Mechanika Nieba

Misja na Jowisza

z asystą grawitacyjną Marsa

Andrzej Odziemkowski

2018

Spis treści

1. Wstęp

1.1. Cel projektu

Celem poniższego projektu było stworzenie programu, który przeprowadzi obliczenia misji międzyplanetarnej. Podstawowym założeniem było przeprowadzenie misji polegającej na transferze satelity z orbity Ziemi na orbitę Jowisza. Program stworzono w języku *Python* używając bibliotek wspierających mechanikę orbitalną i astronomię, np.: *Poliastro, astropy.*

Wynikami obliczeń są sumaryczne przyrosty prędkości niezbędne do udanego przeprowadzenia zaplanowanych manewrów. Dodatkowo zakładając impuls właściwy napędu, używając równania Ciołkowskiego jest możliwe określenie jaka ilość materiału pędnego zostanie użyta podczas misji.

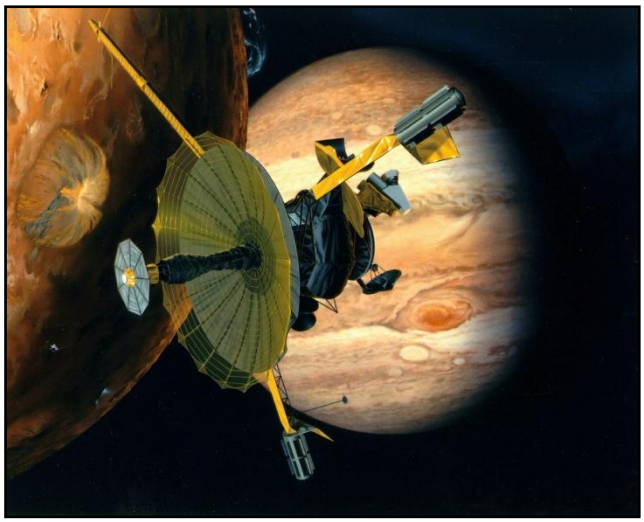
Podczas misji międzyplanetarnych manewr zwany asystą grawitacyjną, pozwala wykorzystać ciała niebieskie w celu zwiększenie prędkości satelity bez zużywania materiału pędnego. Możliwe asysty są determinowane przez pozycje ciał niebieskich w układzie słonecznym w związku z czym zależnie od daty manewr transferu może być mniej lub bardziej kosztowny pod względem zużycia paliwa. Bardzo ważnym zagadnieniem jest określenie tych furtek, gdy transfer jest najbardziej optymalny.

Aby zrealizować wszystkie wspomniane wcześniej cele, opisywanemu satelicie nadano następujące parametry:

- masa własna satelity
- impuls właściwy napędu
- wysokość początkowej orbity kołowej dookoła Ziemi
- data startu, zakończenia misji

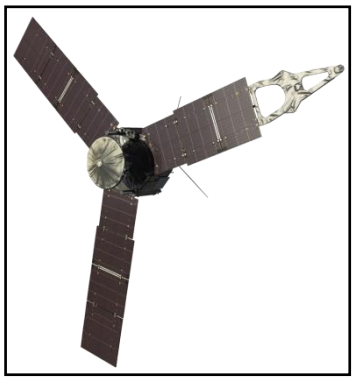
1.2. Misja Galileo i Juno

Jowisz od czasów starożytnych był interesującym obiektem na nieboskłonie. Dzięki użyciu teleskopu, włoski astronom i filozof Galileusz, zaobserwował 4 największe księżyce Jowisza zwane obecnie galileuszowymi: Io, Europa, Ganimedes i Kallisto. Największa planeta naszego układu słonecznego charakteryzuje się również układem barwnych pasów i 'oczek' opasających planetę. Najbardziej znanym z nich jest Wielka Czerwona Plama, ogromny antycyklon zaobserwowany już w XXVII wieku, od wieków pobudzał ludzką wyobraźnię.



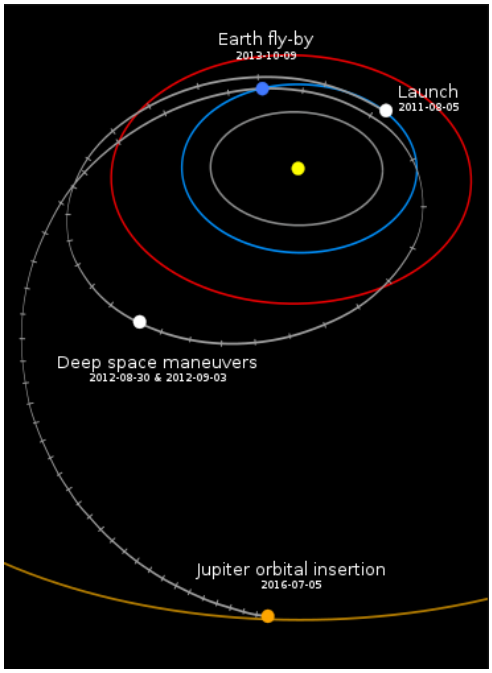
Rys 1 - satelita Galileo

Wraz z nadejściem czasów nowożytnych, Jowisz stał się naturalnym celem agencji kosmicznych. Galileo - bezzałogowa sonda kosmiczna została wystrzelona w 1989 roku przez agencję kosmiczną NASA w celu wykonania badań Jowisza, jego księżyców i pierścieni. W grudniu 1995 r. sonda stała się pierwszym sztucznym satelitą Jowisza oraz wprowadziła w jego atmosferę próbnik z aparaturą pomiarową.



Rys 2 - satelita Juno

Przesyłając ogromną ilość danych na Ziemię Galileo przyczynił się w ogromnym stopniu do zrozumienia gazowych olbrzymów oraz ich księżyców. W 2002 roku nastąpił koniec misji. Na następcę przyszło czekać do 2016 roku kiedy satelita Juno weszła na orbitę Jowisza. W celu zmniejszenia masy materiały pędnego wykonała ona asystę grawitacyjną Ziemi. Obie te, niezwykle ważne misje stanowiły inspirację dla tematu mojego projektu z Mechaniki Nieba.



Rys 3 - przebieg misji Juno

2. Narzędzia do realizacji programu

2.1. Python

Język programowania wysokiego poziomu ogólnego przeznaczenia, o rozbudowanym pakiecie bibliotek standardowych, którego ideą przewodnią jest czytelność i klarowność kodu źródłowego. Jego składnia cechuje się przejrzystością i zwięzłością. Python wspiera różne paradygmaty programowania: obiektowy, imperatywny oraz w mniejszym stopniu funkcyjny. Posiada w pełni dynamiczny system typów i automatyczne zarządzanie pamięcią.

Ze względu na to, że manualne odwzorowanie w Pythonie układu słonecznego, manewrów orbitalnych itd. byłoby niezwykle problematyczne, zastosowano pewne biblioteki posiadające już takie funkcje.

2.2. Poliastro.

Poliastro jest open-sourcowym pakietem dedykowanym problemom Astrodynamiki i Mechaniki orbitalnej takim jak propagacja na orbicie, rozwiązanie problemu Lambert'a, uzyskanie wektorów prędkości i położenia.

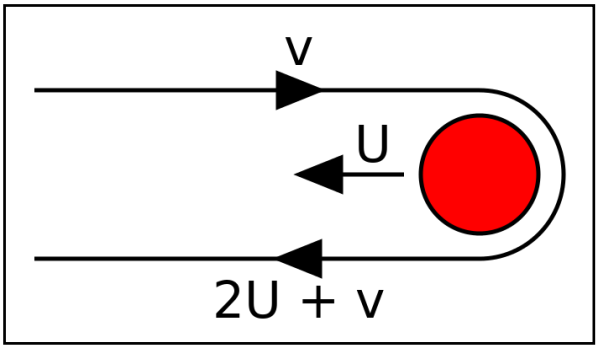
2.3. Astropy

Astropy to open-soursowy pakiet, który zawiela podstawowe narzędzia pozwalające na pisanie programów związanych z astronomią i astrofizyką w Pythonie.

3. Podstawy mechaniki orbitalnej

3.1. Asysta grawitacyjna

Jest to zmiana prędkości i kierunku lotu 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. Manewrowanie w przestrzeni międzyplanetarnej wymaga brania pod uwagę grawitacji Słońca. Pojazdy wysyłane w kierunku wewnętrznych planet – Wenus i Merkurego, zbliżając się do Słońca nabierają prędkości i aby wejść na ich orbitę muszą ją jakoś zmniejszyć. Z kolei pojazdy wysyłane w kierunku zewnętrznych planet muszą nabrać odpowiedniej prędkości, aby móc oddalić się na wystarczającą odległość od Słońca. Realizacja tego przy pomocy napędu rakietowego wymaga dużych ilości paliwa – dlatego poszukuje się innych metod.



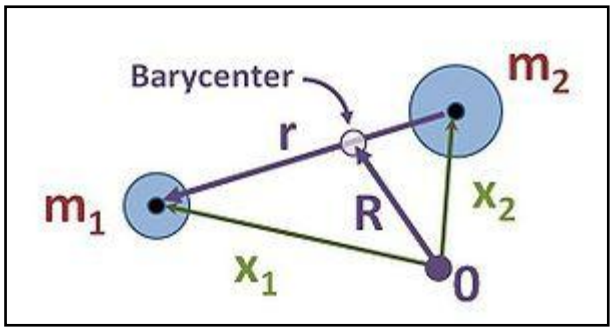
Rys 4 - Uproszczony schemat asysty grawitacyjnej

3.2. Zagadnienie dwóch ciał

Zagadnienie to opisuje ruch dwóch ciał względem wspólnego środka ciężkości. Ruch układu składającego się z dwóch ciał wygodnie jest rozpatrywać w układzie odniesienia, w którym środek masy układu jest w spoczynku. W tym układzie każde z nich porusza się po trajektorii, która jest krzywą stożkową, a środek masy znajduje się w jednym z jej ognisk.

Jeśli krzywe, po których poruszają się ciała, są zamknięte, to są one elipsami. Energia potencjalna układu ciał (jest ona ujemna), przewyższa co do wartości bezwzględnej sumaryczną energię kinetyczną układu, a całkowita energia układu jest ujemna (pomijamy tutaj energię kinetyczną obrotu ciał wokół swych osi).

Jeżeli orbity są otwarte, ruch ciał odbywa się po hiperboli lub paraboli.



Rys 5 - Schemat zagadnienia dwóch ciał

Rozwiązanie tego zagadnienia zawiera zię w pakiecie poliastro.twobody. Zdefiniowanie orbit i obliczenia przeprowadzane są na obiekcie klasy nazwanym *Orbit*. W poniższym projekcie analizowany układ składa się ze Słońca, Ziemi, Marsa i Jowisza. Zakładamy, że początkowo orbita dookoła Ziemi jest kołowa.

3.3. Równanie Ciołkowskiego

Celem optymalizacji jest uzyskanie minimalnego zużycia paliwa podczas manewrów. W praktyce oznacza to minimalizację wektora zmiany prędkości tzw. Δv. W połączeniu ze znanym impulsem właściwym napędu możemy obliczyć jaka masa paliwa jest niezbędna do wykonania zadanego manewru. Używając równania Ciołkowskiego:

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

gdzie:

I_{sp} – impuls właściwy napędu [m/s]

m – masa własna satelity [kg]

m_0 – początkowa masa satelity = masa własna + masa paliwa [kg]

Przekształcając wzór otrzymujemy:

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

gdzie:

m_p – masa paliwa [kg]

Otrzymana masa to masa paliwa zużyta podczas manewru. Znając masy zużyte podczas kolejnych manewrów jesteśmy w stanie optymalizować je tak, by sprostać założeniom misji.

Weryfikując prawdziwość wyników uzyskanych za pomocą powyższego wzoru należy mieć w pamięci, że nie uwzględnia on czasu zmiany prędkości, wszelkie jej zmiany odbywają się w

sposób natychmiastowy. W rzeczywistości manewry takie odbywają się w pewnym skończonym czasie, który dla napędów takich jak np. jonowe, może być bardzo długi.

4. Program

Program działa bazując na głównych założeniach projektu, wyniki wyświetlane są w formie tekstowej i wykresów.

Schemat jego działania można streścić w następujący sposób:

- zadanie parametrów wejściowych
- optymalizacja daty wylotu
- optymalizacja transferów między planetarnych
- osiągnięcie orbity Jowisz
- wyświetlenie wyników w postaci tekstowej i wykresów

4.1. Parametry wejściowe

Parametry zadane zostały w sposób bezpośredni, warunki początkowe należy zmienić bezpośrednio w kodzie programu.

Na rysunku 6 przedstawione zostały zadane parametry wejściowe symulacji. Zmiennym *date_0* i *date_1*, które symbolizują początek i koniec okresu, w którym będzie przeprowadzana analiza możliwych transferów, przypisuje się daty, za pomocą klasy *Time*. Daty te są następnie zamieniane na format juliański.

Następnie zmiennym *m_ship*, *I_sp* oraz *H* przypisywane są wartości: masy własnej satelity, impulsu właściwego i wysokości początkowej orbity kołowej.

4.2. Funkcja optymalizująca transfer pomiędzy planetami

Główną częścią funkcji *optimal transit* jest pętla *while*. Odbywa się ona od najwcześniejszej zadanej daty do daty zamykającej analizowane okno startowe. Krok tej pętli jest zadany w głównej części programu. W pętli zadawane są data startu i data przylotu. Następnie dla tych dat obliczane są pozycja i prędkość planety 1 dla daty startu i analogicznie pozycja i prędkość planety 2 dla daty przylotu. Dla tych wartości rozwiązywane jest zagadnienie Lambert'a i uzyskiwany jest niezbędny do wykonania manewru przyrost prędkości satelity. następnie za pomocą instrukcji warunkowej *if else* następuje sprawdzenie, czy przyrost prędkości jest mniejszy w kolejnym kroku czasowym. Funkcja zwraca wektor przyrostów prędkości i datę przylotu.

4.3. Funkcja optymalizująca datę startu

Funkcja *optimal date* ma za zadanie wyznaczyć optymalną datę startu ze względu na koszt transferów. Początkowo zadaje się maksymalne i minimalne czasy trwania poszczególnych transferów. Po inicjalizacji zmiennych rozpoczyna się pętla *while*, która trwa dopóki *date_0* nie zrówna się z *date_1* . Po zakończeniu każdego obiegu pętli czas jest 'przesuwany' do przodu o zadany krok *step*. Na jej początku ustalana jest pozycja i prędkość satelity na początkowej orbicie kołowej wokół Ziemi dla danego czasu. Następnie używając wcześniej zdefiniowanej funkcji *optimal transit* optymalizuje się manewry, dzięki czemu otrzymywane są całkowity koszty paliwa i zmiana prędkości. Kolejne dane istotne z punktu widzenia wyników są dodawane do zdefiniowanych wcześniej zmiennych. Na końcu pętli *while* występuje instrukcja sterująca *if else* mająca za zadanie optymalizację daty startu w każdym kolejnym kroku.

4.4. Główna część programu

Głowna program używa funkcji *optimal_date* do obliczenia przyrostów prędkości, mas paliwa, dat asysty grawitacyjnej Marsa. Następnie używając funkcji sprawdzającej sprawdzany jest warunek osiągnięcia orbity Marsa. Dalsza część głównego programu to wizualizacja wykresów paliwa i przyrostów prędkości.

5. Obliczenia

Obliczenia przeprowadzono dla kilku zestawów danych obrazujących wpływ parametrów początkowych symulacji na wyniki.

Przypadek 1

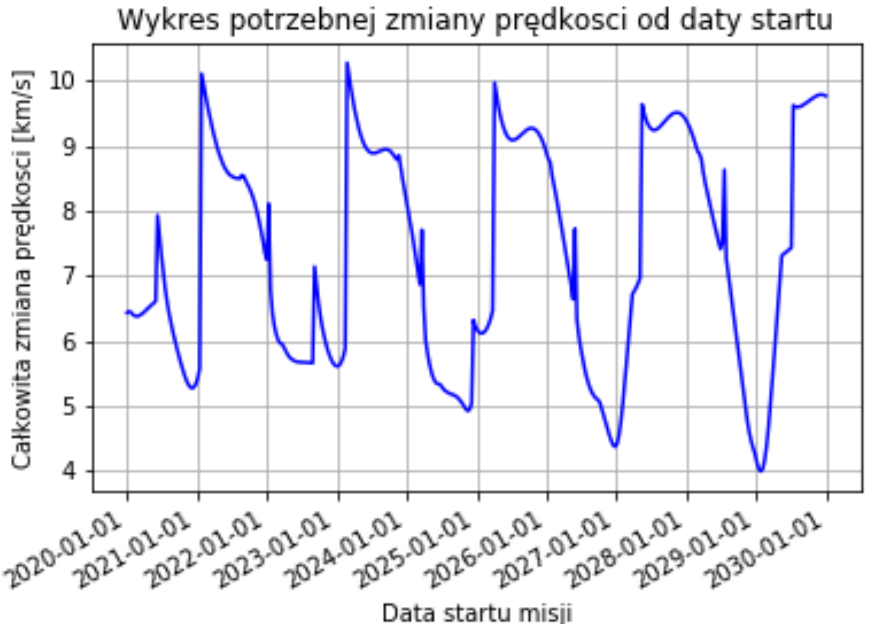
Analizowany przedział czasowy: od 01-01-2020r. g.12 do 01-01-2030r. g.12 .

Minimalny/maksymalny czas trwania transferu Ziemia-Mars: 50/100 dni

Minimalny/maksymalny czas trwania transferu Mars-Jowisz: 400/700 dni

Parametry satelity i początkowej orbity kołowej:

masa	I_sp	H
500	1000	300



Przypadek 2

Analizowany przedział czasowy: od 01-01-2020r. g.12 do 01-01-2030r. g.12 .

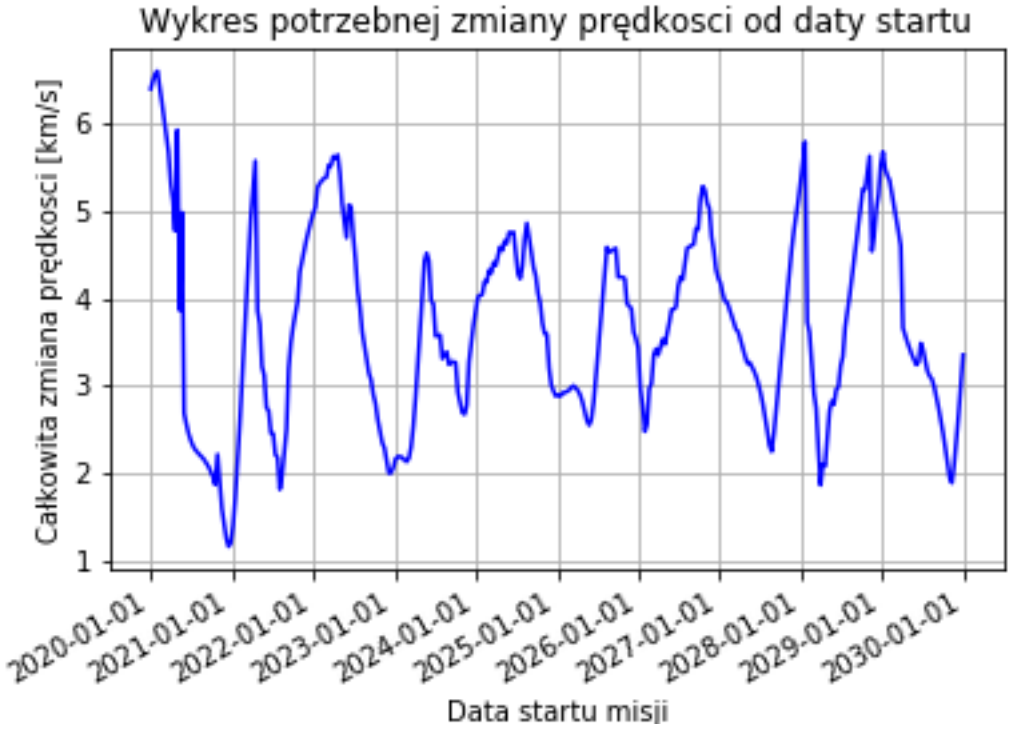
Minimalny/maksymalny czas trwania transferu Ziemia-Mars: 50/400 dni

Minimalny/maksymalny czas trwania transferu Mars-Jowisz: 300/1000 dni

Parametry satelity i początkowej orbity kołowej:

masa	I_sp	H

500	1000	300



Przypadek 3

Analizowany przedział czasowy: od 01-01-2020r. g.12 do 01-01-2030r. g.12 .

Minimalny/maksymalny czas trwania transferu Ziemia-Mars: 50/400 dni

Minimalny/maksymalny czas trwania transferu Mars-Jowisz: 300/1000 dni

Parametry satelity i początkowej orbity kołowej:

masa	I_sp	H
1000	1000	300

Przypadek 4

Analizowany przedział czasowy: od 01-01-2020r. g.12 do 01-01-2030r. g.12 .

Minimalny/maksymalny czas trwania transferu Ziemia-Mars: 50/400 dni

Minimalny/maksymalny czas trwania transferu Mars-Jowisz: 300/1000 dni

Parametry satelity i początkowej orbity kołowej:

masa	I_sp	H
500	500	300

6. Wnioski

Analizując wyniki dla dwóch różnych przedziałów czasowych transferu widać, że średnie przyrosty prędkości, niezbędne do wykonania transferu z asystą, są większe w przypadku, gdy ograniczamy długość trwania. Należy pamiętać, że choć całkowity przyrost prędkości może być mniejszy, to musimy mieć na względzie czas misji. Zbyt długa misja może mieć wpływ na instrumenty lub załogi jeżeli analizujemy lot załogowy.

Zgodnie z równaniem Ciołkowskiego masa niezbędna do wykonania manewru zwiększyła się liniowo, gdy zwiększono masę własną satelity.

Zmniejszając impuls właściwy znacznie zwiększa się masa materiału pędnego niezbędnego do wykonania manewru.

7. Bibliografia

1) Wikipedia

2) Poliastro dokumentacja

3) Astropy dokumentacja

8. Kod źródłowy

```python

#########################################################################################################################

import numpy as np
from astropy import units as u
from astropy import time
from poliastro.bodies import Sun
from poliastro import iod

import pandas as p
import math
from poliastro.bodies import Earth, Mars, Jupiter
from poliastro.twobody import Orbit

import matplotlib.pyplot as plt
import warnings
from matplotlib.dates import DateFormatter as DF
import datetime

#########################################################################################################################

# Wyznaczenie optymalnej daty tranzytu

def optimal_transit(date, transit_min, transit_max, planet1, planet2, vs0, step):


    date_arrival = date + transit_min   # minimalna data wykonania tranzytu
    date_max = date + transit_max       # maksymalna data wykonania, zakonczenie petli
    date_arrival_final = date_arrival

    vs_temp = 0 * u.km / u.s
    dv_final = 0 * u.km / u.s
    step_first = True

    # petla idaca po datach z okreslonym krokiem
    while date_arrival < date_max:
        tof = date_arrival - date                                       # tof - time of flight
        date_iso = time.Time(str(date.iso), scale='utc')                # data startu
        date_arrival_iso = time.Time(str(date_arrival.iso), scale='utc')    # data przylotu

        r1= Orbit.from_body_ephem(planet1, date_iso)
        r2= Orbit.from_body_ephem(planet2, date_arrival_iso)
        r_1, v_1 = r1.rv()                                              # pozycja i predkosc planety poczatkowej
        r_2, v_2 = r2.rv()                                              # pozycja i predkosc planety koncowej
        (vs1, vs2), = iod.lambert(Sun.k, r_1, r_2, tof)                 # rozwiazanie zagadnienia lamberta

        dv_vector = vs1 - (vs0 + (v_1 / (24*3600) * u.day / u.s))       # zmiana predkosci niezbedna do udanego wykonania manewru
        dv = np.linalg.norm(dv_vector/10) * u.km / u.s                  # modul wektora zmiany predkosci

        if step_first:                                                    # zapis wynikow z pierwszego kroku
            dv_final = dv
            vs_temp = vs2

            step_first = False
        else:
            if dv < dv_final:                                           # sprawdzenie czy kolejny krok jest bardziej korzystna
                dv_final = dv
                date_arrival_final = date_arrival
                vs_temp = vs2

        date_arrival += step * u.day

    return dv_final, date_arrival_final, vs_temp                          # funkcja zwraca niezbedny przyrost predkosci, date przybycia

#########################################################################################################################

# przypisanie minimalnych i maksymalnych dlugosci trwania manewru ziemia-mars oraz mars-jowisz

transit_minEM = 50 * u.day
transit_maxEM = 400 * u.day
transit_minMJ = 300 * u.day
transit_maxMJ = 1000 * u.day

def optimal_date(H, date0, date1, m, Isp, step):

    delta_v = 0 * u.km / u.s
    v_out = 0 * u.km / u.s
    m_prop = 0 * u.kg
    date_in = date0
    date_out = date0

    step_one0 = True
    x1 =[]
    x2 =[]
    x3 =[]
    x4 =[]

    # petla szukajaca daty startu z najmniejszym kosztem
    while date0 < date1:
        epoch0 = date0.jyear                            # przypisanie poczatkowej daty
        ss0 = Orbit.circular(Earth, H, epoch=epoch0)    # poczatkowa kolowa orbita dookola Ziemii
        vsE = ss0.rv()[1]                               # wektor predkosci na orbicie poczatkowej

        # optymalizacja kolejnych manewrow pod wzgledem daty zakonczenia tranzytu

```

```
93
94          # optymalizacja kolejnych manewrow pod wzgledem daty zakonczenia tranzytu
95
96          dvEM, date_arrivalM, vsM = optimal_transit(date0, transit_minEM, transit_maxEM, Earth, Mars, vsE, step)
97
98          dvMJ, date_arrivalJ, vsJ = optimal_transit(date_arrivalM, transit_minMJ, transit_maxMJ, Mars, Jupiter, vsM, step)
99
100         # koszty paliwa:
101         m_pMJ = m * (math.exp((dvMJ) / Isp) - 1)
102         m_pEM = (m + m_pMJ) * (math.exp((dvEM) / Isp) - 1)
103
104         dv_total = (dvEM) + (dvMJ)                      # calkowita zmiana predkosci potrzebna do wykonania manewru
105         m_p = m_pEM + m_pMJ                             # masa paliwa potrzebna do wykonania manewrow
106
107         x1.append(date0.iso[0:10])
108         x2.append(int((date_arrivalJ - date0).jd))
109         x3.append(float(dv_total / u.km * u.s))
110         x4.append(int(m_p / u.kg))
111         #print(x1)
112
113         #x={'1': date0.iso[0:10],'2': int((date_arrivalU - date0).jd),'3': float(dv_total / u.km * u.s),'4': int(m_p / u.kg)}
114         #lista = pd.DataFrame(,index=[i])
115
116
117         # wyswietlenie wynikow dla kolejnych dni
118         print(date0.iso[0:10], ', %i days, %.3f km/s, %i kg' % (int((date_arrivalJ - date0).jd),float(dv_total / u.km * u.s),int(m_p / u.kg)))
119
120         # zapisanie wynikow z pierwszego kroku
121         if step_one0:
122             delta_v = dv_total
123             m_prop = m_p
124             v_out = vsJ
125             date_out = date_arrivalJ
126
127             step_one0 = False
128             # sprawdzenie czy nowa konfiguracja jest bardziej korzystna
129         else:
130             if dv_total < delta_v:
131                 delta_v = dv_total
132                 m_prop = m_p
133                 v_out = vsJ
134                 date_in = date0
135                 date_out = date_arrivalJ
136
137         date0 += step * u.day
138     listt = {'x1': x1,'x2': x2, 'x3': x3, 'x4': x4}
139     listt = p.DF(listt)
140
```

```
144 ##########################################################################################################################
145
146 def orbit_check(date, v, m, Isp):
147
148     # sprawdzenie czy orbita została osiągnieta
149
150     date_iso = time.Time(str(date.iso), format='iso', scale='utc')
151     r_out=Orbit.from_body_ephem(Jupiter, date_iso)      # polozenie Jowisza po asyscie
152     r_out1, vp_out1 = r_out.rv()
153     v_exit = v + (vp_out1 / (24 * 3600) * u.day / u.s)       # predkosc satelity po manewrach
154     epoch_out = date.jyear
155
156     ss_out = Orbit.from_vectors(Sun, r_out1, v_exit, epoch=epoch_out)        # wyjsciowe parametry orbity
157
158     print('Sprawdzanie osiągnięcia orbity ...')
159     print()
160
161     if ss_out.ecc >= 1:     # ekscentrycznosc orbity
162         print('Predkosc jest okej')
163     else:
164         print('Predkosc jest za mała')
165         print()
166         print('Dostosuj się do minimalnej orbity wyjściowej ')
167
168         # minimalna orbita wyjsciowa paraboliczna:
169         ss_out_new = Orbit.parabolic(Sun, ss_out.p, ss_out.inc, ss_out.raan, ss_out.argp, ss_out.nu, epoch=epoch_out)
170
171         v_out_new = ss_out_new.rv()[1] - v_exit     # obliczenia nowej predkosci
172         dv_out_new = np.linalg.norm(v_out_new) * u.km / u.s
173         m_p_new = m * (math.exp(dv_out_new / Isp) - 1)        # obliczenia brakujace masy paliwa
174
175         # Odpowiedz:
176         print('Potrzebna delta V: %.3f km/s' % float(dv_out_new / u.km * u.s))
177         print('Potrzebna dod. masa paliwa: %i kg' % int(m_p_new / u.kg))
178
179 ##########################################################################################################################
180
181 # funkcja mająca na celu wylaczenie wyswietlania warningow
182 def fxn():
183     warnings.warn("deprecated", DeprecationWarning)
184
185 with warnings.catch_warnings():
186     warnings.simplefilter("ignore")
187     fxn()
188
189 ##########################################################################################################################
190
191     # Paremetry wejsciowe
```

```
189 ##########################################################################################################
190
191     # Paremetry wejsciowe
192
193     date_0 = '2018-01-01 12:00'
194     date_1 = '2020-01-01 12:00'
195     date_0 = time.Time(date_0, scale='utc')
196     date_1 = time.Time(date_1, scale='utc')
197     date_0 = time.Time(date_0.jd, format='jd', scale='utc')
198     date_1 = time.Time(date_1.jd, format='jd', scale='utc')
199     m_ship = 500 * u.kg
200     I_sp = 500 * u.km / u.s
201     H = 300 * u.km
202
203     # czesc obliczeniowa programu
204
205     step = 10 # krok czasowy symulacji w dniach
206
207     # Asysta grawitacyjna Marsa (_M)
208
209     delta_v_M, v_out_M, date_in_M, date_out_M, m_prop_M, list_M  = optimal_date(H, date_0, date_1, m_ship, I_sp, step)
210     print (delta_v_M, v_out_M, date_in_M.iso, date_out_M.iso, m_prop_M)
211
212     figure1, axis1 = plt.subplots()
213     figure2, axis2 = plt.subplots()
214
215     date_list = list_M.get('x1')
216     dv_list = list_M.get('x3')
217     mp_list = list_M.get('x4')
218
219     # Rysowanie wykresow
220
221     # wykres potrzebnej do wykonania manewru zmiany prędkosci
222
223     axis1.grid(True)
224     axis1.plot([datetime.datetime.strptime(val, '%Y-%m-%d') for val in date_list, dv_list, color='blue')
225
226     figure1.autofmt_xdate()
227     myFmt = DF("%Y-%m-%d")
228     axis1.xaxis.set_major_formatter(myFmt)
229
230     axis1.set_title('dV (start_date)')
231     axis1.set_xlabel('start_date')
232     axis1.set_ylabel('dV [km/s]')
233     figure1.savefig("graph_dv.png")
234
235     # wykres potrzebnej do wykonania manewru zmiany masy paliwa
236
```

```
235     # wykres potrzebnej do wykonania manewru zmiany masy paliwa
236
237     axis2.grid(True)
238     axis2.plot([datetime.datetime.strptime(val, '%Y-%m-%d') for val in date_list], mp_list, color='red')
239
240     figure2.autofmt_xdate()
241     myFmt = DF("%Y-%m-%d")
242     axis2.xaxis.set_major_formatter(myFmt)
243
244     axis2.set_title('m_fuel (')
245     axis2.set_xlabel('start_date')
246     axis2.set_ylabel('mass_fuel [kg]')
247     figure2.savefig("graph_mfuel.png")
248
249
250
```