v))
230
231     ax.plot_surface(x, y, z, rstride=4, cstride=4, color='b')
232     ax.set_xlim3d(-3e7, 3e7)
233     ax.set_ylim3d(-3e7, 3e7)
234     ax.set_zlim3d(-3e7, 3e7)
235     plt.show()
```

2.3.2 Пункт 2

Построим трехмерные графики положения спутника на протяжении суток.

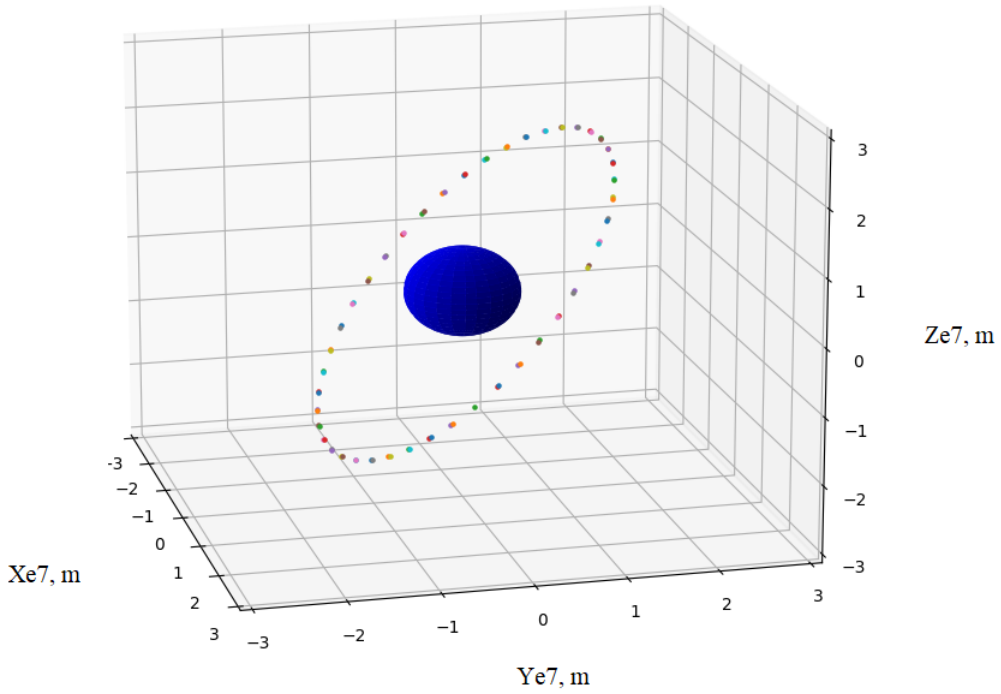

Рисунок 2.3.2 — Трехмерный график положения спутника GPS в ECEF WGS-84

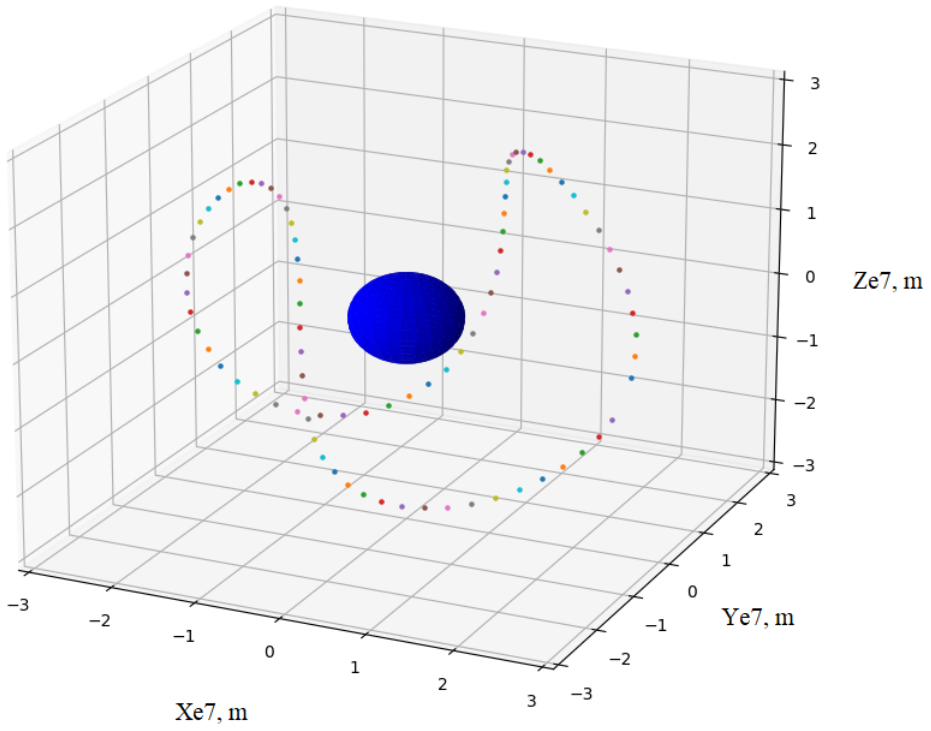

Рисунок 2.3.3 — ТрехмерныЙ график положения спутника GPS в инерциальной системе координат ECI

Переход из системы ECEF в систему ECI был осуществлен также согласно алгоритму из ИКД.

2.3.3 Пункт 3

Помимо траектории спутников в трехмерном виде получим эту траекторию в полярной системе координат(рис. 2.3.4) и сравним ее с результатом из Trimble GNSS Planning Online(рис. 2.3.5).

SkyView стоится относительно положения приемника из RTKNAVI.

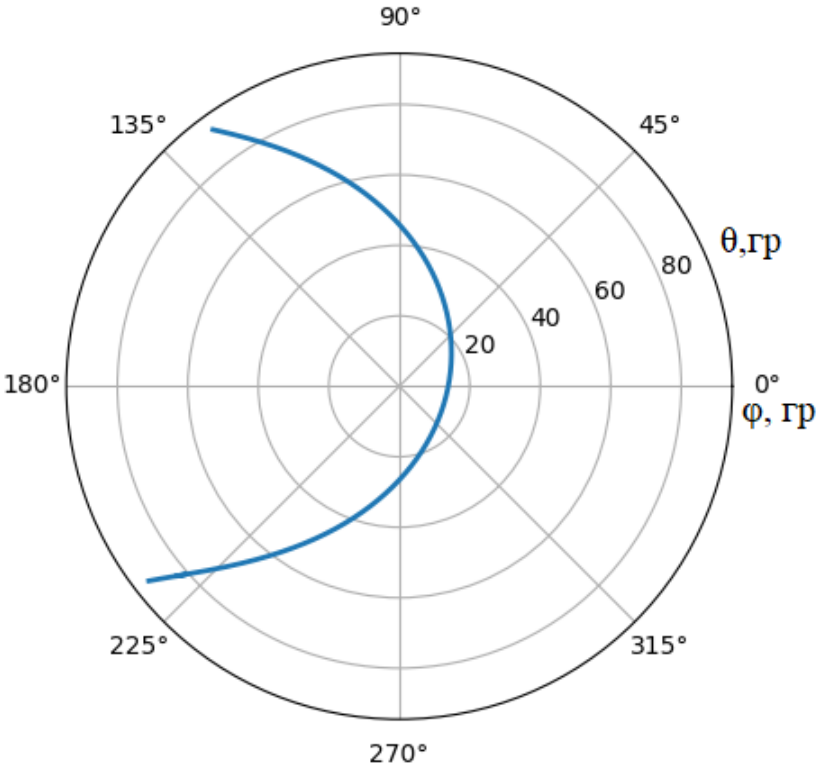

Рисунок 2.3.4 — Рассчитанный SkyPlot

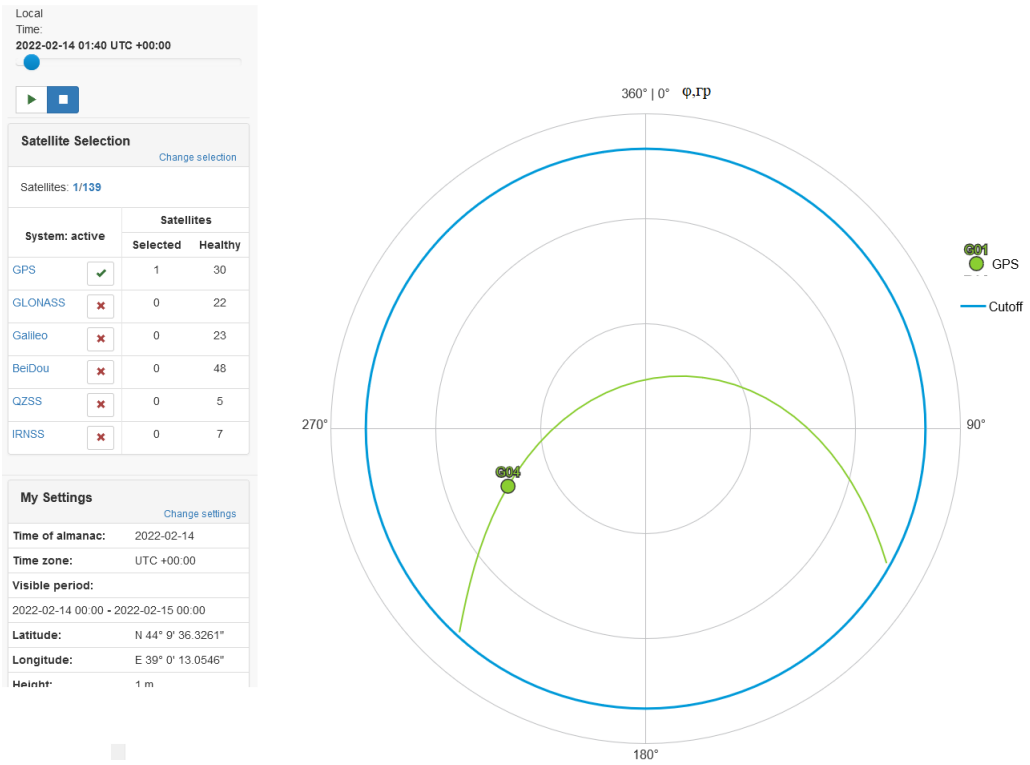

Рисунок 2.3.5 — Trimble SkyPlot

2.4 Выводы

На втором этапе по эфемеридам были рассчитаны положения спутника в разных системах координат, а также построен график SkyView.

Литература

[1] Interface Control Contractor: SAIC (GPS SEI) 200 N. Pacific Coast Highway, Suite 1800 El Segundo, CA 90245