WYDZIAŁ MECHANICZNY ENERGETYKI I LOTNICTWA

Praca przejściowa

Analiza misji do planet Układu Słonecznego

Autor

Filip Perczyński

Prowadzący

dr inż. Mateusz Żbikowski

WARSZAWA 2018

Spis treści

1. Wprowadzenie

1.1. Cel projektu

W ramach niniejszego projektu został opracowany program w języku Python, w opraciu o bibliotekę Poliastro, który pozwala na podstawową analizę misji kosmicznych do planet Układu Słonecznego. W pierwszym etapie tworzenia programu głównym celem było umożliwienie podstawowych analiz dla każdej z planet Układu Słonecznego oraz odpowiednia, estetyczna forma wizualizacji wyników w postaci dwu- i trójwymiarowych wykresów orbit oraz trajektorii lotu, a także wykresów przedstawiających zmianę w czasie parametrów i umożliwiających ich porównanie.

Program umożliwia określenie trajektorii lotu bezpośredniego z Ziemi do wybranej planety Układu Słonecznego oraz pozwala na wybranie najkorzystniejszej daty startu ze względu na ilość zużytego materiału pędnego.

W kolejnych etapach pracy nad programem planowane jest rozszerzanie jego funkcjonalności np. wprowadzenie obliczeń transferu Hohmanna i asysty grawitacyjnej. Pozwoli to na bardziej różnorodne analizy i symulację misji bliższą rzeczywistości, szczególnie w przypadku odległych planet, gdzie w misjach wykorzystywana jest asysta grawitacyjna.

2. Przegląd wybranych misji międzyplanetarnych

Pierwszą w pełni udaną misją planetarną była misja NASA sondy Mariner 2, która zbliżyła się do Wenus 14 grudnia 1962 roku. Kolejnymi planetami, do których wysłane zostały z sukcesem sondy były: Mars - sonda Mariner 4 (zbliżenie w 1965 roku), Jowisz - Pioneer 10 (zbliżenie w 1973 roku), Saturn - Pioneer 11 (zbliżenie w 1979 roku). W 1973 roku Mariner 10 został pierwszym próbnikiem, który dotarł do Merkurego oraz pierwszym, który wykorzystał manewr asysty grawitacyjnej. Do Urana oraz Neptuna dotarła sonda Voyager 2 - zbliżenie do Urana w 1986 roku i Neptuna w 1989 roku.

W XXI wieku misje międzyplanetarne skupiają się przede wszystkim na Marsie ze względu na jego bliskie położenie oraz rozpatrywane misje załogowe. Oprócz Marsa celem pojedynczych misji były również planety Wenus czy Jowisz. W poniższej tabeli przedstawione zostały wybrane udane misje międzyplanetarne oraz określony został cel misji np. sonda orbitująca wokół planety, lądownik lub tylko przelot, czyli asysta grawitacyjna.

Tabela. 1 Przegląd wybranych misji kosmicznych do planet Układu Słonecznego. [1][2]

Planeta	Nazwa misji	Start	Osiągnięcie celu	Dni	Rodzaj misji
Merkury	MESSENGER	2004-08-03	2006-10-24	812	Asysta grawitacyjna
Wenus	Magellan	1989-05-04	1990-08-10	463	Orbiter
	Venus Express	2005-11-09	2006-04-11	153	Orbiter
Mars	ExoMars TGO	2016-03-14	2016-10-19	219	Orbiter, lądownik
	MAVEN	2013-11-18	2014-09-22	308	Orbiter
	Mars Orbiter Mission	2013-11-05	2014-11-24	384	Orbiter
	MSL Curiosity	2011-11-26	2012-08-06	254	łazik
	Phoenix	2007-08-04	2008-05-25	295	lądownik
	Mars Reconnaissance Orbiter	2005-08-12	2006-03-10	210	Orbiter
	MER-B Opportunity	2003-07-07	2004-01-25	202	łazik
	MER-A Spirit	2003-06-10	2004-01-04	208	łazik
	Beagle 2	2003-06-02	2003-12-25	206	lądownik
	Mars Express			206	Orbiter
	2001 Mars Odyssey	2001-04-07	2001-10-24	200	Orbiter
Jowisz	Galileo	1989-10-18	1995-10-08	2181	Orbiter
	Cassini-Huygens	1997-10-15	2000-12-30	1172	Asysta grawitacyjna
	New Horizons	2006-01-19	2007-02-28	405	Asysta grawitacyjna
	Juno	2011-08-05	2016-07-05	1796	Orbiter
Saturn	Voyager 1	1977-09-05	1980-11-12	1164	Asysta grawitacyjna
	Voyager 2	1977-08-20	1981-08-25	1466	Asysta grawitacyjna
	Cassini-Huygens	1997-10-15	2001-07-01	1355	Asysta grawitacyjna
Uran	Voyager 2	1977-08-20	1986-01-24	3079	Asysta grawitacyjna
Neptun	Voyager 2	1977-08-20	1989-08-25	4388	Asysta grawitacyjna

2.1.Misja Venus Express

Venus Express to pierwsza sonda Europejskiej Agencji Kosmicznej wysłana w kierunku planety Wenus. Sonda o masie właściwej 577 kg została wystrzelona na orbitę okołoziemską na pokładzie rakiety Soyuz 9-11-2005 roku. Sonda osiągnęła orbitę planety 11-04-2006 roku. Po dalszych manewrach z użyciem silnika głównego sonda znalazła się na docelowej orbicie 7 maja 2006 roku. Zmiana prędkości sondy konieczna do wejścia na orbitę wyniosła około 1.3 km/s. [3]

Badania naukowe sondy zostały przedłużone z 2007 do 2014 roku. Jednym z głównych osiągnięć misji Venus Express było wykazanie, że wiry polarne na Wenus są najbardziej zmiennymi w całym Układzie Słonecznym. 18-01-2015 roku ustał kontakt z sondą, która zgodnie z planem weszła w atmosferę planety.

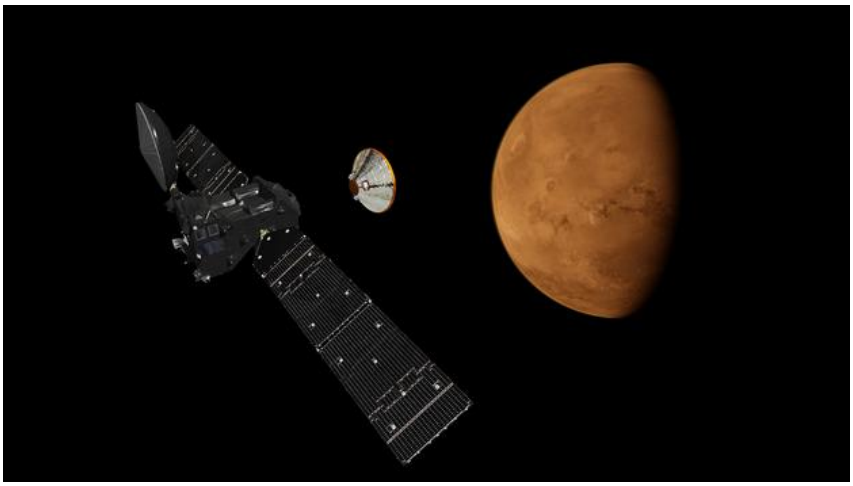
Rys. 1 Wizualizacja sondy Venus Express na tle planety Wenus. [4]

2.2. Misja ExoMars

Misja ExoMars to wspólna inicjatywa badawcza Europejskiej Agencji Kosmicznej i Rosyjskiej Agencji Kosmicznej Roskosmos, szukająca śladów procesów biologicznych i geologicznych na Marsie. Misja składająca się z orbitera TGO - Trace Gas Oriter i lądownika Schiaparelli wystartowała 14 marca 2016 z kosmodromu Bajkonur na rakiecie Proton M.

Sonda dotarła do Marsa 19-10-2016, a orbiter wszedł na orbitę Marsa. Niestety lądownik Schiaparelli w wyniku nieudanego lądowania rozbił się na powierzchni planety.

Na pokładzie sondy i lądownika znajdują się instrumenty wykonane w Polsce np. zasilacz do kamery CaSSIS wykonany przez Centrum Badań Kosmicznych PAN.



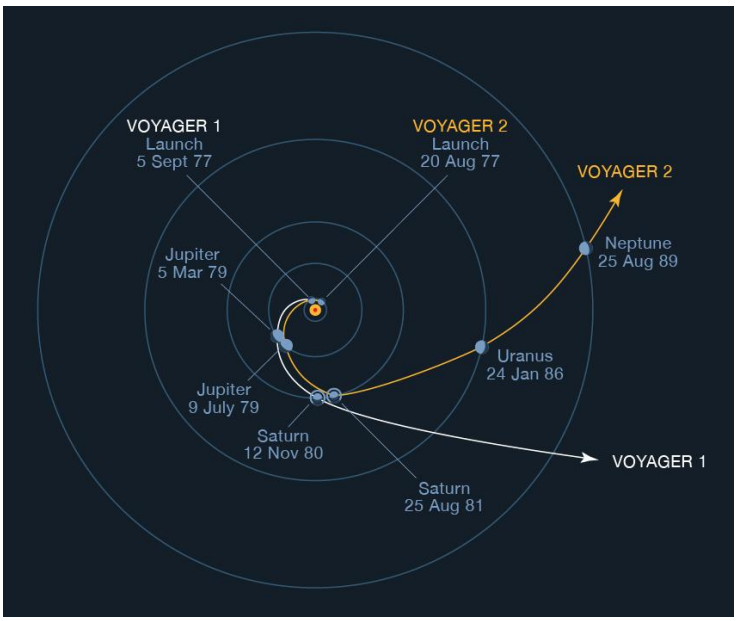
Rys. 2 Wizualizacja lądownika Schiaparelli oddzielającego się od orbitera TGO. [5]

2.3.Misje Voyager

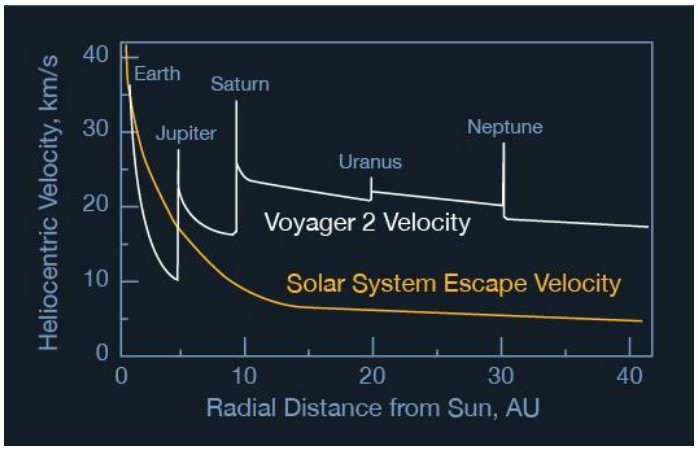
Voyager to bezzałogowy program badawczy, poświęcony badaniom zewnętrznych planet Układu Słonecznego i zewnętrznej części heliosfery za pomocą dwóch bliźniaczych sond kosmicznych.

Sondy Voyager 1 i Voyager 2 wystrzelono w 1977 roku przy użyciu rakiet Titan 3E-Centaur. Pierwszymi celami były badania Jowisza (w 1979 roku) i Saturna (w latach 1980 i 1981). Voyager 2 przeleciał także obok Urana (w 1986 roku) i Neptuna (w 1989 roku), czego nie dokonała dotąd żadna inna sonda. Kombinacja asyst grawitacyjnych użyta przez sondę Voyager 2 (Jowisz, Saturn, Uran i Neptun) zdarza się raz na 176 lat. Dzięki wykorzystaniu manewrów c zas przelotu do Neptuna lub Plutona został skrócony o około 20 lat w stosunku do lotu bezpośredniego.

Voyager 2 bada obecnie najdalsze obszary heliosfery, natomiast Voyager 1 przekroczył w 2012 roku heliopauzę i przesyła dane z przestrzeni międzygwiezdnej. Obie misje dostarczyły bardzo wielu informacji o planetach-olbrzymach Układu Słonecznego, ich księżycach i pierścieniach. W roku 1998 Voyager 1 pod względem oddalenia od Słońca wyprzedził sondę Pioneer 10 (zmierzającą w przeciwnym kierunku) i stał się najdalszym sztucznym obiektem w kosmosie



Rys. 3 Trajektoria lotu misji Voyager 1 i Voyager 2. [1]



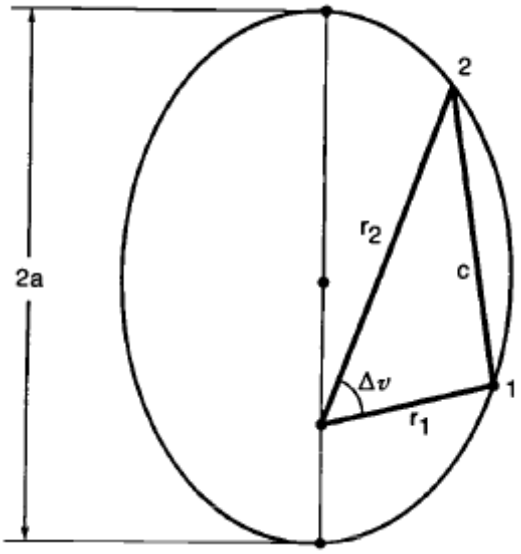
Rys. 4 Wykres prędkości Voyager'a 1 i 2. [1]

3. Wyznaczanie trajektorii międzyplanetarnych

3.1. Zagadnienie Lamberta

Zagadnienie Lamberta opisuje problem znalezienia orbity transferowej łączącej dwa wektory pozycji w określonym przedziale czasu. Teoria Lamberta zakłada, że czas transferu zależy jedynie od półosi wielkiej orbity transferowej, sumy promieni do punktów 1 i 2 ($r_1 + r_2$) oraz odległości cięciwy między punktami. Na poniższym rysunku przedstawiona została geometria orbity transferowej.

$$c = \sqrt{r_1^2 + r_2^2 - 2r_1 r_2 cos\Delta v}$$



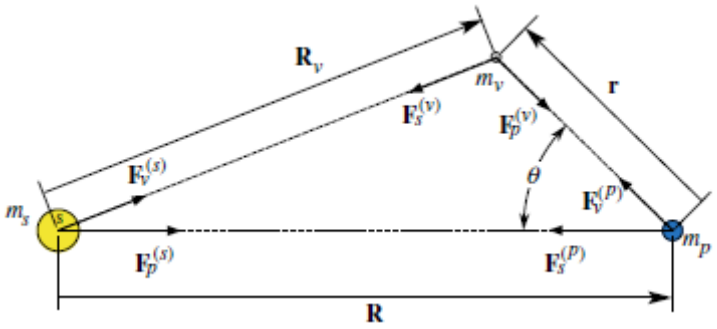
Rys. 5 Geometria orbity transferowej według zagadnienia Lamberta. [6]

3.2. Metoda "patched conics"

Trajektorię lotu można wyznaczać również za pomocą metody "patched conics'. Pozwala ona na podzielenie misji międzyplanetarnej na trzy etapy:

- wydostanie się z wpływu grawitacyjnego planety – trajektoria hiperboliczna,
- lot po eliptycznej orbicie heliocentrycznej,
- wejście w oddziaływanie grawitacyjne planety docelowej – trajektoria hiperboliczna.

Z metodą tą wiąże się pojęcie sfery oddziaływania inaczej wpływu - SOI - Sphere of Influence. Jest to obszar wokół planety, w którym ruch pojazdu kosmicznego przestajemy traktować jako heliocentryczny, ze względu na dominujące działanie siły przyciągania planety. [7]



Rys. 6 Schemat oddziaływania między trzema ciałami. [8]

Zasięg strefy wpływu grawitacyjnego r_{SOI} można obliczyć ze wzoru:

$$\frac{r_{SOI}}{R} = \left(\frac{m_p}{m_s}\right)^{2/5}$$

gdzie R- odległość planety od Słońca,

m_p - masa planety,

m_s - masa Słońca.

Dla Ziemi $r_{SOI} = 925 * 10^3\ km = 145\ R_e$

3.3. Manewry międzyplanetarne

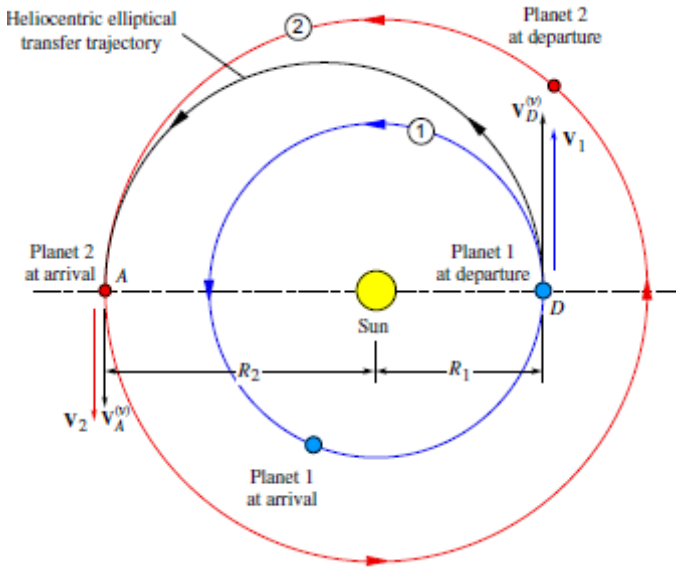
3.3.1. Transfer międzyplanetarny Hohmanna

Transfer Hohmanna stanowi najbardziej efektywny transfer pomiędzy planetami. Zmiana prędkości wymagana do manewru Hohmanna wynosi dla impulsu w perycentrum elipsy łączącej obie orbity (początkową i docelową):

$$\Delta V_1 = \sqrt{\frac{\mu}{R_1}}\left(\sqrt{\frac{2R_2}{R_1+R_2}} - 1\right)$$

Natomiast dla impulsu w apocentrum orbity:

$$\Delta V_2 = \sqrt{\frac{\mu}{R_2}}\left(1 - \sqrt{\frac{2R_2}{R_1+R_2}}\right)$$



Rys. 7 Transfer międzyplanetarny Hohmanna z planety wewnętrznej do zewnętrznej.[8]

Aby zastosować transfer Hohmanna musi wystąpić odpowiednia konfiguracja planet (okno startowe). W tym celu wprowadzony został parametr okres synodyczny T_{syn} , który określa przedział czasu, po którym powtórzy się konfiguracja planet:

$$T_{syn} = \frac{T_1 T_2}{|T_1 - T_2|}$$

gdzie T_1, T_2 - okresy obiegu planet wokół Słońca

Przykładowo dla Ziemi i Marsa okres synodyczny jest równy 777.9 dnia.

3.3.2. Transfer z małym ciągiem

Napęd jonowy w przeciwieństwie do napędu chemicznego nie wykorzystuje do zmiany orbity dużej impulsowej zmiany ciągu, ale ciągłą małą zmianę ciągu przez długi okres czasu. Całkowita zmiana prędkości potrzebna do wykonania transferu z małym ciągiem między orbitami kołowymi jest przybliżana równaniem Edelbauma:

$$\Delta V^2 = V_{c1}^2 + V_{c2}^2 - 2V_{c1}V_{c2}\cos\left(\frac{\pi\alpha}{2}\right)$$

$$\alpha = \Delta i$$

gdzie, ΔV - całkowita zmiana prędkości potrzebna do wykonania transferu małym ciągiem,
V_{c1} - prędkość na kołowej orbicie początkowej,
V_{c2} - prędkość na kołowej orbicie końcowej,
Δi - zmiana inklinacji między orbitami.

W przypadku, gdy orbity początkowa i końcowa mają tę samą inklinację zmiana prędkości:

$$\Delta V = \left|\overrightarrow{V_{c1}} - \overrightarrow{V_{c2}}\right|$$

Do obliczenia trajektorii lotu z orbity wokół Ziemi na orbitę Księżyca zastosowano metodę opisaną w [8]. Najpierw obliczone zostały prędkości początkowa i końcowa:

$$v_{c1} = \sqrt{\frac{\mu}{r_1}}$$

$$v_{c2} = \sqrt{\frac{\mu}{r_2}}$$

gdzie μ - parametr grawitacyjny Ziemi, r - odległość od środka Ziemi.

Następnie na podstawie wymaganej zmiany inklinacji między orbitą startową i końcową obliczane jest przyspieszenie transferu z małym ciągiem - f , prędkość początkowa - v_o, końcowa - v_f , całkowita zmiana prędkości i początkowy kąt odchylenia ciągu - β_o.

$$tan\beta_o = \frac{\sin\left(\frac{\pi}{2}\Delta i\right)}{\frac{v_o}{v_f} - \cos\left(\frac{\pi}{2}\Delta i\right)}$$

Czas transferu t_f można oszacować z zależności:

$$t_f = \frac{\Delta v}{f}$$

Parametry statku kosmicznego wzdłuż trajektorii lotu są obliczane za pomocą kolejnych równań z określonym krokiem czasowym:

$$\Delta V = ft$$

$$\beta = \tan^{-1}\left(\frac{v_o \sin(\beta_o)}{v_o \cos(\beta_o) - ft}\right)$$

$$v = \sqrt{(v_o^2 - 2v_o ft cos(\beta_o) + f^2 t^2)}$$

$$\Delta i = \frac{2}{\pi}\left[\tan^{-1}\left(\frac{ft - v_o cos\beta_o}{v_o sin\beta_o}\right) + \frac{\pi}{2} - \beta_o\right]$$

Zakładane jest, że statek w każdej chwili znajduje się na orbicie kołowej o promieniu r obliczanym z zależności:

$$r = \frac{\mu}{v^2}$$

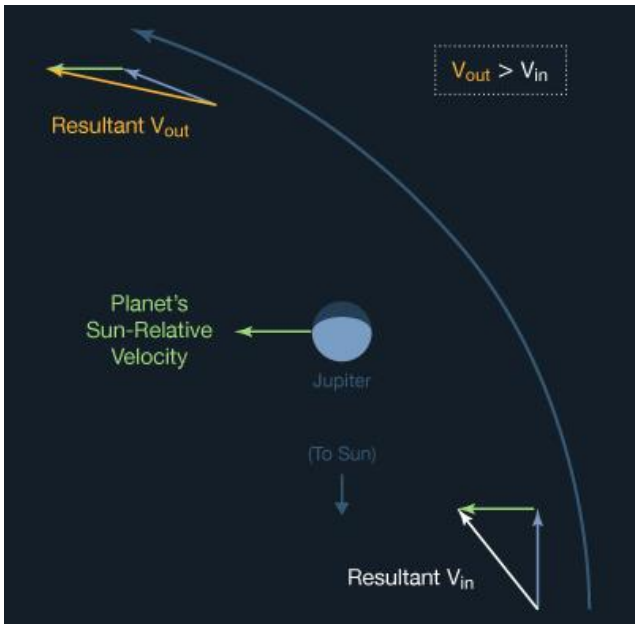
Anomalia prawdziwa obliczana jest ze wzoru:

$$\dot{\theta} = \sqrt{\frac{\mu}{r^3}}$$

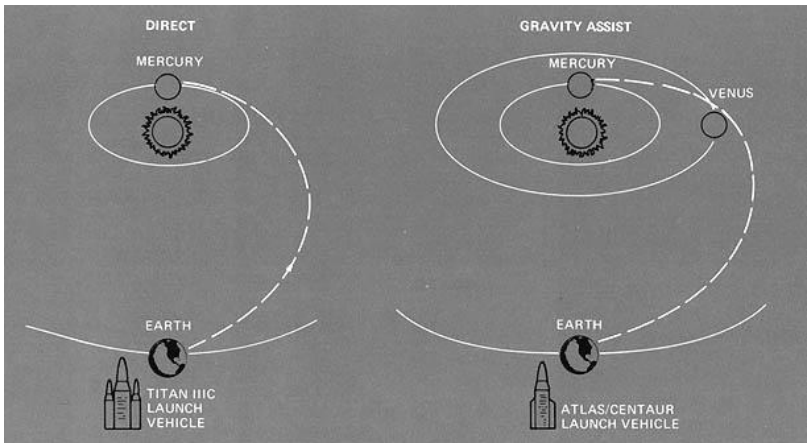
3.3.3. Asysta grawitacyjna

Asysta grawitacyjna to zmiana prędkości i kierunku lotu statku kosmicznego przy użyciu pola grawitacyjnego planety lub innego dużego ciała niebieskiego. Jest to obecnie powszechnie używana metoda uzyskiwania prędkości pozwalających osiągnąć zewnętrzne planety Układu Słonecznego

Asysta grawitacyjna zmienia kierunek, w którym porusza się pojazd, nie zmieniając jego prędkości względem planety. Umożliwia to zwiększenie prędkości względem Słońca maksymalnie o dwukrotność prędkości orbitalnej planety. Głównym ograniczeniem asysty grawitacyjnej jest konieczność dostosowania się do aktualnego położenia planet.



Rys. 8 Schemat działania asysty grawitacyjnej. [1]



Rys. 9 Porównanie trajektorii lotu bezpośredniego do Merkurego oraz misji Marinera 10 z użyciem asysty grawitacyjnej Wenus. [2]

4. Program analizujący misje do planet Układu Słonecznego

Program został opracowany przy wykorzystaniu bibliotek „open-source” do obliczeń mechaniki nieba. W pracy nad programem bazowano na dostępnych przykładach, a głównym celem było umożliwienie wykonywania obliczeń dla każdej z planet Układu Słonecznego i wprowadzenie wyboru różnych rodzajów analiz. Obecnie w programie możliwy jest wybór między dwoma opcjami:

- obliczenia trajektorii lotu bezpośredniego do wybranej planety na podstawie podanej daty startu i lądowania,

- optymalizacja daty startu i lądowania ze względu na koszt transferu dla lotu bezpośredniego do wybranej planety.

Planowane jest rozwijanie programu i wprowadzanie nowych funkcjonalności np. obliczeń dla misji wykorzystujących transfer Hohmanna.

4.1. Środowisko programistyczne

Program powalający na podstawową analizę wybranych aspektów misji kosmicznych został stworzony w środowisku Python 3.6 z wykorzystaniem biblioteki Poliastro 0.9.0 oraz Astropy.

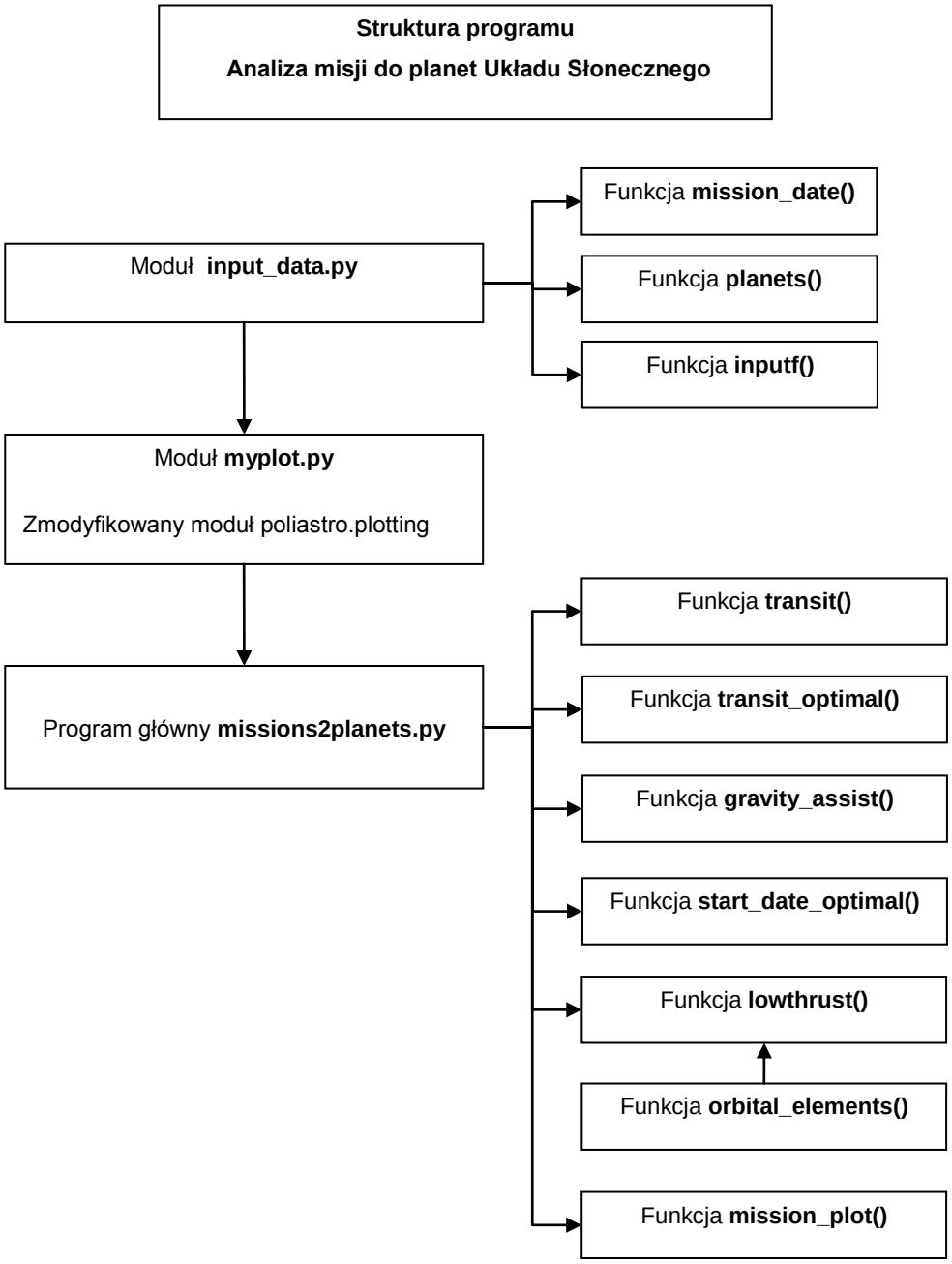
Astropy [9] jest pakietem przeznaczonym do obliczeń astronomicznych o szerokim zakresie funkcji. W projekcie wykorzystany został moduł jednostek ułatwiający obliczenia oraz moduł określający pozycję planet na podstawie efemeryd udostępnianych przez NASA Jet Propulsion Laboratory.

Poliastro [10] to projekt „open-source” do wykonywania obliczeń związanych z mechaniką nieba , którego autorem jest Juan Luis Cano Rodríguez. Poliastro umożliwia określenie orbity na podstawie parametrów orbitalnych lub wektorów prędkości i pozycji, rozwiązuje zagadnienie dwóch ciał, problem Lamberta, zagadnienia związane z manewrami orbitalnymi. Pozwala również na wykonywanie wykresów dwu- i trójwymiarowych.

Wykresy dwuwymiarowe orbit tworzone są przy użyciu wbudowanej funkcji Poliastro oraz biblioteki graficznej Matplotlib. Natomiast do innych wykresów dwuwymiarowych oraz wykresów trójwymiarowych wykorzystywana jest biblioteka Plotly [11] wizualizująca interaktywne wykresy w przeglądarce i umożliwiająca dalszą analizę danych po założeniu konta. Plotly działa również w trybie offline, jednak otwierające się okno przeglądarki może być dla użytkownika minusem i zaburzać jego pracę. Dużym atutem biblioteki plotly pozostaje ogromna liczba funkcji oraz możliwość interaktywnej obróbki danych.

4.2.Struktura programu

Program skład się z trzech modułów: modułu wprowadzania danych, modułu rysowania wykresów oraz programu głównego a także różnych funkcji odpowiadających za kolejne kroki obliczeń. Na poniższym schemacie przedstawiona została struktura programu.



Rys. 10 Schemat struktury programu.

4.2.1. Moduł wprowadzania danych - input_data.py

Funkcja pozwalająca na wprowadzanie danych przez użytkownika i sprawdzenie ich poprawności została opracowana w formie osobnego modułu ze względu na jej dość dużą złożoność. Dane wejściowe muszą mieścić się w określonych granicach, aby zapewnić poprawne działanie programu. Funkcja wprowadzania danych inputf() zawarta w module zwraca następujące dane wprowadzone przez użytkownika:

- numer wybranej opcji obliczeń – 1, 2, 3, 4

- masa statku kosmicznego (tylko dla opcji 2) w granicach 0 - 100 000 kg,
- impuls właściwy napędu (tylko dla opcji 2) w granicach 0 - 100 000 m/s,
- wysokość początkowej orbity kołowej (tylko dla opcji 2) w granicach 100 - 10 000 km,
- cel misji – jedna z planet Układu Słonecznego,
- data startu misji w granicach 1900 – 2100 rok,
- data zakończenia misji w granicach 1900 – 2100 rok.

Funkcja inputf() zwraca również wartości: transit_min i transit_max są to niezbędne do analizy optymalizacyjnej wartości minimalnej i maksymalnej liczby dni – czasu dotarcia do wybranej planety. Zwracane są również nazwy planet wykorzystywanych podczas asysty grawitacyjnej.

W module znajdują się również dwie dodatkowe funkcje:

- mission_date() – funkcja wprowadzania daty przez użytkownika i sprawdzania jej poprawności m.in. formatu daty, liczby dni, miesięcy, lat przestępnych
- planets() – funkcja, której argumentem jest nazwa planety wprowadzona przez użytkownika; zamienia ona nazwę planety na format uznawany przez funkcje Poliastro i przypisuje wybranej planecie odpowiednią wartość transit_min i transit_max.

4.2.2. Moduł tworzenia wykresów - myplot.py

Moduł tworzenia wykresów jest zmodyfikowanym modułem biblioteki Poliastro – poliastro.plotting. Modyfikacja modułu była konieczna, aby umożliwić tworzenie wykresów za pomocą biblioteki Plotly w edytorze Pythona - Spyder. Moduł poliastro.plotting przystosowany był bowiem do pracy w edytorze Jupiter Notebook. Stworzenie osobnego modułu myplot.py pozwala także na modyfikację sposobu wyświetlania wykresu, wielkości planet.

4.2.3. Program główny – missions2planets.py

Program główny składa się z trzech głównych funkcji wykonujących kolejne kroki obliczeń, funkcji tworzenia wykresów oraz wyświetlania wyników w konsoli. Podstawowe funkcje wykonujące obliczenia:

- funkcja transit() – funkcja ta określa położenie planet oraz oblicza rozwiązanie problemu Lamberta, argumentami funkcji są:
    - data początkowa ora data końcowa,
    - nazwy planet.

Funkcja zwraca wektory prędkości i położenia planet, a także prędkości na początku i na końcu transferu otrzymane z rozwiązania zagadnienia Lamberta.

- funkcja transit_optimal() – optymalizuje datę zakończenia misji, argumentami funkcji są:
  - data startu,
  - wartości transit_min i transit_max czyli minimalna i maksymalna liczba dni trwania misji,
  - nazwy planet,
  - prędkość na orbicie kołowej,
  - krok czasowy obliczeń: dla analizy podstawowej wynosi 20 dni, dla analizy szczegółowej 1 dzień.

  Funkcja zwraca optymalną datę zakończenia misji oraz wymaganą zmianę prędkości.

- funkcja start_date_optimal – optymalizuje datę startu misji, w funkcji tej wywoływana jest funkcja transit_optimal(), argumentami funkcji są:
  - data startu i końca misji,
  - wysokość orbity kołowej,
  - masa statku kosmicznego,
  - impuls właściwy silnika,
  - krok czasowy

  Funkcja oblicza optymalną datę startu i końca misji, a także masę materiału pędnego konieczną do wykonania transferu i zmianę prędkości.

Funkcja gravity_assist() oblicza transfer z wykorzystaniem asysty grawitacyjnej wraz z optymalizacją daty startu i lądowania.

Funkcja lowthrust() oblicza parametry transferu małym ciągiem. W funkcji tej wykorzystywana jest funkcja orbitalelements(), która zczytuje z pliku elementy orbitalne planet Układu Słonecznego pobrane ze strony internetowej Jet Propulsion Laboratory NASA.

Funkcja mission_plot() tworzy dla opcji 1 – dwa wykresy: dwuwymiarowy wykres orbit oraz trójwymiarowy wykres położenia planet i trajektorii lotu. Natomiast dla opcji 2 oprócz wymienionych wyżej wykresów funkcja tworzy również wykres obliczanej zmiany prędkości oraz masy materiału pędnego w funkcji daty startu.

Ostatnim elementem programu głównego jest wyświetlanie wyników cząstkowych oraz wyników ostatecznych optymalizacji.

4.2.4.Algorytm obliczeń

Podstawową opcją programu jest wyznaczenie trajektorii podróży między planetami znając datę rozpoczęcia misji i zakończenia transferu. Na podstawie tych danych w funkcji transit() uzyskiwana jest pozycja Ziemi oraz planety docelowej, a także ich wektory prędkości. Następnie rozwiązywane jest zagadnienie Lamberta, z którego otrzymuje się prędkość na początku i na końcu transferu. Dane te pozwalają na wyznaczenie orbit oraz trajektorii lotu i stworzenie wykresów.

Analiza 2 umożliwia znalezienie najlepszej konfiguracji daty startu i lądowania (w podanym przez użytkownika początkowo zakresie) ze względu na zużycie materiału pędnego.

Masa materiału pędnego obliczana jest na podstawie wzoru Ciołkowskiego na prędkość ciała ze zmienną masą:

$$\Delta V = I_{sp} \ln\left(\frac{m_0}{m}\right)$$

Przybliżona masa użytego materiału pędnego wynosi:

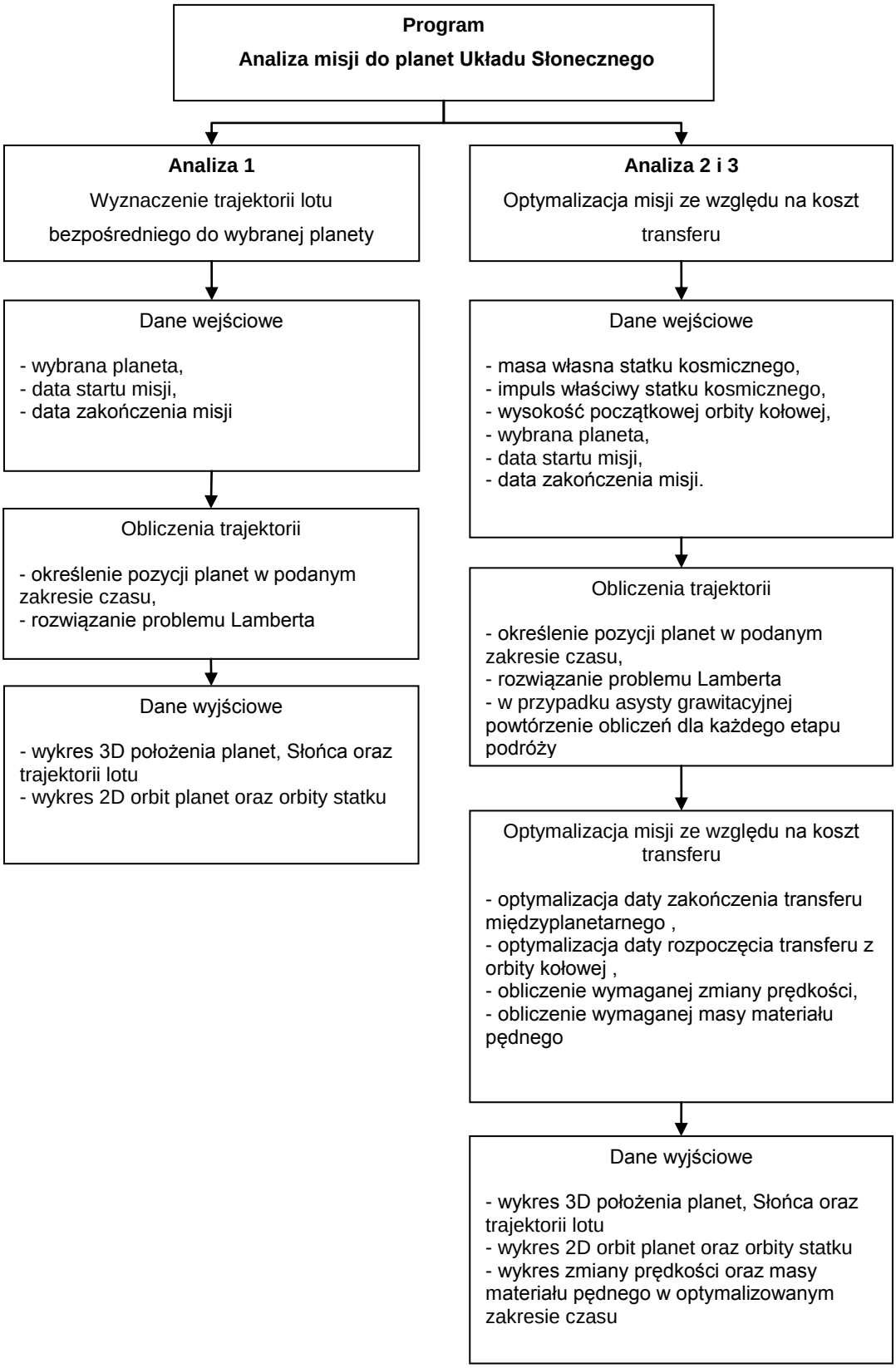
$$m_p = m\left[\exp\left(\frac{\Delta V}{I_{sp}}\right) - 1\right]$$

Masa ta jest silnie zależna od wartości wprowadzonej masy statku kosmicznego i impulsu właściwego silnika.

Obliczenia wykonywane są dla zadanego kroku czasowego, wynoszącego początkowo 20 dni. W pętli, w której data startu zwiększa się o krok czasowy aż do osiągnięcia daty końcowej znajdowana jest data, dla której zmiana prędkości konieczna do wykonania manewru jest najmniejsza oraz obliczana jest wymagana ilość materiału pędnego. Jednocześnie dla każdego dnia startu optymalizowana jest data końca misji:

- określany jest zakres możliwej daty końca misji z uwzględnieniem zadanej minimalnej i maksymalnej liczby dni podróży
- w pętli, w której data końca misji jest zwiększana o krok 20 dni aż do osiągnięcia wartości maksymalnej obliczana jest:
    - wektory pozycji i prędkości planet dla danego zakresu czasu,
    - prędkości na początku i na końcu transferu z zagadnienia Lamberta,
    - zmiana prędkości konieczna do wykonania transferu,
- następnie wybierana jest data końca misji, dla której zmiana prędkości jest najmniejsza.

Po wykonaniu obliczeń dla kroku czasowego 20 dni możliwe jest przeprowadzenie szczegółowych obliczeń z krokiem czasowym 1 dzień dla zakresu +/- 20 dni od dat początku i końca misji obliczony w początkowej analizie.



Rys. 11 Ogólny schemat działania programu.

5. Wyniki analiz misji

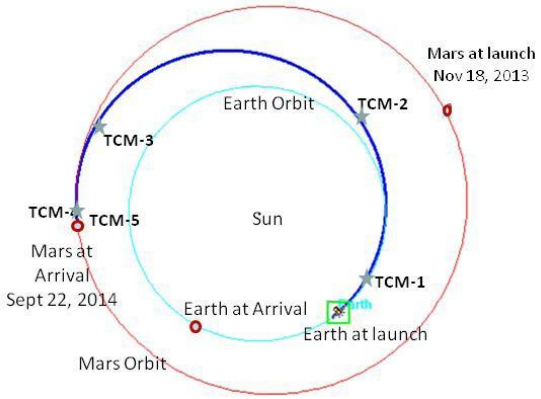
5.1. Analiza 1 – trajektoria lotu bezpośredniego

Wyznaczenie trajektorii lotu bezpośredniego możliwe jest dla każdej z planet Układu Słonecznego, jednak w przypadku planet bardzo odległych, rozpoczynając od Jowisza, może powodować błędy obliczeń przy niekorzystnej konfiguracji dat. Loty bezpośrednie są wykonywane do planet wewnętrznych: Merkurego, Wenus oraz do Marsa, dlatego obliczenia dla Jowisza, Saturna, Urana i Neptuna należy rozpatrywać jedynie teoretycznie. Przeprowadzana analiza jest również bardzo uproszczona, np. pomija wpływ innych planet. Pomimo to uzyskiwane przy pomocy programu trajektorie lotu dla bliższych planet są bardzo zbliżone do rzeczywistych misji.

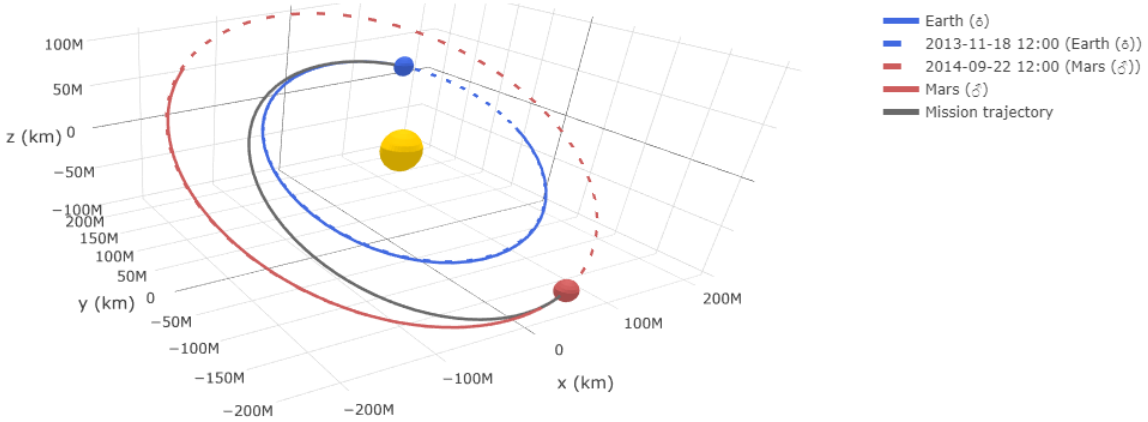
Porównanie trajektorii sondy MAVEN i obliczeń w programie

Data startu: 2013-11-18

Data zakończenia transferu: 2014-09-22



Rys. 12 Rzeczywista trajektoria lotu sondy MAVEN. [12]

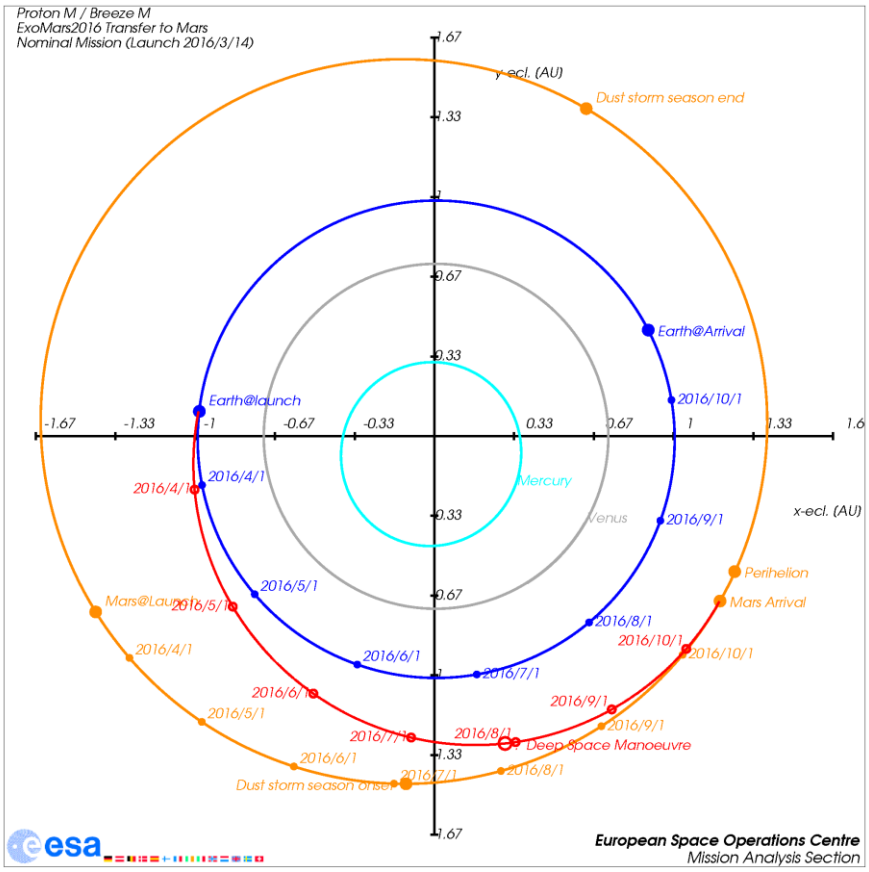


Rys. 13 Trajektoria lotu sondy MAVEN obliczona w programie.

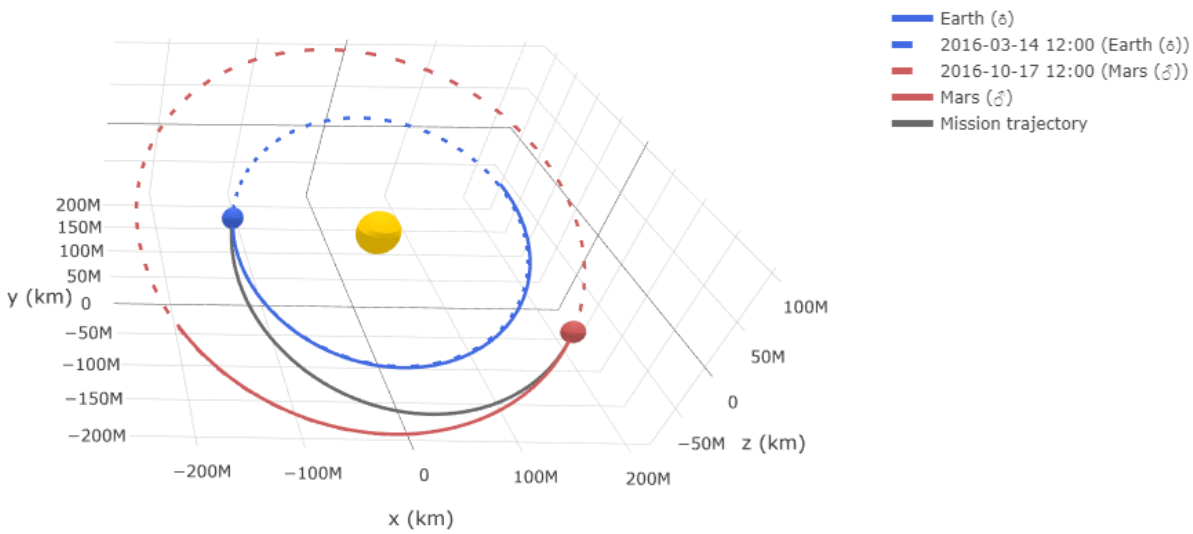
Porównanie trajektorii sondy ExoMars i obliczeń w programie

Data startu: 2016-03-14

Data zakończenia transferu: 2016-10-17



Rys. 14 Rzeczywista trajektoria lotu sondy ExoMars. [5]

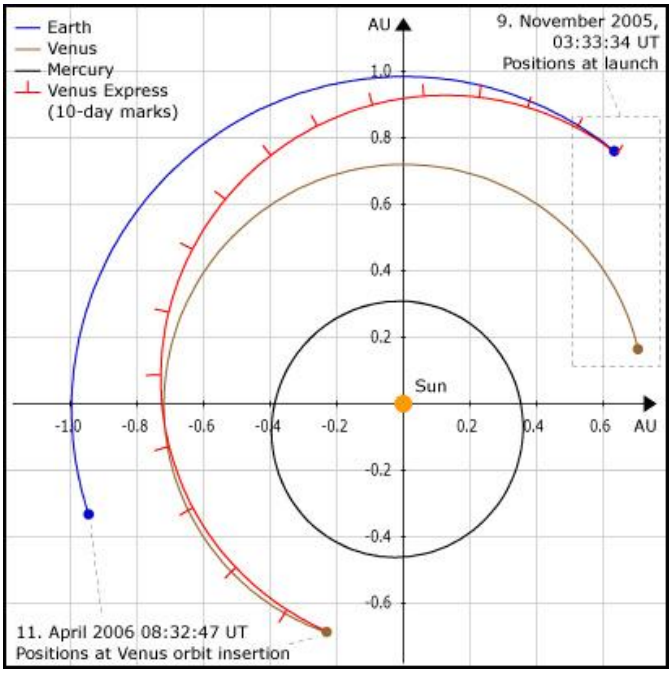


Rys. 15 Trajektoria lotu sondy ExoMars obliczona w programie.

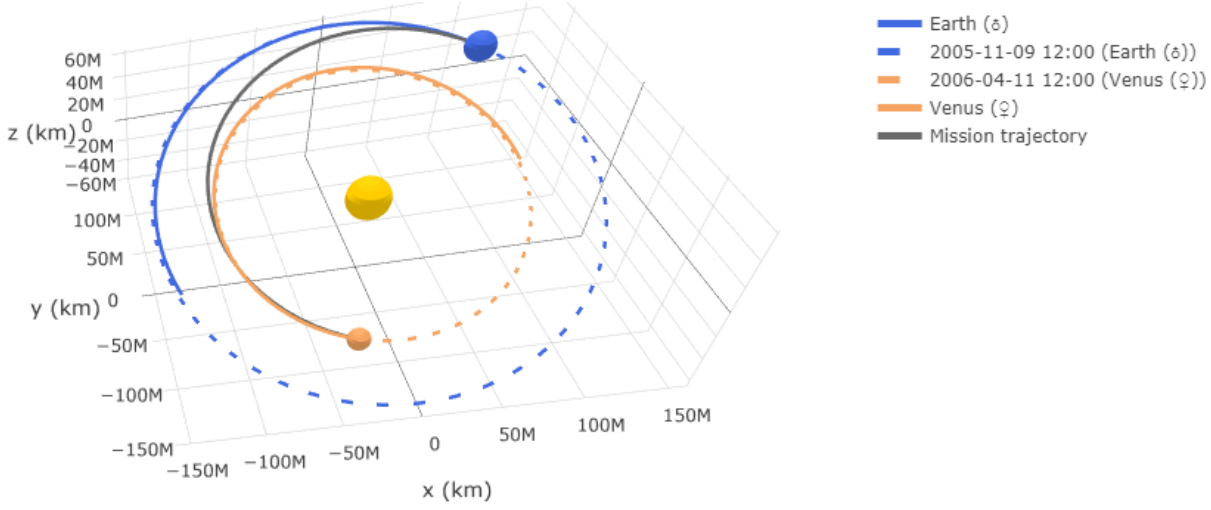
Porównanie trajektorii sondy Venus Express i obliczeń w programie

Data startu: 2005-11-09

Data zakończenia transferu: 2006-04-11



Rys. 16 Rzeczywista trajektoria lotu sondy Venus Express. [13]

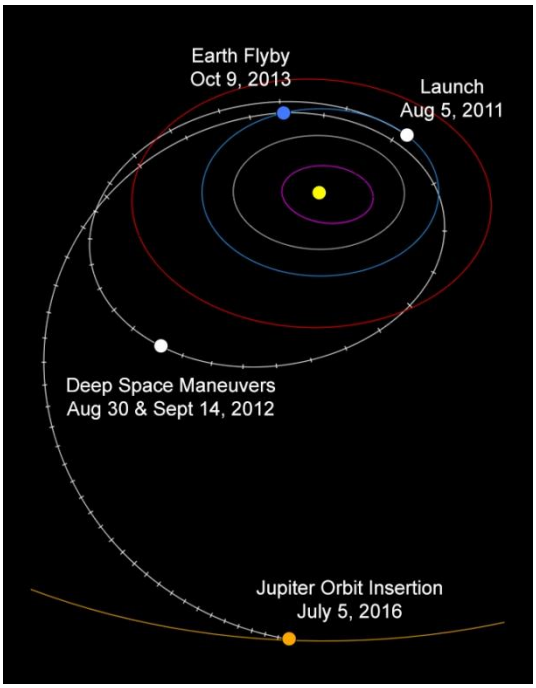


Rys. 17 Trajektoria lotu sondy Venus Express obliczona w programie.

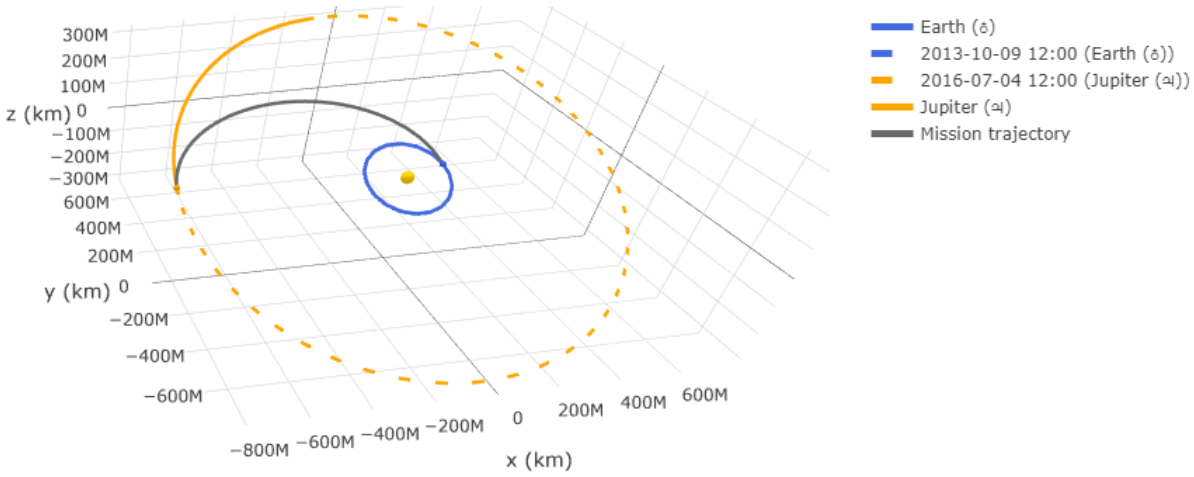
Porównanie trajektorii sondy Juno po wykonaniu asysty grawitacyjnej Ziemi i obliczeń w programie

Data startu: 2013-10-09

Data zakończenia transferu: 2016-07-24



Rys. 18 Rzeczywista trajektoria lotu sondy Juno. [14]



Rys. 19 Fragment trajektorii lotu sondy Juno obliczony w programie.

5.2. Analiza 2 – optymalizacja kosztów lotu

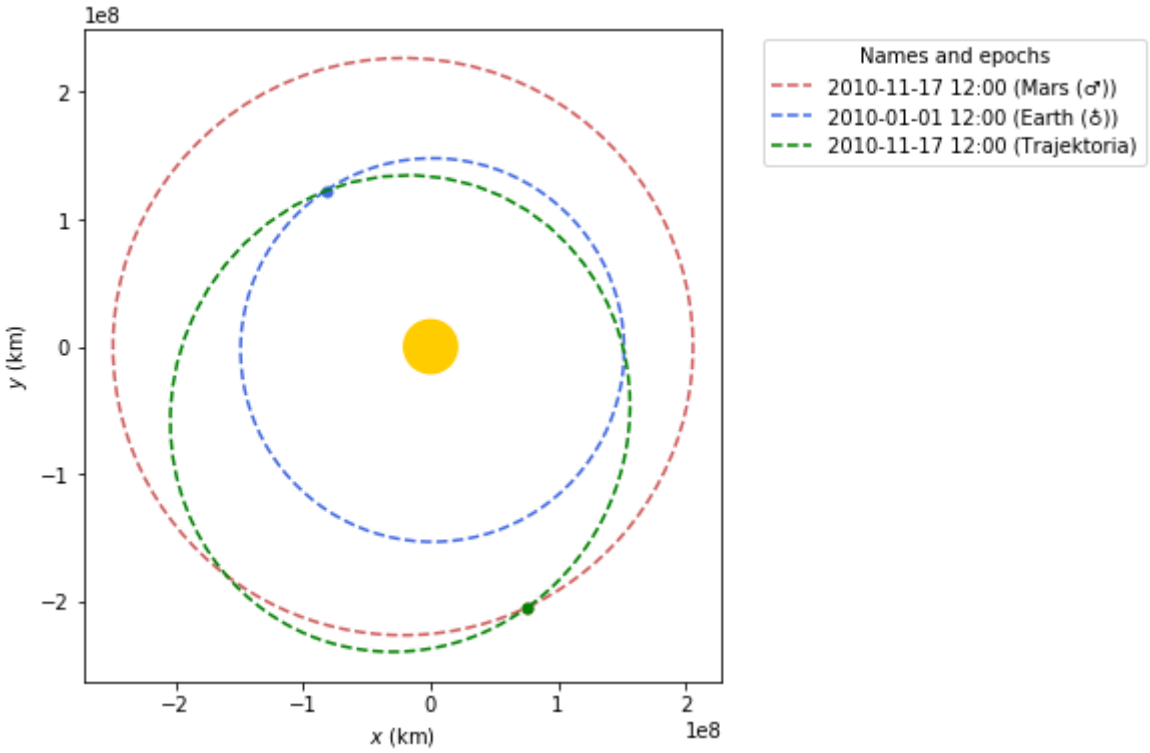
Dane wejściowe:

- masa statku kosmicznego $M = 1000\,kg$,
- impuls właściwy silnika: $I_{sp} = 4400\,m/s$
- wysokość początkowej orbity kołowej: $H = 250\,km$
- cel: Mars,
- start misji: 2010-01-01
- koniec misji: 2020-01-01

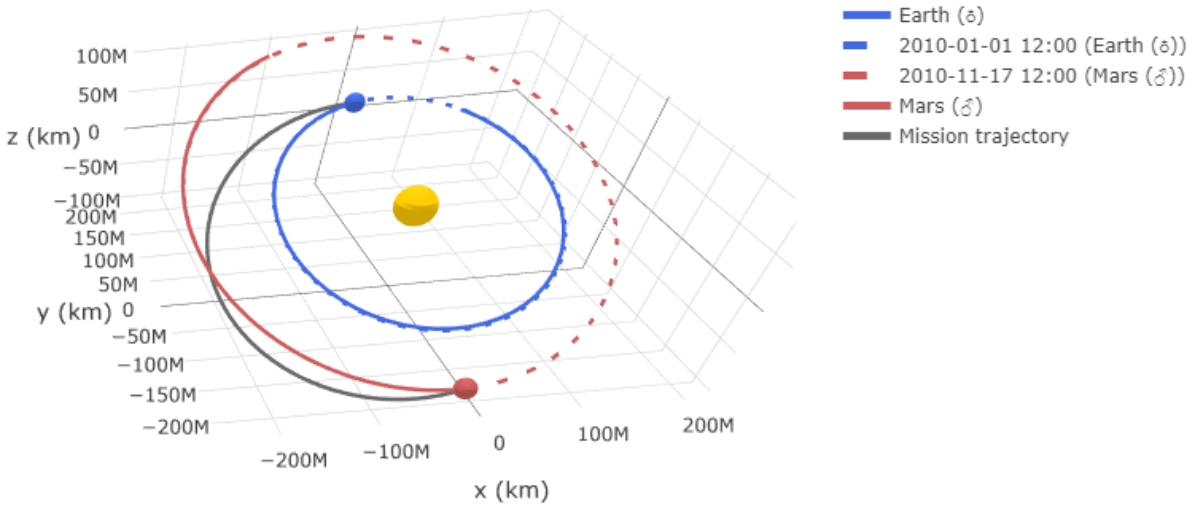
Wyniki analizy optymalizacyjnej misji na Marsa:

- optymalna data startu: 2010-01-01
- data końca misji: 2010-11-17
- czas lotu: 320 dni
- zmiana prędkości: 3.267 km/s
- masa zużytego materiału pędnego: 1101 kg

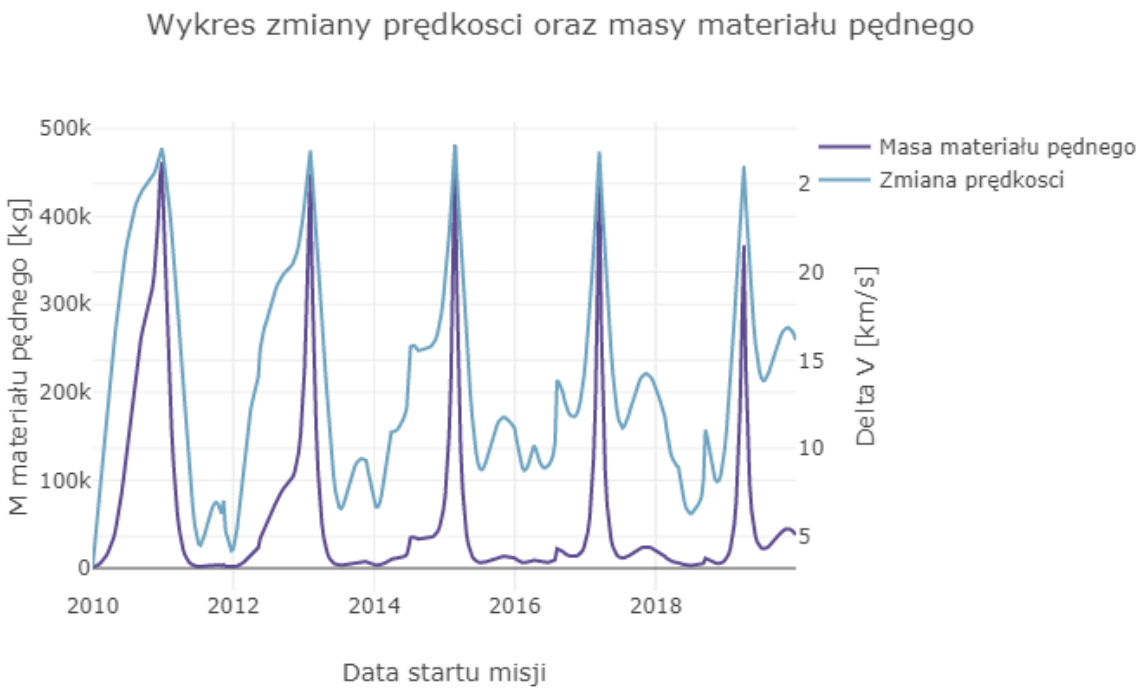
Wizualizacja orbit Ziemi i Marsa oraz obliczonej orbity transferowej.



Rys. 20 Wykres 2D orbit planet i orbity transferowej.



Rys. 21 Wizualizacja trójwymiarowa zoptymalizowanej trajektorii lotu.



Rys. 22 Wykres zmiany prędkości oraz masy materiału pędnego w funkcji daty startu misji.

Na wykresie zmiany prędkości oraz masy materiału pędnego zauważyć można okresowe zmiany parametrów związane z ustawieniem planet. Najkorzystniejsza konfiguracja została otrzymana dla daty startu równej początkowi analizowanego okresu: 01-01-2010 rok. Zmiany prędkości wahają się od najmniejszej wartości 3.267 km/s do około 25 km/s. Natomiast masa materiału pędnego zawiera się w granicach 1001 kg - 500000 kg. Otrzymane wartości parametrów są porównywalne do danych rzeczywistych, jednak w większości przypadków przyjmują większe wartości, na co mają wpływ założenia upraszczające oraz założenia wartości wejściowych np. impulsu oraz masy statku.

5.3. Analiza 3 – asysta grawitacyjna

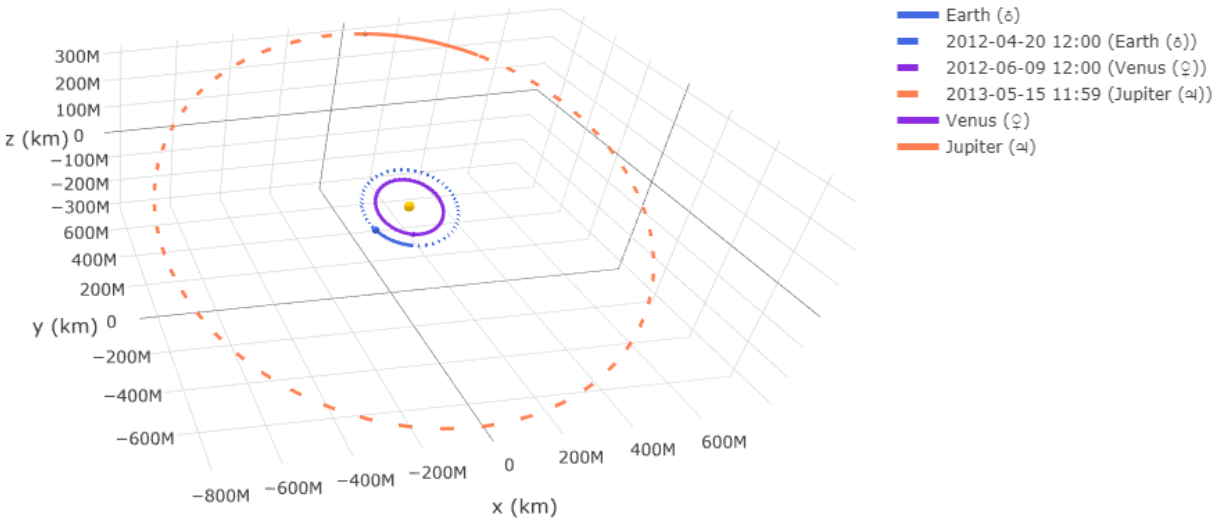
Dane wejściowe:

- masa statku kosmicznego $M = 1000\,kg$,
- impuls właściwy silnika: $I_{sp} = 4400\,m/s$
- wysokość początkowej orbity kołowej: $H = 250\,km$
- cel: Jowisz,
- asysta planety: Wenus,
- start misji: 2010-01-01
- koniec misji: 2020-01-01

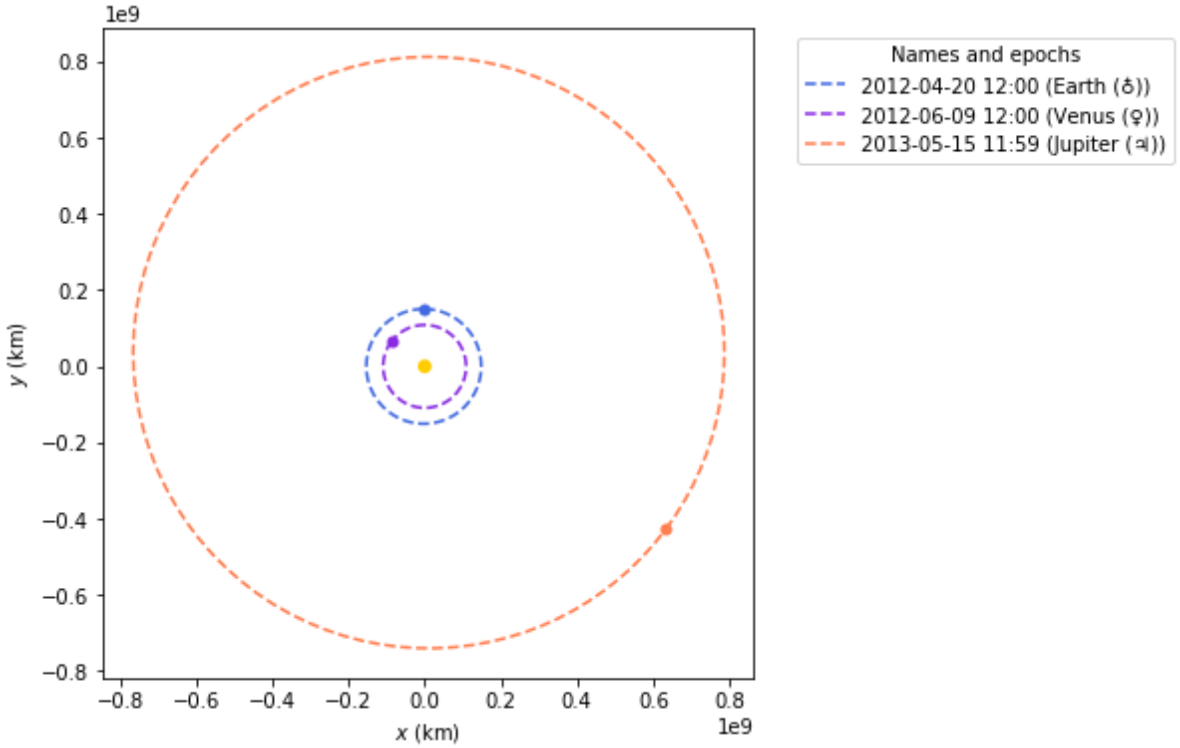
Wyniki analizy optymalizacyjnej misji na Marsa:

- optymalna data startu: 2012-04-20
- data końca misji: 2013-05-15
- czas lotu: 390 dni
- zmiana prędkości: 1.82 km/s
- masa zużytego materiału pędnego: 512 kg

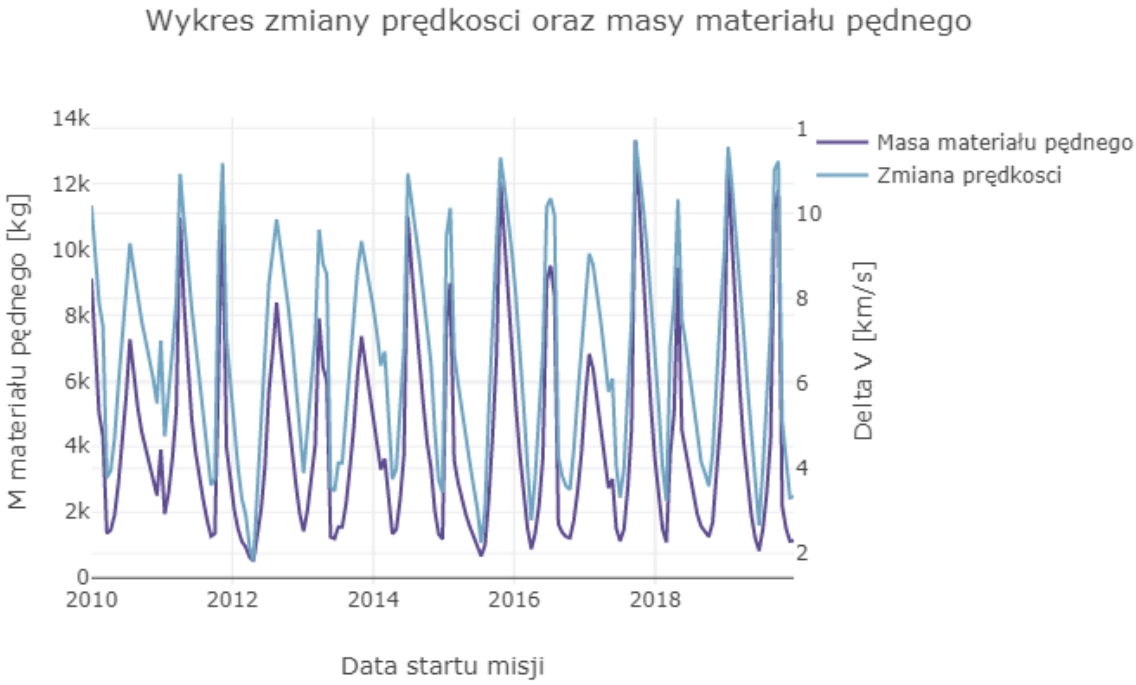
Wizualizacja orbit Ziemi i Marsa.



Rys. 23 Wizualizacja trójwymiarowa układu planet podczas trwania misji.



Rys. 24 Dwuwymiarowy wykres orbit planet.



Rys. 25 Wykres zmiany prędkości oraz masy materiału pędnego w funkcji daty startu misji.

5.4. Analiza 4 – transfer małym ciągiem

Dane wejściowe:

- masa statku kosmicznego $M = 1000\ kg$,
- impuls właściwy silnika: $I_{sp} = 4700\ m/s$
- wysokość początkowej orbity kołowej: $H = 250\ km$
- cel: Mars,
- start misji: 2010-01-01
- koniec misji: 2012-01-01

Wyniki analizy optymalizacyjnej misji na Marsa:

- przyśpieszenie: 0.145 m/s^2
- zmiana prędkości: 21,9 km/s
- masa zużytego materiału pędnego: 0.47 kg
- czas lotu: 1753 dni

6. Podsumowanie

Opracowany w oparciu o bibliotekę Poliastro program pozwala na wykonywanie różnorodnych analiz dla wszystkich planet Układu Słonecznego. Umożliwia wizualizację trajektorii rzeczywistych w przypadku lotu bezpośredniego oraz analizę misji pod kątem najmniejszego kosztu transferu. Zaproponowana wizualizacja wyników umożliwia porównanie parametrów transferu oraz porównywanie różnych konfiguracji misji. Wyniki otrzymane z analizy misji planet: Merkurego, Wenus, Marsa, Jowisza są porównywalne z rzeczywistymi misjami, natomiast w przypadku Saturna, Urana, Neptuna zmiana prędkości i masa użytego materiału pędnego są bardzo wysokie, mogą pojawiać się również problemy z obliczeniem trajektorii lotu dla podanego zbyt krótkiego analizowanego okresu trwania misji. Program może zostać rozwinięty poprzez wprowadzenie dodatkowych obliczeń zmian trajektorii za pomocą międzyplanetarnego transferu Hohmanna, metody "patch conics", co powinno zwiększyć dokładność wykonywanych obliczeń, szczególnie dla najbardziej odległych planet oraz umożliwi porównywanie wyników obliczeń za pomocą różnych metod.

7. Bibliografia

[1] “Interplanetary Trajectories | Basics of Space Flight - NASA Solar System Exploration.” [Online]. Available: https://solarsystem.nasa.gov/basics/chapter4-1. [Accessed: 16-Jan-2018].

[2] “cnes | How interplanetary trajectories work.” [Online]. Available: https://cnes.fr/en/how-interplanetary-trajectories-work. [Accessed: 16-Jan-2018].

[3] ESA, “Mission Definition Report,” 2001.

[4] “Venus Express factsheet / Venus Express / Space Science / Our Activities / ESA.” [Online]. Available: http://www.esa.int/Our_Activities/Space_Science/Venus_Express/Venus_Express_factsheet. [Accessed: 16-Jan-2018].

[5] “Schiaparelli separating from the Trace Gas Orbiter.”

[6] V. A. Chobotov, *Orbital Mechanics, Third Edition*. 2002.

[7] Ł. Mężyk, “Wykłady - Techniki Kosmiczne 2017,” 2017.

[8] H. D. Curtis, *Orbital Mechanics for Engineering Students, third edition*. 2013.

[9] “Astropy.” [Online]. Available: http://www.astropy.org/. [Accessed: 17-Jan-2018].

[10] “poliastro - Astrodynamics in Python — poliastro 0.9.dev0 documentation.” [Online]. Available: http://docs.poliastro.space/en/latest/. [Accessed: 17-Jan-2018].

[11] “Modern Visualization for the Data Era - Plotly.” [Online]. Available: https://plot.ly/. [Accessed: 17-Jan-2018].

[12] “Mission Profile – MAVEN.” [Online]. Available: http://spaceflight101.com/maven/mission-profile/. [Accessed: 17-Jan-2018].

[13] “Venus Express is up and away | Astronomy.com.” [Online]. Available: http://www.astronomy.com/news/2005/11/venus-express-is-up-and-away. [Accessed: 17-Jan-2018].

[14] “File:Juno cruise trajectory.jpg - Wikimedia Commons.” [Online]. Available: https://commons.wikimedia.org/wiki/File:Juno_cruise_trajectory.jpg. [Accessed: 17-Jan-2018].

8. Załącznik 1

Program główny missions2planets.py

```
"""
Created: 2017-12-10
Author: Filip Perczyński
-------------------------------------------------------------------------
-
Program główny, w którym zawierają sie:
- funkcja transit() zwraca: - wektory pozycji i predkosci planet, rozwiazanie
    problemu Lamberta
- funkcja transit_optimal() - zwraca zoptymalizowana date konca misji i
    zmiane predkosci
- funkcja gravity_assist() - oblicza transfer z wykorzystaniem asysty
grawitacyjnej
- funkcja start_date_optimal() - zwraca zoptymalizowana date startu, konca
misji,
    najmniejsza zmiane predkosci, zuzycie materialu pednego i dane do wykresu
- funkcja lowthrust() - oblicza parametry transferu z małym ciagiem
- funkcja planets2plot() - zwraca etykiete planety i kolor do narysowania
wykresu
- funkcja mission_plot() - zwraca wykresy 2D i 3D trajektorii lotu
- funkcja opt_plot() - zwraca wykresy zmiany predkosci i masy materialu
pednego
    w funkcji czasu dla analizy 2
-------------------------------------------------------------------------
-
"""
import numpy as np
import astropy.units as u
from astropy.coordinates import solar_system_ephemeris,
get_body_barycentric_posvel
from astropy import constants as const

from poliastro import iod
from poliastro.bodies import *
from poliastro.twobody import Orbit
from poliastro.plotting import OrbitPlotter
from poliastro.util import time_range

import plotly.graph_objs as go
from plotly.offline import plot

import input_data
import myplot
import orbital_elements

```

```
38.
39. def transit(date, date_arrival, planet1, planet2):
40.     #czas trwania misji
41.     tof = date_arrival - date
42.     N = 50
43.     tof.to(u.h)
44.     times_vector = time_range(date, end=date_arrival, periods=N)
45.
46. # okreslenie pozycji planet w przedziale czasu: od startu do końca misji
47.     rr_planet1, vv_planet1 = get_body_barycentric_posvel(planet1,
    times_vector)
48.     rr_planet2, vv_planet2 = get_body_barycentric_posvel(planet2,
    times_vector)
49.
50.     r0 = rr_planet1[0].xyz #wektor pozycji Ziemi w momencie startu
51.     v0 = vv_planet1[0].xyz #wektor predkosci Ziemi w momencie startu
52.     rf = rr_planet2[-1].xyz #wektor pozycji planety docelowej w momencie
    końca misji
53.     vf = vv_planet2[-1].xyz #wektor predkosci planety docelowej w momencie
    końca misji
54.
55. # rozwiazanie problemu Lamberta
56.     (va, vb), = iod.lambert(Sun.k, r0, rf, tof, numiter=1000)
57.
58.     return(r0, v0, rf, vf, va, vb, rr_planet1, rr_planet2, times_vector)
59.
60.
61. def transit_optimal(date, transit_min, transit_max, planet1, planet2, vs0,
    step):
62.     #wartosci poczatkowe
63.     date_out = date + transit_min
64.     date_max = date + transit_max
65.     date_out_final = date_out
66.     dv_final = 0 * u.km / u.s
67.
68.     step_one = True  #sprawdzenie pierwszego kroku
69.
70.     while date_out < date_max:  #poszukiwanie optymalnej daty konca misji
71.
72.         r0, v0, rf, vf, va, vb, rr_planet1, rr_planet2, times_vector =
    transit(date, date_out, planet1, planet2)
73.
74.         dv_vector = va - (vs0 + (v0 / (24*3600) * u.day / u.s)) #wektor
    wymaganej zmiany predkosci
75.         dv = np.linalg.norm(dv_vector/10) * u.km / u.s  #zmiana predkosci
76.
77.         if step_one:
78.             dv_final = dv
```

```
79.             step_one = False
80.         else:
81.             if dv < dv_final: #wybór koniguracji z najmniejsza zmniana
    predkosci
82.                 dv_final = dv
83.                 date_out_final = date_out
84.
85.         date_out += step
86.
87.     return dv_final, date_out_final, vb
88.
89. def gravity_assist(date, vsE, transit_min, transit_max, as1, as2, as3, m,
    Isp, step):
90.
91.     if nr == 2:
92.         dv_final, date_out_final, vs_p1 = transit_optimal(date,
    transit_min[0], transit_max[0], 'earth', planet, vsE, step)
93.         date_assist = 0
94.         m_p_tot = m * (np.exp(dv_final/ Isp)-1)
95.
96.     if nr == 3:
97.         if as3 == 0:
98.             if as2 == 0:
99.                 dv_final1, date_out_final1, vs_p1 = transit_optimal(date,
    transit_min[1], transit_max[1], 'earth', as1, vsE, step)
100.                    dv_final2, date_out_final2, vs_p2 =
    transit_optimal(date_out_final1, transit_min[4], transit_max[4], as1, planet,
    vs_p1, step)
101.                    date_assist = [date_out_final1, 0, 0]
102.                    date_out_final = date_out_final2
103.                    m_p2 = m  * (np.exp((dv_final2) / Isp) - 1)
104.                    m_p1 = (m + m_p2) * (np.exp((dv_final1) / Isp) - 1)
105.                    m_p_tot = m_p1 + m_p2
106.                    dv_final = dv_final1 + dv_final2
107.
108.                else:
109.                    dv_final1, date_out_final1, vs_p1 =
    transit_optimal(date, transit_min[1], transit_max[1], 'earth', as1, vsE,
    step)
110.                    dv_final2, date_out_final2, vs_p2 =
    transit_optimal(date_out_final1, transit_min[2], transit_max[2], as1, as2,
    vs_p1, step)
111.                    dv_final3, date_out_final3, vs_p3 =
    transit_optimal(date_out_final2, transit_min[4], transit_max[4], as2, planet,
    vs_p2, step)
112.                    date_assist = [date_out_final1, date_out_final2, 0]
113.                    date_out_final = date_out_final3
114.                    m_p3 = m * (np.exp((dv_final3) / Isp) - 1)
```

```
115.                     m_p2 = (m + m_p3) * (np.exp((dv_final2) / Isp) - 1)
116.                     m_p1 = (m + m_p3 + m_p2) * (np.exp((dv_final1) / Isp)
    - 1)
117.
118.                     m_p_tot = m_p1 + m_p2 + m_p3
119.                     dv_final = dv_final1 + dv_final2 + dv_final3
120.
121.             else:
122.                 dv_final1, date_out_final1, vs_p1 = transit_optimal(date,
    transit_min[1], transit_max[1], 'earth', as1, vsE, step)
123.                 dv_final2, date_out_final2, vs_p2 =
    transit_optimal(date_out_final1, transit_min[2], transit_max[2], as1, as2,
    vs_p1, step)
124.                 dv_final3, date_out_final3, vs_p3 =
    transit_optimal(date_out_final2, transit_min[3], transit_max[3], as2, as3,
    vs_p2, step)
125.                 dv_final4, date_out_final4, vs_p4 =
    transit_optimal(date_out_final3, transit_min[4], transit_max[4], as3, planet,
    vs_p3, step)
126.                 date_assist = [date_out_final1, date_out_final2,
    date_out_final3]
127.                 date_out_final = date_out_final4
128.                 m_p4 = m * (np.exp(dv_final4 / Isp)-1)
129.                 m_p3 = (m + m_p4) * (np.exp((dv_final3) / Isp) - 1)
130.                 m_p2 = (m + m_p4 + m_p3) * (np.exp((dv_final2) / Isp) - 1)
131.                 m_p1 = (m + m_p4 + m_p3 + m_p2) * (np.exp((dv_final1) /
    Isp) - 1)
132.
133.                 m_p_tot = m_p1 + m_p2 + m_p3 + m_p4
134.                 dv_final = dv_final1 + dv_final2 + dv_final3 + dv_final4
135.
136.         return m_p_tot, dv_final, date_out_final, date_assist
137.
138.
139.
140.     def start_date_optimal(H, date0, date1, m, Isp, step):
141.         #wartosci poczatkowe
142.         delta_v = 0 * u.km / u.s
143.         dv_final = 0 * u.km / u.s
144.         m_prop = 0 * u.kg
145.         date_launch_final = date0
146.         date_arrival_final = date0
147.         date_assist_final = []
148.
149.         step_one0 = True  #sprawdzenie pierwszego kroku
150.
151.         print('Data startu, Czas lotu [dni], Zmiana predkosci dV [km/s],
    Material pedny [kg]')
```

```
152.
153.          # inicjalizacja zmiennych do tworzenia wykresu
154.          plot_date = np.array([])
155.          plot_mprop = np.array([])
156.          plot_dv = np.array([])
157.
158.          while date0 < date1: #poszukiwanie optymalnej daty poczatku misji
159.              epoch0 = date0.jyear_str
160.              ss0 = Orbit.circular(Earth, H, epoch=epoch0) #poczatkowa
    orbita kolowa
161.              vsE = ss0.rv()[1] #predkosc na orbicie kolowej
162.
163.              m_p_tot, dv_final, date_out_final, date_assist =
    gravity_assist(date0, vsE, transit_min, transit_max, as1, as2, as3, m, Isp,
    step)
164.
165.              # zmiana formatu danych do przedstawienia na wykresie
166.              x = str(date0.iso[0:10])
167.              y = m_p_tot.value
168.              y2 = dv_final.value
169.              # macierze zmiennych do stworzenia wykresu
170.              plot_date = np.append(plot_date,x)
171.              plot_mprop = np.append(plot_mprop,y)
172.              plot_dv = np.append(plot_dv,y2)
173.
174.              print(date0.iso[0:10], ', %i dni, %.3f km/s, %i kg' %
    (int((date_out_final - date0).jd),
175.
    float(dv_final / u.km * u.s),
176.
    int(m_p_tot / u.kg)))
177.              if step_one0:
178.                  delta_v = dv_final
179.                  m_prop = m_p_tot
180.                  date_arrival_final = date_out_final
181.                  step_one0 = False
182.              else:
183.                  if dv_final < delta_v: #wybór koniguracji z najmniejsza
    zmniana predkosci
184.                      delta_v = dv_final
185.                      m_prop = m_p_tot
186.                      date_launch_final = date0 #optymalna data startu
187.                      date_arrival_final = date_out_final #optymalana data
    konca
188.                      date_assist_final = date_assist
189.
190.              date0 += step
191.
```

```
192.          return delta_v, date_launch_final, date_arrival_final,
    date_assist_final, m_prop, plot_date, plot_mprop, plot_dv
193.
194.
195.      def lowthrust(planet, h_LEO, Isp, mass, date_launch, date_arrival):
196.
197.          # Inklinacja i półos wielka planety
198.          a, inc = orbital_elements.orbitalements(planet.title())
199.
200.          a_p = a # polos wielka planety
201.          i_p = inc  # inklinacja planety względem ekliptyki, stopnie
202.          i_E = np.deg2rad(23.5)  # inklinacja równika Ziemi względem
    ekliptyki, stopnie
203.          i_LEO = np.deg2rad(28.5) # inklinacja orbity LEO, stopnie
204.
205.          r_E = 6378.1363 #const.R_earth/1000 # Promien rownikowy Ziemi, km
206.
207.          g = const.g0/u.m*u.s*u.s
208.          pi = np.pi
209.
210.          T = 0.000145
211.
212.          r_LEO = r_E + h_LEO/u.km # Promien orbity LEO, km
213.          r_planet = a_p # Promien orbity planety, km
214.
215.          epoch0 = date_launch.jyear_str
216.          orb_E = Orbit.circular(Earth, h_LEO, epoch=epoch0)
217.          v_E1 = orb_E.rv()[1] #predkosc na orbicie kolowej
218.          v_LEO = np.linalg.norm(v_E1)
219.
220.          r0, v0, rf, vf, va, vb, rr_planet1, rr_planet2, times_vector =
    transit(date_launch, date_arrival, 'earth', planet)
221.          v_planet_vec = vf / (24*3600) * u.day / u.s
222.          v_planet = np.linalg.norm(v_planet_vec)
223.
224.          #Całkowita zmiana inklinacji
225.          di_LEO_p = i_LEO + i_E - i_p
226.
227.          # Obliczanie przyspieszenia
228.          mass = mass/u.kg
229.          accel = T/mass
230.
231.          # Obliczenie kąta odchylenia ciagu
232.          beta_o = np.arctan((np.sin(pi*di_LEO_p/2))/(v_LEO/v_planet-
    np.cos(pi*di_LEO_p/2)));
233.
234.          # Obliczenie całkowitej zmiany predkosci, km/s
```

```
235.          deltaVtotal = np.sqrt(v_LEO**2+v_planet**2-
   2*v_LEO*v_planet*np.cos(pi*di_LEO_p/2))
236.          deltaV2total = v_LEO*np.cos(beta_o)-
   (v_LEO*np.sin(beta_o))/(np.tan(pi*di_LEO_p/2+beta_o))
237.
238.          # Obliczenie masy materiału pednego, kg
239.          Isp1 = Isp*1000* u.s / u.km
240.          mp = mass*(np.exp(deltaVtotal/(Isp1*g))-1)
241.
242.          # Obliczenie czasu transferu
243.          time_s = mp*Isp1*g/T # Transfer w sekunach
244.          time_m = time_s/60 # Transfer w minutach
245.          time_h = time_m/60 # Transfer w godzinach
246.          time_d = time_h/24 # Transfer w dniach
247.          time_y = time_d/365.25 # Transfer w latach
248.          t2_s = deltaVtotal/accel
249.          t2_m = t2_s/60
250.          t2_h = t2_m/60
251.          t2_d = t2_h/24
252.          t2_y = t2_d/365.25
253.
254.          return mp, t2_d, deltaVtotal, accel
255.
256.      def planets2plot(planet):
257.          if planet == 'mercury':
258.              pl_planet = Mercury
259.              color_planet = 'sandybrown'
260.          if planet == 'venus':
261.              pl_planet = Venus
262.              color_planet = 'blueviolet'
263.          if planet == 'earth':
264.              pl_planet = Earth
265.              color_planet = 'royalblue'
266.          if planet == 'mars':
267.              pl_planet = Mars
268.              color_planet = 'indianred'
269.          if planet == 'jupiter':
270.              pl_planet = Jupiter
271.              color_planet = 'coral'
272.          if planet == 'saturn':
273.              pl_planet = Saturn
274.              color_planet = 'springgreen'
275.          if planet == 'uranus':
276.              pl_planet = Uranus
277.              color_planet = 'skyblue'
278.          if planet == 'neptune':
279.              pl_planet = Neptune
280.              color_planet = 'navy'
```

```
281.
282.          return (pl_planet, color_planet)
283.
284.
285.      def mission_plot(date_launch, date_arrival, date_assist, planet):
286.
287.          color_planet_e = 'royalblue'
288.          color_trans = 'dimgrey'
289.          color_orbit_trans = 'orchid'
290.
291.          if nr in [1,2]:
292.              r0, v0, rf, vf, va, vb, rr_planet1, rr_planet2, times_vector =
    transit(date_launch, date_arrival, 'earth', planet)
293.
294.              # Obliczenie orbit transferowych oraz orbit planet
295.              ss0_trans = Orbit.from_vectors(Sun, r0, va, date_launch)
296.              ssf_trans = Orbit.from_vectors(Sun, rf, vb, date_arrival)
297.              ss_e= Orbit.from_vectors(Sun, r0, v0, date_launch)
298.              ss_p = Orbit.from_vectors(Sun, rf, vf, date_arrival)
299.
300.
301.              pl_planet, color_planet = planets2plot(planet)
302.
303.              #wykres orbit 2D
304.              orb = OrbitPlotter()
305.              orb.plot(ss_p, label= pl_planet, color=color_planet)
306.              orb.plot(ss_e, label= Earth, color=color_planet_e)
307.              orb.plot(ssf_trans, label='Orbita transferowa',
    color=color_orbit_trans)
308.
309.              #wykres orbit i trajektorii lotu 3D
310.              frame = myplot.OrbitPlotter3D()
311.              frame.set_attractor(Sun)
312.              frame.plot_trajectory(rr_planet1, label=Earth,
    color=color_planet_e)
313.              frame.plot(ss_e, label= Earth, color=color_planet_e)
314.              frame.plot(ss_p, label= pl_planet, color=color_planet)
315.              frame.plot_trajectory(rr_planet2, label=pl_planet,
    color=color_planet)
316.              frame.plot_trajectory(ss0_trans.sample(times_vector),
    label="Mission trajectory", color=color_trans)
317.              frame.set_view(30 * u.deg, 260 * u.deg, distance=3* u.km)
318.              frame.show(title="Mission to Solar System Planet")
319.
320.          if nr == 3:
321.              if as3 == 0:
322.                  if as2 == 0:
323.
```

```
                    r0, v0, rf, vf, va, vb, rr_planet1, rr_planet2,
    times_vector = transit(date_launch, date_assist[0], 'earth', as1)
                    r02, v02, rf2, vf2, va2, vb2, rr_planet12,
    rr_planet22, times_vector2 = transit(date_assist[0], date_arrival, as1,
    planet)

                    ss0_trans = Orbit.from_vectors(Sun, r0, va,
    date_launch)
                    ssf_trans = Orbit.from_vectors(Sun, rf, vb,
    date_assist[0])
                    ss0_trans2 = Orbit.from_vectors(Sun, r02, va2,
    date_assist[0])
                    ssf_trans2 = Orbit.from_vectors(Sun, rf2, vb2,
    date_arrival)

                    ss_e = Orbit.from_vectors(Sun, r0, v0, date_launch)
                    ss_p1 = Orbit.from_vectors(Sun, rf, vf,
    date_assist[0])
                    ss_p2 = Orbit.from_vectors(Sun, rf2, vf2,
    date_arrival)

                    orb = OrbitPlotter()
                    orb.plot(ss_e, label= Earth, color=color_planet_e)
                    pl_planet, color_planet = planets2plot(as1)
                    orb.plot(ss_p1, label= pl_planet, color=color_planet)
                    pl_planet, color_planet = planets2plot(planet)
                    orb.plot(ss_p2, label= pl_planet, color=color_planet)

                    frame = myplot.OrbitPlotter3D()
                    frame.set_attractor(Sun)
                    frame.plot_trajectory(rr_planet1, label=Earth,
    color=color_planet_e)
                    frame.plot(ss_e, label= Earth, color=color_planet_e)

                    pl_planet, color_planet = planets2plot(as1)
                    frame.plot(ss_p1, label= pl_planet,
    color=color_planet)
                    pl_planet, color_planet = planets2plot(planet)
                    frame.plot(ss_p2, label= pl_planet,
    color=color_planet)

                    pl_planet, color_planet = planets2plot(as1)
                    frame.plot_trajectory(rr_planet12, label= pl_planet,
    color=color_planet)
                    #frame.plot_trajectory(ss0_trans.sample(times_vector),
    label="Mission trajectory", color=color_trans)

                    pl_planet, color_planet = planets2plot(planet)
```

```
358.                     frame.plot_trajectory(rr_planet22, label=pl_planet,
    color=color_planet)
359.
    #frame.plot_trajectory(ss0_trans2.sample(times_vector2), label="Mission
    trajectory2", color=color_trans)
360.                     frame.set_view(30 * u.deg, 260 * u.deg, distance=3 *
    u.km)
361.                     frame.show(title=" Mission: from Earth to Solar System
    Planet with gravity assist")
362.
363.                 else:
364.                     r0, v0, rf, vf, va, vb, rr_planet1, rr_planet2,
    times_vector = transit(date_launch, date_assist[0], 'earth', as1)
365.                     r02, v02, rf2, vf2, va2, vb2, rr_planet12,
    rr_planet22, times_vector2 = transit(date_assist[0], date_assist[1], as1,
    as2)
366.                     r03, v03, rf3, vf3, va3, vb3, rr_planet13,
    rr_planet23, times_vector3 = transit(date_assist[1], date_arrival, as2,
    planet)
367.
368.                     ss0_trans = Orbit.from_vectors(Sun, r0, va,
    date_launch)
369.                     ssf_trans = Orbit.from_vectors(Sun, rf, vb,
    date_assist[0])
370.                     ss0_trans2 = Orbit.from_vectors(Sun, r02, va2,
    date_assist[0])
371.                     ssf_trans2 = Orbit.from_vectors(Sun, rf2, vb2,
    date_assist[1])
372.                     ss0_trans3 = Orbit.from_vectors(Sun, r03, va3,
    date_assist[1])
373.                     ssf_trans3 = Orbit.from_vectors(Sun, rf3, vb3,
    date_arrival)
374.
375.                     ss_e = Orbit.from_vectors(Sun, r0, v0, date_launch)
376.                     ss_p1 = Orbit.from_vectors(Sun, rf, vf,
    date_assist[0])
377.                     ss_p2 = Orbit.from_vectors(Sun, rf2, vf2,
    date_assist[1])
378.                     ss_p3 = Orbit.from_vectors(Sun, rf3, vf3,
    date_arrival)
379.
380.                     orb = OrbitPlotter()
381.                     orb.plot(ss_e, label= Earth, color=color_planet_e)
382.                     pl_planet, color_planet = planets2plot(as1)
383.                     orb.plot(ss_p1, label= pl_planet, color=color_planet)
384.                     pl_planet, color_planet = planets2plot(as2)
385.                     orb.plot(ss_p2, label= pl_planet, color=color_planet)
386.                     pl_planet, color_planet = planets2plot(planet)
```

```
387.                     orb.plot(ss_p3, label= pl_planet, color=color_planet)
388.
389.                     frame = myplot.OrbitPlotter3D()
390.                     frame.set_attractor(Sun)
391.
392.                     frame.plot_trajectory(rr_planet1, label=Earth,
    color=color_planet_e)
393.                     frame.plot(ss_e, label= Earth, color=color_planet_e)
394.                     pl_planet, color_planet = planets2plot(as1)
395.                     frame.plot(ss_p1, label= pl_planet,
    color=color_planet)
396.
397.                     pl_planet, color_planet = planets2plot(as2)
398.                     frame.plot(ss_p2, label= pl_planet,
    color=color_planet)
399.
400.                     pl_planet, color_planet = planets2plot(planet)
401.                     frame.plot(ss_p3, label= pl_planet,
    color=color_planet)
402.
403.                     pl_planet, color_planet = planets2plot(as1)
404.                     frame.plot_trajectory(rr_planet2, label=pl_planet,
    color=color_planet)
405.                     #frame.plot_trajectory(ss0_trans.sample(times_vector),
    label="Mission trajectory", color=color_trans)
406.
407.                     pl_planet, color_planet = planets2plot(as2)
408.                     frame.plot_trajectory(rr_planet22, label= pl_planet,
    color=color_planet)
409.
    #frame.plot_trajectory(ss0_trans2.sample(times_vector2), label="Mission
    trajectory2", color=color_trans)
410.
411.                     pl_planet, color_planet = planets2plot(planet)
412.                     frame.plot_trajectory(rr_planet23, label= pl_planet,
    color=color_planet)
413.
    #frame.plot_trajectory(ss0_trans3.sample(times_vector3), label="Mission
    trajectory3", color=color_trans)
414.
415.                     frame.set_view(30 * u.deg, 260 * u.deg, distance=3 *
    u.km)
416.                     frame.show(title=" Mission: from Earth to Solar System
    Planet with gravity assist")
417.
418.
419.             else:
```

```
                r0, v0, rf, vf, va, vb, rr_planet1, rr_planet2,
    times_vector = transit(date_launch, date_assist[0], 'earth', as1)
                r02, v02, rf2, vf2, va2, vb2, rr_planet12, rr_planet22,
    times_vector2 = transit(date_assist[0], date_assist[1], as1, as2)
                r03, v03, rf3, vf3, va3, vb3, rr_planet13, rr_planet23,
    times_vector3 = transit(date_assist[1], date_assist[2], as2, as3)
                r04, v04, rf4, vf4, va4, vb4, rr_planet14, rr_planet24,
    times_vector4 = transit(date_assist[2], date_arrival, as3, planet)

                ss0_trans = Orbit.from_vectors(Sun, r0, va, date_launch)
                ssf_trans = Orbit.from_vectors(Sun, rf, vb,
    date_assist[0])
                ss0_trans2 = Orbit.from_vectors(Sun, r02, va2,
    date_assist[0])
                ssf_trans2 = Orbit.from_vectors(Sun, rf2, vb2,
    date_assist[1])
                ss0_trans3 = Orbit.from_vectors(Sun, r03, va3,
    date_assist[1])
                ssf_trans3 = Orbit.from_vectors(Sun, rf3, vb3,
    date_assist[2])
                ss0_trans4 = Orbit.from_vectors(Sun, r04, va4,
    date_assist[2])
                ssf_trans4 = Orbit.from_vectors(Sun, rf4, vb4,
    date_arrival)

                ss_e = Orbit.from_vectors(Sun, r0, v0, date_launch)
                ss_p1 = Orbit.from_vectors(Sun, rf, vf, date_assist[0])
                ss_p2 = Orbit.from_vectors(Sun, rf2, vf2, date_assist[1])
                ss_p3 = Orbit.from_vectors(Sun, rf3, vf3, date_assist[2])
                ss_p4 = Orbit.from_vectors(Sun, rf4, vf4, date_arrival)

                orb = OrbitPlotter()
                orb.plot(ss_e, label= Earth, color=color_planet_e)
                pl_planet, color_planet = planets2plot(as1)
                orb.plot(ss_p1, label= pl_planet, color=color_planet)
                pl_planet, color_planet = planets2plot(as2)
                orb.plot(ss_p2, label= pl_planet, color=color_planet)
                pl_planet, color_planet = planets2plot(as3)
                orb.plot(ss_p3, label= pl_planet, color=color_planet)
                pl_planet, color_planet = planets2plot(planet)
                orb.plot(ss_p4, label= pl_planet, color=color_planet)

                frame = myplot.OrbitPlotter3D()
                frame.set_attractor(Sun)

                frame.plot_trajectory(rr_planet1, label=Earth,
    color=color_planet_e)
                frame.plot(ss_e, label= Earth, color=color_planet_e)
```

```
                pl_planet, color_planet = planets2plot(as1)
                frame.plot(ss_p1, label= pl_planet, color=color_planet)


                pl_planet, color_planet = planets2plot(as2)
                frame.plot(ss_p2, label= pl_planet, color=color_planet)

                pl_planet, color_planet = planets2plot(as3)
                frame.plot(ss_p3, label= pl_planet, color=color_planet)

                pl_planet, color_planet = planets2plot(planet)
                frame.plot(ss_p4, label= pl_planet, color=color_planet)

                pl_planet, color_planet = planets2plot(as1)
                frame.plot_trajectory(rr_planet2, label=pl_planet,
color=color_planet)
                #frame.plot_trajectory(ss0_trans.sample(times_vector),
label="Mission trajectory", color=color_trans)

                pl_planet, color_planet = planets2plot(as2)
                frame.plot_trajectory(rr_planet22, label= pl_planet,
color=color_planet)
                #frame.plot_trajectory(ss0_trans2.sample(times_vector2),
label="Mission trajectory2", color=color_trans)

                pl_planet, color_planet = planets2plot(as3)
                frame.plot_trajectory(rr_planet23, label= pl_planet,
color=color_planet)
                #frame.plot_trajectory(ss0_trans3.sample(times_vector3),
label="Mission trajectory3", color=color_trans)

                pl_planet, color_planet = planets2plot(planet)
                frame.plot_trajectory(rr_planet24, label= pl_planet,
color=color_planet)
                #frame.plot_trajectory(ss0_trans4.sample(times_vector4),
label="Mission trajectory4", color=color_trans)

                frame.set_view(30 * u.deg, 260 * u.deg, distance=3 * u.km)
                frame.show(title="Mission: from Earth to Solar System
Planet with gravity assist")


    def opt_plot(x,y1,y2):
        trace1 = go.Scatter(
            x = plot_date,
            y = plot_mprop,
            name='Masa materiału pędnego',
            line = dict(
```

```
                color = ('rgb(93,75,144)')
            )
        )
        trace2 = go.Scatter(
            x = plot_date,
            y = plot_dv,
            name='Zmiana prędkosci',
            yaxis='y2',
            line = dict(
                color = ('rgb(110,164,193)')
            )
        )
        data =[trace1, trace2]

        layout = go.Layout(

            title='Wykres zmiany prędkosci oraz masy materiału pędnego',
            yaxis=dict(
                title='M materiału pędnego [kg]'
            ),
            xaxis=dict(
                title='Data startu misji'
            ),
            yaxis2=dict(
                title='Delta V [km/s]',
                overlaying='y',
                side='right'
                )
        )
        fig = go.Figure(data=data, layout=layout)
        plot_url = plot(fig, filename='parametry.html')

        return()
    """
    ------------------------------------------------------------------------
--------
    """

    print ('='*80)
    print('Analiza misji do planet Układu Słonecznego')
    print ('='*80)
    print()

    nr, m_spacecraft, I_sp, H, planet, date_launch, date_arrival, as1,
as2, as3, transit_min, transit_max = input_data.inputf()

    solar_system_ephemeris.set("jpl")

```

```
541.      if nr==1:
542.          date_assist = 0
543.          mission_plot(date_launch, date_arrival, date_assist, planet)
544.
545.      if nr in [2,3]:
546.          step1 = 20 * u.day #krok analizy optymalizacyjnej
547.          step2 = 1 * u.day
548.
549.          print()
550.          print('='*80)
551.          print('Wyniki analizy wstepnej')
552.          print('='*80)
553.          # analiza wstępna dla kroku 1
554.          delta_v, date_launch_final, date_arrival_final, date_assist_final,
    m_prop, plot_date, plot_mprop, plot_dv = start_date_optimal(H, date_launch,
    date_arrival, m_spacecraft, I_sp, step1)
555.
556.          print('-'*80)
557.          print('Koniec analizy wstepnej')
558.          print('Optymalna data startu:', date_launch_final.iso[0:10])
559.          print('-'*80)
560.
561.          print('='*80)
562.          print('Czy przeprowadzić analizę szczegółową?')
563.          an = input('Wpisz "tak" lub "nie": ')
564.          an = str(an)
565.          check = True
566.          while check:
567.              if an in ['tak','nie']:
568.                  check = False
569.                  check2 = True
570.                  while check2:
571.                      if an == 'tak':
572.                          print('='*80)
573.                          print('Wyniki analizy szczegolowej')
574.                          print('='*80)
575.                          date0_prec = date_launch_final - 20 * u.day
    #zawezenie czasu analizy
576.                          date1_prec = date_arrival_final + 20 * u.day
577.                          # analiza szczegolowa
578.                          delta_v, date_launch_final, date_arrival_final,
    date_assist_final, m_prop, plot_date, plot_mprop, plot_dv =
    start_date_optimal(H, date0_prec, date1_prec, m_spacecraft, I_sp, step2)
579.
580.                          print()
581.                          print('Koniec analizy szczegolowej')
582.                          print('-'*80)
583.                          check2 = False
```

```
584.                     else:
585.                         check2 = False
586.             else:
587.                 print('Wprowadzone słowo jest niepoprawne')
588.         #prezentacja wynikow
589.         print()
590.         print('='*80)
591.         print('Wyniki analizy optymalizacyjnej')
592.         print('='*80)
593.         print()
594.         print('Optymalna data startu:', date_launch_final.iso[0:10])
595.         print('Data dolotu do planety:', date_arrival_final.iso[0:10])
596.         print('Czas lotu:', int((date_arrival_final -
    date_launch_final).jd),'dni')
597.         print('Wymagana zmiana predkosci: %.3f km/s' % float(delta_v /
    u.km * u.s))
598.         print('Masa zuzytego materialu pednego: %i kg' % int(m_prop /
    u.kg))
599.         print('-'*80)
600.
601.         opt_plot( plot_date, plot_mprop, plot_dv)
602.
603.         mission_plot(date_launch_final, date_arrival_final,
    date_assist_final, planet)
604.
605.     if nr == 4:
606.         mp, t2_s, deltaVtotal, accel = lowthrust(planet,H, I_sp,
    m_spacecraft, date_launch, date_arrival)
607.
608.         print()
609.         print('='*80)
610.         print('Analiza lotu z małym ciagiem')
611.         print('='*80)
612.         print()
613.         print('Przyspieszenie: %.10f km/s^2' % float(accel))
614.         print('Całkowita zmiana prędkosci: %f km/s' % float(deltaVtotal))
615.         print('Masa zuzytego materialu pednego: %f kg' % float(mp))
616.         print('Czas lotu: %i dni' % int(t2_s))
```

Funkcja orbital_elements.py

```
"""
Created: 2018-02-04
Author: Filip Perczyński
-------------------------------------------------------------------------------
Funkcja wczytująca parametry orbitalne planet ze strony internetowej JPL:
    http://ssd.jpl.nasa.gov/txt/p_elem_t1.txt
-------------------------------------------------------------------------------
"""
import requests
import re
import astropy.units as u
import numpy as np

def orbitalements(planet):
    tle_url = "http://ssd.jpl.nasa.gov/txt/p_elem_t1.txt"
    orb_elem = requests.get(tle_url).text
    orb_file = open('Orbital_elements.txt','w')
    orb_file.write(orb_elem)
    orb_file.close()
    orbital_elem = open('Orbital_elements.txt', 'r').readlines()

    orbital_elem = [a for a in orbital_elem if a != '\n']
    orbital_elem = ''.join(orbital_elem)
    lines = orbital_elem.split("\n")

    lines = [re.split("\s{2,}", line) for line in lines[12:]]

    for i in range(0, len(lines)-1, 2):
        name = lines[i][0]
        elements = lines[i][1:]

        if name == "EM Bary":
            name = "Earth"

        (semi_major_axis, eccentricity, inclination, mean_longitude,
            longitude_of_periapsis, longitude_of_ascending_node) = elements

        #body = bodies.setdefault(name, {})
        if name == planet:
            body = {"name": name,
                "semi_major_axis": float(semi_major_axis) ,
                "eccentricity": eccentricity,
                "inclination": float(inclination),
                "longitude_of_periapsis": longitude_of_periapsis,
```

```
                "longitude_of_ascending_node": longitude_of_ascending_node,
            }

    a = body["semi_major_axis"]
    a = a
    inc = body['inclination']
    inc = np.deg2rad(inc)

    return(a,inc)
```

Funkcja input_data.py

```
"""
Created: 2018-01-05
Author: Filip Perczyński
--------------------------------------------------------------------------------
Funkcja wprowadzania danych wejsciowych:
- funkcja mission_date() - wprowadzenie daty startu i rozpoczęcia misji
    w formacie [rrrr]-[mm]-[dd], sprawdzenie poprawnosci danych np. formatu,
    liczby dni, miesiecy, lat przestepnych
- funkcja planets() - zwraca nazwę planety w formacie odczytywanym przez
poliastro oraz
    wartosc minimalnej i maksymalnej liczby dni potrzebnych na podroz do
planety
- funkcja inputf() - wprowadzenie danych przez użytkownika oraz sprawdzenie
ich
    poprawnosci, funkcja zwraca następujące dane:
        - nr - wybór opcji obliczen,
        - m_spacecraft - masa statku kosmicznego,
        - I_sp - impuls wlasciwy,
        - H - wysokosc orbity poczatkowej,
        - planet - wybrana planeta,
        - date_launch - data startu,
        - date_arrival - data końca misji,
        - transit_min - minimalna liczba dni podrozy,
        - transit_max - maksymalna liczba dni podrozy,
        - as1, as2, as3 - planety wykorzystywane do asysty grawitacyjnej.
--------------------------------------------------------------------------------
"""
import astropy.units as u
from astropy import time

def mission_date():
    check = True
    while check:
```

```
        print()
        print('Data w granicach 1900 - 2100 rok')
        date_ymd =input(('Rok-Miesiac-Dzien [rrrr]-[mm]-[dd]: '))

        A=[]
        A=date_ymd.split("-")
        if "" in A:
            A.remove("")
        else:
            A = A

        if date_ymd.count("-") != 2:
            print()
            print (('Zly format daty'))
        elif len(A) != 3:
            print()
            print (('Zly format daty'))
        else:
            y,m,d = date_ymd.split("-")

            if 1899 < int(y) < 2101:
                if 0 < int(m) < 13:
                    if int(m) in [1, 3, 5, 7, 8, 10, 12]:
                        if 0 < int(d) < 32:
                            check = False
                        else:
                            print()
                            print(('Podany dzien jest bledny'))
                    elif int(m) in [4, 6, 9, 11]:
                        if 0 < int(d) < 31:
                            check = False
                        else:
                            print()
                            print(('Podany dzien jest bledny'))
                    elif int(m) == 2:
                        if (int(y) % 4) == 0:
                            if (int(y) % 100) == 0:
                                if (int(y) % 400) == 0:
                                    if 0 < int(d) < 30:
                                        check = False
                                    else:
                                        print()
                                        print(('Podany dzien jest bledny'))
                                else:
                                    if 0 < int(d) < 29:
                                        check = False
                                    else:
                                        print()
```

```
                                        print(('Podany dzien jest bledny'))
                                else:
                                    if 0 < int(d) < 30:
                                        check = False
                                    else:
                                        print()
                                        print(('Podany dzien jest bledny'))
                            else:
                                 if 0 < int(d) < 29:
                                     check = False
                                 else:
                                     print()
                                     print(('Podany dzien jest bledny'))
                    else:
                        print()
                        print(('Podany miesiac jest bledny'))
                 else:
                     print()
                     print(('Podany rok jest poza zakresem'))

    date = str(date_ymd) + ' 12:00'
            date = time.Time(date, format='iso', scale='utc')
            return (date)


        def planets(planet):
            if planet == 'Merkury':
                planet = 'mercury'
                as1 = 0
                as2 = 0
                as3 = 0
                trans_min = 100
                trans_min_as1 = 0
                trans_min_as2 = 0
                trans_min_as3 = 0
                trans_min_v = 0
                trans_max = 200
                trans_max_as1 = 0
                trans_max_as2 = 0
                trans_max_as3 = 0
                trans_max_v = 0
            if planet == 'Wenus':
                planet = 'venus'
                as1 = 0
                as2 = 0
                as3 = 0
                trans_min = 50
                trans_min_as1 = 0
```

```
127.             trans_min_as2 = 0
128.             trans_min_as3 = 0
129.             trans_min_v = 0
130.             trans_max = 100
131.             trans_max_as1 = 0
132.             trans_max_as2 = 0
133.             trans_max_as3 = 0
134.             trans_max_v = 0
135.         if planet == 'Mars':
136.             planet = 'mars'
137.             as1 = 'venus'
138.             as2 = 0
139.             as3 = 0
140.             trans_min = 100
141.             trans_min_as1 = 50
142.             trans_min_as2 = 0
143.             trans_min_as3 = 0
144.             trans_min_v = 100
145.             trans_max = 150
146.             trans_max_as1 = 100
147.             trans_max_as2 = 0
148.             trans_max_as3 = 0
149.             trans_max_v = 300
150.         if planet == 'Jowisz':
151.             planet = 'jupiter'
152.             as1 = 'venus'
153.             as2 = 0
154.             as3 = 0
155.             trans_min = 300
156.             trans_min_as1 = 50
157.             trans_min_as2 = 200
158.             trans_min_as3 = 0
159.             trans_min_v = 300
160.             trans_max = 500
161.             trans_max_as1 = 100
162.             trans_max_as2 = 600
163.             trans_max_as3 = 0
164.             trans_max_v = 500
165.         if planet == 'Saturn':
166.             planet = 'saturn'
167.             as1 = 'venus'
168.             as2 = 'jupiter'
169.             as3 = 0
170.             trans_min = 1000
171.             trans_min_as1 = 50
172.             trans_min_as2 = 700
173.             trans_min_as3 = 700
174.             trans_min_v = 1000
```

```
175.             trans_max = 2000
176.             trans_max_as1 = 100
177.             trans_max_as2 = 1000
178.             trans_max_as3 = 1000
179.             trans_max_v = 2000
180.         if planet == 'Uran':
181.             planet = 'uranus'
182.             as1 = 'venus'
183.             as2 = 'jupiter'
184.             as3 = 'saturn'
185.             trans_min = 4000
186.             trans_min_as1 = 50
187.             trans_min_as2 = 2000
188.             trans_min_as3 = 2000
189.             trans_min_v = 3000
190.             trans_max = 5000
191.             trans_max_as1 = 100
192.             trans_max_as2 = 2500
193.             trans_max_as3 = 2500
194.             trans_max_v = 3500
195.         if planet == 'Neptun':
196.             planet = 'neptune'
197.             as1 = 'jupiter'
198.             as2 = 'saturn'
199.             as3 = 'uranus'
200.             trans_min = 5000
201.             trans_min_as1 = 1000
202.             trans_min_as2 = 2000
203.             trans_min_as3 = 3000
204.             trans_min_v = 3000
205.             trans_max = 6000
206.             trans_max_as1 = 1500
207.             trans_max_as2 = 2500
208.             trans_max_as3 = 3500
209.             trans_max_v = 3500
210.
211.         transit_min = [trans_min, trans_min_as1, trans_min_as2,
    trans_min_as3, trans_min_v]* u.day
212.         transit_max = [trans_max, trans_max_as1, trans_max_as2,
    trans_max_as3, trans_max_v]* u.day
213.         return planet, as1, as2, as3, transit_min, transit_max
214.
215.
216.     def inputf():
217.
218.         print ('Możliwe są dwie rodzaje analiz:')
219.         print ('1 - trajektoria lotu bezposredniego do wybranej planety na
    podstawie daty startu i ladowania')
```

```
220.         print ('2 - optymalizacja daty startu i lądowania ze względu na
     koszt transferu dla lotu bezporedniego')
221.         print ('3 - optymalizacja daty startu i lądowania ze względu na
     koszt transferu dla lotu z wykorzystaniem asysty grawitacyjnej dla
     zewnętrznych planet Układu Słonecznego')
222.         print ('4 - obliczenia transferu z małym ciągiem')
223.         opt = input(('Wybierz odpowiedni numer:  '))
224.         nr = int(opt)
225.         print()
226.         print('-'*80)
227.         print (('Wprowadź parametry statku kosmicznego'))
228.         print('-'*80)
229.
230.         if nr in [2,3,4]:
231.             check = True
232.             while check:
233.                 print()
234.                 print ('Masa własna statku kosmicznego w granicach 0-150
     000 kg')
235.                 m_spacecraft = int(input(('M [kg]: ')))
236.
237.                 if 0 < m_spacecraft < 100000:
238.                     check = False
239.                 else:
240.                     print(('Podana masa jest poza zakresem'))
241.
242.             check = True
243.             while check:
244.                 print()
245.                 print ('Impuls własciwy statku kosmicznego w granicach 0-
     150 000 kg')
246.                 I_sp = int(input(('Isp [m/s]: ')))
247.
248.                 if 0 < I_sp < 100000:
249.                     check = False
250.                 else:
251.                     print(('Podany impuls jest poza zakresem'))
252.
253.             check = True
254.             while check:
255.                 print()
256.                 print('Wysokosc poczatkowej orbity kolowej w granicach 100
     - 10 000 km')
257.                 H = int(input(('H [km]: ')))
258.
259.                 if 100 < H < 10000:
260.                     check = False
261.                 else:
```

```
                    print(('Podana wysokosc jest poza zakresem'))

        check = True
        while check:
            print()
            print(('Wybierz cel misji - jedna z planet Ukladu Slonecznego'))
            print(('Merkury, Wenus, Mars, Jowisz, Saturn, Uran, Neptun'))
            planet = input(('Cel: '))

            if planet in ['Merkury', 'Wenus', 'Mars', 'Jowisz', 'Saturn', 'Uran', 'Neptun']:
                check = False
                planet, as1, as2, as3, transit_min, transit_max = planets(planet)
                if nr == 3:
                    print()
                    if as2 == 0:
                        print('Lot z wykorzystaniem asysyty grawitacyjnej:', as1.title())
                    elif as3 ==0:
                        print('Lot z wykorzystaniem asysyty grawitacyjnej:', as1.title(), 'i', as2.title())
                    else:
                        print('Lot z wykorzystaniem asysyty grawitacyjnej:', as1.title(), ',', as2.title(), 'i', as3.title())

            else:
                print(('Podana planeta jest bledna'))

        print()
        print(('Wprowadź datę startu misji'))
        date_launch = mission_date()
        print()
        print(('Wprowadź datę zakończenia misji'))
        date_arrival = mission_date()

    # Parametry niewykorzystywane w analize 1
        if nr == 1:
            m_spacecraft = 0
            H = 0
            I_sp = 0
    # nadanie wprowadzonym parametrom jednostek
        m_spacecraft = m_spacecraft * u.kg
        H = H * u.km
        I_sp = I_sp/1000 * u.km / u.s
        date_launch = time.Time(date_launch.jd, format='jd', scale='utc')
```

```
303.          date_arrival = time.Time(date_arrival.jd, format='jd',
    scale='utc')
304.
305.          return nr, m_spacecraft, I_sp, H, planet, date_launch,
    date_arrival, as1, as2, as3, transit_min, transit_max
306.
```

Funkcja myplot.py

```
1. """
2. Created on Sun Jan  7 16:48:00 2018
3. """
4.
5. """ Plotting utilities.
6. """
7. import os.path
8. from itertools import cycle
9.
10. import numpy as np
11.
12. from typing import List, Tuple
13. import plotly.colors
14. from plotly.offline import plot
15.
16. from plotly.graph_objs import Scatter3d, Surface, Layout
17.
18. from astropy import units as u
19.
20. from poliastro.util import norm
21. from poliastro.twobody.orbit import Orbit
22.
23. BODY_COLORS = {
24.     "Sun": "#ffcc00",
25.     "Earth": "#204a87",
26.     "Jupiter": "#ba3821",
27. }
28.
29.
30.
31.
32. def plot3d(orbit, *, label=None, color=None):
33.     frame = OrbitPlotter3D()
34.     frame.plot(orbit, label=label, color=color)
35.     frame.show()
36.
37.     return frame
38.
39. def _generate_label(orbit, label):
```

```
    orbit.epoch.out_subfmt = 'date_hm'
    label_ = '{}'.format(orbit.epoch.iso)
    if label:
        label_ += ' ({})'.format(label)

    return label_


def _generate_sphere(radius, center, num=20):
    u1 = np.linspace(0, 2 * np.pi, num)
    v1 = u1.copy()
    uu, vv = np.meshgrid(u1, v1)

    x_center, y_center, z_center = center

    xx = x_center + radius * np.cos(uu) * np.sin(vv)
    yy = y_center + radius * np.sin(uu) * np.sin(vv)
    zz = z_center + radius * np.cos(vv)

    return xx, yy, zz


def _plot_sphere(radius, color, name, center=[0, 0, 0] * u.km):
    xx, yy, zz = _generate_sphere(radius, center)
    sphere = Surface(
        x=xx.to(u.km).value, y=yy.to(u.km).value, z=zz.to(u.km).value,
        name=name,
        colorscale=[[0, color], [1, color]],
        cauto=False, cmin=1, cmax=1, showscale=False,  # Boilerplate
    )
    return sphere



class OrbitPlotter3D:
    """OrbitPlotter3D class.
    """
    def __init__(self):
        self._layout = Layout(
            autosize=True,
            scene=dict(
                xaxis=dict(
                    title="x (km)",
                ),
                yaxis=dict(
                    title="y (km)",
                ),
```

```
                zaxis=dict(
                    title="z (km)",
                ),
                aspectmode="data",  # Important!
            ),
        )
        self._data = []  # type: List[dict]

        # TODO: Refactor?
        self._attractor = None
        self._attractor_data = {}  # type: dict
        self._attractor_radius = np.inf * u.km

            self._color_cycle = cycle(plotly.colors.DEFAULT_PLOTLY_COLORS)

        @property
        def figure(self):
            return dict(
                data=self._data + [self._attractor_data],
                layout=self._layout,
            )

        def _redraw_attractor(self, min_radius=0 * u.km):
            # Select a sensible value for the radius: realistic for low orbits,
            # visible for high and very high orbits
            radius = max(self._attractor.R.to(u.km), min_radius.to(u.km))

            # If the resulting radius is smaller than the current one, redraw it
            if radius < self._attractor_radius:
                sphere = _plot_sphere(
                    radius, BODY_COLORS.get(self._attractor.name, "#999999"), self._attractor.name)

                # Overwrite stored properties
                self._attractor_radius = radius
                self._attractor_data = sphere

        def _plot_trajectory(self, trajectory, label, color, dashed):
            trace = Scatter3d(
                x=trajectory.x.to(u.km).value, y=trajectory.y.to(u.km).value, z=trajectory.z.to(u.km).value,
                name=label,
                line=dict(
                    color=color,
                    width=5,
                    dash='dash' if dashed else 'solid',
```

```
                ),
                mode="lines",  # Boilerplate
            )
            self._data.append(trace)

        def set_attractor(self, attractor):
            """Sets plotting attractor.
            Parameters
            ----------
            attractor : ~poliastro.bodies.Body
                Central body.
            """
            if self._attractor is None:
                self._attractor = attractor

            elif attractor is not self._attractor:
                raise NotImplementedError("Attractor has already been set to {}.".format(self._attractor.name))

        @u.quantity_input(elev=u.rad, azim=u.rad)
        def set_view(self, elev, azim, distance=5):
            x = distance * np.cos(elev) * np.cos(azim)
            y = distance * np.cos(elev) * np.sin(azim)
            z = distance * np.sin(elev)

            self._layout.update({
                "scene": {
                    "camera": {
                        "eye": {
                            "x": x.to(u.km).value, "y": y.to(u.km).value, "z": z.to(u.km).value
                        }
                    }
                }
            })

        def plot(self, orbit, *, label=None, color=None):
            """Plots state and osculating orbit in their plane.
            """
            if color is None:
                color = next(self._color_cycle)

            self.set_attractor(orbit.attractor)

            self._redraw_attractor(orbit.r_p * 0.15)  # Arbitrary threshold

            label = _generate_label(orbit, label)
```

```
            trajectory = orbit.sample()

            self._plot_trajectory(trajectory, label, color, True)

            # Plot sphere in the position of the body
            radius = min(self._attractor_radius * 0.5, (norm(orbit.r) -
orbit.attractor.R) * 0.3)  # Arbitrary thresholds
            sphere = _plot_sphere(
                radius, color, label, center=orbit.r)

            self._data.append(sphere)

        def plot_trajectory(self, trajectory, *, label=None, color=None):
            """Plots a precomputed trajectory.
            An attractor must be set first.
            Parameters
            ----------
            trajectory : ~astropy.coordinates.CartesianRepresentation
                Trajectory to plot.
            """
            if self._attractor is None:
                raise ValueError("An attractor must be set up first,
please use "
                                 "set_attractor(Major_Body).")
            else:
                self._redraw_attractor(trajectory.norm().min() * 0.15)  #
Arbitrary threshold

            self._plot_trajectory(trajectory, str(label), color, False)

        def _prepare_plot(self, **layout_kwargs):
            # If there are no orbits, draw only the attractor
            if not self._data:
                self._redraw_attractor()

            if layout_kwargs:
                self._layout.update(layout_kwargs)

        def show(self, **layout_kwargs):
            self._prepare_plot(**layout_kwargs)

            plot(self.figure)

        def savefig(self, filename, **layout_kwargs):
            self._prepare_plot(**layout_kwargs)

            basename, ext = os.path.splitext(filename)
            export(
```

```
222.                 self.figure,
223.                 image=ext[1:], image_filename=basename,
224.                 show_link=False,  # Boilerplate
225.             )
```