Politechnika Warszawska

WYDZIAŁ MECHANICZNY
ENERGETYKI I LOTNICTWA

Instytut Instytut Techniki Lotniczej i Mechaniki Stosowanej

Praca przejściowa inżynierska

na kierunku Lotnictwo i Kosmonautyka
w specjalności Automatyka i Systemy Pokładowe

Wyznaczanie orbity statku kosmicznego na podstawie obserwacji naziemnych

Paweł Pietrzak
Numer albumu 304263

opiekun
mgr inż. Mateusz Sochacki

WARSZAWA 2024

Spis treści

1. Wstęp

Wyznaczenie orbity obiektu kosmicznego polega na określeniu wektora stanu (położenia oraz prędkości) obserwowanego ciała na podstawie zbioru obserwacji [1]. Zagadnienie to jest jednym z najważniejszych zadań mechaniki nieba już od początków istnienia tej dziedziny nauki. Pierwsze skuteczne metody zostały opracowane na przełomie XVIII i XIX wieku m.in. przez Laplace'a [2] oraz Gaussa, który wykorzystując swoją metodę wyznaczył orbitę Ceres [3].

Współcześnie znajomość orbit potrzebna jest nie tylko do katalogowania Układu Słonecznego, ale również do planowania manewrów orbitalnych, korekcji orbit satelitów, unikania kosmicznych śmieci. Mimo upływu lat, historyczne metody w rozwiniętej wersji wciąż są wykorzystywane jako pierwsze przybliżenie dla metod numerycznych pozwalających wyznaczyć dokładniejsze rozwiązania [1].

1.1. Definicja wektora stanu i elementów orbitalnych

Wektor stanu orbitalnego statku kosmicznego składa się z wektorów położenia (**r**) i prędkości (**v**). Zgodnie z konwencją ruch statków kosmicznych w okolicy Ziemi opisuję się względem układu ECI (Earth-Centered Inertial). Jego środek znajduje się w środku Ziemi, płaszczyzna XY jest zgodna z płaszczyzną równika, oś X wskazuje na Punkt Barana, a oś Z jest zgodna z osią obrotu planety. Oś Y dopełnia układ tak, by był on prawoskrętny.

Korzystając z prawa powszechnego ciążenia oraz drugiej zasady dynamiki można wyznaczyć równania różniczkowe ruchu obiektu i je rozwiązać, używając wektora stanu jako warunku brzegowego. Prowadzi to do wyznaczenia orbity możliwej do opisania przy pomocy elementów orbitalnych, czyli zestawu sześciu wielkości, pięciu stałych w czasie opisujących jej tor oraz szóstej, zmiennej, wskazującej aktualne położenie obiektu na orbicie [4]. Klasyczne elementy orbitalne składają się z:

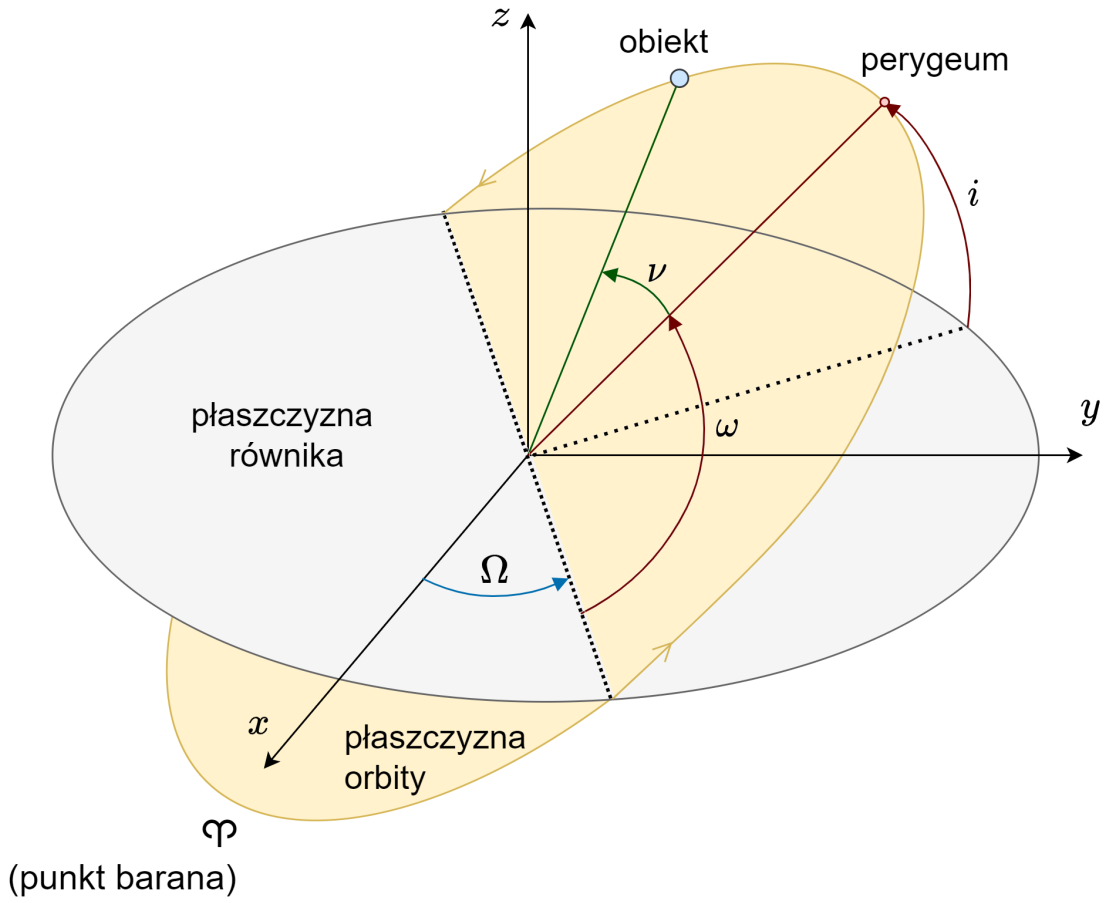
- półosi wielkiej (a)
- inklinacji (i)
- długości węzła wstępującego (Ω)
- mimośrodu (e)
- argumentu perygeum (ω)
- anomalii prawdziwej (ν)

Elementy orbitalne oraz układ ECI ilustruje rysunek 1.1.

1.2. Transformacja wektora stanu na elementy orbitalne

Dysponując wektorem stanu (wektor położenia **r** i wektor prędkości **v** w układzie ECI) można wyznaczyć elementy orbitalne. Potrzebny jest też powiązany z prawem powszechnego ciążenia standardowy parametr grawitacyjny μ, będący iloczynem stałej grawitacyjnej i sumy mas ciała orbitującego i centralnego.

$$\mu = G(M+m) \overset{M>>m}{\approx} GM \tag{1.1}$$



Rysunek 1.1. Układ referencyjny ECI oraz część elementów orbitalnych: inklinacja i, długość węzła wstępującego Ω, argument perygeum ω i anomalia prawdziwa ν

Procedura wyznaczenia elementów orbitalnych z wektora stanu:
Wyznaczenie momentu pędu:

$$\mathbf{h} = \mathbf{r} \times \mathbf{v} \tag{1.2}$$

Wyznaczenie inklinacji:

$$i = \cos^{-1}\left(\frac{h_z}{\|\mathbf{h}\|}\right) \tag{1.3}$$

Wyznaczenie: osi apsyd i długości węzła wstępującego:

$$\mathbf{N} = [0,0,1] \times \mathbf{h} \tag{1.4}$$

$$\Omega = \cos^{-1}\left(\frac{N_x}{\|\mathbf{N}\|}\right) \tag{1.5}$$

Wyznaczenie mimośrodu:

$$\mathbf{e} = \frac{1}{\mu}\left(\mathbf{v} \times \mathbf{h} - \mu\frac{\mathbf{r}}{\|\mathbf{r}\|}\right) \tag{1.6}$$

$$e = \|\mathbf{e}\| \tag{1.7}$$

Wyznaczenie argumentu perygeum:

$$\omega = cos^{-1}\left(\frac{\mathbf{N}\cdot\mathbf{e}}{\|\mathbf{N}\|\|\mathbf{e}\|}\right) \tag{1.8}$$

Jeśli $e_Z < 0$:

$$\omega = 360° - \omega \tag{1.9}$$

Wyznaczenie anomalii prawdziwej:

$$\theta = cos^{-1}\left(\frac{\mathbf{e}\cdot\mathbf{r}}{\|\mathbf{e}\|\|\mathbf{r}\|}\right) \tag{1.10}$$

Jeśli $\mathbf{r}\cdot\mathbf{v} < 0$:

$$\theta = 360° - \theta \tag{1.11}$$

Wyznaczenie półosi wielkiej:

$$r_p = \frac{h^2}{\mu}\frac{1}{1+\|\mathbf{e}\|\cos(0°)} \tag{1.12}$$

$$r_a = \frac{h^2}{\mu}\frac{1}{1+\|\mathbf{e}\|\cos(180°)} \tag{1.13}$$

$$a = \frac{r_p + r_a}{2} \tag{1.14}$$

1.3. Metoda Gaussa

Metoda Gaussa pozwala na wyznaczenie wektora stanu obiektu na podstawie trzech obserwacji [5]. Wymaga to pomiaru składającego się z się z czasu pomiaru t_i, wektora położenia obserwatora R_i oraz wersora kierunkowego obserwacji $\hat{\rho}_i$. Wersor kierunkowy można wyznaczyć na podstawie zaobserwowanych współrzędnych astronomicznych [6]. Schemat obserwacji pokazuje Rysunek. 1.2.

Kolejne kroki metody to:
Wyznaczenie międzyczasów:

$$\tau_1 = t_1 - t_2 \tag{1.15}$$

$$\tau = t_3 - t_1 \tag{1.16}$$

$$\tau_3 = t_3 - t_2 \tag{1.17}$$

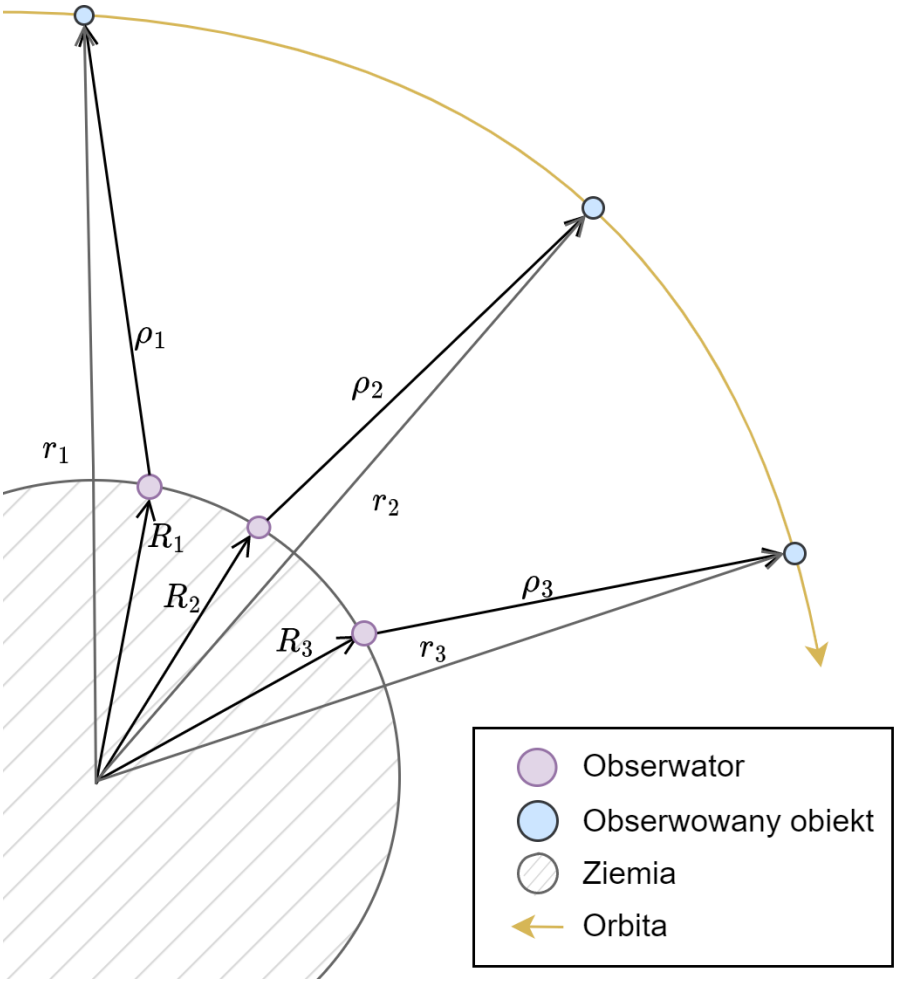
Wyznaczenie iloczynów:

$$\mathbf{p}_1 = \hat{\boldsymbol{\rho}}_2 \times \hat{\boldsymbol{\rho}}_3 \tag{1.18}$$

$$\mathbf{p}_2 = \hat{\boldsymbol{\rho}}_1 \times \hat{\boldsymbol{\rho}}_3 \tag{1.19}$$

$$\mathbf{p}_3 = \hat{\boldsymbol{\rho}}_1 \times \hat{\boldsymbol{\rho}}_2 \tag{1.20}$$

$$D_0 = \hat{\boldsymbol{\rho}}_1 \cdot \mathbf{p}_1 \tag{1.21}$$



Rysunek 1.2. Schemat wektorów używanych do wyznaczenia orbity metodą Gaussa

Wyznaczenie macierzy $\boldsymbol{D}_{3x3}$:

$$D_{ij} = \boldsymbol{R}_i \cdot \boldsymbol{p}_j \tag{1.22}$$

Wyznaczenie współczynników A i B:

$$A = \frac{1}{D_0}\left(-D_{12}\frac{\tau_3}{\tau} + D_{22} + D_{32}\frac{\tau_1}{\tau}\right) \tag{1.23}$$

$$B = \frac{1}{6D_0}\left(D_{12}(\tau_3^2 - \tau^2)\frac{\tau_3}{\tau} + D_{32}(\tau^2 - \tau_1^2)\frac{\tau_1}{\tau}\right) \tag{1.24}$$

Wyznaczenie współczynnika E:

$$E = \mathbf{R}_2 \cdot \hat{\boldsymbol{\rho}}_2 \tag{1.25}$$

Wyznaczenie współczynników a, b i c:

$$a = -(A^2 + 2AE + \|\mathbf{R}_2\|^2) \tag{1.26}$$

$$b = -2\mu B(A + E) \tag{1.27}$$

$$c = -\mu^2 B^2 \tag{1.28}$$

Wyznaczenie rzeczywistego, nieujemnego pierwiastka wielomianu i przyjęcie go jako wartość $\|\boldsymbol{r}_2\|$:

$$x^8 + ax^6 + bx^3 + c = 0 \tag{1.29}$$

Wyznaczenie wartości wektora obserwacji:

$$\|\boldsymbol{\rho}_2\| = A + \frac{\mu B}{\|\mathbf{r}_2\|^3} \tag{1.30}$$

$$\|\boldsymbol{\rho}_1\| = \frac{1}{D_0}\left(\frac{6\left(D_{31}\frac{\tau_1}{\tau_3} + D_{21}\frac{\tau}{\tau_3}\right)\|\mathbf{r}_2\|^3 + \mu D_{31}(\tau^2 - \tau_1^2)\frac{\tau_1}{\tau_3}}{6\|\mathbf{r}_2\|^3 + \mu(\tau^2 - \tau_3^2)} - D_{11}\right) \tag{1.31}$$

$$\|\boldsymbol{\rho}_3\| = \frac{1}{D_0}\left(\frac{6\left(D_{13}\frac{\tau_3}{\tau_1} + D_{23}\frac{\tau}{\tau_1}\right)\|\mathbf{r}_2\|^3 + \mu D_{13}(\tau^2 - \tau_3^2)\frac{\tau_3}{\tau_1}}{6\|\mathbf{r}_2\|^3 + \mu(\tau^2 - \tau_3^2)} - D_{33}\right) \tag{1.32}$$

$$\tag{1.33}$$

Wyznaczenie wektora położenia:

$$\mathbf{r}_1 = \mathbf{R}_1 + \|\boldsymbol{\rho}_1\|\hat{\boldsymbol{\rho}}_1 \tag{1.34}$$

$$\mathbf{r}_2 = \mathbf{R}_2 + \|\boldsymbol{\rho}_2\|\hat{\boldsymbol{\rho}}_2 \tag{1.35}$$

$$\mathbf{r}_3 = \mathbf{R}_3 + \|\boldsymbol{\rho}_3\|\hat{\boldsymbol{\rho}}_3 \tag{1.36}$$

Wyznaczenie współczynników Lagrange'a

$$f_1 = 1 - \frac{1}{2}\frac{\mu}{\|\mathbf{r}_2\|^3}\tau_1^2 \tag{1.37}$$

$$f_3 = 1 - \frac{1}{2}\frac{\mu}{\|\mathbf{r}_2\|^3}\tau_3^2 \tag{1.38}$$

$$f_1 = \tau_1 - \frac{1}{6}\frac{\mu}{\|\mathbf{r}_2\|^3}\tau_1^3 \tag{1.39}$$

$$g_3 = \tau_3 - \frac{1}{6}\frac{\mu}{\|\mathbf{r}_2\|^3}\tau_3^3 \tag{1.40}$$

$$\tag{1.41}$$

Wyznaczenie wektora prędkości:

$$\mathbf{v}_2 = \frac{1}{f_1 g_3 - f_3 g_1}\left(-f_3\mathbf{r_1} + f_1\mathbf{r_3}\right) \tag{1.42}$$

2. Cel i zakres projektu

Celem pracy było wykorzystanie wybranej metody wyznaczania orbity statku kosmicznego na podstawie obserwacji naziemnych. Zaimplementowano metodę Gaussa wraz z poprawką iteracyjną. Do sprawdzenia poprawności działania metody stworzono generator obserwacji, który na podstawie modelu keplerowskiego i założonych elementów orbitalnych wyznacza dane obserwacyjne. Wykonano również walidację korzystając z danych wygenerowanych przy pomocy zewnętrznego programu do symulacji nocnego nieba Stellarium [7].

3. Walidacja działania metody

Do wygenerowania danych użyto informacji z TLE (Two Line Element) Międzynarodowej Stacji Kosmicznej z 19 grudnia 2022, pobranych ze strony archiwizującej informacje o satelitach CelesTrak [8]:

```
1 25544U 98067A   22353.42711160  .00011227  00000-0  20403-3 0  9992
2 25544  51.6432 138.9346 0003592 174.7295 234.7524 15.50052736373915
```

Format TLE nie zawiera informacji o elementach orbitalnych zdefiniowanych tak jak w rozdziale 1.1. Zamiast półosi wielkiej podaje dodatkowe informacje takie jak średnia częstość kołowa, jej pierwszą i drugą pochodną, współczynnik ciśnienia promieniowania, anomalię średnią oraz informacje o obiekcie i orbicie [9].

Półoś wielką można wyznaczyć na podstawie częstości kołowej zgodnie ze wzorem:

$$a = \frac{\mu^{\frac{1}{3}}}{\left(\frac{2\pi}{24 \cdot 60 \cdot 60} n\right)^{\frac{2}{3}}} \tag{3.1}$$

Gdzie:

- n – częstość kołowa [obr/dzień]
- μ – GM standardowy parametr grawitacyjny dla Ziemi

Na podstawie danych TLE uzyskano klasyczne elementy orbitalne. Następnie wygenerowano trzy zestawy danych obserwacyjnych i wykonano 5 porównań:

- trzy obserwacje w pobliżu górowania, o niewielkich różnicach czasu między obserwacjami
- trzy obserwacje dokonane podczas wschodu, górowania i zachodu obserwowanego obiektu
- powyższe dwa przypadki z symulowanym błędem pomiaru
- sytuacja hipotetyczna - obserwacje równomiernie oddzielone na całej orbicie
- obserwacje dokonane w symulatorze nocnego nieba Stellarium

3.1. Wspólne stałe

Podczas wszystkich porównań przyjęto następujące wspólne stale:

$$\begin{aligned} \phi &= 51°23'15.49" \\ H &= 153m \\ R_E &= 6378km \\ f &= 0.003353 \\ \mu &= 3,986004418 \cdot 10^{14} \frac{m^3}{s^2} \end{aligned}$$

Gdzie:

- ϕ – szerokość geograficzna obserwatora
- H – wysokość nad poziomem morza obserwatora
- R_E – promień Ziemi
- f – spłaszczenie Ziemi
- θ – lokalny czas gwiazdowy obserwacji

Współrzędne kartezjańskie obserwatora dla każdej obserwacji wyznaczono ze wzoru:

$$\begin{aligned}\mathbf{R} = &\left[\frac{R_e}{\sqrt{1-(2f-f^2)sin^2\phi}}+H\right]\cdot\cos\phi(\cos\theta\hat{I}+\sin\theta\hat{J}) \\ &+\left[\frac{R_e(1-f)^2}{\sqrt{1-(2f-f^2)sin^2\phi}}+H\right]\cdot\sin\phi\end{aligned} \tag{3.2}$$

3.2. Obserwacja w pobliżu górowania

3.2.1. Dane obserwacyjne

Czas obserwacji odpowiada zmianie anomalii o kąt 10°:

$$t_1 = 0s$$
$$t_2 = 77s$$
$$t_3 = 154s$$

Czas gwiazdowy obserwacji:

$$\theta_1 = 301.7692°$$
$$\theta_2 = 302.0933°$$
$$\theta_3 = 302.4175°$$

Wykorzystane pozycje [km]:

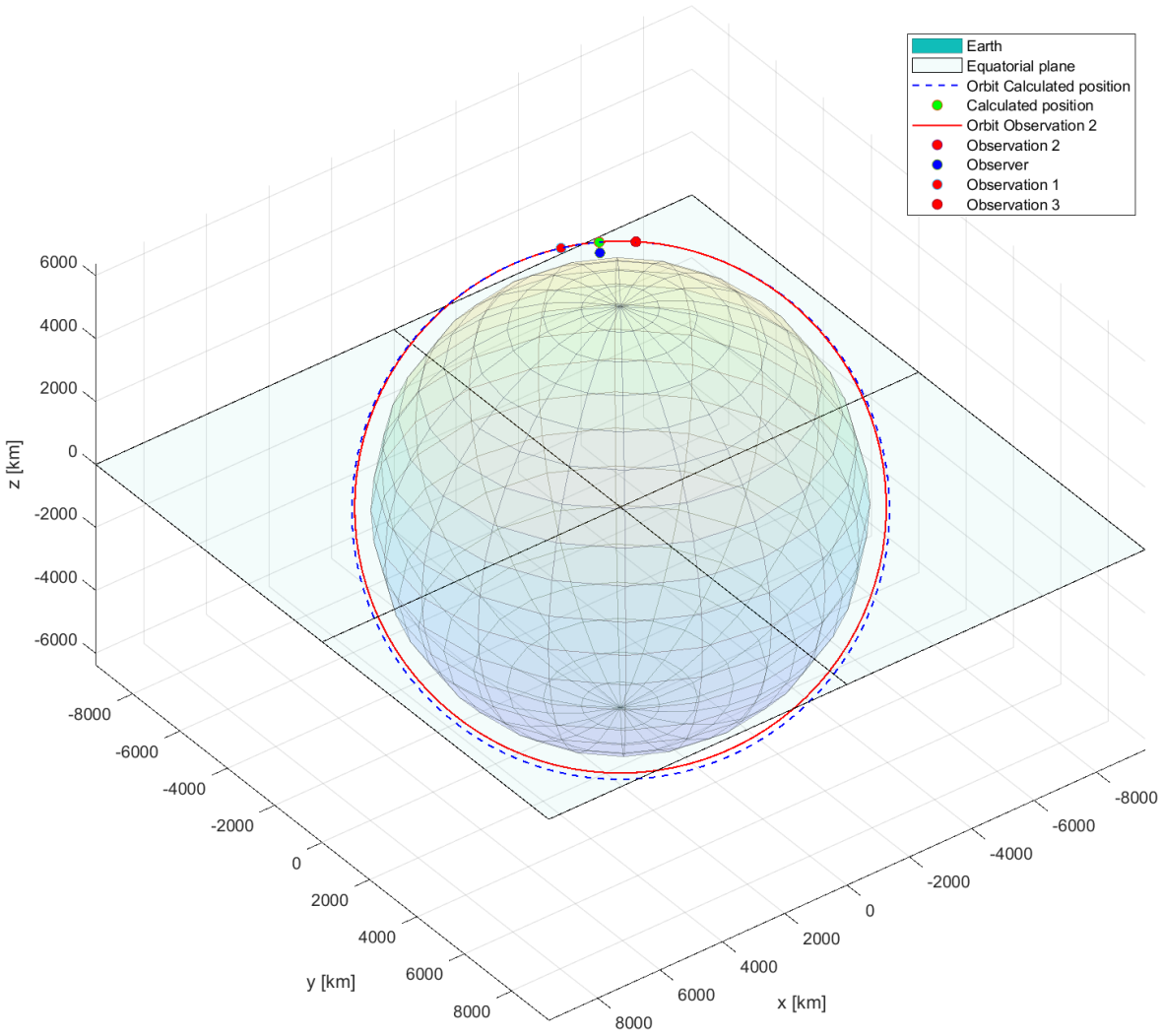
$$\begin{bmatrix} R_{1x} & R_{1y} & R_{1z} \\ R_{2x} & R_{2y} & R_{2z} \\ R_{3x} & R_{3y} & R_{3z} \end{bmatrix} = \begin{bmatrix} 338 & -5050 & 4530 \\ 843 & -5216 & 4269 \\ 1342 & -5342 & 3975 \end{bmatrix}$$

3.2.2. Wyniki

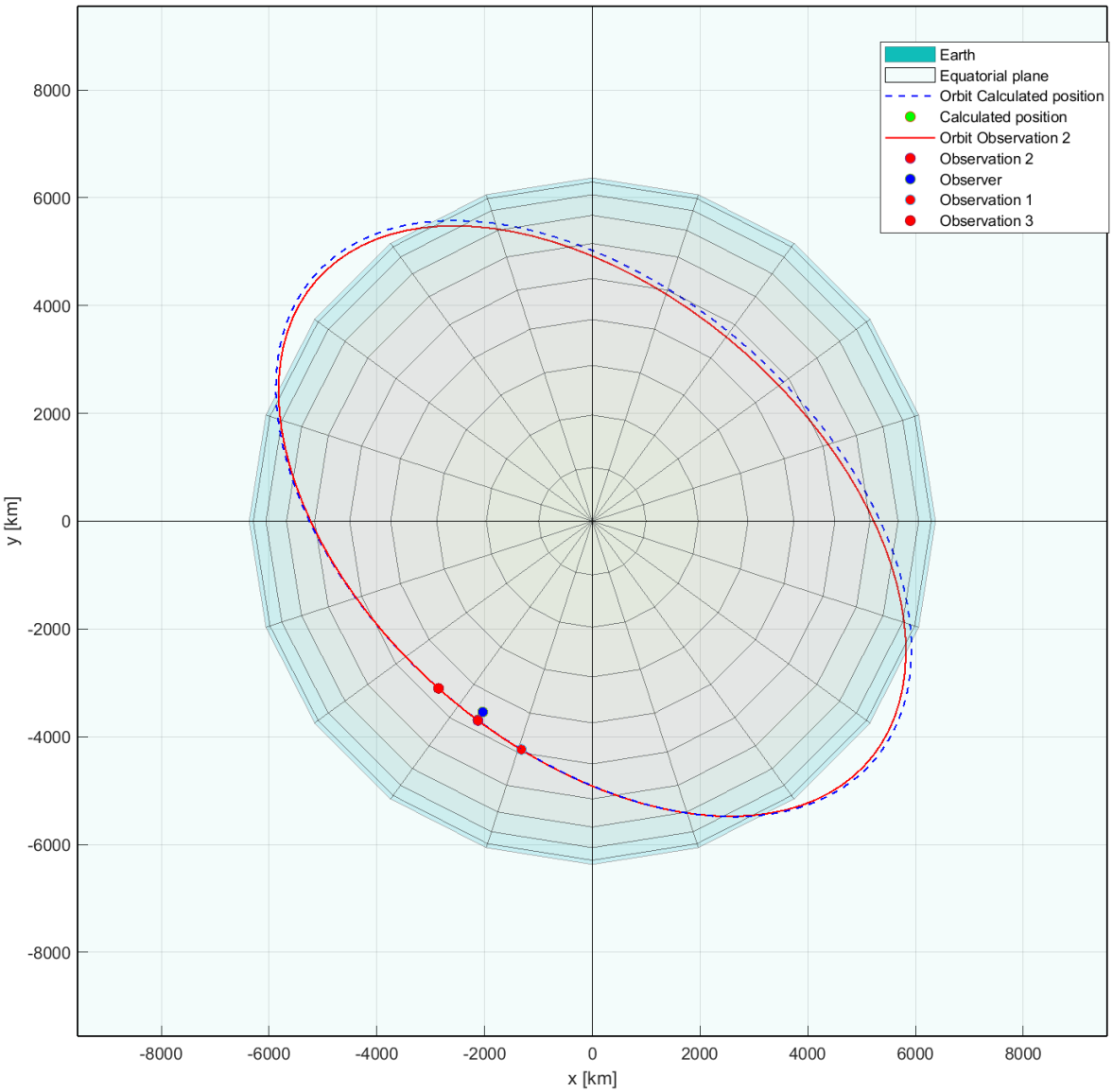
Wszystkie błędy są mniejsze niż 1e-06. Jest to zgodne z oczekiwaniami, ponieważ model obserwacyjny i metoda wyznaczania korzystają z idealnego modelu keplerowskiego.

Tabela 3.1. Krótki czas między obserwacjami - porównanie wyników

Parametr 1	Wartość referencyjna	Wartość wyznaczona	Błąd
Moduł położenia w 2 pozycji	6793 km	6793 km	1e-09 km
Moduł prędkości w 2 pozycji	7.7 km/s	7.7 km/s	5e-11 km/s
Mimośród	3.6e-04	3.6e-04	2e-12
Półoś wielka	6794 km	6794 km	2e-08
Inklinacja	51.64°	51.64°	3e-11°
Rektascensja węzła wstępującego	138.9°	138.9°	4e-11 °
Argument perycentrum	174.7°	174.7°	4e-07
Anomalia prawdziwa	312°	312°	4e-07



Rysunek 3.1. Krótki czas między obserwacjami - wizualizacja



Rysunek 3.2. Krótki czas między obserwacjami - rzut na płaszczyznę równikową

3.3. Obserwacja podczas pełnego przelotu

3.3.1. Dane obserwacyjne

Czas obserwacji odpowiada zmianie anomalii o kąt 30°:

$$t_1 = 0s$$
$$t_2 = 309s$$
$$t_3 = 619s$$

Czas gwiazdowy obserwacji:

$$\theta_1 = 300.7965°$$
$$\theta_2 = 302.0933°$$
$$\theta_3 = 303.3900°$$

Wykorzystane pozycje [km]:

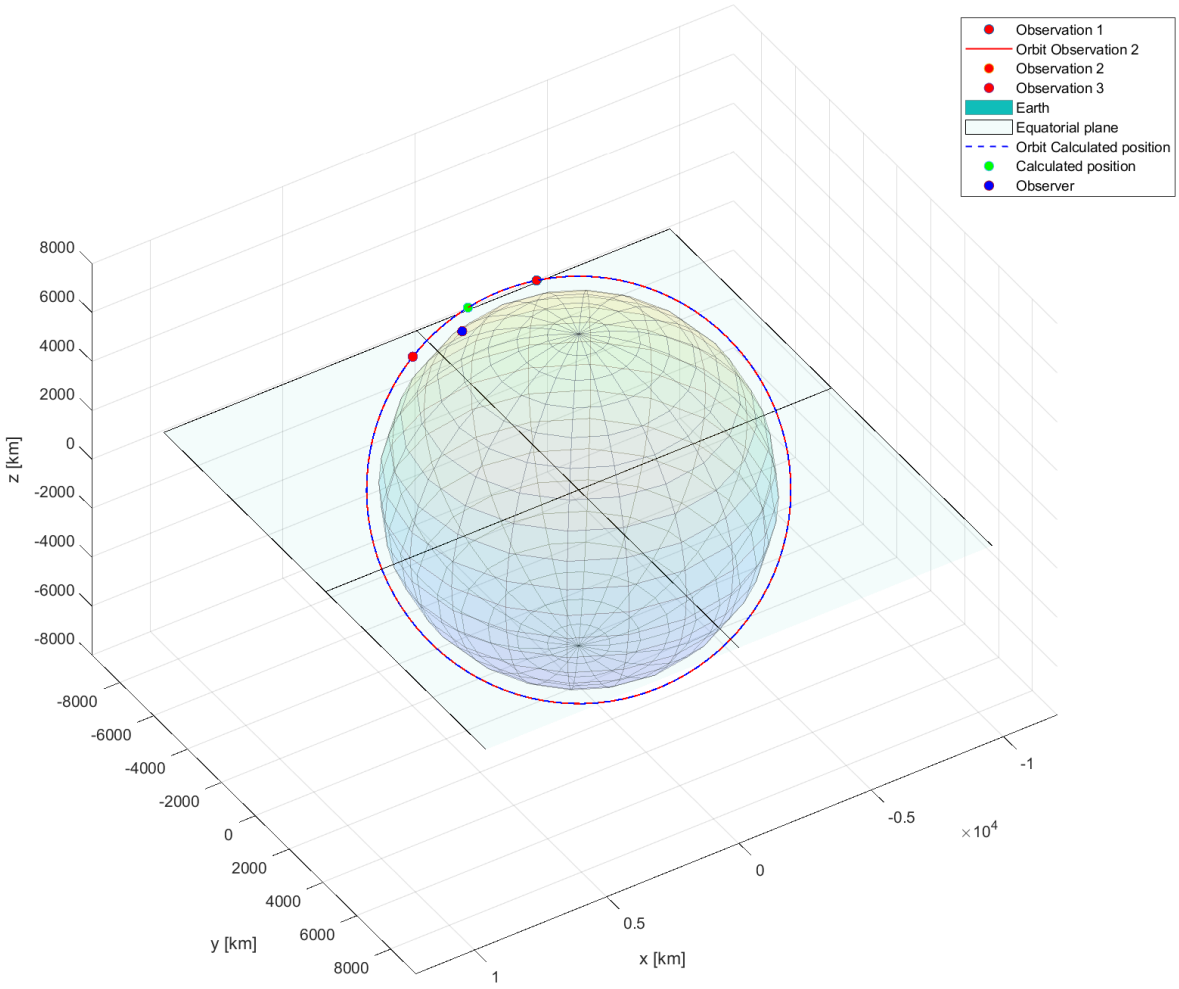
$$\begin{bmatrix} R_{1x} & R_{1y} & R_{1z} \\ R_{2x} & R_{2y} & R_{2z} \\ R_{3x} & R_{3y} & R_{3z} \end{bmatrix} = \begin{bmatrix} -1177 & -4328 & 5101 \\ 843 & -5216 & 4269 \\ 2762 & -5474 & 2922 \end{bmatrix}$$

3.3.2. Wyniki

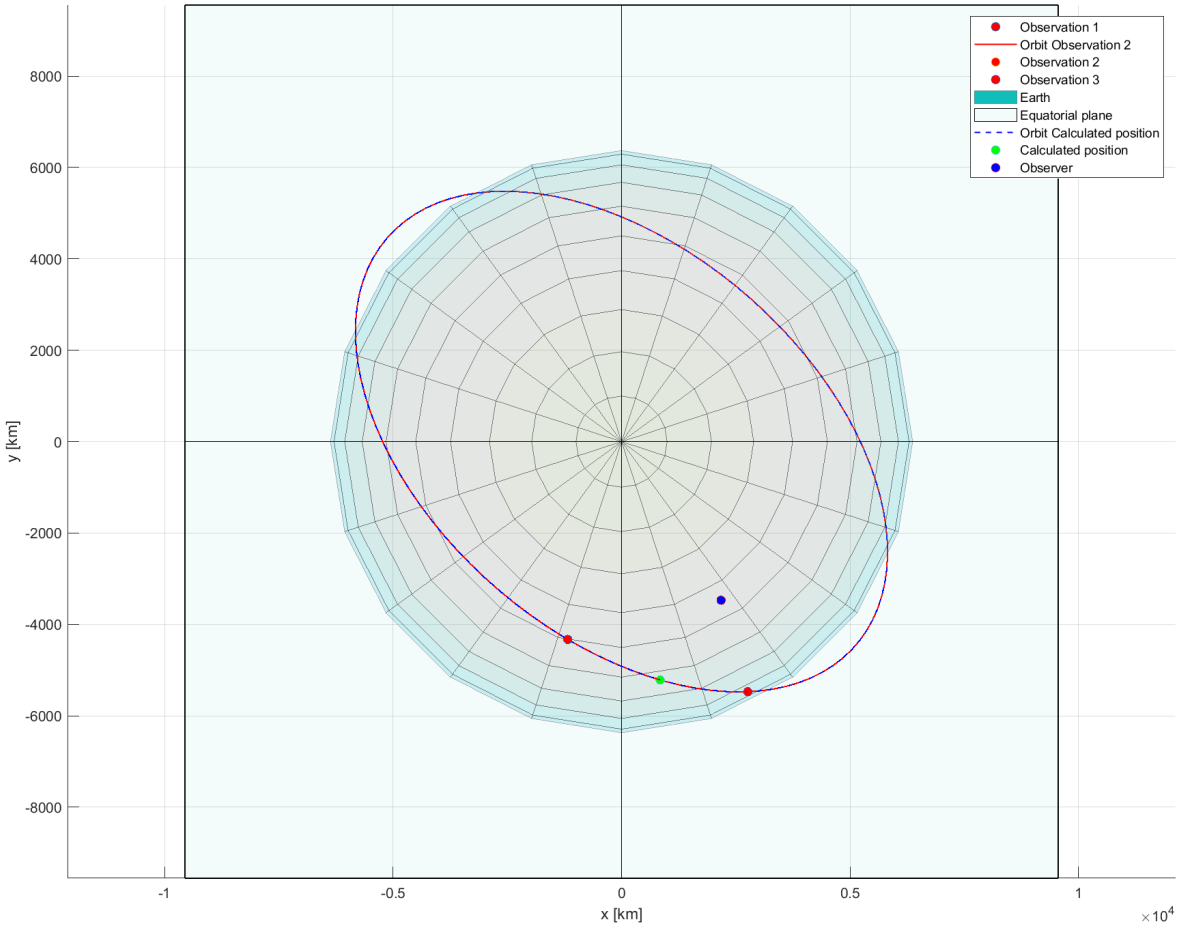
Wszystkie błędy są mniejsze niż 1e-06, podobnie jak dla obserwacji o krótkich odstępach między pomiarami.

Tabela 3.2. Obserwacje rozproszone - porównanie wyników

Parametr 1	Wartość referencyjna	Wartość wyznaczona	Błąd
Moduł położenia w 2 pozycji	6793 km	6793 km	1e-09 km
Moduł prędkości w 2 pozycji	7.7 km/s	7.7 km/s	5e-11 km/s
Mimośród	3.6e-04	3.6e-04	1e-11
Półoś wielka	6794 km	6794 km	9e-08
Inklinacja	51.64°	51.64°	3e-11°
Rektascensja węzła wstępującego	138.9°	138.9°	7e-13 °
Argument perycentrum	174.7°	174.7°	1e-06
Anomalia prawdziwa	312°	312°	1e-06



Rysunek 3.3. Obserwacje rozproszone - wizualizacja



Rysunek 3.4. Obserwacje rozproszone - rzut na płaszczyznę równikową

3.4. Obserwacje zakłócone

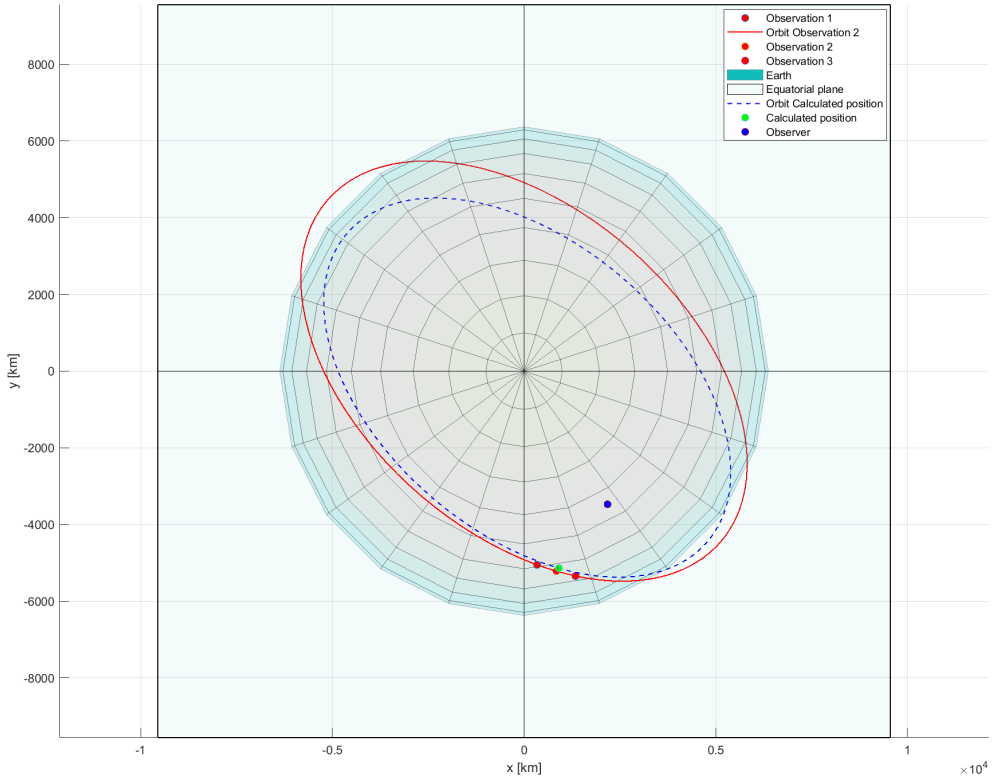
Wyniki uzyskane w podrozdziałach 3.2 i 3.3 sugerują, że rozkładanie obserwacji w czasie może nie mieć wpływu na wyniki, ponieważ nawet dla krótkich obserwacji błędy praktycznie nie istnieją. Jest to jednak spowodowane tym, że dane obserwacyjne są generowane w sposób idealny na podstawie idealnego modelu. Poniżej przedstawiono wyniki dla pomiarów nieidealnych.

3.4.1. Dane obserwacyjne

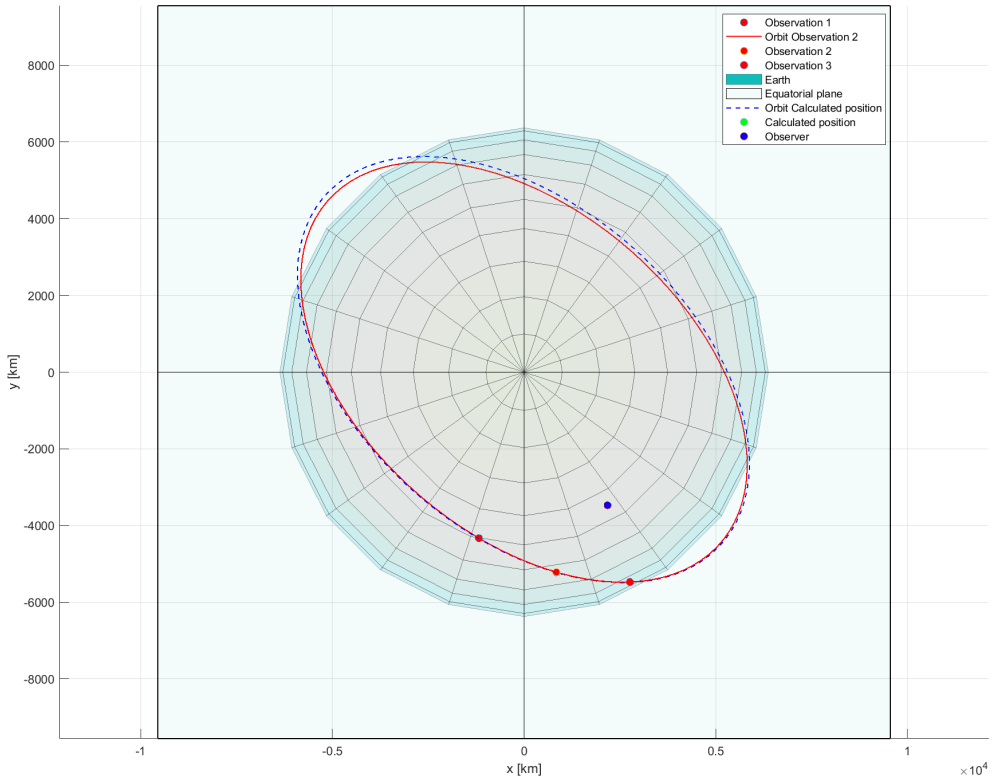
Takie jak w rozdziałach 3.2 i 3.3. Wersory kierunkowe zaszumiono dodając do każdej współrzędnej losowe liczby z przedziału [0; 0,002] i ponowownie normalizując.

3.4.2. Wyniki

Wyznaczone orbity zostały jakościowo porównane na wykresie 3.5. Zgodnie z oczekiwaniami przeprowadzenie pomiarów z większymi interwałami czasowymi pozwala na wyznaczenie orbity mimo nieidealnych danych.



(a) Krótki czas między obserwacjami



(b) Obserwacje rozproszone w czasie

Rysunek 3.5. Porównanie wpływu dodanego błędu do wersora kierunkowego obserwacji na określoną orbitę w dla dwóch częstości dokonywania pomiarów

3.5. Obserwacja hipotetyczna

3.5.1. Dane obserwacyjne

Czas obserwacji odpowiada zmianie anomalii o kąt 162:°:

$$t_1 = 0s$$
$$t_2 = 1269s$$
$$t_3 = 2538s$$

Czas gwiazdowy obserwacji:

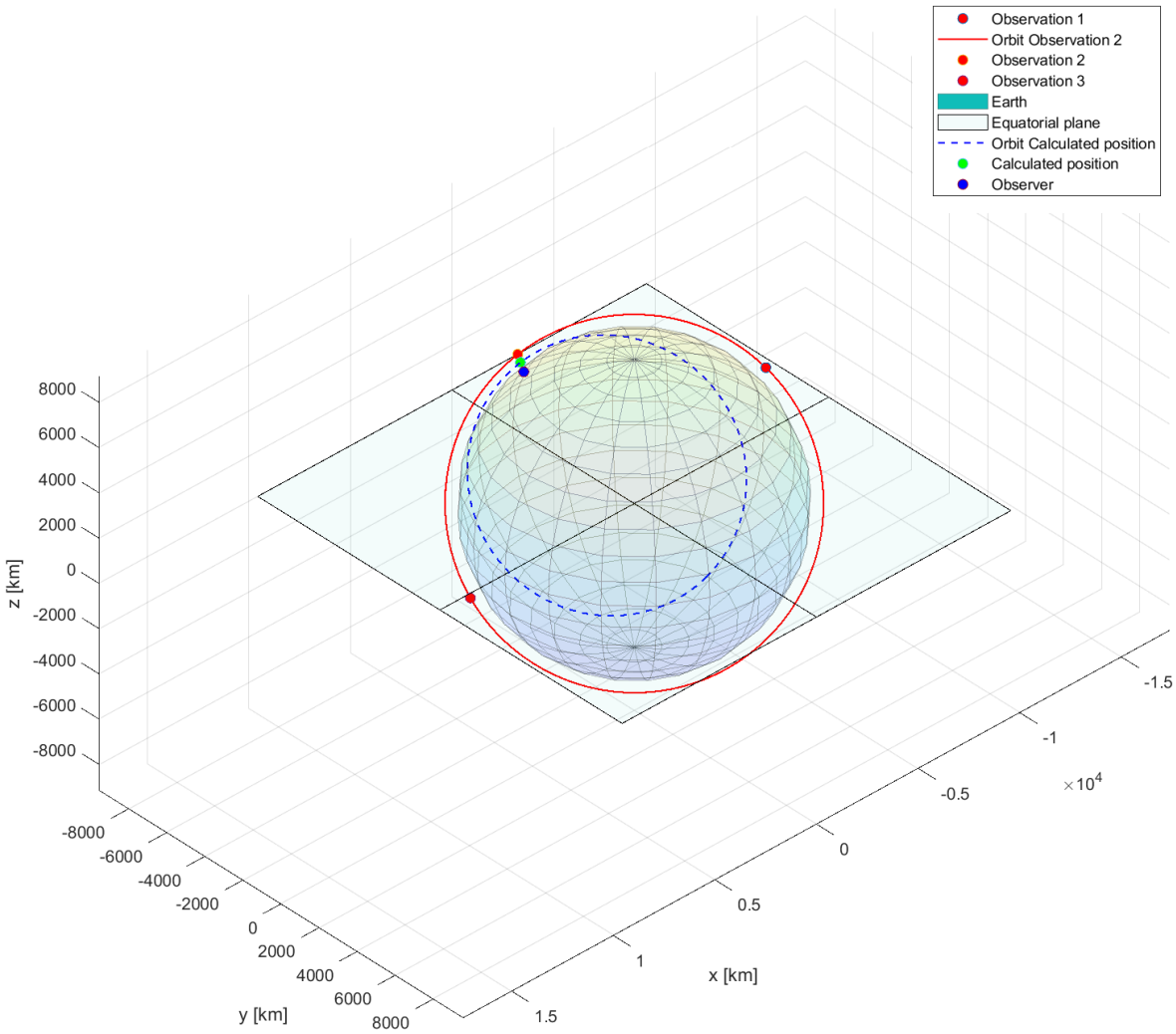
$$\theta_1 = 296.7743°$$
$$\theta_2 = 302.0933°$$
$$\theta_3 = 307.4090°$$

Wykorzystane pozycje [km]:

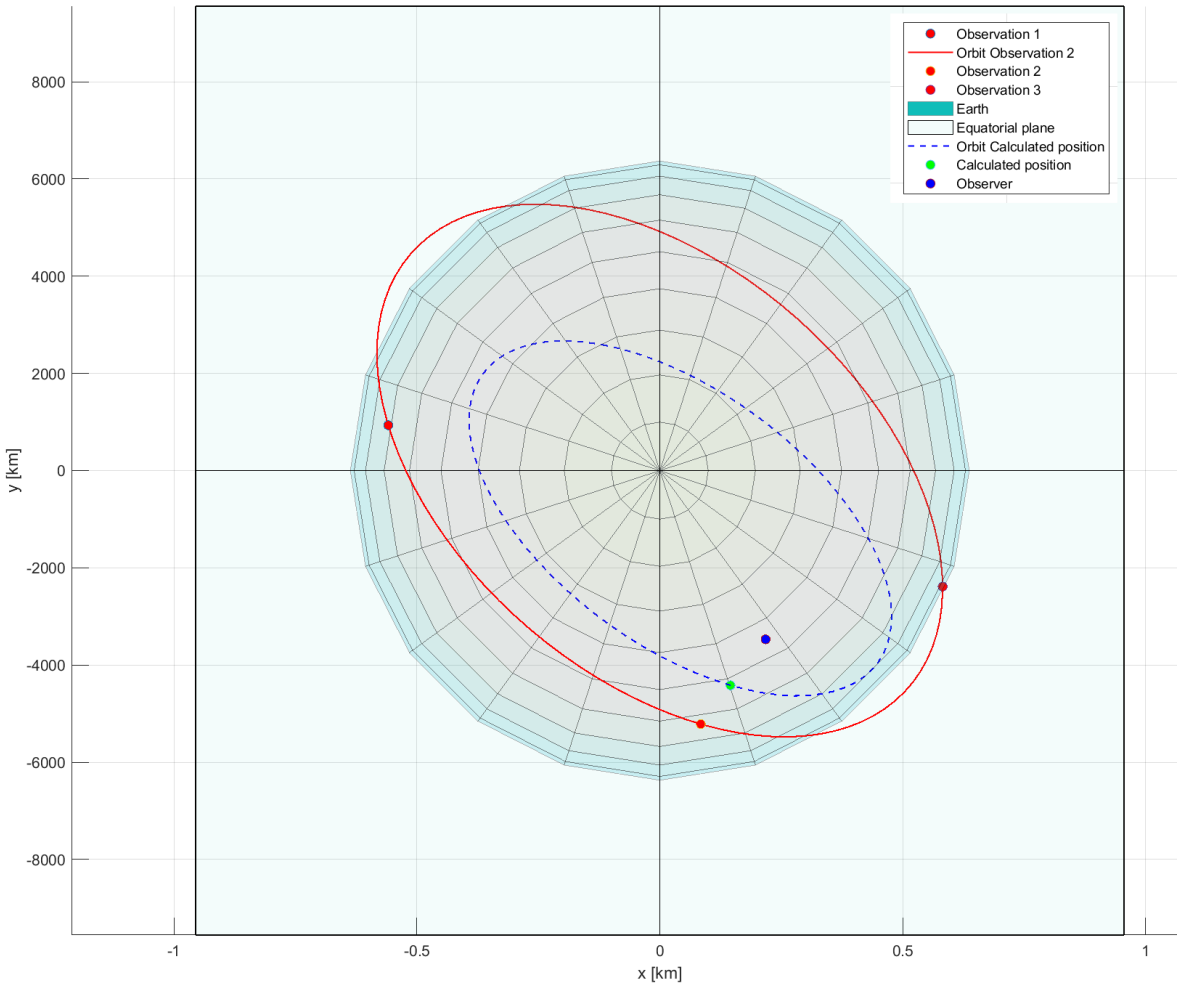
$$\begin{bmatrix} R_{1x} & R_{1y} & R_{1z} \\ R_{2x} & R_{2y} & R_{2z} \\ R_{3x} & R_{3y} & R_{3z} \end{bmatrix} = \begin{bmatrix} -5590 & 934 & 3750 \\ 843 & -5216 & 4269 \\ 5821 & -2385 & -2560 \end{bmatrix}$$

3.5.2. Wyniki

Ponieważ wyprowadzenie metody Gaussa wykorzystuje założenie, że czas między pomiarami jest niewielki względem okresu orbity [5], to jest ona nieskuteczna dla długich obserwacji. Zastosowanie poprawki iteracyjnej pozwala na powiększenie zakresu stosowalności, ale ostatecznie i ona zawodzi, gdy metoda podstawowa przestaje generować wystarczająco zbliżone wyniki. Dla przeprowadzonych symulacji maksymalna zmiana anomalii prawdziwej, dla których metoda zwracała poprawne wyniki, wynosiła 160°.



Rysunek 3.6. Obserwacje hipotetyczne - wizualizacja



Rysunek 3.7. Obserwacje hipotetyczne - rzut na płaszczyznę równikową

3.6. Obserwacja rzeczywista

Do obserwacji wykorzystano program Stellarium. Interfejs aplikacji pokazuje Rys. 3.8. Obserwatora umieszczono na współrzędnych 51°23’ 15"N, 18°34’ 10"E. Zaobserwowano przelot, który odbył się 19 grudnia 2022 między godziną 8:52 i 8:57 UTC. Czas wybrano z uwagi na zgodność daty z TLE i długi okres widoczności.

3.6.1. Dane obserwacyjne

Czas obserwacji:

$$t_1 = 52'24"$$
$$t_2 = 54'26"$$
$$t_3 = 56"35'$$

Czas gwiazdowy obserwacji:

$$\theta_1 = 15°28'27.1"$$
$$\theta_2 = 16°00'29.8"$$
$$\theta_3 = 16°02'39.0"$$

Wykorzystane pozycje [km]:

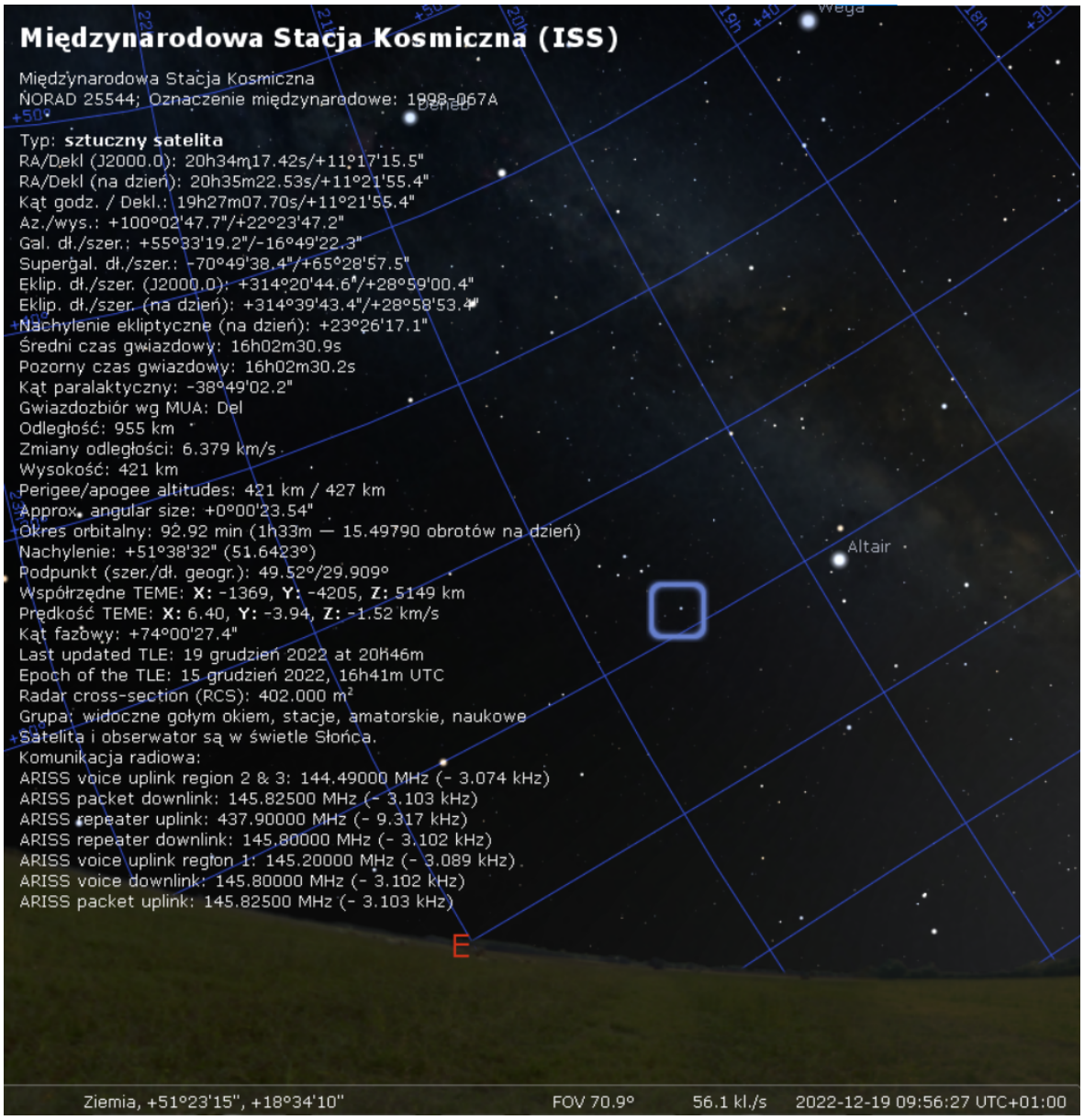
$$\begin{bmatrix} R_{1x} & R_{1y} & R_{1z} \\ R_{2x} & R_{2y} & R_{2z} \\ R_{3x} & R_{3y} & R_{3z} \end{bmatrix} = \begin{bmatrix} -2854 & -3102 & 5302 \\ -2127 & -3692 & 5284 \\ -1317 & -4237 & 5136 \end{bmatrix}$$

3.6.2. Wyniki

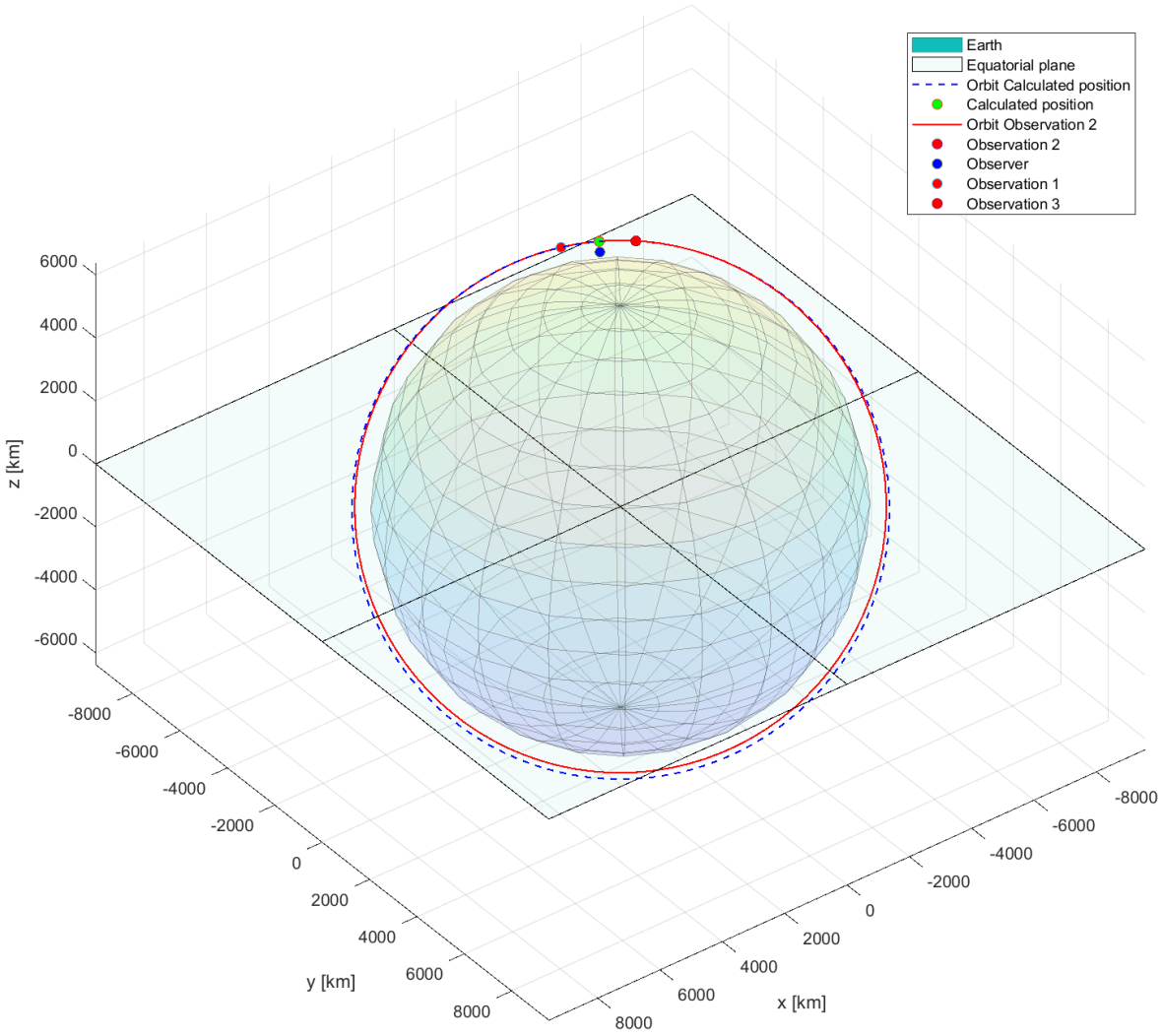
Pozycja wyznaczona przez program Stellarium pochodzi z bardziej zaawansowanego modelu, niż ten wykorzystywany do aproksymacji, co tłumaczy brak zgodności wyników. Mimo tego orbita została wyznaczone w przybliżeniu poprawnie, chociaż jest ona bardziej ekscentryczna.

Tabela 3.3. Obserwacja rzeczywista - porównanie wyników

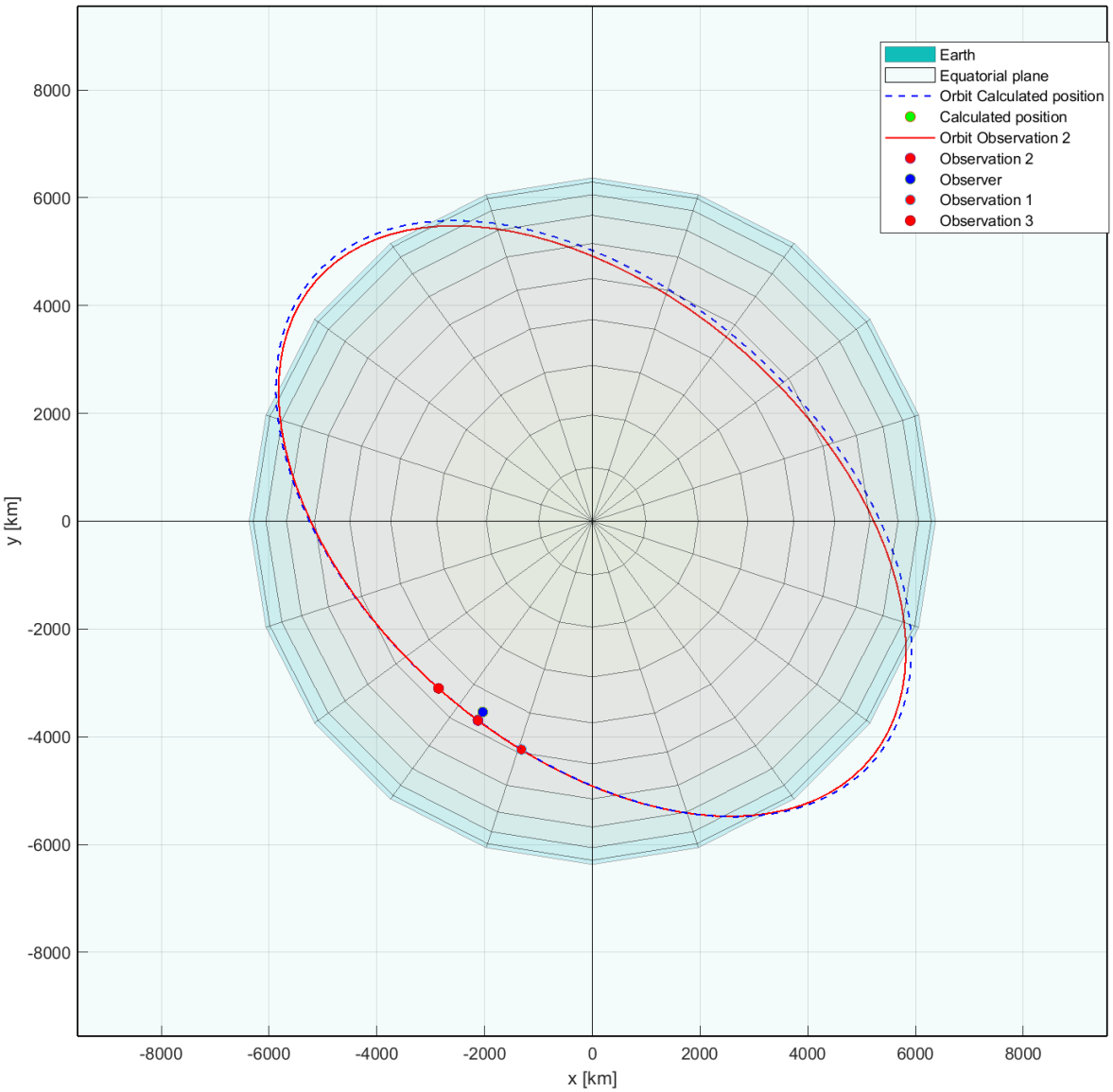
Parametr 1	Wartość referencyjna	Wartość wyznaczona	Błąd
Moduł położenia w 2 pozycji	6788 km	6789	0.77 km
Moduł prędkości w 2 pozycji	7.7 km/s	7.7 km/s	0.04 km/s
Mimośród	0.0004	0.0123	0.0119
Półoś wielka	6794 km	6873 km	78 km
Inklinacja	51.64°	51.63°	-0.018°
Rektascensja węzła wstępującego	138.9°	139.2°	0.23 °
Argument perycentrum	174.7°	97.6°	-76.9
Anomalia prawdziwa	282°	356°	74



Rysunek 3.8. Interfejs symulatora nocnego nieba Stellarium wykorzystanego do dokonania obserwacji



Rysunek 3.9. Obserwacja na podstawie symulatora nieba



Rysunek 3.10. Obserwacja na podstawie symulatora nieba - rzut na płaszczyznę równikową

4. Podsumowanie

W ramach przedstawionego zagadnienia sprawdzono stosowalność metody Gaussa do wstępnego wyznaczania orbit obiektów kosmicznych na podstawie obserwacji naziemnych. W tym celu zaimplementowano metodę w formie programu oraz przygotowano zbiory danych walidacyjnych. Metoda wykorzystuje założenie, że czasy między kolejnymi pomiarami są niewielkie względem okresu orbity i dla przypadków zgodnych z założeniami jest skuteczna. Jeśli międzyczasy są zbyt małe, to nawet drobne błędy w obserwacji mogą spowodować duże rozbieżności w obliczonej orbicie. Wykonanie pomiarów w trakcie jednego przebiegu orbity - podczas wschodu, zenitu i zachodu obiektu pozwala uzyskać optymalne wyniki, jednocześnie zachowując pewność, że obserwowany jest ten sam obiekt.

Bibliografia

[1] J. Vetter, „Fifty Years of Orbit Determination: Development of Modern Astrodynamics Methods”, *J. Hopkins Appl. Tech. D*, t. 27, sty. 2007.

[2] „Laplacian Orbit Determination and Differential Corrections”, *Celestial Mechanics and Dynamical Astronomy 2005 93:1*, t. 93, s. 53–68, 1 wrz. 2005, ISSN: 1572-9478. DOI: 10.1007/S10569-005-3242-6.

[3] D. Teets i K. Whitehead, „The Discovery of Ceres: How Gauss Became Famous”, *Mathematics Magazine*, t. 72, s. 83, 2 kw. 1999, ISSN: 0025570X. DOI: 10.2307/2690592.

[4] H. D. Curtis, *Orbital Mechanics for Engineering Students*. Elsevier Ltd, 2013, s. 159–161, ISBN: 9780080977478. DOI: 10.1016/C2011-0-69685-1.

[5] H. D. Curtis, *Orbital Mechanics for Engineering Students*. Elsevier Ltd, 2013, s. 238–250, ISBN: 9780080977478. DOI: 10.1016/C2011-0-69685-1.

[6] H. Bradt, *Astronomy Methods: A Physical Approach to Astronomical Observations*. Cambridge University Press, 2004, s. 34–44, ISBN: 9780521535519.

[7] *Stellarium - a free open source planetarium for your computer*, Dostęp zdalny (14.12.2022): https://stellarium.org/, 2022.

[8] *CelesTrak: Current GP Element Sets*, Dostęp zdalny (14.12.2022): https://celestrak.org/NORAD/elements/, 2022.

[9] F. R. Hoots i R. L. Roehrich, „SPACETRACK REPORT NO. 3 Models for Propagation of NORAD Element Sets”, 1980.

Spis rysunków

Spis tabel

Załączniki

4.1. Warianty symulacji

```
clc
clear
cla
%% Constants
mu = 3.986004418E14 /1000^3;
deg = pi/180;
Re = 6378; % Earth Radius
f = 0.003353; % Earth oblateness
%% ISS model
% Values from the web
TLE_first_line = {1 '25544U' '98067A' 22353.42711160  .00011227
    00000-0  20403-3 0  9992};
TLE_second_line = [2 25544  51.6432 138.9346 0003592 174.7295
    234.7524 15.50052736373915];
```

```
ISS = ISSClass(TLE_first_line, TLE_second_line);
ISS_OE = ISS.getOrbitalElements;
%% Observer's data
phi = deg*(51 + 13/60 + 15.49/3600); % geodetic longtitude east
lambda = deg*(18 + 34/60 + 10.70/3600); % latitude
H = 153; % [m] local altitude
%% Generated test data
cla

improvement = true; % iterative improvement

central_anomaly = 312;
danomaly = [5, 15, 82]; %
anomalies_sets = ...
    [central_anomaly-danomaly(1) central_anomaly central_anomaly+
        danomaly(1); ...
    central_anomaly-danomaly(2) central_anomaly central_anomaly+
        danomaly(2); ...
    central_anomaly-danomaly(3) central_anomaly central_anomaly+
        danomaly(3)].*deg;

%%= Test cases solving
for test_case = 1:height(anomalies_sets)
    clf

    anomalies = anomalies_sets(test_case, :);

    % Memory allocation
    t = zeros(1, 3);
    theta = zeros(3, 1);
    R = zeros(3, 3);
    ISS_rho = zeros(3, 3);
    rho = zeros(3, 3);

    % Observer's position vector at each observation
    %     R = findStationPosition(phi, theta, Re, f, H);
    %     R = ones(3)*100;
    first_obs_time = datetime("19-Dec-2022 9:30:00");

    % Time and direction cosines of each observation
    for i = 1:3
        t(i) = anomaly2time(anomalies(i), ISS_OE, mu);
        currentState = ISS.getStateVector(anomalies(i));
        ISS_position = currentState(:, 1);

        theta(i) = date2siderealTime(first_obs_time+seconds(t(i)),
            phi);
        R(:,i) = findStationPosition(phi, theta(i), Re, f, H);
        rho(:, i) = position2dirCosine(ISS_position, R(:,i));

        ISS.setAnomaly(anomalies(i));
```

```
61          %           if i == 2
            ISS.plot(sprintf("Observation %d", i), i==2);
            %           end


66      end
        t = t-t(1);

        % Proper calculations
        [r,v] = observation2state(R, rho, t, mu, improvement);
71      orbitalElements = state2orbitalElements(r, v, mu);

        %% Displays
        % Plots
        %     cla
76      hold on
        axis equal
        plotEarth();
        plotFromOE(orbitalElements, mu, r, "Calculated position", true);
        ISS.setAnomaly(anomalies(2));
81      %           ISS.plot();
        hold on
        %     plot3(R(1,3), R(2,3), R(3,3), 'o', 'MarkerFaceColor', 'red
            ', 'DisplayName', 'Observer')
        plot3(R(1,2), R(2,2), R(3,2), 'o', 'MarkerFaceColor', 'blue', '
            DisplayName', 'Observer')
        %     plot3(R(1,1), R(2,1), R(3,1), 'o', 'MarkerFaceColor', 'red
            ', 'DisplayName', 'Observer')
86

        % Data display
        measuredValues = [norm(r), norm(v), orbitalElements];
        trueValues = [vecnorm(ISS.state), ISS.orbitalElements];
        displayData("Measured", convertOE(measuredValues));
91      displayData("True", convertOE(trueValues));
        displayData("Error [measured-true]", (convertOE(measuredValues)-(
            convertOE(trueValues))));

    end
```

```
1   clc
    clear
    cla
    %% Constants
    improvement = true;
6

    mu = 3.986004418E14 /1000^3;
    deg = pi/180;
    Re = 6378; % Earth Radius
    f = 0.003353; % Earth oblateness
11  %% ISS model
```

```
% Values from the web
TLE_first_line = {1 '25544U' '98067A' 22353.42711160  .00011227
    00000-0  20403-3 0  9992};
TLE_second_line = [2 25544  51.6432 138.9346 0003592 174.7295
    234.7524 15.50052736373915];

ISS = ISSClass(TLE_first_line, TLE_second_line);
ISS_OE = ISS.getOrbitalElements;
%% Observer's data
phi = deg*(51+23/60+15.49/60/60); % geodetic longtitude east
H = 153; % [m] local altitude

theta = deg*[...
    15+28/60+27.1/3600; ...
    16+00/60+29.8/3600;...
    16+02/60+39.0/3600]/24*360; % local (mean) sidereal time [deg]

R = findStationPosition(phi, theta, Re, f, H);

%% Observation
t = [52*60+24, 54*60+26, 56*60+35]; % time [seconds]

real_R = [-2854, -3102, 5320; ...
    -2127, -3692, 5284; ...
    -1317, -4237, 5136]';
real_R_norm = vecnorm(real_R(:, 2));

real_V = [5.74, -5.08, 0.12; ...
    6.13, -4.55, -0.71; ...
    6.41,-3.90, -1.57]';
real_V_norm = vecnorm(real_V(:, 2));

real_state_vector = [real_R_norm, real_V_norm];

ISS.setAnomaly(deg*(-78+16/60+7.4/3600));

rho = zeros(3,3);
for i=1:3
    rho(:, i) = position2dirCosine(real_R(:,i), R(:,i));
end

[r,v] = observation2state(R, rho, t, mu, improvement);
orbitalElements = state2orbitalElements(r, v, mu);

%% Data display
measuredValues = [norm(r), norm(v), orbitalElements];
trueValues = [real_state_vector, ISS.orbitalElements];
displayData("Measured", convertOE(measuredValues));
displayData("True", convertOE(trueValues));
displayData("Error [measured-true]", (convertOE(measuredValues)-(
    convertOE(trueValues))));
```

```
% Plots
hold on
plotEarth();

plotFromOE(orbitalElements, mu, r, 'Calculated position', true)

hold on
axis equal

ISS.plot("Observation 2", true);

plot3(R(1,2), R(2,2), R(3,2), 'o', 'MarkerFaceColor', 'blue', '
    DisplayName', 'Observer')

plot3(real_R(1,3), real_R(2,3), real_R(3,3), ...
    'o', 'MarkerFaceColor', 'red', 'DisplayName', 'Observation 1')
% plot3(real_R(1,2), real_R(2,2), real_R(3,2), ...
%     'o', 'MarkerFaceColor', 'red', 'DisplayName', 'Observation 2')
plot3(real_R(1,1), real_R(2,1), real_R(3,1), ...
    'o', 'MarkerFaceColor', 'red', 'DisplayName', 'Observation 3')
```

4.2. Algorytm Gaussa

```
function [r2, v2] = observation2state(R, rho, t, mu,
    iterativeImprovementOn)
% returns state vector
% R - observer's position vectors
% rho - direction cosine vectors
% t - observation times

% time intervals
tau = [t(1)-t(2); t(3)-t(1); t(3)-t(2)];
% cross products p
p = [cross(rho(:,2), rho(:,3)), cross(rho(:,1), rho(:,3)), cross(rho
    (:,1), rho(:,2))];
% weird D
D0 = dot(rho(:,1), p(:,1));
D = R.'*p;
% A and B
A = 1/D0 * (-D(1,2) * tau(3)/tau(2) + D(2,2) + D(3,2) * tau(1)/tau(2)
    );
B = 1/6/D0 * (D(1,2)*(tau(3)^2-tau(2)^2)*tau(3)/tau(2) + D(3,2)*(tau
    (2)^2-tau(1)^2)*tau(1)/tau(2));
% E
E = dot(R(:,2), rho(:,2));
% abc coefficients
a = -(A^2+2*A*E+dot(R(:,2), R(:,2)));
b = -2*mu*B*(A+E);
c = -mu^2*B^2;
```

```
% roots
r = roots([1, 0, a, 0, 0, b, 0, 0, c]);
r2Norm = max(r(real(r)>0 & imag(r) == 0)); % real solutions larger
    than Earth Radius

if length(r2Norm) > 1
    error("More than one solution to the 8th order polynomial
        calculated. Proceeding is not supported.")
else
    if isempty(r2Norm)
        error("No solution found.");
    end
end

% lengths of cosine directions
rhoNorm(1) = 1/D0 * ( (6*(D(3,1)*tau(1)/tau(3) + D(2,1)*tau(2)/tau(3)
    )*r2Norm^3  ...
    + mu*D(3,1)*(tau(2)^2 - tau(1)^2)*tau(1)/tau(3) ) ...
    / (6*r2Norm^3 + mu*(tau(2)^2 - tau(3)^2)) - D(1,1) ) ;
rhoNorm(2) = A + mu*B/r2Norm^3;
rhoNorm(3) = 1/D0 * ( (6*(D(1,3)*tau(3)/tau(1) - D(2,3)*tau(2)/tau(1)
    )*r2Norm^3  ...
    + mu*D(1,3)*(tau(2)^2 - tau(3)^2)*tau(3)/tau(1) ) ...
    / (6*r2Norm^3 + mu*(tau(2)^2 - tau(3)^2)) - D(3,3) ) ;
% r vector
r = R + rho .* rhoNorm;
% Langrage coefficients
f(1) = 1 - 0.5 * mu/r2Norm^3 * tau(1)^2;
f(3) = 1 - 0.5 * mu/r2Norm^3 * tau(3)^2;
g(1) = tau(1) - mu/r2Norm^3*tau(1)^3/6;
g(3) = tau(3) - mu/r2Norm^3*tau(3)^3/6;
% v2
v2 = (f(1)*r(:,3)-f(3)*r(:,1))/(f(1)*g(3)-f(3)*g(1));
r2 = r(:,2);

if iterativeImprovementOn == true
    iter = 0;
    iterMax = 1000;
    maxError = 1E-8;
    error = [1,1,1];
    fgNew = [f(1), f(3), g(1), g(3)];

    while sum(error < maxError) ~= 3 && iter < iterMax
        [r2, v2, newRhoNorm, fgNew] = refineStateMeasurement(r2, v2,
            tau, rho, R, D, D0, mu, fgNew);
        error = abs(newRhoNorm - rhoNorm);
        rhoNorm = newRhoNorm;
        iter = iter + 1;
    end

end
```

```
end
```

```
function [r2, v2, rhoNorm, fgNew] = refineStateMeasurement(r2, v2,
    tau, rho, R, D, D0, mu, fgOld)
% iterative improvement of the orbit determined by observation2state.
    m
% r2 - position vector of object in second observation
% v2 - velocity vector of object in second observation
% mu - gravitational paremete

% vector magnitudes
rNorm = norm(r2);
vNorm = norm(v2);

% semimajor axis reciprocal (1/a)
alpha = 2/rNorm - vNorm^2/mu;

% radial component of v
vRadial = dot(v2, r2/rNorm);

% universal Kepler's equation (Algorithm 3.3, Equation 3.46)
x1 = solveUniversalKepler(tau(1), rNorm, vRadial, alpha, mu);
x3 = solveUniversalKepler(tau(3), rNorm, vRadial, alpha, mu);

% f, g, c
z1 = alpha * x1^2;
z3 = alpha * x3^2;

f(1) = 1 - x1^2 / rNorm * StumpffC(z1);
f(3) = 1 - x3^2 / rNorm * StumpffC(z3);
g(1) = tau(1) - 1/sqrt(mu) * x1^3 * StumpffS(z1);
g(3) = tau(3) - 1/sqrt(mu) * x3^3 * StumpffS(z3);

f(1) = mean([f(1), fgOld(1)]);
f(3) = mean([f(3), fgOld(2)]);
g(1) = mean([g(1), fgOld(3)]);
g(3) = mean([g(3), fgOld(4)]);
fgNew = [f(1), f(3), g(1), g(3)];

c1 = g(3)/(f(1)*g(3) - f(3)*g(1));
c3 = -g(1)/(f(1)*g(3) - f(3)*g(1));

% new rho
rhoNorm(1) = 1/D0 * (-D(1,1) + D(2,1)/c1 - D(3,1)*c3/c1);
rhoNorm(2) = 1/D0 * (-c1*D(1,2) + D(2,2) - D(3,2)*c3);
rhoNorm(3) = 1/D0 * (-c1/c3*D(1,3) + D(2,3)/c3 - D(3,3));

r = R + rho .* rhoNorm;
r2 = r(:, 2);
```

```
v2 = (f(1)*r(:,3)-f(3)*r(:,1))/(f(1)*g(3)-f(3)*g(1));

end
```

```
function orbitalElements = state2orbitalElements (r, v, mu)
% returns orbital elements from the state vector
% r = [x y z]
% v = [vx vy vz]


% Distance
rNorm = norm(r);
% Speed
vNorm = norm(v);
% Radial velocity
vr = dot(r, v) / rNorm;
% Specific angular momentum
h = cross(r, v);
hNorm = norm(h);
% Inclination
i = acos(h(3)/hNorm);
% Node line vector
N = cross([0 0 1], h);
NNorm = norm(N);
% RA of ascending node
if N(2) >= 0
    omega = acos(N(1)/NNorm);
else
    omega = 2*pi - acos(N(1)/NNorm);
end
% eccentricity
e = 1/mu * (cross(v,h) - mu * r/rNorm);
eNorm = norm(e);
% perigee argument
if e(3) >= 0
    w = acos(dot(N, e)/NNorm/eNorm);
else
    w = 2*pi - acos(dot(N, e)/NNorm/eNorm);
end
% true anomaly
if vr >= 0
    theta = acos(dot(e, r)/eNorm/rNorm);
else
    theta = 2*pi - acos(dot(e, r)/eNorm/rNorm);
end

orbitalElements = [hNorm, i, omega, eNorm, w, theta];
% hNorm = specific angular momentum
% i = inclination
% omega = RA of ascending node
```

```
% eNorm = eccentricity
% w = perigee argument
% theta = true anomaly

return
end
```

4.3. Uniwersalne równanie Keplera

```
function x = solveUniversalKepler(dt, r0, vr0, a, mu)
%SOLVEUNIVERSALKEPLER Solves universal Kepler equation
% dt
% r0 - radial position
% vr0 - radial velocity
% a - reciprocal of semimajor axis
% mu - gravitational parameter
% x - universal anomaly

error = 1E-8;
maxIter = 1000;

% initial X estimate
x = sqrt(mu)*abs(a)*dt;

ratio  = 1;
iter = 0;
while abs(ratio) > error && iter <= maxIter
    % Stumpff functions
    z = a*x*x;
    C = StumpffC(z);
    S = StumpffS(z);

    F = r0*vr0/sqrt(mu)*x^2*C + (1 - a*r0)*x^3*S + r0*x - sqrt(mu)*dt
        ;
    dF = r0*vr0/sqrt(mu)*x*(1 - a*x^2*S) + (1 - a*r0)*x^2*C + r0;

    ratio = F/dF;

    x = x - ratio;
    iter = iter+1;
end
end
```

```
function C = StumpffC(z)
%STUMPFFC calculates Stumpff C coefficient

if z > 0
    C = (1 - cos(sqrt(z))) / z;
elseif z < 0
```

```
    C = (cosh(sqrt(-z)) - 1) / (-z);
else
    C = 0.5;
end

end
```

```
function C = StumpffC(z)
%STUMPFFC calculates Stumpff C coefficient

if z > 0
    C = (1 - cos(sqrt(z))) / z;
elseif z < 0
    C = (cosh(sqrt(-z)) - 1) / (-z);
else
    C = 0.5;
end

end
```

4.4. Konwersja danych

```
function R = findStationPosition(phi, theta, Re, f, H)
%FINDSTATIONPOSITION converts from geocentral coordinates to
    cartesian (ECI)
% phi = geodetic latitude
% theta = local sidereal time
% Re = equatorial radius
% f - oblateness = (Re - Rp) / Re where:
% Rp = polar radius
% H = local altitude

theta = theta.'; % to horizontal
multiplier = Re / sqrt(1-(2*f - f*f)*sin(phi)^2);

R = [(multiplier + H) * cos(phi) * cos(theta); ...
    (multiplier + H) * cos(phi) * sin(theta);...
    (multiplier * (1-f)^2 + H) * sin(phi) .* ones(length(theta),1)
        .'];

end
```

```
function rho = angles2directionCosines(alfa, delta)
%ANGLES2DIRECTIONCOSINES converts astro coordinates angles (
    declination and RA) to direction
% vectors
rho = [cos(delta).*cos(alfa) cos(delta).*sin(alfa) sin(delta)].';
end
```

```
function orbitalElements = convertOE(orbitalElements)
% CONVERTOE converts OE from h to semimajor axis (a)

mu = 3.986004418E14 /1000^3;
h = orbitalElements(3);
e = orbitalElements(6);

rp = h^2/mu * 1/(1+e*cos(0));
ra = h^2/mu * 1/(1+e*cos(pi));
orbitalElements(3) = 0.5*(ra+rp);


end
```

4.5. Symulacja ISS

```
classdef ISSClass < handle
    %ISSCLASS simulates ISS state based on TLE

    properties
        orbitalElements
        state
        epochTLE
        mu = 3.986004418E14 /1000^3;
    end

    methods
        function obj = ISSClass(TLE_first_line, TLE_second_line)
            %             TLE_first_line = {1 '25544U' '98067A'
                22250.60379146  .00007699  00000-0  14232-3 0
                9993};
            %             TLE_second_line = [2 25544  51.6443
                288.2831 0002670 203.2971 293.6689
                15.50084907357968];
            obj.updateOrbitalElements(TLE_second_line );
            obj.state = obj.getStateVector(obj.orbitalElements(6));
            obj.epochTLE = obj.readEpoch(convertStringsToChars(string
                (TLE_first_line{4})));
        end

        function updateOrbitalElements(obj, TLE_second_line)
            deg = pi/180;

            inclination = TLE_second_line(3) * deg;
            RAofAscendingNode = TLE_second_line(4) * deg;
            eccentricity = TLE_second_line(5) * 10E-8;
            perigeeArgument = TLE_second_line(6) * deg;

            meanMotion = TLE_second_line(8) * 2*pi/24/60/60; %
                revolutions/day -> rad/s
```

```
            semiMajorAxis = (obj.mu / meanMotion^2)^(1/3);
            semiMinorAxis = semiMajorAxis * (1-eccentricity^2)^0.5;
            h = meanMotion * semiMajorAxis * semiMinorAxis;

            obj.orbitalElements = [h, inclination, RAofAscendingNode,
                 eccentricity, perigeeArgument, 0];
        end

        function epochTLE = readEpoch(obj, epoch)
            day = epoch(1:5);
            epochTLE = ...
                datetime(day, 'InputFormat', 'yyDDD') + ...
                days(mod(str2double(epoch), 1));
        end

        function plot(obj, point_name, plotAdditional)
            plotFromOE(obj.orbitalElements, obj.mu, obj.state(:,1),
                point_name, plotAdditional);
        end

        function plotSinglePoint(obj)
            name = sprintf("ISS - true anomaly %f deg", obj.
                orbitalElements(6)/pi*180);
            plot3(obj.state(1,1), obj.state(2,1), obj.state(3,1), 'o'
                , 'DisplayName', name);
        end

        %% set
        function setAnomaly(obj, anomaly)
            obj.orbitalElements(6) = anomaly;
            obj.state = obj.getStateVector(obj.orbitalElements(6));
        end
        %% get
        function stateVector = getStateVector(obj, anomaly)
            obj.orbitalElements(6) = anomaly;
            [r,v] = orbitalElements2state(obj.orbitalElements, obj.mu
                );
            stateVector = [r,v];
        end

        function orbitalElements = getOrbitalElements(obj)
            orbitalElements = obj.orbitalElements;
        end

        function mu = getMu(obj)
            mu = obj.mu;
        end

    end
end
```

4.6. Wyświetlanie wyników

```
function displayData(adjective, measuredValues)
%DISPLAYDATA fprints vector in form of [rNorm, vNorm, orbitalElements
    ]

% rad2deg
deg = 180/pi;
measuredValues(4) = measuredValues(4) * deg;
measuredValues(5) = measuredValues(5) * deg;
measuredValues(7) = measuredValues(7) * deg;
measuredValues(8) = measuredValues(8) * deg;

fprintf(['-------------------\n%s parameters:\n' ...
    'R(2): %g km \n' ...
    'V(2): %.2g km/s \n' ...
    'semimajor axis: %g km \n' ...
    'inclination: %.4g deg \n' ...
    'RA of ascending node: %.4g deg \n' ...
    'eccentricity: %g  \n' ...
    'perigee argument: %.4g deg \n' ...
    'true anomaly(2): %.4g deg \n-------------------\n'], ...
    adjective, measuredValues);

end
```

```
function plotEarth()
%PLOTEARTH plots Earth

% plotting Earth

earthR = 6371; %km
[X,Y,Z] = sphere;
X = X * earthR;
Y = Y * earthR;
Z = Z * earthR;
surf(X,Y,Z, 'DisplayName', 'Earth', 'FaceAlpha', 0.1, 'EdgeAlpha',
    0.3);
% Equatorial plane
% maxR = max(abs(R), [], 'all');
maxR = max(abs(earthR*1.5), [], 'all');
[x, y] = meshgrid(-maxR:maxR:maxR);
z = zeros(size(x,1));
surf(x,y,z, 'FaceAlpha', 0.2, 'DisplayName', 'Equatorial plane', '
    FaceAlpha', 0.05);

grid on
legend()
```

```
22 end
```

```
function orbitPlot = plotFromOE(orbitalElements, mu, r, name,
    plotAdditional)
%PLOTFROMOE plots the orbit from orbirtal elements vector
% orbitalElements:
% (1) hNorm = specific angular momentum
% (2) i = inclination
% (3) omega = RA of ascending node
% (4) eNorm = eccentricity
% (5) w = perigee argument
% (6) theta = true anomaly
% name - name of plotted point
% plotAdditional - true/false - should plot apse line and orbit?
h = orbitalElements(1);
i = orbitalElements(2);
omega = orbitalElements(3);
e = orbitalElements(4);
w = orbitalElements(5);

% perigee
rp = h^2/mu * 1/(1+e*cos(0));
% apogee
ra = h^2/mu * 1/(1+e*cos(pi));

% perifocal r
theta = linspace(0, 2*pi);
R = h^2/mu * 1 ./ (1+ e*cos(theta)) .* [cos(theta); sin(theta); zeros
    (1, length(theta))];

rotation = ...
    [cos(w) sin(w) 0; -sin(w) cos(w) 0; 0 0 1] * ...
    [1 0 0; 0 cos(i) sin(i); 0 -sin(i) cos(i)] * ...
    [cos(omega) sin(omega) 0; -sin(omega) cos(omega) 0; 0 0 1];

% geocentric R
R = rotation.' * R;

%% plotting
hold on

if name == "Calculated position"
    plotstyle = 'b--';
    dotcolor = "green";
else
    dotcolor = "red";
    plotstyle = 'r-';
end
```

```
47 if plotAdditional
       % plotting the orbit

       plot3(R(1,:), R(2,:), R(3,:), plotstyle, 'DisplayName', strcat("
           Orbit ", name), 'LineWidth', 1);

52


       % % Apse line
       % apsePoints = [ [rp*cos(0); 0; 0], [0; 0; 0], [ra*cos(pi); 0; 0]
            ];
       % apsePoints = rotation.' * apsePoints;
57     % plot3(apsePoints(1,:), apsePoints(2,:), apsePoints(3,:), 'o--k
           ', 'MarkerFaceColor', 'black', ...
       %     'DisplayName', 'Apse line')

   end

62 % Initial state
   plot3(r(1), r(2), r(3), 'o', 'MarkerFaceColor', dotcolor, '
       DisplayName', name)
   % Axes
   axis equal
   xlabel('x [km]')
67 ylabel('y [km]')
   zlabel('z [km]')

   % Initial state geo

72

   end
```