Tymon Tomczykowski
Numer albumu: 320995
Opiekun pracy: mgr inż. Mateusz Sochacki

Praca przejściowa inżynierska

Porównanie metod modelowania pola grawitacyjnego

Data oddania: 13.09.2024 Ocena: .....

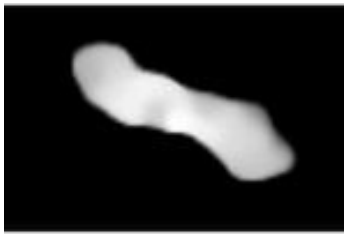
Spis treści

1 Wstęp

1.1 Metody

Modelowanie pola grawitacyjnego jest kluczowym elementem wielu aktywności podejmowanych w dziedzinie eksploracji i badania przestrzeni kosmicznej, takich jak planowanie misji kosmicznych czy prowadzenie obserwacji obiektów kosmicznych. Metody omawiane w tej pracy są stosowane najczęściej do relatywnie małych obiektów, których grawitacja nie jest na tyle silna aby ukształtować je w kształt zbliżony do sfery. W wyniku tego obiekty te mają często bardzo nieregularne kształty co sprawia, że ich pole grawitacyjne również jest nieregularne a jego modelowanie może być wyzwaniem.

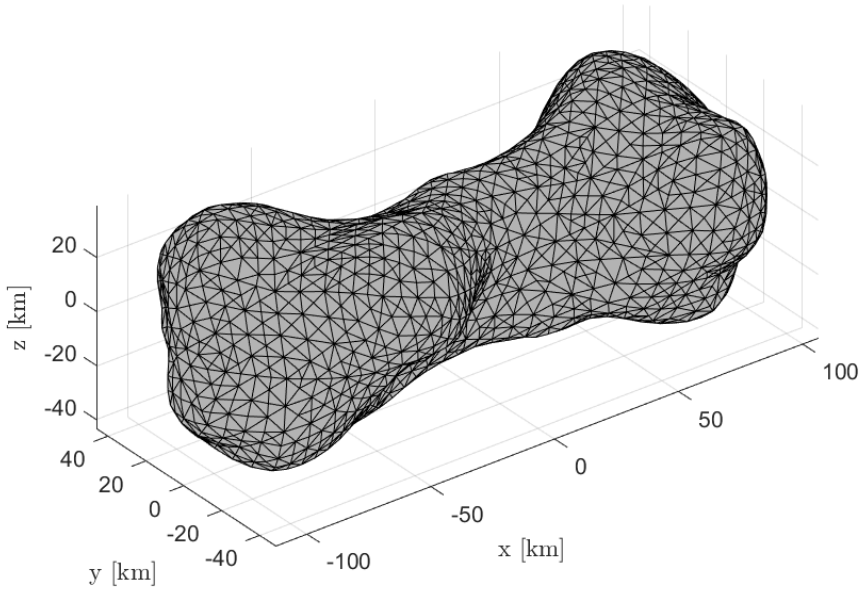
(a) 433 Eros [1]



(b) 216 Kleopatra [2]

Rysunek 1: Przykłady asteroid o nieregularnych kształtach

Modelowanie pola grawitacyjnego odbywa się w oparciu o trójkątną siatkę przybliżającą rzeczywisty obiekt uzyskaną w wyniku obserwacji astronomicznych. Sprawia to iż niemożliwe jest dokładne uzyskanie modelu pola grawitacyjnego i każda z porównywanych metod jedynie przybliża rzeczywiste pole.



Rysunek 2: Model trójkątnej siatki asteroidy 216 Kleopatra [3]

1.1.1 Metoda wielościanów (Polyhedral method)

Metoda wielościanów przedstawiona przez Robert A. Werner i Daniel J. Scheeres [4] opiera się na geometrii siatki i oferuje dokładne rozwiązenie równania potencjału przy założeniu jednorodnej gęstości obiektu. Oznacza to, że błąd w przypadku tej metody wynika jedynie z przybliżenia kształtu obiektu oraz w minimalnym stopniu z błędów obliczeniowych. Błąd metody maleje wraz z rosnącą rozdzielczością modelu siatki. Kolejną zaletą tej metody jest poprawność rozwiązania nie tylko poza obszarem obiektu ale również wewnątrz oraz na jego powierzchni. Sprawia to iż metoda ta jest bardzo użyteczna przy wyznaczaniu sił grawitacji działających na powierzchnię obiektu oraz przy badaniu dynamiki obiektów wystrzelonych z lub na powierzchnię obiektu. Wadą tej metody jest duży koszt obliczeniowy, który rośnie wraz z rozdzielczością modelu.

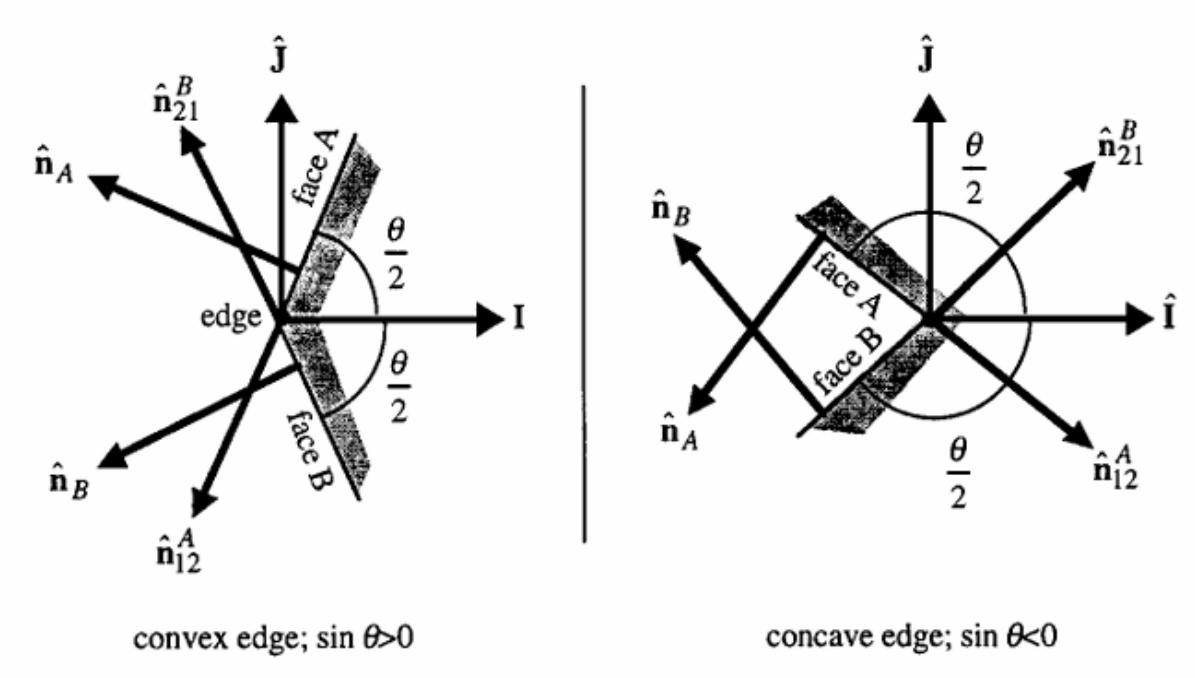
Ogólny wzór metody wygląda następująco:

$$U = \frac{1}{2} G\sigma \sum_{e \in \text{ edges}} \mathbf{r}_e \bullet \mathbf{E}_e \cdot \mathbf{r}_e \cdot L_e - \frac{1}{2} G\sigma \sum_{f \in \text{ faces}} \mathbf{r}_f \bullet \mathbf{F}_f \bullet \mathbf{r}_f \cdot \omega_f \tag{1}$$

Gdzie:

- U - potencjał grawitacyjny,
- G - stała grawitacji,
- σ - gęstość obiektu,
- e - kolejne krawędzie modelu,
- $\mathbf{r}_e$ - wektor poprowadzony z punktu obliczeniowego do dowolnego punktu na krawędzi,
- $\mathbf{E}_e = \mathbf{E}_{12} = \hat{\mathbf{n}}_A \hat{\mathbf{n}}_{12}^A + \hat{\mathbf{n}}_B \hat{\mathbf{n}}_{21}^B$ - iloczyn diadyczny, interpretacja wektorów użytych we wzorze pokazano na rysunku 3
- $L_e^f = \int_e \frac{1}{r}\, \mathrm{d}s = \ln \frac{a+b+e}{a+b-e}$ - gdzie a i b to odległości od punktu obliczeniowego do dwóch końców krawędzi a e to długość krawędzi ścianki,
- f - kolejne ścianki modelu,
- $\mathbf{r}_f$ - wektor poprowadzony z punktu obliczeniowego do dowolnego punktu leżącego w płaszczyźnie ścianki,
- $\mathbf{F}_f \equiv \hat{\mathbf{n}}_f \hat{\mathbf{n}}_f$ - iloczyn diadyczny jednostkowych wektorów normalnych każdej ścianki skierowanych od środka modelu,
- $\omega_f = 2 \operatorname{arc\,tg} \frac{\hat{\mathbf{r}}_1 \bullet \hat{\mathbf{r}}_2 \times \hat{\mathbf{r}}_3}{1 + \hat{\mathbf{r}}_1 \bullet \hat{\mathbf{r}}_2 + \hat{\mathbf{r}}_2 \bullet \hat{\mathbf{r}}_3 + \hat{\mathbf{r}}_3 \bullet \hat{\mathbf{r}}_1}$ - wielkość, która może być interpretowana jako pole powierzchni rzutu danej ścianki na jednostkową sferę o środku w punkcie obliczeniowym. Wektory znajdujące się we wzorze to wektory poprowadzone od punktu obliczeniowego do trzech kolejnych narożników ścianki.

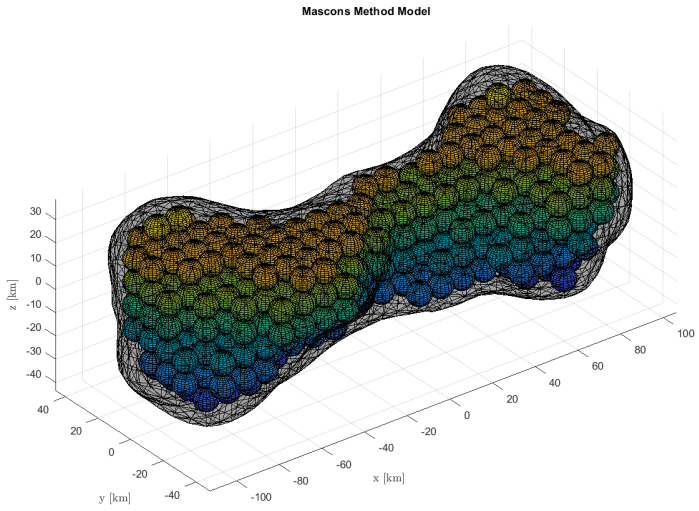
Szczegółowe wyprowadzenie powyższego wzoru wraz ze wskazówkami do implementacji można znaleźć w cytowanym wcześniej artykule autorstwa Robert A. Werner i Daniel J. Scheeres [4].



Rysunek 3: Interpretacja wektorów potrzebnych do obliczenia $\mathbf{E}_e$

1.1.2 Metoda punktów masy (Point masses method)

Metoda punktów masy polega na przybliżeniu oddziaływania pola grawitacyjnego obiektu poprzez wypełnienie go dużą ilością punktów oraz przypisaniu im odpowiedniej masy tak aby sumaryczna masa punktów odpowiadała masie obiektu. Wynikiem jest model sumerycznego pola grawitacyjnego każdego z tych punktów. Zaletą tej metody jest jej prostota. Przydaje się przy początkowym przybliżaniu pola grawitacyjnego. Natomiast przy dużej ilości punktów koszt obliczeniowy może znacznie wzrosnąć i może się okazać, że inna metoda zagwarantuje większą dokładność jednocześnie niższym nakładem obliczeniowym.



Rysunek 4: Przykład podziału obiektu na punktowe masy

Wzór na potencjał grawitacyjny uzyskany z punktowych mas wraz z wyjaśnieniem symboli przedstawiono poniżej:

$$U = \sum_{i=1}^{N} \frac{\mathrm{G}M_i}{|\boldsymbol{r} - \boldsymbol{r}_i|} \tag{2}$$

Gdzie:

- U - potencjał grawitacyjny,
- G - stała grawitacji,
- N - liczba punktów podziału,
- i - indeks symbolizujący kolejne punktowe masy,
- M_i - masa pojedyńczego punktu dobrana tak, że zachodzi zależność $\sum_{i=1}^{N} M_i = M_\mathrm{A}$ gdzie M_A to masa całego obiektu,
- $\boldsymbol{r}$ - wektor poprowadzony do punktu obliczeniowego z początku układu współrzędnych,
- $\boldsymbol{r}_i$ - wektor poprowadzony do kolejnego punktu podziału z początku układu współrzędnych.

Podział na punktu odbywał się w następujący sposób:

1. Znaleziono prostopadłościan opisany na modelu obiektu,
2. Prostopadłościan podzielono na punkty odległe od siebie o zadaną wartość $2a$,
3. Odrzucono punkty, które znajdują się poza modelem obiektu lub które znajdują się w odległości mniejszej niż a od powierzchni modelu.

1.1.3 Metoda harmonik sferycznych (Spherical harmonics method)

Metoda harmonik sferycznych polega na rozwinięciu równania potencjału grawitacyjnego przy użyciu harmonik sferycznych. Metoda ta ma bogatą historię i jest powszechnie używana do modelowania potencjału (nie tylko grawitacyjnego). Głównym celem tej metody jest wyznaczenie współczynników, przy pomocy których można następnie małym nakładem obliczeniowym odtworzyć model potencjału grawitacyjnego. Istnieje wiele sposobów wyznaczania tych współczynników. W tej pracy wykorzystano sposób przedstawiony przez Roberta A. Werner [5]. Ogromną zaletą tej metody jest fakt iż główny koszt obliczeniowy dotyczy jedynie wyznaczenia współczynników. Na podstawie wyznaczonych wcześniej współczynników można następnie szybko i przy użyciu niskiej mocy obliczeniowej uzyskać wartość potencjału co sprawia iż metoda ta jest niezwykle użyteczna podczas misji kosmicznych gdy moc obliczeniowa komputera pokładowego satelity lub statku kosmicznego jest szczególnie cenna. Wadą tej metody jest brak możliwości łatwego wyznaczenia pola grawitacyjnego na powierzchni oraz wewnątrz obiektu. W tym celu należałoby prawdopodobnie korzystać z innych równań, których jednak nie udało się znaleźć w dostępnej literaturze zatem porównywana w tej pracy metoda harmonik sferycznych dotyczy jedynie punktów poza obszarem obiektu.

Poniżej podano wzór na potencjał grawitacyjny z użyciem tej metody oraz krótkie wyjaśnienie dotyczące obliczania współczynników.

$$U = \frac{GM}{r}\left\{1 + \sum_{n=1}^{\infty}\left(\frac{a}{r}\right)^n \sum_{m=0}^{n} P_{n,m}(\sin\phi)\left[C_{n,m} \times \cos(m\lambda) + S_{n,m}\sin(m\lambda)\right]\right\} \tag{3}$$

Gdzie:

- U - potencjał grawitacyjny,
- G - stała grawitacji,
- M - masa obiektu,
- r - odległość punktu obliczeniowego od początku układu współrzędnych,
- n - stopień wielomianu (dla $m = 0$) lub funkcji (dla $m > 0$) Legendre'a,
- m - rząd funckji Legendre'a,
- a - promień sfery odniesienia czyli sfery opisanej na modelu,
- $P_{n,m}$ - wielomian (dla $m = 0$) lub funkcja (dla $m > 0$) Legendre'a obliczana przy pomocy funkcji ze środowiska *Matlab*,
- ϕ oraz λ - współrzędne kątowe punktu obliczeniowego,
- $C_{n,m}$ oraz $S_{n,m}$ - znormalizowane współczynniki harmoniczne.

Współczynniki harmoniczne obliczono przy pomocy następujących wzorów rekurencyjnych: Współczynniki diagonalne:

$$n = 0 : \begin{bmatrix} \bar{c}_{0,0} \\ \bar{s}_{0,0} \end{bmatrix} = \frac{1}{M}\begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{4}$$

$$n = 1 : \begin{bmatrix} \bar{c}_{1,1} \\ \bar{s}_{1,1} \end{bmatrix} = \frac{1}{\sqrt{3}M}\begin{bmatrix} x'/a \\ y'/a \end{bmatrix} \tag{5}$$

$$n > 1 : \begin{bmatrix} \bar{c}_{n,n} \\ \bar{s}_{n,n} \end{bmatrix} = \frac{2n-1}{\sqrt{2n(2n+1)}} \times \begin{bmatrix} x'/a & -y'/a \\ y'/a & x'/a \end{bmatrix}\begin{bmatrix} \bar{c}_{n-1,n-1} \\ \bar{s}_{n-1,n-1} \end{bmatrix} \tag{6}$$

Współczynniki wertykalne ($m < n$ oraz $n > 1$):

$$\begin{aligned}\begin{bmatrix} \bar{c}_{n,m} \\ \bar{s}_{n,m} \end{bmatrix} &= (2n-1)\sqrt{\frac{(2n-1)}{(2n+1)(n+m)(n-m)}}\frac{z'}{a}\begin{bmatrix} \bar{c}_{n-1,m} \\ \bar{s}_{n-1,m} \end{bmatrix} \\ &-\sqrt{\frac{(2n-3)(n+m-1)(n-m-1)}{(2n+1)(n+m)(n-m)}}\left(\frac{r'}{a}\right)^2 \times \begin{bmatrix} \bar{c}_{n-2,m} \\ \bar{s}_{n-2,m} \end{bmatrix}\end{aligned} \tag{7}$$

Współczynniki subdiagonalne ($m = n - 1$):

$$\begin{bmatrix} \bar{c}_{n,n-1} \\ \bar{s}_{n,n-1} \end{bmatrix} = \frac{2n-1}{\sqrt{2n+1}}\frac{z'}{a}\begin{bmatrix} \bar{c}_{n-1,n-1} \\ \bar{s}_{n-1,n-1} \end{bmatrix} \tag{8}$$

Gdzie x', y' i $'$ to współrzędne narożników kolejnych ścianek, których mnożenie z innymi współczynnikami odbywa się w sposób szczególny wyjaśniony w pracy autorstwa Roberta A. Werner [5]. Działanie to wiąże się z przedstawieniem współrzędnych jako specyficzne wielomiany trzeciego stopnia zwane z angielskiego *trinomials*. Otrzymane w ten sposób współczynniki należy jeszcze zmodyfikować w następujący sposób:

$$\begin{bmatrix} \bar{C}_{n,m} \\ \bar{S}_{n,m} \end{bmatrix} = \sigma \sum_{\text{simplices}} \left(\frac{\det J}{(n+3)!} \sum_{i+j+k=n} i!j!k! \begin{bmatrix} \bar{\alpha}_{i,j,k} \\ \bar{\beta}_{i,j,k} \end{bmatrix} \right) \tag{9}$$

Gdzie $\bar{\alpha}_{i,j,k}$ oraz $\bar{\beta}_{i,j,k}$ są odpowiednikami współczynników $\bar{c}_{n,m}$ oraz $\bar{s}_{n,m}$ zapisanymi znowu w postaci pewnych wielomianów trzeciego stopnia. Ostatecznie otrzymane współczynniki należy jeszcze znormalizować przy użyciu innego współczynnika:

$$N_{n,m} = \sqrt{(2-\delta_{0,m})\,(2n+1)(n-m)!/(n+m)!} \tag{10}$$

Gdzie $\delta_{0,m}$ to delta Kroneckera. Szczegółowe wyprowadzenie powyższych wzorów, interpretacje zmiennych oraz wskazówki implementacji można znaleźć w cytowanej wcześniej pracy [5].

2 Cel i zakres pracy

Celem pracy jest ogólne porównanie trzech powszechnych metod modelowania pola grawitacyjnego, przedstawienie ich zalet oraz wad.

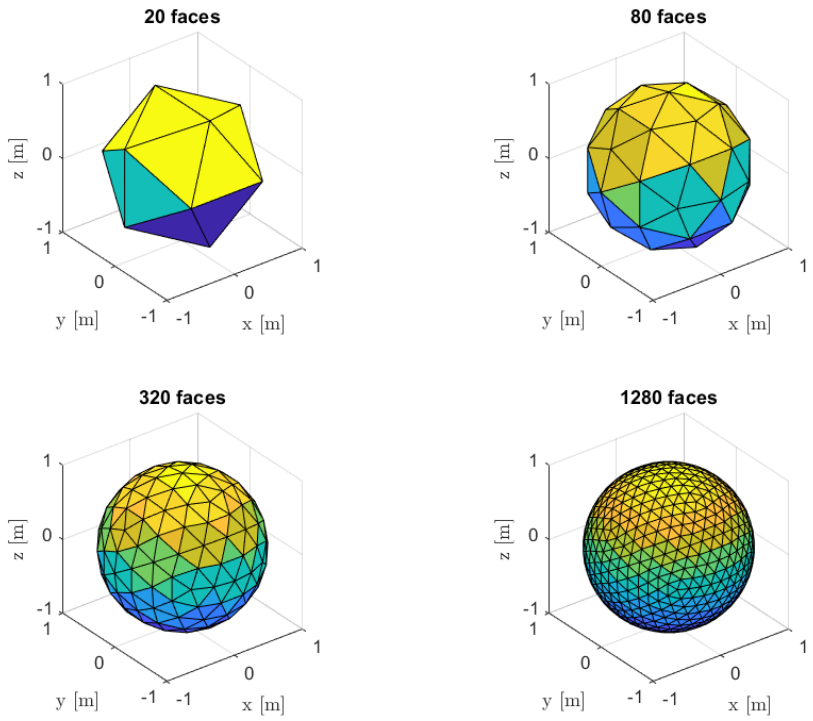
Wybrano trzy metody, które zostaną porównane zgodnie z metodyką opisaną poniżej. Przedstawione zostaną wyniki i opisane będą wnioski z nich wynikające. W celu porównania metody te zostały zaimplementowane z wykorzystaniem środowiska *Matlab*.

W pierwszej kolejności metody zostaną porównane z analitycznym rozwiązaniem równania potencjału dla sfery. W tym celu przygotowane zostaną cztery modele siatki przybliżające kształt sfery o różnej rozdzielczości przedstawione na rysunku 5. Dla każdej metody obliczony zostanie błąd względny wyznaczania potencjału oraz czas obliczeniowy. Błąd względny obliczony zostanie poprzez wyznaczenie wartości potencjału dla stu losowo wygenerowanych punktów i rozpatrzone zostaną dwa przypadki: gdy punkty znajdują się wewnątrz sfery oraz poza nią. Dla metody harmonik sferycznych zbadany zostanie jedynie przypadek gdy punkty znajdują się poza sferą gdyż metoda ta nie obejmuje innej możliwości. Należy również zaznaczyć, że czas obliczoniowy dla tej metody dotyczy wyznaczania wartości potencjału na podstawie współczynników a nie obliczania tych współczynników.

Dla metody wielościanów badany gędzie wpływ rozdzielczości modelu siatki na dokładność wyników oraz czas obliczeniowy.

W przypadku metody punktów masy badany będzie natomiast wpływ ilości punktów podziału modelu.

Dla metody harmonik sferycznych rozpatrzone będą dodatkowe dwa przypadki: wpływ rozdzielczości modelu siatki oraz wpływ stopnia rozwinięcia równania potencjału.



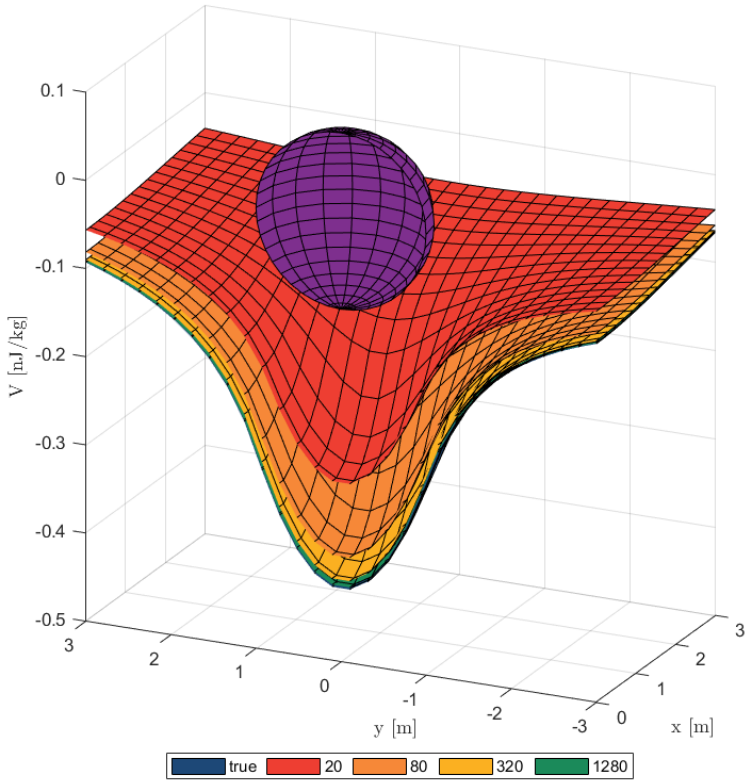
Rysunek 5: Modele siatki przybliżające kształt sfery o różnej rozdzielczości

Ze względu na matematyczny charakter metody harmonik sferycznych przypadek sfery jest wyjątkowo korzystny. To samo dotyczy metod punktów masy gdzie punkty w przypadku sfery rozkładane są symetrycznie, co skutkuje bardzo wysoką dokładnością dla przypadku punktów obliczeniowych poza aferą. Z tego względu przeprowadzone będzie dodatkowego porównania wszystkich trzech metod na przykładzie modelu asteroidy 216 Kleopatra. W tym przypadku zaniedbany zostanie błąd wynikający z przybliżenia kształtu rzeczywistego obiektu (model traktowany będzie jako dokładny na potrzeby porównania) co sprawi, że wynik uzyskany metodą wielościanów będzie mógł być uznany jako dokładny. Podejście takie jest stosowane w wielu pracach ([4], [6–8]).

3 Analiza metod

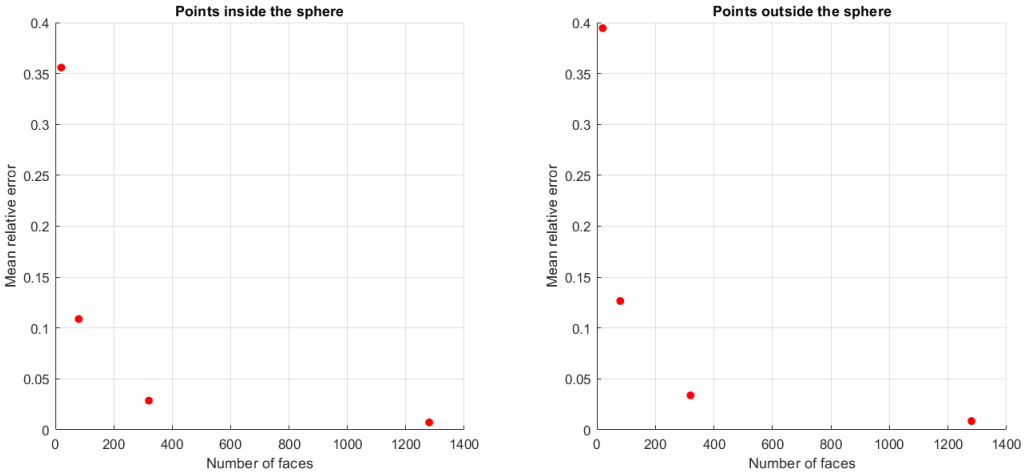
3.1 Polyhedral method

W metodzie wielościanów został przeanalizowany wpływ rozdzielczości modelu siatki na dokładność wyników oraz czas obliczeniowy. W pierwszej kolejności pokazano studnię potencjału dla czterech rozdzielczości siatki w porównaniu do wyniku dokładnego. Porównanie to przedstawiono na rysunku 6. Jak widać większa rozdzielczość siatki skutkuje lepiej dopasowanym kształtem studni.

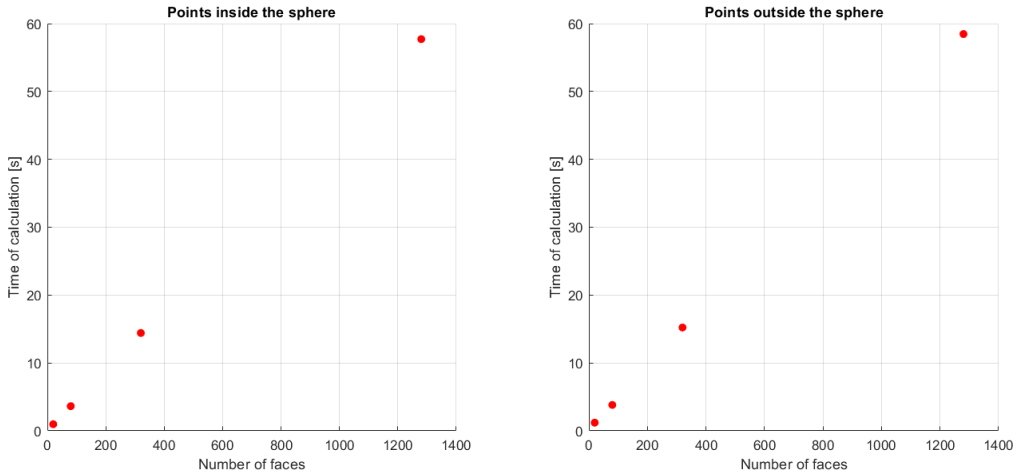


Rysunek 6: Porównanie studni potencjału dla różnej dokładności modelu

Kolejno dla 100 losowo wygenerowanych punktów obliczono średni błąd względny potencjału grawitacyjnego oraz sprawdzono czas obliczeń. Wyniki przedstawiono na rysunkach 7 oraz 8. Rozpatrzono również dwa przypadki: punkty obliczeniowe znajdujące się wewnątrz sfery oraz poza nią. Jak można zauważyć wykresy dla tych dwóch przypadków są bardzo zbliżone co potwierdza poprawność działania tej metody dla obu przypadków. Wartości na wykresach przedstawiających błąd względny maleją nieliniowo wraz ze wzrostem rozdzielczości. Na początku spadek ten jest duży aczkolwiek od pewnego momentu uzyskano niewielką poprawę dokładności przy dalszym zwiększaniu rozdzielczości siatki. Czas obliczeniowy rośnie natomiast w przybliżeniu liniowo.



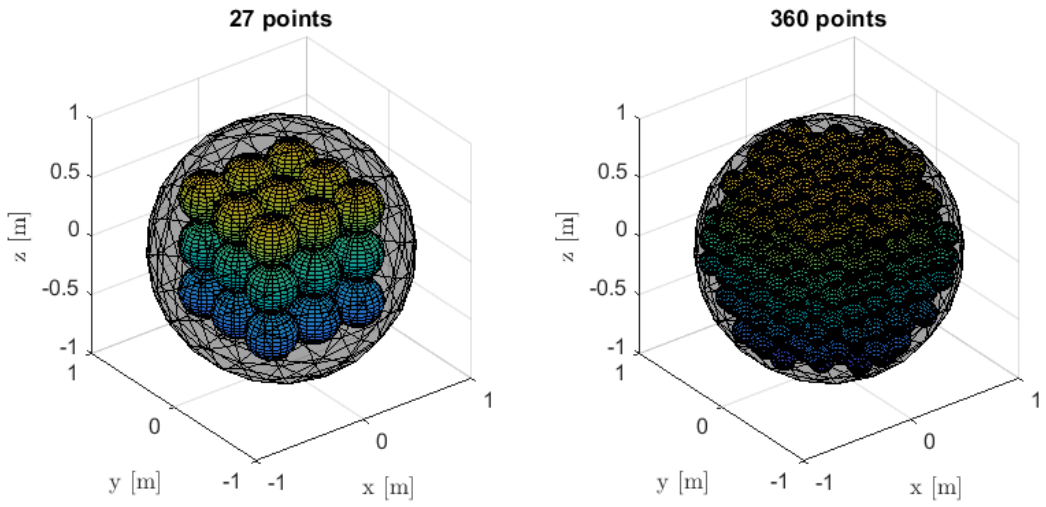
Rysunek 7: Wykresy zależności błędu względnego od liczby ścian modelu



Rysunek 8: Wykresy zależności czasu obliczeniowego od liczby ścian modelu

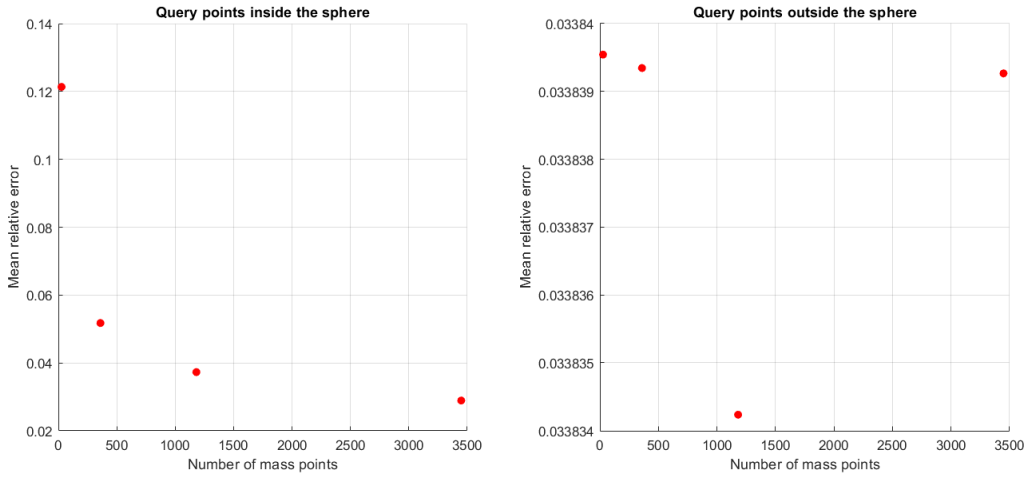
3.2 Point masses method

W metodzie punktów masy przeanalizowany został wpływ ilości punktów masy na dokładność oraz czas obliczeń. Sferę podzielono na kolejno 27, 360, 1181 oraz 3448 punktów. Wizualizację podziału dla pierwszych dwóch ilości punktów przedstawiono na rysunku 9.

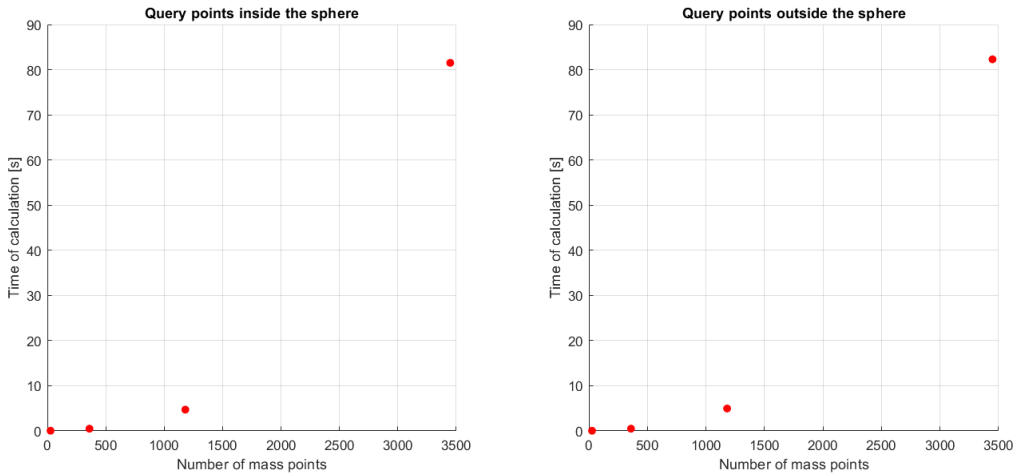


Rysunek 9: Wizualizacje podziału sfery na punkty

Na rysunku 10 przedstawiono wyniki dotyczące dokładności obliczeń, a na rysunku 11 wyniki dotyczące czasu obliczeniowego. Dla przypadku punktów obliczeniowych wewnątrz sfery można zaobserwować nieliniowy wzrost dokładności wraz ze wzrostem ilości punktów. Natomiast dla przypadku punktów obliczeniowych poza sferą błąd względny jest niemal zaniedbywalny. Wiąże się to z symetrycznym rozkładem punktów masy wewnątrz sfery co matematycznie sprowadza się do idealnego rozwiązania. Przypadek ten należy zatem traktować jako nieważny. Czas obliczeniowy dla obu przypadków nieliniowo wzrasta wraz z liczbą punktów masy.



Rysunek 10: Wykresy zależności błędów wzlędnych od liczby punktów podziału



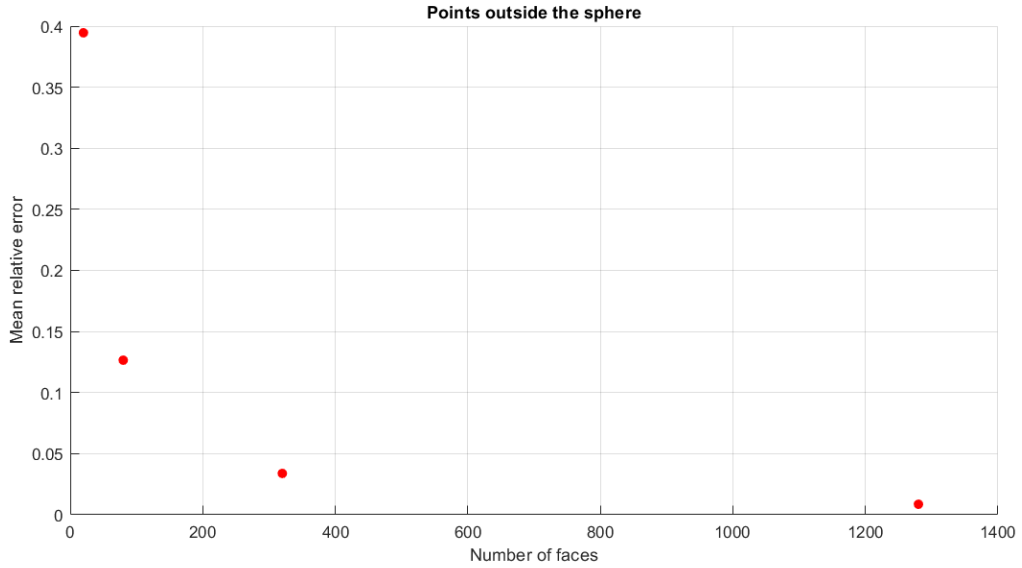
Rysunek 11: Wykresy zależności czasu obliczeniowego od liczby punktów podziału

3.3 Spherical Harmonics

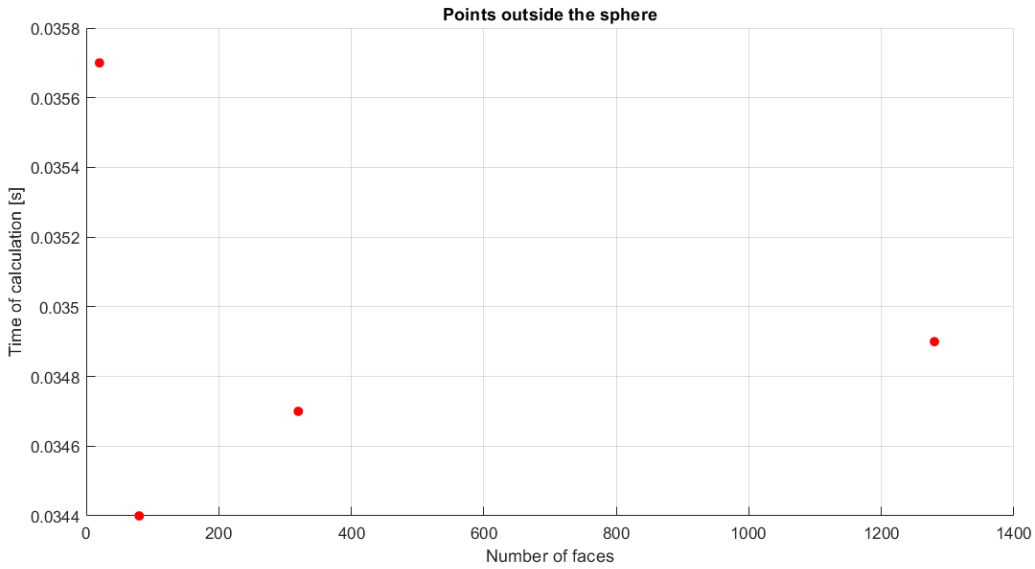
W metodzie harmonik sferycznych został zbadany wpływ rozdzielczości modelu siatki oraz stopnia rozwinięcia równania potencjału na dokładność i czas obliczeń.

3.3.1 Wpływ rozdzielczości modelu siatki

Wraz ze wzrostem rozdzielczości modelu, zgodnie z przewidywaniami, otrzymano wzrost dokładnosci rozwiązania co widać na rysunku 12. Czas obliczeniowy, z wyłączeniem pierwszego punktu, również wzrasta wraz z rozdzielczością co pokazane jest na rysunku 13. Są to natomiast na tyle małe różnice, że niekoniecznie należy sugerować się tym przypadkiem.



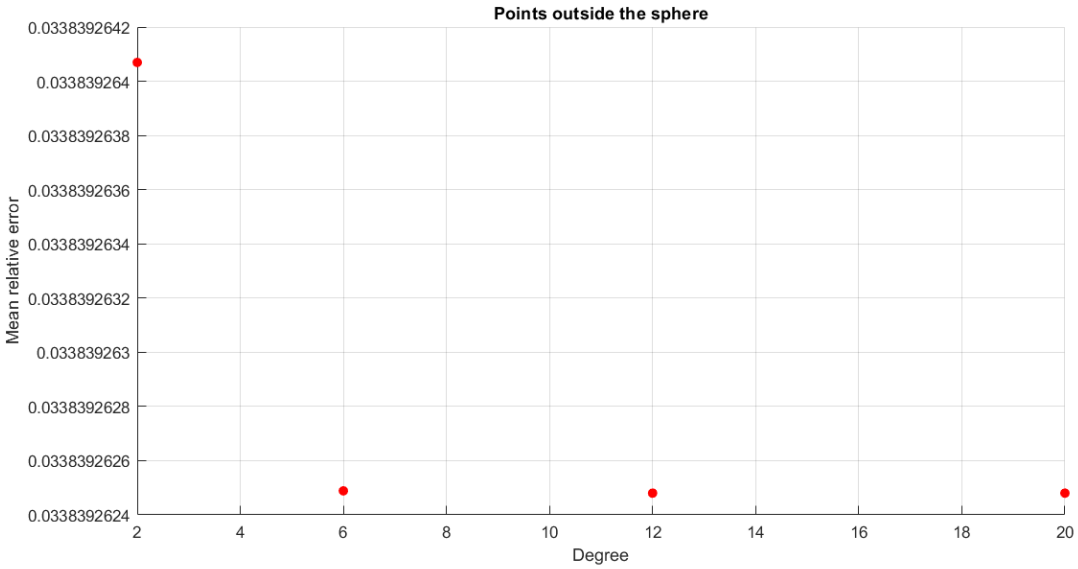
Rysunek 12: Wykres zależności błędu względnego od liczby ścian modelu



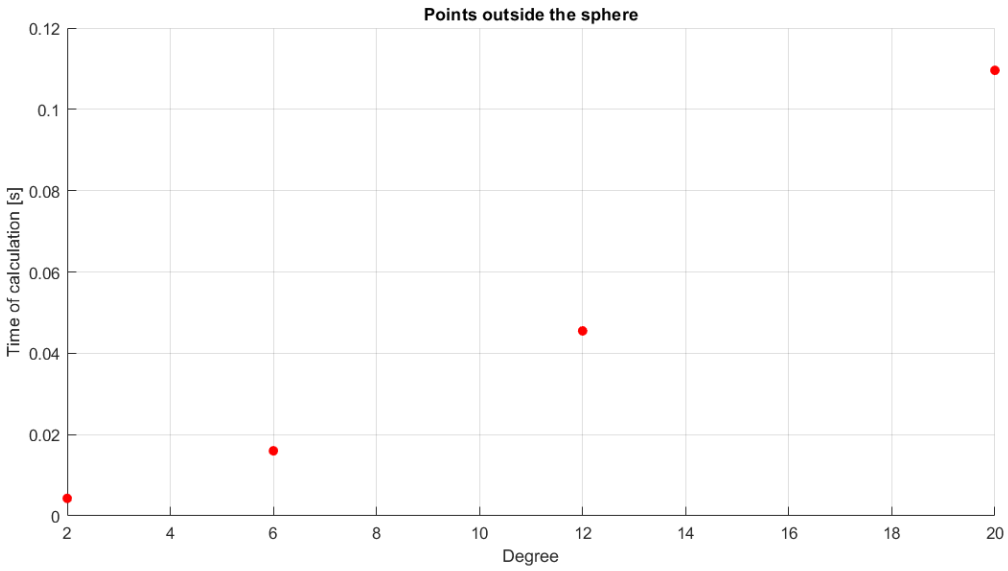
Rysunek 13: Wykres zależności czasu obliczeniowego od liczby ścian modelu

3.3.2 Wpływ stopnia rozwinięcia równania potencjału

Wpływ stopnia rozwinięcia równania potencjału badano dla modelu posiadającego 320 ścian. Jak widać na rysunku 14, już dla drugiego stopnia otrzymano bardzo wysoką dokładność a następnie dalsze zwiększanie wartości stopnia rozwinięcia nie skutkowało w poprawie wyników. Wiąże się to prawdopodobnie z przyjęciem sfery jako model testowy co idealnie wpisuje się w charakter matematyczny metody i już pierwszy stopień rozwinięcia opisywałby bardzo dobrze model pola grawitacyjnego. Błąd osiągnięty dla 6, 12 oraz 20 stopnia prawdopodobnie wynika w dużym stopniu jedynie z modelu siatki. Czas obliczeniowy natomiast, nieliniowo rośnie wraz ze zwiększaniem stopnia rozwinięcia co widać na rysunku 15.



Rysunek 14: Wykres zależności błędu względnego od stopnia rozwinięcia równania potencjału

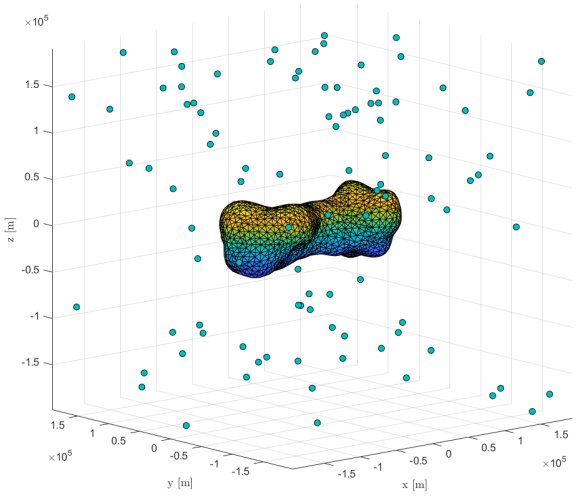


Rysunek 15: Wykres zależności czasu obliczeniowego od stopnia rozwinięcia równania potencjału

4 Analiza na podstawie modelu (216) Kleopatra

Ze względu na osobliwości opisane w **??**, 3.2 oraz 3.3.2, rozpatrzono dodatkowy przypadek na przykładzie asteroidy (216) Kleopatra [9]. Zdecydowano się na ten obiekt z powodu wyjątkowo nieregularnego kształtu oraz dużej dostępności danych. Użyty model siatki asteroidy pochodzi z [3]. Gęstość asteroidy założono jako stałą w całej objętości i równą $\rho = 3600\frac{kg}{m^3}$ [10].

Model asteroidy, na potrzeby analizy metod, został uznany jako dokładny co sprawia, że wyniki uzyskane metodą wielościanów również można uznać za dokładne gdyż błędy w tej metodzie wynikają głównie z niedokładności modelu. W ten sposób metody punktów masy oraz harmonik sferycznych mogą być porównane w odniesieniu do metody wielościanów. Podejście to jest powszechne w literaturze ([4], [6–8]).



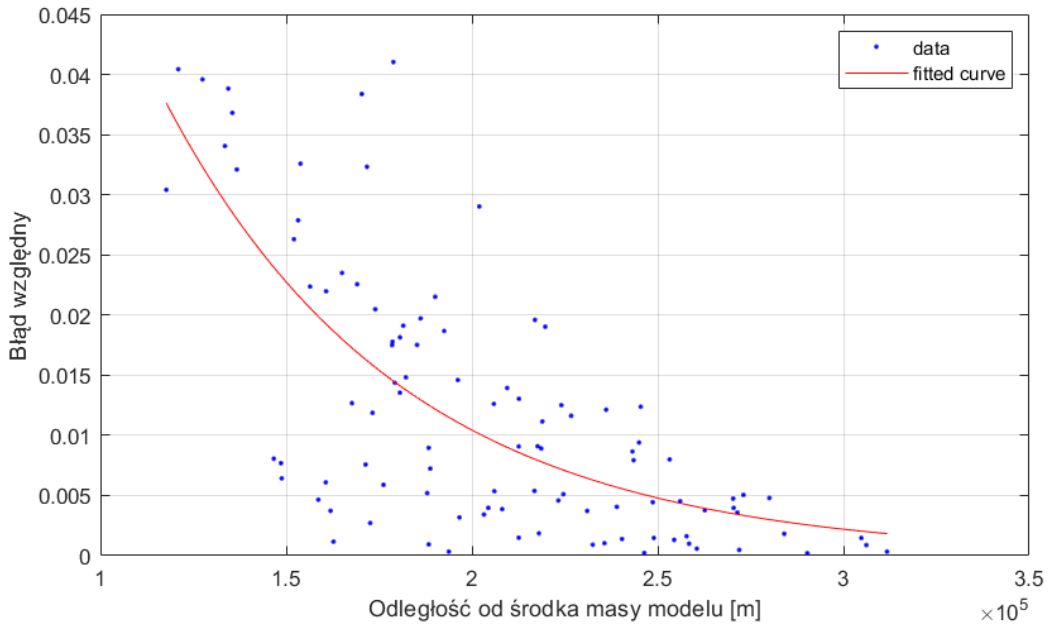
Rysunek 16: Wizualizacja wygenerowanych punktów

W celu analizy metod, zostało losowo wygenerowane 100 punktów znajdujących się w zakresie od 114 km do 312 km od środka masy modelu asteroidy. Wizualizacja wygenerowanych punktów przedstawiona jest na rysunku 16. Dolna granica jest jednocześnie promieniem odniesienia dla metody harmonik sferycznych czyli sferą opisaną na modelu asteroidy. Metoda harmonik sferycznych dla punktów znajdujących się poniżej tej granicy może generować bardzo wysokie błędy, które zaburzyłyby ogólny wynik. Dla wygenerowanych w ten sposób punktów obliczeno potencjał grawitacyjny przy użyciu metod punktów masy oraz harmonik sferycznych, a następnie policzono błędy względne względem metody wielościanów oraz zmierzono czas obliczeniowy. W metodzie punktów masy analizowano trzy przypadki: 13, 86 oraz 1018 punktów podziału modelu. W metodzie harmonik sferycznych analizowano trzy stopnie rozwinięcia równania potencjału: 4, 10 oraz 50 stopnia. Wyniki zostały przedstawione w tabeli 1.

Tabela 1: Wyniki analizy na przykładzie (216) Kleopatra

Metoda	Wielościanów	Punktów masy			Harmonik sferycznych		
Przypadek	-	13 punktów	86 punktów	1018 punktów	4 stopnia	10 stopnia	50 stopnia
Czas obliczeniowy [s]	179.16	0.013	0.037	0.394	0.028	0.041	0.699
Błąd względny [%]	-	1.27	0.6	0.13	1.23	1.19	1.19

Dodatkowo na rysunku 17 przedstawiono wykres zależności błędu względnego od odległości punktów obliczeniowych od środka masy modelu dla metody harmonik sferycznych 50 stopnia.

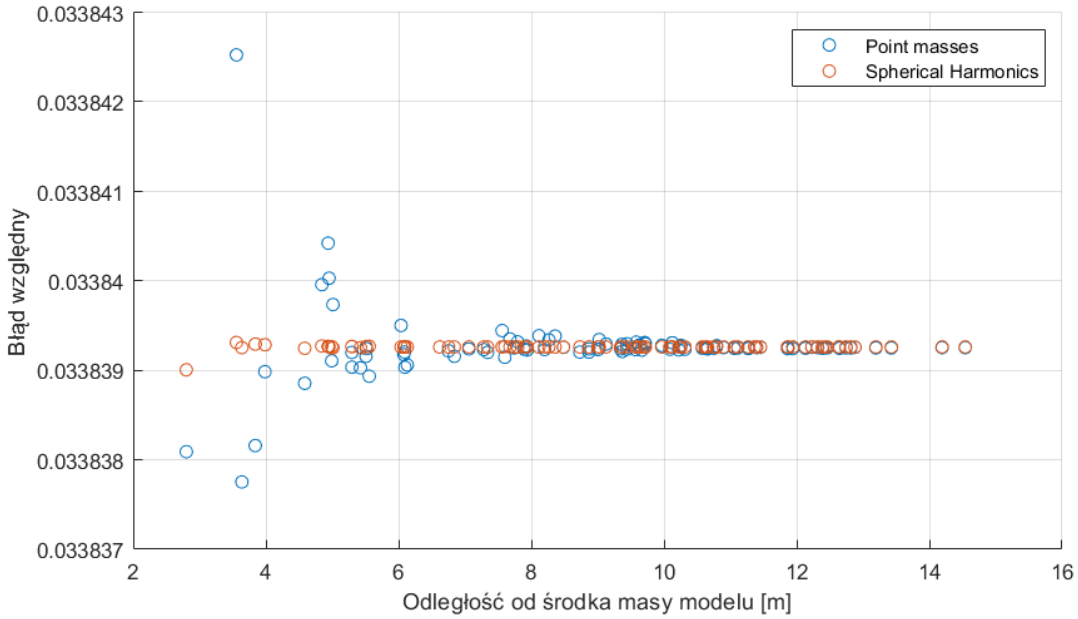


Rysunek 17: Wykres błędu względnego dla metody harmonik sferycznych 50 stopnia

5 Podsumowanie wyników

Główną zaletą metody wielościanów jest fakt, iż niedokładność tej metody zależy głównie od dokładności modelu. W przypadku gdy dysponuje się modelem o wysokiej dokładności może ona się okazać najtrafniejszą do pewnych zastosowań. Może także służyć jako odniesienie przy porównywaniu innych metod. Dużą zaletą jest również brak ograniczeń jeśli chodzi punkty obliczeniowe oraz kształt modelu czego nie można powiedzieć o na przykład metodzie harmonik sferycznych. Wadą natomiast jest czas obliczeniowy potrzebny do uzyskania danej dokładności wyników, który jest znacznie większy niż w przypadku pozostałych dwóch metod. Ten fakt mocno ogranicza zastosowanie tej metody.

Metoda punktów masy cechuje się wyjątkową prostą. Sprawdziła się lepiej niż metoda harmonik sferycznych w przypadku na przykładzie asteroidy (216) Kleopatra. Nie posiada również ograniczeń jeśli chodzi o położenie punktów obliczeniowych a potencjalny kulisty kształt modelu może działać na korzyść w przypadku punktów obliczeniowych oddalonych od powierzchni modelu. Potencjalne wady mogą pojawić się w przypadku punktów obliczeniowych zlokalizowanych blisko powierzchni modelu. Przejaw tego widać na dodatkowym rysunku 18. Wyniki przedstawione na tym wykresie zostały obliczone dla modelu sfery posiadającego 320 ścian. Liczba punktów podziału dla metody punktów masy wynosi 3448 natomiast stopień metody harmonik sferycznych 20. Jak widać w bliższym otoczeniu sfery błąd metody punktów masy jest bardziej rozstrzelony. Obliczenia dla metody punktów masy zajęły ponad 80 sekund gdzie dla metody harmonik sferycznych było to mniej niż 0.12 sekundy.



Rysunek 18: Porównanie zachowania metody punktów masy i harmonik sferycznych dla punktów w pobliżu powierzchni

Głównymi zaletami metody harmonik sferycznych są szybki czas obliczeniowy, wygodny format przechowywania wartości współczynników oraz rekurencyjny charakter metody co sprawia, że jest ona tak popularna. Ograniczeniem jest potencjalny brak możliwości modelowania pola

grawitacyjnego wewnątrz modelu lub konieczność użycia innych zależności matematycznych. Wadą są również potencjalne wysokie wartości błędów wewnątrz sfery odniesienia dla obiektów o kształcie bardzo odbiegającym od sfery o czym mowa w rozdziale 4. Natomiast podczas analizy na przykładzie sfery metoda ta sprawdziła się lepiej w pobliżu powierzchni niż metoda punktów masy.

6 Kod programu

6.1 Klasy

```
classdef polyhedron_obj < handle
properties
    stl_model
    V {mustBeNumeric}
    r {mustBeNumeric}
end

methods
    function obj = polyhedron_obj(model)
        arguments
            model triangulation
        end

        obj.stl_model = model;
    end

    function V = calculate(obj,r,bulk_density, opt1)
        arguments
            obj polyhedron_obj
            r double
            bulk_density double
            opt1.Test_plot char
        end
        if ~isfield(opt1,"Test_plot")
            opt1.Test_plot = 'off';
        end


        set_of_edges = obj.stl_model.edges;
        set_of_faces = obj.stl_model.ConnectivityList;
        set_of_n_f = faceNormal(obj.stl_model);
        G = 6.6743e-11;

        sum_e = zeros(length(r(:,1)),1);
        sum_f = zeros(length(r(:,1)),1);

        w_f_check = 0;

        for i = 1:length(r(:,1))
            rq = r(i,:);
            for k = 1:length(set_of_edges(:,1))
                edge = set_of_edges(k,:);
```

```
                p_a = obj.stl_model.Points(edge(1),:);
                p_b = obj.stl_model.Points(edge(2),:);
                E_e = get_E_e(obj.stl_model,edge,opt1.Test_plot
                    );
                L_e = get_L_e(rq,p_a,p_b);
                r_e = get_r_ef(rq,p_a);

                sum_e(i,1) = sum_e(i,1) + (L_e * r_e * E_e *
                    r_e');

            end

            for k = 1:length(set_of_faces(:,1))
                face = set_of_faces(k,:);
                face_points = obj.stl_model.Points(face',:);
                n_f = set_of_n_f(k,:);
                F_f = get_F_f(n_f);
                w_f = get_w_f(rq,face_points);
                w_f_check = w_f_check + w_f;
                r_f = get_r_ef(rq,face_points(1,:));
                sum_f(i,1) = sum_f(i,1) + (w_f * r_f * F_f *
                    r_f');
            end
            disp([num2str(i) '/' num2str(length(r(:,1)))]);
        end

        V = -0.5 * G * bulk_density * (sum_e - sum_f);
        obj.V = V;
        obj.r = r;
    end



    function axes = plot_V(obj, r, scale, opt2)
        arguments
            obj polyhedron_obj
            r double
            scale double
            opt2.Units char
        end
        if ~isfield(opt2,"Units")
            opt2.Units = 'm';
        end

        unit_scale = 1;

        model_temp = obj.stl_model;
```

```
87              if strcmp(opt2.Units,'km')
88                  unit_scale = 1/1000;
89                  model_temp = triangulation(model_temp.
                        ConnectivityList,model_temp.Points * unit_scale)
                        ;
90              end
91
92              figure();
93              % trisurf(model_temp,'FaceColor','black','FaceAlpha
                    ',0.2);
94              % hold on;
95
96              if strcmp(opt2.Units,'km')
97                  unit_scale = 1/1000;
98                  x_surf = r(:,1)*unit_scale;
99                  y_surf = r(:,2)*unit_scale;
100                 z_surf = obj.V*scale*unit_scale;
101             elseif strcmp(opt2.Units,'m')
102                 x_surf = r(:,1)*unit_scale;
103                 y_surf = r(:,2)*unit_scale;
104                 z_surf = obj.V*scale*unit_scale;
105             end
106
107
108             % xv = linspace(min(x_surf), max(x_surf), 50);
109             % yv = linspace(min(y_surf), max(y_surf), 50);
110             % [X_surf,Y_surf] = meshgrid(xv, yv);
111             % Z_surf = griddata(x_surf,y_surf,z_surf,X_surf,Y_surf)
                    ;
112             X_surf = reshape(x_surf,[sqrt(length(x_surf)),sqrt(
                    length(x_surf))]);
113             Y_surf = reshape(y_surf,[sqrt(length(x_surf)),sqrt(
                    length(x_surf))]);
114             Z_surf = reshape(z_surf,[sqrt(length(x_surf)),sqrt(
                    length(x_surf))]);
115             surf(X_surf, Y_surf, Z_surf);
116             xlabel(['x [',opt2.Units,']'],'Interpreter','latex');
117             ylabel(['y [',opt2.Units,']'],'Interpreter','latex');
118             zlabel(['z [',opt2.Units,']'],'Interpreter','latex');
119             title('Polyhedron Method Gravitational Potential');
120             % axis equal;
121             cb = colorbar;
122             set(cb,'TickLabels',cb.Ticks/(scale*unit_scale));
123             cb.Label.String = 'V [J/kg]';
124             axes = gca;
125         end
126
```

```
        function write_V(obj,fname)
            V_fname = ['poly_V_',fname,'.txt'];
            V_fpath = fullfile(pwd,"Input_data","Potential_data",
               V_fname);
            writematrix([obj.r,obj.V],V_fpath);
        end
    end
end
```

```
    classdef mascons_obj < handle
    properties
        stl_model % model of the asteroid
        points {mustBeNumeric} % positions of the mascons points
        R {mustBeNumeric} % radius of the spheres
        V {mustBeNumeric} % gravitational potential
        M_i {mustBeNumeric} % mass of one point
        r {mustBeNumeric} % query points
    end

    methods
        function obj = mascons_obj(model,mas_r, bulk_density) %
           constructor, divides objects into mascons
            arguments
                model triangulation
                mas_r double
                bulk_density double
            end

            cub_minmax = [min(model.Points); max(model.Points)];

            mas_p_x = (cub_minmax(1,1)+mas_r:mas_r*2:cub_minmax
               (2,1)-mas_r)';
            mas_p_y = (cub_minmax(1,2)+mas_r:mas_r*2:cub_minmax
               (2,2)-mas_r)';
            mas_p_z = (cub_minmax(1,3)+mas_r:mas_r*2:cub_minmax
               (2,3)-mas_r)';

            [mas_grid_x, mas_grid_y, mas_grid_z] = ndgrid(mas_p_x,
               mas_p_y, mas_p_z);
            mas_grid_p = [mas_grid_x(:) mas_grid_y(:) mas_grid_z(:)
               ];

            in = inpolyhedron(model.ConnectivityList,model.Points,
               mas_grid_p);

            mas_grid_p = [mas_grid_p(in==1,1),mas_grid_p(in==1,2),
               mas_grid_p(in==1,3)];
```

```
31
32              model_new.vertices = model.Points;
33              model_new.faces = model.ConnectivityList;
34
35              err = 1e-15;
36
37              [dist, ~] = point2trimesh(model_new,'QueryPoints',
                    mas_grid_p);
38              in2 = (abs(dist)+err)>=mas_r;
39
40              mas_grid_p = [mas_grid_p(in2==1,1),mas_grid_p(in2==1,2)
                    ,mas_grid_p(in2==1,3)];
41
42              obj.stl_model = model;
43              obj.points = mas_grid_p;
44              obj.R = mas_r;
45
46              Vol = stlVolume(obj.stl_model.Points',obj.stl_model.
                    ConnectivityList');
47              M = bulk_density * Vol;
48              obj.M_i = M/length(obj.points(:,1));
49          end
50
51          function calculate(obj,r) % method calculating
                gravitational potential
52
53              G = 6.6743e-11;
54
55              obj.V = zeros(length(r(:,1)),1);
56
57              for i = 1:length(r(:,1))
58                  rq = r(i,:);
59                  for k = 1:length(obj.points(:,1))
60                      obj.V(i,1) = obj.V(i,1) + -(G * obj.M_i/(norm(
                            rq - obj.points(k,:))));
61                  end
62              end
63              obj.r = r;
64          end
65
66          function axes = plot_mascons(obj,opt1,opt2) % method
                plotting mascons model
67              arguments
68                  obj mascons_obj
69                  opt1.Spheres char
70                  opt2.Units char
71              end
```

```
72
73              if ~isfield(opt1,"Spheres")
74                  opt1.Spheres = 'off';
75              end
76              if ~isfield(opt2,"Units")
77                  opt2.Units = 'm';
78              end
79
80              model_temp = obj.stl_model;
81              if strcmp(opt2.Units,'km')
82                  unit_scale = 1/1000;
83                  model_temp = triangulation(model_temp.
                        ConnectivityList,model_temp.Points * unit_scale)
                        ;
84              end
85
86              trisurf(model_temp,'FaceColor','black','FaceAlpha',0.2)
                    ;
87              hold on;
88              if strcmp(opt1.Spheres,'on')
89
90                  for i = 1:length(obj.points(:,1))
91                      [x_s, y_s, z_s] = sphere;
92                      x_s = x_s * obj.R + obj.points(i,1);
93                      y_s = y_s * obj.R + obj.points(i,2);
94                      z_s = z_s * obj.R + obj.points(i,3);
95                      if strcmp(opt2.Units,'km')
96                          surf(x_s./1000,y_s./1000,z_s./1000);
97                      elseif strcmp(opt2.Units,'m')
98                          surf(x_s,y_s,z_s,'LineWidth',0.1);
99                      end
100                 end
101             else
102                 % scatter3(obj.points(:,1), obj.points(:,2), obj.
                        points(:,3),'red');
103             end
104             hold off;
105             xlabel(['x [',opt2.Units,']'],'Interpreter','latex');
106             ylabel(['y [',opt2.Units,']'],'Interpreter','latex');
107             zlabel(['z [',opt2.Units,']'],'Interpreter','latex');
108             title([num2str(length(obj.points(:,1))), ' points']);
109             axis equal;
110             axes = gca;
111         end
112
113         function axes = plot_V(obj, r, scale, opt2) % method
                plotting gravitational potential
```

```
        arguments
            obj mascons_obj
            r double
            scale double
            opt2.Units char
        end
        if ~isfield(opt2,"Units")
            opt2.Units = 'm';
        end

        unit_scale = 1;

        model_temp = obj.stl_model;
        if strcmp(opt2.Units,'km')
            unit_scale = 1/1000;
            model_temp = triangulation(model_temp.
               ConnectivityList,model_temp.Points * unit_scale)
               ;
        end

        figure();
        trisurf(model_temp,'FaceColor','black','FaceAlpha',0.2)
           ;
        hold on;

        if strcmp(opt2.Units,'km')
            unit_scale = 1/1000;
            x_surf = r(:,1)*unit_scale;
            y_surf = r(:,2)*unit_scale;
            z_surf = r(:,3)*unit_scale+obj.V*scale*unit_scale;
        elseif strcmp(opt2.Units,'m')
            x_surf = r(:,1)*unit_scale;
            y_surf = r(:,2)*unit_scale;
            z_surf = r(:,3)*unit_scale+obj.V*scale*unit_scale;
        end



        xv = linspace(min(x_surf), max(x_surf), 50);
        yv = linspace(min(y_surf), max(y_surf), 50);
        [X_surf,Y_surf] = meshgrid(xv, yv);
        Z_surf = griddata(x_surf,y_surf,z_surf,X_surf,Y_surf);
        surf(X_surf, Y_surf, Z_surf);
        xlabel(['x [',opt2.Units,']'],'Interpreter','latex');
        ylabel(['y [',opt2.Units,']'],'Interpreter','latex');
        zlabel(['z [',opt2.Units,']'],'Interpreter','latex');
        title('Mascons Method Gravitational Potential');
```

```
            axis equal;
            cb = colorbar;
            set(cb,'TickLabels',cb.Ticks/(scale*unit_scale));
            cb.Label.String = 'V [J/kg]';
            axes = gca;
        end

        function write_V(obj,fname)
            V_fname = ['mass_V_',fname,'.txt'];
            V_fpath = fullfile(pwd,"Input_data","Potential_data",
               V_fname);
            writematrix([obj.r,obj.V],V_fpath);
        end
    end
end
```

```
    classdef spherical_harmonics_obj < handle
    properties
        stl_model
        V {mustBeNumeric}
        C_nm
        S_nm
        M
        a
        n_max
        bulk_density
        r
    end

    methods
        function obj = spherical_harmonics_obj(model, bulk_density,
            n_max, a)
            arguments
                model triangulation
                bulk_density double
                n_max double
                a double
            end
            obj.n_max = n_max;
            obj.bulk_density = bulk_density;
            obj.stl_model = model;
            Vol = stlVolume(obj.stl_model.Points',obj.stl_model.
               ConnectivityList');
            obj.M = bulk_density * Vol;
            obj.a = a;
        end

```

```
30          function calculate_coefs_point_masses(obj,mas_grid_p,M_i)
31              arguments
32                  obj spherical_harmonics_obj
33                  mas_grid_p double
34                  M_i double
35              end
36
37              G = 6.6743e-11;
38
39              n_max = obj.n_max;
40              obj.C_nm = zeros(n_max+1,n_max+1);
41              obj.S_nm = zeros(n_max+1,n_max+1);
42
43              c = zeros(n_max+1,n_max+1,length(mas_grid_p(:,1)));
44              s = zeros(n_max+1,n_max+1,length(mas_grid_p(:,1)));
45
46              a = obj.a;
47
48                  for n = 0:n_max
49                      for m = 0:n
50                          for i = 1:length(mas_grid_p(:,1))
51                              mas_point = mas_grid_p(i,:);
52                              x_prim = mas_point(1);
53                              y_prim = mas_point(2);
54                              z_prim = mas_point(3);
55                              r_prim = norm(mas_point);
56                              if n == m
57                                  if n == 0
58                                      c(1,1,i) = 1/obj.M;
59                                      s(1,1,i) = 0;
60                                  elseif n == 1
61                                      c(2,2,i) = 1/(sqrt(3)*obj.M*a)*
                                            x_prim;
62                                      s(2,2,i) = 1/(sqrt(3)*obj.M*a)*
                                            y_prim;
63                                  else
64                                      h_temp = (2*n-1)/(sqrt(2*n*(2*n
                                            +1))) * [x_prim/a, -y_prim/a
                                            ; y_prim/a, x_prim/a] * [c(n
                                            ,n,i); s(n,n,i)];
65                                      c(n+1,n+1,i) = h_temp(1);
66                                      s(n+1,n+1,i) = h_temp(2);
67                                      h_temp = [];
68                                  end
69                              elseif m == n-1
70                                  h_temp = (2*n-1)/(sqrt(2*n+1)) *
                                        z_prim/a * [c(n,n,i); s(n,n,i)];
```

```
                                c(n+1,n,i) = h_temp(1);
                                s(n+1,n,i) = h_temp(2);
                                h_temp = [];
                            elseif m < n-1
                                h_temp = (2*n-1)*sqrt((2*n-1)/((2*n
                                   +1)*(n+m)*(n-m))) * z_prim/a * [
                                   c(n,m+1,i); s(n,m+1,i)] - sqrt
                                   ((2*n-3)*(n+m-1)*(n-m-1)/((2*n
                                   +1)*(n+m)*(n-m))) * (r_prim/a)^2
                                    * [c(n-1,m+1,i); s(n-1,m+1,i)];
                                c(n+1,m+1,i) = h_temp(1);
                                s(n+1,m+1,i) = h_temp(2);
                                h_temp = [];
                            end
                        end
                    end
                end

                for i = 1:length(mas_grid_p(:,1))
                    if i == 1
                        obj.C_nm(:,:) = c(:,:,i) .* M_i;
                        obj.S_nm(:,:) = s(:,:,i) .* M_i;
                    else
                        obj.C_nm(:,:) = obj.C_nm(:,:) + c(:,:,i) .*
                            M_i;
                        obj.S_nm(:,:) = obj.S_nm(:,:) + s(:,:,i) .*
                            M_i;
                    end
                end




        end

        function calculate_coefs(obj)
            arguments
                obj spherical_harmonics_obj
            end
            set_of_edges = obj.stl_model.edges;
            set_of_faces = obj.stl_model.ConnectivityList;
            set_of_n_f = faceNormal(obj.stl_model);
            num_of_faces = length(set_of_faces(:,1));
            G = 6.6743e-11;

            n_max = obj.n_max;
            obj.C_nm = zeros(n_max+1);
```

```
111             obj.S_nm = zeros(n_max+1);
112
113             for n = 0:n_max
114                 if n > 1
115                     c_prev2 = [];
116                     s_prev2 = [];
117                     c_prev2 = c_prev1;
118                     s_prev2 = s_prev1;
119                 end
120                 if n > 0
121                     c_prev1 = [];
122                     s_prev1 = [];
123                     c_prev1 = c;
124                     s_prev1 = s;
125                 end
126                 t = (n+2)*(n+1)*0.5;
127                 c = zeros(t,n+1,num_of_faces);
128                 s = zeros(t,n+1,num_of_faces);
129                 for m = 0:n
130                     for f = 1:num_of_faces
131                         face = set_of_faces(f,:);
132                         face_points = obj.stl_model.Points(face',:)
                            ;
133                         J = face_points';
134                         x_prim = J(1,:);
135                         y_prim = J(2,:);
136                         z_prim = J(3,:);
137                         if n == m
138                             if n == 0
139                                 c(:,1,f) = 1/obj.M;
140                                 s(:,1,f) = 0;
141                             elseif n == 1
142                                 c(:,2,f) = 1/(sqrt(3)*obj.M*obj.a)*
                                    x_prim;
143                                 s(:,2,f) = 1/(sqrt(3)*obj.M*obj.a)*
                                    y_prim;
144                             else
145                                 c(:,m+1,f) = (2*n-1)/(sqrt(2*n*(2*n
                                    +1))*obj.a) * (tri_mult(x_prim,
                                    c_prev1(:,m,f)) - tri_mult(
                                    y_prim,s_prev1(:,m,f)));
146                                 s(:,m+1,f) = (2*n-1)/(sqrt(2*n*(2*n
                                    +1))*obj.a) * (tri_mult(y_prim,
                                    c_prev1(:,m,f)) + tri_mult(
                                    x_prim,s_prev1(:,m,f)));
147                             end
148                         elseif m == n-1
```

```
149                         c(:,m+1,f) = (2*n - 1)/(sqrt(2*n+1) *
                               obj.a) * tri_mult(z_prim,c_prev1(:,m
                                + 1,f));
150                         s(:,m+1,f) = (2*n - 1)/(sqrt(2*n+1) *
                               obj.a) * tri_mult(z_prim,s_prev1(:,m
                                + 1,f));
151                     elseif m < n-1
152                         c(:,m+1,f) = (2*n-1)*sqrt((2*n-1)/((2*n
                               +1)*(n+m)*(n-m)))/obj.a * tri_mult(
                               z_prim,c_prev1(:,m + 1,f)) - sqrt
                               ((2*n-3)*(n+m-1)*(n-m-1)/((2*n+1)*(n
                               +m)*(n-m)))/(obj.a^2) * (tri_mult((
                               tri_mult(x_prim,x_prim) + tri_mult(
                               y_prim,y_prim) + tri_mult(z_prim,
                               z_prim)),c_prev2(:,m + 1,f)));
153                         s(:,m+1,f) = (2*n-1)*sqrt((2*n-1)/((2*n
                               +1)*(n+m)*(n-m)))/obj.a * tri_mult(
                               z_prim,s_prev1(:,m + 1,f)) - sqrt
                               ((2*n-3)*(n+m-1)*(n-m-1)/((2*n+1)*(n
                               +m)*(n-m)))/(obj.a^2) * (tri_mult((
                               tri_mult(x_prim,x_prim) + tri_mult(
                               y_prim,y_prim) + tri_mult(z_prim,
                               z_prim)),s_prev2(:,m + 1,f)));
154
155                     end
156                     obj.C_nm(n+1,m+1) = obj.C_nm(n+1,m+1) +
                           integrate_simplex(c(:,m+1,f),det(J));
157                     obj.S_nm(n+1,m+1) = obj.S_nm(n+1,m+1) +
                           integrate_simplex(s(:,m+1,f),det(J));
158                     disp(['face ', num2str(f), ' of ', num2str(
                           num_of_faces), ', n = ', num2str(n), ',
                           m = ', num2str(m)]);
159                 end
160             end
161         end
162         obj.C_nm = obj.C_nm*obj.bulk_density;
163         obj.S_nm = obj.S_nm*obj.bulk_density;
164
165     end
166
167     function calculate(obj,C,S,r_query,centroid)
168         obj.V = zeros(length(r_query(:,1)),1);
169         G = 6.6743e-11;
170         for i = 1:length(r_query(:,1))
171             rq = r_query(i,:);
172             r = norm(rq);
173             lat = asin(rq(3)/r);
```

```
174                 lon = atan2(rq(2),rq(1));
175                 n_max = obj.n_max;
176                 sum_m = 0;
177                 sum_n = 0;
178                 for n = 1:n_max
179                     P = legendre(n,sin(lat));
180                     for m = 0:n
181                         N = sqrt((2-double(n==m)) * (2*n+1) *
                               factorial(n-m) / factorial(n+m));
182                         sum_m = sum_m + N * P(m+1) * (C(n+1,m+1)*
                               cos(m*lon) + S(n+1,m+1)*sin(m*lon));
183                     end
184                     sum_n = sum_n + (obj.a/r)^n * sum_m;
185                     sum_m = 0;
186                 end
187                 obj.V(i,1) = G*obj.M/r * (1+sum_n);
188                 sum_n = 0;
189                 obj.r = r_query;
190             end
191         end
192
193         function write_coefs_werner(obj)
194             C_fname = ['C_werner_',num2str(obj.n_max),'.txt'];
195             C_fpath = fullfile(pwd,"Input_data","Coefs",C_fname);
196             writematrix(obj.C_nm,C_fpath)
197
198             S_fname = ['S_werner_',num2str(obj.n_max),'.txt'];
199             S_fpath = fullfile(pwd,"Input_data","Coefs",S_fname);
200             writematrix(obj.S_nm,S_fpath)
201         end
202
203         function write_coefs_mas(obj, mas_r)
204             C_fname = ['C_mas_',num2str(obj.n_max),'_',num2str(
                    mas_r),'.txt'];
205             C_fpath = fullfile(pwd,"Input_data","Coefs",C_fname);
206             writematrix(obj.C_nm,C_fpath)
207
208             S_fname = ['S_mas_',num2str(obj.n_max),'_',num2str(
                    mas_r),'.txt'];
209             S_fpath = fullfile(pwd,"Input_data","Coefs",S_fname);
210             writematrix(obj.S_nm,S_fpath)
211         end
212
213         function read_coefs_werner(obj)
214             C_fname = ['C_werner_',num2str(obj.n_max),'.txt'];
215             C_fpath = fullfile(pwd,"Input_data","Coefs",C_fname);
216             S_fname = ['S_werner_',num2str(obj.n_max),'.txt'];
```

```
            S_fpath = fullfile(pwd,"Input_data","Coefs",S_fname);

            obj.C_nm = readmatrix(C_fpath);
            obj.S_nm = readmatrix(S_fpath);
        end

        function read_coefs_mas(obj, mas_r)
            C_fname = ['C_mas_',num2str(obj.n_max),'_',num2str(
               mas_r),'.txt'];
            C_fpath = fullfile(pwd,"Input_data","Coefs",C_fname);
            S_fname = ['S_mas_',num2str(obj.n_max),'_',num2str(
               mas_r),'.txt'];
            S_fpath = fullfile(pwd,"Input_data","Coefs",S_fname);

            obj.C_nm = readmatrix(C_fpath);
            obj.S_nm = readmatrix(S_fpath);
        end


        function axes = plot_V(obj, r, scale, opt2)
            arguments
                obj spherical_harmonics_obj
                r double
                scale double
                opt2.Units char
            end
            if ~isfield(opt2,"Units")
                opt2.Units = 'm';
            end

            unit_scale = 1;

            model_temp = obj.stl_model;
            if strcmp(opt2.Units,'km')
                unit_scale = 1/1000;
                model_temp = triangulation(model_temp.
                   ConnectivityList,model_temp.Points * unit_scale)
                   ;
            end

            figure();
            trisurf(model_temp,'FaceColor','black','FaceAlpha',0.2)
               ;
            hold on;

            if strcmp(opt2.Units,'km')
                unit_scale = 1/1000;
```

```
                x_surf = r(:,1)*unit_scale;
                y_surf = r(:,2)*unit_scale;
                z_surf = r(:,3)*unit_scale-obj.V*scale*unit_scale;
            elseif strcmp(opt2.Units,'m')
                x_surf = r(:,1)*unit_scale;
                y_surf = r(:,2)*unit_scale;
                z_surf = r(:,3)*unit_scale-obj.V*scale*unit_scale;
            end


            xv = linspace(min(x_surf), max(x_surf), 50);
            yv = linspace(min(y_surf), max(y_surf), 50);
            [X_surf,Y_surf] = meshgrid(xv, yv);
            Z_surf = griddata(x_surf,y_surf,z_surf,X_surf,Y_surf);
            surf(X_surf, Y_surf, Z_surf);
            xlabel(['x [',opt2.Units,']'],'Interpreter','latex');
            ylabel(['y [',opt2.Units,']'],'Interpreter','latex');
            zlabel(['z [',opt2.Units,']'],'Interpreter','latex');
            title('Polyhedron Method Gravitational Potential');
            axis equal;
            cb = colorbar;
            set(cb,'TickLabels',cb.Ticks/(scale*unit_scale));
            cb.Label.String = 'V [J/kg]';
            axes = gca;
        end

        function write_V(obj,fname)
            V_fname = ['sh_V_',fname,'.txt'];
            V_fpath = fullfile(pwd,"Input_data","Potential_data",
               V_fname);
            writematrix([obj.r,obj.V],V_fpath);
        end
    end
end
```

6.2 Funkcje

```
    function r_t = vec_model2face(r,n_f,p_a,p_b)
    k = n_f;
    i = (p_b - p_a)/norm(p_b - p_a);
    j = cross(k,i);
    F = [i', j', k'];
    M = [[1,0,0]',[0,1,0]',[0,0,1]'];
    r_t = (F \ M * r')';
    end
```

```
    function tri_result = tri_mult(tri1, tri2)
    n1 = (-3+sqrt(9-(8*(1-length(tri1)))))/2;
    n2 = (-3+sqrt(9-(8*(1-length(tri2)))))/2;
    n = n1 + n2;
    t_result = (n+2)*(n+1)*0.5;
    tri_result = zeros(1,t_result);
    for i = n1:-1:0
        for k = 0:(n1-i)
            j = n1-i-k;
            for r = n2:-1:0
                for s = 0:(n2-r)
                    t = n2-r-s;
                    index = (j+s+k+t)*(j+s+k+t+1)/2 + k + t + 1;
                    index1 = (j+k)*(j+k+1)/2 + k + 1;
                    index2 = (s+t)*(s+t+1)/2 + t + 1;
                    tri_result(index) = tri_result(index) + tri1(
                       index1) * tri2(index2);
                end
            end
        end
    end
end
```

```
    function centroid = calculate_centroid(stl_model)
    points = stl_model.Points;
    centroid = mean(points);
end
```

```
    function [rq, rqn, i] = gen_points(a,b,n,mode,r)
    if strcmp(mode,"out")
        rq = (b-a).*rand(n,3) + a;
        rqn = vecnorm(rq,2,2);
        while any(rqn < r)
            n_temp = length(rqn(rqn < r));
            rq(rqn < r,:) = (b-a).*rand(n_temp,3) + a;
            rqn = vecnorm(rq,2,2);
```

```
        end
        i = n;
    elseif strcmp(mode,"in")
        rq = (b-a).*rand(n,3) + a;
        rqn = vecnorm(rq,2,2);
        while any(rqn > r)
            n_temp = length(rqn(rqn > r));
            rq(rqn > r,:) = (b-a).*rand(n_temp,3) + a;
            rqn = vecnorm(rq,2,2);
        end
        i = 0;
    else
        rq = (b-a).*rand(n,3) + a;
        rqn = vecnorm(rq,2,2);
        i = length(rqn(rqn > r));
    end
end
```

```
    function rq = gen_points_kleo(a,b,n,model)
    rq = (b-a).*rand(n,3) + a;
    in = inpolyhedron(model.ConnectivityList,model.Points,rq);

    while any(in == 1)
    n_temp = length(in(in == 1));
    rq(in == 1,:) = (b-a).*rand(n_temp,3) + a;
    in = inpolyhedron(model.ConnectivityList,model.Points,rq);
    end
end
```

```
    function E_e = get_E_e(model,edge,plot)
    p_a = model.Points(edge(1),:);
    p_b = model.Points(edge(2),:);

    faces_id = cell2mat(edgeAttachments(model,edge(1),edge(2)))';
    n_f = faceNormal(model,faces_id);

    n_ab = (p_b - p_a)/norm(p_b - p_a);

    n_e(1,:) = cross(n_ab*(-1),n_f(1,:));
    n_e(1,:) = n_e(1,:)/norm(n_e(1,:));
    n_e(2,:) = cross(n_ab,n_f(2,:));
    n_e(2,:) = n_e(2,:)/norm(n_e(2,:));

    face_points_id(1,:) = model.ConnectivityList(faces_id(1),:);

    color_i = 1;
    p_c = model.Points(face_points_id(1,:),:);
```

```
    for i = 3:-1:1
        if p_c(i,:) == p_a
            p_c(i,:) = [];
        elseif p_c(i,:) == p_b
            p_c(i,:) = [];
        end
    end
    v_ac = p_c - p_a;
    v_aq = dot(v_ac,n_ab)*n_ab;
    p_q = p_a + v_aq;
    v_qc = p_c - p_q;
    if dot(v_qc,n_e(1,:)) > 0
        n_e = n_e * (-1);
        color_i = 3;
    end

    E_e = n_f(1,:)' * n_e(1,:) + n_f(2,:)' * n_e(2,:);


    if strcmp(plot,'on')
        color = ['r','b','y','g'];
    p_test = (p_a + p_b)/2;
    scale = 1/10;
    hold on;
    quiver3(p_test(1,1),p_test(1,2),p_test(1,3), n_f(1,1)*scale,n_f
       (1,2)*scale,n_f(1,3)*scale,0.5,'color',color(color_i));
    quiver3(p_test(1,1),p_test(1,2),p_test(1,3), n_f(2,1)*scale,n_f
       (2,2)*scale,n_f(2,3)*scale,0.5,'color',color(color_i+1));
    quiver3(p_test(1,1),p_test(1,2),p_test(1,3), n_e(1,1)*scale,n_e
       (1,2)*scale,n_e(1,3)*scale,0.5,'color',color(color_i));
    quiver3(p_test(1,1),p_test(1,2),p_test(1,3), n_e(2,1)*scale,n_e
       (2,2)*scale,n_e(2,3)*scale,0.5,'color',color(color_i+1));
    axis equal;
    end
    end
```

```
    function F_f = get_F_f(n_f)
    F_f = n_f' * n_f;
end
```

```
    function L_e = get_L_e(r,p_a,p_b)
    e = norm(p_a-p_b);
    a = norm(r-p_a);
    b = norm(r-p_b);
    L_e = log((a+b+e)/(a+b-e));
    end
```

```
    function r_ef = get_r_ef(r,p_a)
    r_ef = p_a - r;
    end
```

```
    function w_f = get_w_f(r,face_points)
    r_1 = face_points(1,:) - r;
    r_1 = r_1/norm(r_1);
    r_2 = face_points(2,:) - r;
    r_2 = r_2/norm(r_2);
    r_3 = face_points(3,:) - r;
    r_3 = r_3/norm(r_3);




    w_f = 2 * atan2((dot(r_1,cross(r_2,r_3))),(1 + dot(r_1,r_2) +
       dot(r_2,r_3) + dot(r_3,r_1)));
    end
```

```
    function int_result = integrate_simplex(tri,det_J)
    n = (-3+sqrt(9-(8*(1-length(tri)))))/2;
    for i = n:-1:0
        for k = 0:(n-i)
            j = n-i-k;
            index = (j+k)*(j+k+1)/2 + k + 1;
            mix_factor = factorial(i)*factorial(j)*factorial(k)/
               factorial(n+3);
            tri(index) = tri(index)*mix_factor;
        end
    end
    int_result = det_J * sum(tri);
end
```

```
    function p_c = p_model2face(p,n_f,p_a,p_b,rq)
    k = n_f;
    i = (p_b - p_a)/norm(p_b - p_a);
    j = cross(k,i);
    F = [i', j', k'];
    M = [[1,0,0]',[0,1,0]',[0,0,1]'];

    p_r =  - rq;
    p_c = (F \ M * p_r')';
    end
```

6.3 Skrypty

```
clear;
% close all;
addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data'

%% Reading model
unit_scale = 1000;

model_fname = 'Kleopatra_Ostro.stl';
cd_splitted = split(mfilename('fullpath'),'\');
model_path = fullfile(cd_splitted{1:end-2},"model3d",
   model_fname);
model_temp = stlread(model_path);
model_temp = triangulation(model_temp.ConnectivityList,
   model_temp.Points * unit_scale);
centroid = calculate_centroid(model_temp);
model = triangulation(model_temp.ConnectivityList,model_temp.
   Points - centroid);
clear model_fname cd_splitted model_path model_temp centroid


a = -200000;
b = 200000;
n = 100;
rq = gen_points_kleo(a,b,n,model);

trimesh(model);
hold on;
scatter3(rq(:,1),rq(:,2),rq(:,3));
axis equal;

writematrix(rq,"rand_kleo.txt");
```

```
close all;
clear;

base = IcosahedronMesh();



f1 = figure();
f1.Position = [100 100 800 600];
tiledlayout(f1,2,2);
nexttile
trisurf(base);
```

```
    xlabel('x [m]','Interpreter','latex');
    ylabel('y [m]','Interpreter','latex');
    zlabel('z [m]','Interpreter','latex');
    xlim([-1,1]);
    daspect([1 1 1]);
    title('20 faces');


    complex1 = SubdivideSphericalMesh(base,1);

    nexttile;
    trisurf(complex1);
    axis equal;
    xlabel('x [m]','Interpreter','latex');
    ylabel('y [m]','Interpreter','latex');
    zlabel('z [m]','Interpreter','latex');
    title('80 faces');

    complex2 = SubdivideSphericalMesh(base,2);

    nexttile
    trisurf(complex2);
    axis equal;
    xlabel('x [m]','Interpreter','latex');
    ylabel('y [m]','Interpreter','latex');
    zlabel('z [m]','Interpreter','latex');
    title('320 faces');

    complex3 = SubdivideSphericalMesh(base,3);

    nexttile
    trisurf(complex3);
    axis equal;
    xlabel('x [m]','Interpreter','latex');
    ylabel('y [m]','Interpreter','latex');
    zlabel('z [m]','Interpreter','latex');
    title('1280 faces');
```

```
    clear;
    close all;
    addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
    addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data\
       Coefs'

    V_poly_path = fullfile(pwd,"Input_data","Potential_data","
       poly_V_kleo.txt");
    V_mass1_path = fullfile(pwd,"Input_data","Potential_data","
```

```
   mass_V_kleo1.txt");
V_mass2_path = fullfile(pwd,"Input_data","Potential_data","
   mass_V_kleo2.txt");
V_mass3_path = fullfile(pwd,"Input_data","Potential_data","
   mass_V_kleo3.txt");
V_sh1_path = fullfile(pwd,"Input_data","Potential_data","
   sh_V_kleo_coef_4.txt");
V_sh2_path = fullfile(pwd,"Input_data","Potential_data","
   sh_V_kleo_coef_10.txt");
V_sh3_path = fullfile(pwd,"Input_data","Potential_data","
   sh_V_kleo_coef_50.txt");

V_poly = readmatrix(V_poly_path);
rq = V_poly(:,1:3);
rqn = vecnorm(rq,2,2);
V_poly = V_poly(:,4);


V_mass1 = readmatrix(V_mass1_path);
V_mass1 = V_mass1(:,4);
V_mass2 = readmatrix(V_mass2_path);
V_mass2 = V_mass2(:,4);
V_mass3 = readmatrix(V_mass3_path);
V_mass3 = V_mass3(:,4);

V_sh1 = readmatrix(V_sh1_path);
V_sh1 = -V_sh1(:,4);
V_sh2 = readmatrix(V_sh2_path);
V_sh2 = -V_sh2(:,4);
V_sh3 = readmatrix(V_sh3_path);
V_sh3 = -V_sh3(:,4);

V = [V_mass1, V_mass2, V_mass3, V_sh1, V_sh2, V_sh3];

for i = 1:length(V(1,:))
    err(i) = mean(abs(V(:,i)-V_poly)./abs(V_poly));
end

t = []


err_sh = abs(V_sh3-V_poly)./abs(V_poly);
p = fit(rqn,err_sh,'exp1');
px = sort(rqn);

plot(p,rqn,err_sh);
grid on;
```

```
clear;
close all;
addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data'

% Configuration file name
config_fname = "sh.json";

%% Reading config
config_path = fullfile(pwd,"Input_data","Config",config_fname);
config_fid = fopen(config_path);
config_raw = fread(config_fid);
config = jsondecode(char(config_raw'));
fclose(config_fid);
clear config_raw config_fid config_path config_fname

r_fname = config.main_configuration.query_points_file;
r_path = fullfile(pwd,"Input_data","Query_points",r_fname);
r = readmatrix(r_path);
clear r_fname r_path

%% Reading model
model_fname = config.main_configuration.model_file;
cd_splitted = split(mfilename('fullpath'),'\');
model_path = fullfile(cd_splitted{1:end-2},"model3d",
   model_fname);
model_temp = stlread(model_path);
model_temp = triangulation(model_temp.ConnectivityList,
   model_temp.Points * config.main_configuration.unit_scale);
centroid = calculate_centroid(model_temp);
model = triangulation(model_temp.ConnectivityList,model_temp.
   Points - centroid);
clear model_fname cd_splitted model_path model_temp centroid

%% Calculating gracitational potential
method = config.method.method_name;
if strcmp(method,"Spherical Harmonics")
        spherical_harmonics_method = spherical_harmonics_obj(
           model,config.main_configuration.bulk_density,config.
           method.degree,config.method.a);
        if strcmp(config.method.calculation_algorithm,"werner")
            if exist(fullfile(pwd,"Input_data","Coefs",['
               C_werner_',num2str(config.method.degree),'.txt'
               ]),'file') == 2
                spherical_harmonics_method.read_coefs_werner();
```

```
39                  else
40                  spherical_harmonics_method.calculate_coefs();
41                  spherical_harmonics_method.write_coefs_werner();
42                  end
43              else
44                  mas_r = config.method.mas_r;
45                  if exist(fullfile(pwd,"Input_data","Coefs",['C_mas_
                        ',num2str(config.method.degree),'_',num2str(
                        mas_r),'.txt']),'file') == 2
46                      spherical_harmonics_method.read_coefs_mas(mas_r
                            )
47                  else
48                  mascons_method = mascons_obj(model,mas_r,config.
                        main_configuration.bulk_density);
49                  spherical_harmonics_method.
                        calculate_coefs_point_masses(mascons_method.
                        points,mascons_method.M_i);
50                  spherical_harmonics_method.write_coefs_mas(mas_r);
51                  end
52              end
53      elseif strcmp(method,"Mascons")
54      mascons_method = mascons_obj(model,config.method.mas_r,config.
            main_configuration.bulk_density);
55      mascons_method.calculate(r);
56      elseif strcmp(method,"Polyhedron")
57      polyhedron_method = polyhedron_obj(model);
58      polyhedron_method.calculate(r,config.main_configuration.
            bulk_density);
59
60      else
61          disp("Wrong method chosen. Available methods: Spherical
                Harmonics, Mascons or Polyhedron");
62      end
63      clear method
```

```
1       % close all;
2       clear;
3
4       %% Initial data (TODO in json file)
5       % Asteroid data
6       model_fname = "Kleopatra_Ostro.stl";
7       bulk_density = 3600; % kg/m^3
8
9       % Field query points
10      x = linspace(-300000,300000,30); % m
11      y = x;
12      [X,Y] = meshgrid(x);
```

```
    r = [X(:),Y(:),-6e+4 * ones(length(X(:)),1)];

    % Mascons method configuration
    mas_r = 5000; % radius of the spheres
    unit_scale = 1000; % for models provided not in meters

    % Ploting configuration
    plot_unit = 'm'; % units of plots (km or m)
    V_scale = 10^2; % scale of the potential plot

    %% Main section
    % Creating mascons model
    mascons_method = mascons_obj(model_fname,mas_r,unit_scale,
       bulk_density);
    % Plotting mascons model
    mascons_method.plot_mascons('Spheres','on','Units',plot_unit);
    % Calculating gravitational potential
    mascons_method.calculate(r);
    % Plotting results
    mascons_method.plot_V(r,V_scale,'Units',plot_unit);
```

```
    clear;
    close all;
    addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
    addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data'

    r_fname = 'rand_kleo.txt';
    r_path = fullfile(pwd,"Input_data","Query_points",r_fname);
    rq = readmatrix(r_path);
    clear r_fname r_path

    unit_scale = 1000;

    model_fname = 'Kleopatra_Ostro.stl';
    cd_splitted = split(mfilename('fullpath'),'\');
    model_path = fullfile(cd_splitted{1:end-2},"model3d",
       model_fname);
    model_temp = stlread(model_path);
    model_temp = triangulation(model_temp.ConnectivityList,
       model_temp.Points * unit_scale);
    centroid = calculate_centroid(model_temp);
    model = triangulation(model_temp.ConnectivityList,model_temp.
       Points - centroid);
    clear model_fname cd_splitted model_path model_temp centroid

    %% Creating objects
    bulk_density = 3600;
```

```
24
25      a1 = 12000;
26      a2 = 8000;
27      a3 = 4000;
28
29      mass_1 = mascons_obj(model,a1,bulk_density);
30      mass_2 = mascons_obj(model,a2,bulk_density);
31      mass_3 = mascons_obj(model,a3,bulk_density);
32
33
34      %% Calculating potential
35
36      tic
37      mass_1.calculate(rq);
38      t1 = toc
39      tic
40      mass_2.calculate(rq);
41      t2 = toc
42      tic
43      mass_3.calculate(rq);
44      t3 = toc
45
46      mass_1.write_V('kleo1');
47      mass_2.write_V('kleo2');
48      mass_3.write_V('kleo3');
49
50      f1 = figure();
51      f1.Position = [100 100 700 700];
52      mass_1.plot_mascons("Spheres",'on','Units','km');
```

```
1       clear;
2       close all;
3       addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
4       addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data'
5
6       r_fname = 'rand_in.txt';
7       r_path = fullfile(pwd,"Input_data","Query_points",r_fname);
8       rq_in = readmatrix(r_path);
9       clear r_fname r_path
10
11      r_fname = 'rand_out.txt';
12      r_path = fullfile(pwd,"Input_data","Query_points",r_fname);
13      rq_out = readmatrix(r_path);
14      clear r_fname r_path
15
16      V_fpath_1 = fullfile(pwd,"Input_data","Potential_data","
          mass_V_rand_in1.txt");
```

```
17      V_fpath_2 = fullfile(pwd,"Input_data","Potential_data","
            mass_V_rand_in2.txt");
18      V_fpath_3 = fullfile(pwd,"Input_data","Potential_data","
            mass_V_rand_in3.txt");
19      V_fpath_4 = fullfile(pwd,"Input_data","Potential_data","
            mass_V_rand_in4.txt");
20
21      V1_in = readmatrix(V_fpath_1);
22      V1_in = V1_in(:,4);
23      V2_in = readmatrix(V_fpath_2);
24      V2_in = V2_in(:,4);
25      V3_in = readmatrix(V_fpath_3);
26      V3_in = V3_in(:,4);
27      V4_in = readmatrix(V_fpath_4);
28      V4_in = V4_in(:,4);
29
30      V_fpath_1 = fullfile(pwd,"Input_data","Potential_data","
            mass_V_rand_out1.txt");
31      V_fpath_2 = fullfile(pwd,"Input_data","Potential_data","
            mass_V_rand_out2.txt");
32      V_fpath_3 = fullfile(pwd,"Input_data","Potential_data","
            mass_V_rand_out3.txt");
33      V_fpath_4 = fullfile(pwd,"Input_data","Potential_data","
            mass_V_rand_out4.txt");
34
35      V1_out = readmatrix(V_fpath_1);
36      V1_out = V1_out(:,4);
37      V2_out = readmatrix(V_fpath_2);
38      V2_out = V2_out(:,4);
39      V3_out = readmatrix(V_fpath_3);
40      V3_out = V3_out(:,4);
41      V4_out = readmatrix(V_fpath_4);
42      V4_out = V4_out(:,4);
43
44      a = 1; % radius
45      M = 4/3 * pi * a^3; % mass of a sphere
46      G = 6.6743e-11; % gravitational constant
47      for i = 1:length(rq_in(:,1))
48          r = norm(rq_in(i,:));
49          if r > 1
50              V_in(i) = -G*M/r;
51          else
52              V_in(i) = -G*M* (3*a^2 - r^2)/(2*a^3);
53          end
54      end
55      for i = 1:length(rq_out(:,1))
56          r = norm(rq_out(i,:));
```

```
        if r > 1
            V_out(i) = -G*M/r;
        else
            V_out(i) = -G*M* (3*a^2 - r^2)/(2*a^3);
        end
    end
    V_in = V_in';
    V_out = V_out';

    err_1_in = mean(abs(V1_in-V_in)./abs(V_in));
    err_2_in = mean(abs(V2_in-V_in)./abs(V_in));
    err_3_in = mean(abs(V3_in-V_in)./abs(V_in));
    err_4_in = mean(abs(V4_in-V_in)./abs(V_in));

    err_1_out = mean(abs(V1_out-V_out)./abs(V_out));
    err_2_out = mean(abs(V2_out-V_out)./abs(V_out));
    err_3_out = mean(abs(V3_out-V_out)./abs(V_out));
    err_4_out = mean(abs(V4_out-V_out)./abs(V_out));

    points = [27, 360, 1181, 3448];
    err_in = [err_1_in, err_2_in, err_3_in, err_4_in];
    err_out = [err_1_out, err_2_out, err_3_out, err_4_out];

    f1 = figure();
    f1.Position = [100 100 1200 500];
    tiledlayout(1,2);
    nexttile;
    scatter(points,err_in,'filled','red');
    title('Query points inside the sphere');
    ylabel('Mean relative error');
    xlabel('Number of mass points');
    grid on;
    nexttile;
    scatter(points,err_out,'filled','red');
    title('Query points outside the sphere');
    ylabel('Mean relative error');
    xlabel('Number of mass points');
    grid on;

    t_in = [0.04, 0.49, 4.71, 81.52];
    t_out = [0.04, 0.49, 4.96, 82.3];

    f2 = figure();
    f2.Position = [100 100 1200 500];
    tiledlayout(1,2);
    nexttile;
    scatter(points,t_in,'filled','red');
```

```
    title('Query points inside the sphere');
    ylabel('Time of calculation [s]');
    xlabel('Number of mass points');
    grid on;
    nexttile;
    scatter(points,t_out,'filled','red');
    title('Query points outside the sphere');
    ylabel('Time of calculation [s]');
    xlabel('Number of mass points');
    grid on;

    rqn = vecnorm(rq_out,2,2);
    err_sh = abs(V4_out-V_out)./abs(V_out);
    f3 = figure();
    scatter(rqn,err_sh);
    grid on;
```

```
    close all;
    clear;


    V_fpath_1 = fullfile(pwd,"Input_data","Potential_data","
       mass_V_1.txt");
    V_fpath_2 = fullfile(pwd,"Input_data","Potential_data","
       mass_V_2.txt");
    V_fpath_3 = fullfile(pwd,"Input_data","Potential_data","
       mass_V_3.txt");
    V_fpath_4 = fullfile(pwd,"Input_data","Potential_data","
       mass_V_4.txt");

    V1 = readmatrix(V_fpath_1);
    V2 = readmatrix(V_fpath_2);
    V3 = readmatrix(V_fpath_3);
    V4 = readmatrix(V_fpath_4);
    rq = V1(:,1:3);

    x_surf = rq(:,1);
    y_surf = rq(:,2);
    % z_surf = obj.V;

    X_surf = reshape(x_surf,[sqrt(length(x_surf)),sqrt(length(
       x_surf))]);
    Y_surf = reshape(y_surf,[sqrt(length(x_surf)),sqrt(length(
       x_surf))]);
    % Z_surf = reshape(z_surf,[sqrt(length(x_surf)),sqrt(length(
       x_surf))]);

```

```
    %% Calculating true potential of a unit sphere
    a = 1; % radius
    M = 4/3 * pi * a^3; % mass of a sphere
    G = 6.6743e-11; % gravitational constant
    for i = 1:length(rq(:,1))
        r = norm(rq(i,:));
        if r > 1
            V(i) = -G*M/r;
        else
            V(i) = -G*M* (3*a^2 - r^2)/(2*a^3);
        end
    end
    V_true = reshape(V,[sqrt(length(x_surf)),sqrt(length(x_surf))])
       ;
    V1 = reshape(V1(:,4),[sqrt(length(x_surf)),sqrt(length(x_surf))
       ]);
    V2 = reshape(V2(:,4),[sqrt(length(x_surf)),sqrt(length(x_surf))
       ]);
    V3 = reshape(V3(:,4),[sqrt(length(x_surf)),sqrt(length(x_surf))
       ]);
    V4 = reshape(V4(:,4),[sqrt(length(x_surf)),sqrt(length(x_surf))
       ]);

    z_scale = 1/10;
    v_scale = 10^9;

    [x_s,y_s,z_s] = sphere(20);
    z_s = z_s * z_scale;

    f1 = figure();
    surf(X_surf,Y_surf,V_true*v_scale,'FaceColor','#1d4877');
    hold on;
    surf(X_surf,Y_surf,V1*v_scale,'FaceAlpha',1,'FaceColor','#
       ee3e32');
    surf(X_surf,Y_surf,V2*v_scale,'FaceAlpha',1,'FaceColor','#
       f68838');
    surf(X_surf,Y_surf,V3*v_scale,'FaceAlpha',1,'FaceColor','#
       fbb021');
    surf(X_surf,Y_surf,V4*v_scale,'FaceAlpha',1,'FaceColor','#1
       b8a5a');
    surf(x_s,y_s,z_s,'FaceColor',"#7E2F8E");
    daspect([1 1 z_scale]);
    f1.Position = [100 100 700 700];
    xlim([0,3]);
    ylim([-3,3]);
    xlabel('x [m]','Interpreter','latex');
    ylabel('y [m]','Interpreter','latex');
```

```
    zlabel('V [nJ/kg]','Interpreter','latex');
    legend('true','27','87','360','1181','Location','southoutside',
       'Orientation','horizontal');
    view(-65,20)
```

```
    clear;
    close all;
    addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
    addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data'

    r_fname = 'rand_out.txt';
    r_path = fullfile(pwd,"Input_data","Query_points",r_fname);
    rq = readmatrix(r_path);
    clear r_fname r_path


    %% Reading model


    model_fname = 'sphere320.stl';
    cd_splitted = split(mfilename('fullpath'),'\');
    model_path = fullfile(cd_splitted{1:end-2},"model3d",
       model_fname);
    model_temp = stlread(model_path);
    model = triangulation(model_temp.ConnectivityList,model_temp.
       Points);
    clear model_fname cd_splitted model_path model_temp centroid


    %% Creating objects
    bulk_density = 1;

    a1 = 0.2;
    a2 = 0.1;
    a3 = 0.06;
    a4 = 0.05;
    vol = 4/3*pi;

    mass_1 = mascons_obj(model,a1,bulk_density,vol);
    mass_2 = mascons_obj(model,a2,bulk_density,vol);
    mass_3 = mascons_obj(model,a3,bulk_density,vol);
    mass_4 = mascons_obj(model,a4,bulk_density,vol);

    %% Calculating potential

    tic
    mass_1.calculate(rq);
```

```
    t1 = toc
    tic
    mass_2.calculate(rq);
    t2 = toc
    tic
    mass_3.calculate(rq);
    t3 = toc
    tic
    mass_4.calculate(rq);
    t4 = toc

    mass_1.write_V('rand_out1');
    mass_2.write_V('rand_out2');
    mass_3.write_V('rand_out3');
    mass_4.write_V('rand_out4');




    % f1 = figure();
    % scatter3(rq(:,1),rq(:,2),rq(:,3),'filled','red');
    % hold on;
    % [x_s,y_s,z_s] = sphere;
    % surf(x_s,y_s,z_s,'FaceAlpha',0.2);
    % xlim([a,b]);
    % ylim([a,b]);
    % zlim([a,b]);
    % daspect([1 1 1]);
```

```
    clear;
    close all;
    addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
    addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data'


    model_fname = 'sphere320.stl';
    cd_splitted = split(mfilename('fullpath'),'\');
    model_path = fullfile(cd_splitted{1:end-2},"model3d",
       model_fname);
    model_temp = stlread(model_path);
    model = triangulation(model_temp.ConnectivityList,model_temp.
       Points);
    clear model_fname cd_splitted model_path model_temp centroid

    bulk_density = 1;

    a1 = 0.2;
```

```
    a2 = 0.1;
    a3 = 0.07;
    a4 = 0.04;

    mass_1 = mascons_obj(model,a1,bulk_density);
    mass_2 = mascons_obj(model,a2,bulk_density);
    mass_3 = mascons_obj(model,a3,bulk_density);
    mass_4 = mascons_obj(model,a4,bulk_density);

    %% Creating query points
    x = linspace(-3,3,30); % m
    y = x;
    [X,Y] = meshgrid(x);
    rq = [X(:),Y(:),zeros(length(X(:)),1)];

    f1 = figure();
    f1.Position = [100 100 700 700];
    tiledlayout(1,2);
    nexttile;
    mass_1.plot_mascons("Spheres",'on');
    nexttile;
    mass_2.plot_mascons("Spheres",'on');

    %% Calculating potential
    mass_1.calculate(rq);
    mass_2.calculate(rq);
    mass_3.calculate(rq);
    mass_4.calculate(rq);

    mass_1.write_V('1');
    mass_2.write_V('2');
    mass_3.write_V('3');
    mass_4.write_V('4');
```

```
    close all;
    clear;

    model_fname = 'Kleopatra_Ostro.stl';
    cd_splitted = split(mfilename('fullpath'),'\');
    model_path = fullfile(cd_splitted{1:end-2},"model3d",
       model_fname);
    model_temp = stlread(model_path);
    model = triangulation(model_temp.ConnectivityList,model_temp.
       Points);
    clear model_fname cd_splitted model_path model_temp centroid

    trisurf(model,'Facecolor',[.7 .7 .7]);
```

```
    axis equal;
    xlabel('x [km]','Interpreter','latex');
    ylabel('y [km]','Interpreter','latex');
    zlabel('z [km]','Interpreter','latex');
```

```
    clear;
    close all;
    addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
    addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data'

    r_fname = 'rand_kleo.txt';
    r_path = fullfile(pwd,"Input_data","Query_points",r_fname);
    rq = readmatrix(r_path);
    clear r_fname r_path

    unit_scale = 1000;

    model_fname = 'Kleopatra_Ostro.stl';
    cd_splitted = split(mfilename('fullpath'),'\');
    model_path = fullfile(cd_splitted{1:end-2},"model3d",
       model_fname);
    model_temp = stlread(model_path);
    model_temp = triangulation(model_temp.ConnectivityList,
       model_temp.Points * unit_scale);
    centroid = calculate_centroid(model_temp);
    model = triangulation(model_temp.ConnectivityList,model_temp.
       Points - centroid);
    clear model_fname cd_splitted model_path model_temp centroid

    f1 = figure();
    trisurf(model);
    hold on;
    scatter3(rq(:,1),rq(:,2),rq(:,3),'MarkerEdgeColor','k',...
            'MarkerFaceColor',[0 .75 .75]);
    axis equal;
    xlabel(['x [m]'],'Interpreter','latex');
                ylabel(['y [m]'],'Interpreter','latex');
                zlabel(['z [m]'],'Interpreter','latex');
```

```
    clear;
    close all;
    addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
    addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data'

    r_fname = 'rand_kleo.txt';
    r_path = fullfile(pwd,"Input_data","Query_points",r_fname);
    rq = readmatrix(r_path);
```

```
clear r_fname r_path

unit_scale = 1000;

model_fname = 'Kleopatra_Ostro.stl';
cd_splitted = split(mfilename('fullpath'),'\');
model_path = fullfile(cd_splitted{1:end-2},"model3d",
   model_fname);
model_temp = stlread(model_path);
model_temp = triangulation(model_temp.ConnectivityList,
   model_temp.Points * unit_scale);
centroid = calculate_centroid(model_temp);
model = triangulation(model_temp.ConnectivityList,model_temp.
   Points - centroid);
clear model_fname cd_splitted model_path model_temp centroid

poly = polyhedron_obj(model);

bulk_density = 3600;

tic
poly.calculate(rq,bulk_density);
t_poly = toc;

poly.write_V('kleo');
```

```
close all;
clear;


V_fpath_20 = fullfile(pwd,"Input_data","Potential_data","
   poly_V_20.txt");
V_fpath_80 = fullfile(pwd,"Input_data","Potential_data","
   poly_V_80.txt");
V_fpath_320 = fullfile(pwd,"Input_data","Potential_data","
   poly_V_320.txt");
V_fpath_1280 = fullfile(pwd,"Input_data","Potential_data","
   poly_V_1280.txt");

V20 = readmatrix(V_fpath_20);
V80 = readmatrix(V_fpath_80);
V320 = readmatrix(V_fpath_320);
V1280 = readmatrix(V_fpath_1280);
rq = V20(:,1:3);

x_surf = rq(:,1);
y_surf = rq(:,2);
```

```
18      % z_surf = obj.V;
19
20      X_surf = reshape(x_surf,[sqrt(length(x_surf)),sqrt(length(
            x_surf))]);
21      Y_surf = reshape(y_surf,[sqrt(length(x_surf)),sqrt(length(
            x_surf))]);
22      % Z_surf = reshape(z_surf,[sqrt(length(x_surf)),sqrt(length(
            x_surf))]);
23
24      %% Calculating true potential of a unit sphere
25      a = 1; % radius
26      M = 4/3 * pi * a^3; % mass of a sphere
27      G = 6.6743e-11; % gravitational constant
28      for i = 1:length(rq(:,1))
29          r = norm(rq(i,:));
30          if r > 1
31              V(i) = -G*M/r;
32          else
33              V(i) = -G*M* (3*a^2 - r^2)/(2*a^3);
34          end
35      end
36      V_true = reshape(V,[sqrt(length(x_surf)),sqrt(length(x_surf))])
            ;
37      V20 = reshape(V20(:,4),[sqrt(length(x_surf)),sqrt(length(x_surf
            ))]);
38      V80 = reshape(V80(:,4),[sqrt(length(x_surf)),sqrt(length(x_surf
            ))]);
39      V320 = reshape(V320(:,4),[sqrt(length(x_surf)),sqrt(length(
            x_surf))]);
40      V1280 = reshape(V1280(:,4),[sqrt(length(x_surf)),sqrt(length(
            x_surf))]);
41
42      z_scale = 1/10;
43      v_scale = 10^9;
44
45      [x_s,y_s,z_s] = sphere(20);
46      z_s = z_s * z_scale;
47
48      f1 = figure();
49      surf(X_surf,Y_surf,V_true*v_scale,'FaceColor','#1d4877');
50      hold on;
51      surf(X_surf,Y_surf,V20*v_scale,'FaceAlpha',1,'FaceColor','#
            ee3e32');
52      surf(X_surf,Y_surf,V80*v_scale,'FaceAlpha',1,'FaceColor','#
            f68838');
53      surf(X_surf,Y_surf,V320*v_scale,'FaceAlpha',1,'FaceColor','#
            fbb021');
```

```
    surf(X_surf,Y_surf,V1280*v_scale,'FaceAlpha',1,'FaceColor','#1
       b8a5a');
    surf(x_s,y_s,z_s,'FaceColor',"#7E2F8E");
    daspect([1 1 z_scale]);
    f1.Position = [100 100 700 700];
    xlim([0,3]);
    ylim([-3,3]);
    xlabel('x [m]','Interpreter','latex');
    ylabel('y [m]','Interpreter','latex');
    zlabel('V [nJ/kg]','Interpreter','latex');
    legend('true','20','80','320','1280','Location','southoutside',
       'Orientation','horizontal');
    view(-65,20)
```

```
    clear;
    close all;
    addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
    addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data'

    r_fname = 'rand_in.txt';
    r_path = fullfile(pwd,"Input_data","Query_points",r_fname);
    rq_in = readmatrix(r_path);
    clear r_fname r_path

    r_fname = 'rand_out.txt';
    r_path = fullfile(pwd,"Input_data","Query_points",r_fname);
    rq_out = readmatrix(r_path);
    clear r_fname r_path

    V_fpath_20 = fullfile(pwd,"Input_data","Potential_data","
       poly_V_rand_in20.txt");
    V_fpath_80 = fullfile(pwd,"Input_data","Potential_data","
       poly_V_rand_in80.txt");
    V_fpath_320 = fullfile(pwd,"Input_data","Potential_data","
       poly_V_rand_in320.txt");
    V_fpath_1280 = fullfile(pwd,"Input_data","Potential_data","
       poly_V_rand_in1280.txt");

    V20_in = readmatrix(V_fpath_20);
    V20_in = V20_in(:,4);
    V80_in = readmatrix(V_fpath_80);
    V80_in = V80_in(:,4);
    V320_in = readmatrix(V_fpath_320);
    V320_in = V320_in(:,4);
    V1280_in = readmatrix(V_fpath_1280);
    V1280_in = V1280_in(:,4);

```

```
30      V_fpath_20 = fullfile(pwd,"Input_data","Potential_data","
            poly_V_rand_out20.txt");
31      V_fpath_80 = fullfile(pwd,"Input_data","Potential_data","
            poly_V_rand_out80.txt");
32      V_fpath_320 = fullfile(pwd,"Input_data","Potential_data","
            poly_V_rand_out320.txt");
33      V_fpath_1280 = fullfile(pwd,"Input_data","Potential_data","
            poly_V_rand_out1280.txt");
34
35      V20_out = readmatrix(V_fpath_20);
36      V20_out = V20_out(:,4);
37      V80_out = readmatrix(V_fpath_80);
38      V80_out = V80_out(:,4);
39      V320_out = readmatrix(V_fpath_320);
40      V320_out = V320_out(:,4);
41      V1280_out = readmatrix(V_fpath_1280);
42      V1280_out = V1280_out(:,4);
43
44      a = 1; % radius
45      M = 4/3 * pi * a^3; % mass of a sphere
46      G = 6.6743e-11; % gravitational constant
47      for i = 1:length(rq_in(:,1))
48          r = norm(rq_in(i,:));
49          if r > 1
50              V_in(i) = -G*M/r;
51          else
52              V_in(i) = -G*M* (3*a^2 - r^2)/(2*a^3);
53          end
54      end
55      for i = 1:length(rq_out(:,1))
56          r = norm(rq_out(i,:));
57          if r > 1
58              V_out(i) = -G*M/r;
59          else
60              V_out(i) = -G*M* (3*a^2 - r^2)/(2*a^3);
61          end
62      end
63      V_in = V_in';
64      V_out = V_out';
65
66      err_20_in = mean(abs(V20_in-V_in)./abs(V_in));
67      err_80_in = mean(abs(V80_in-V_in)./abs(V_in));
68      err_320_in = mean(abs(V320_in-V_in)./abs(V_in));
69      err_1280_in = mean(abs(V1280_in-V_in)./abs(V_in));
70
71      err_20_out = mean(abs(V20_out-V_out)./abs(V_out));
72      err_80_out = mean(abs(V80_out-V_out)./abs(V_out));
```

```
    err_320_out = mean(abs(V320_out-V_out)./abs(V_out));
    err_1280_out = mean(abs(V1280_out-V_out)./abs(V_out));

    faces = [20, 80, 320, 1280];
    err_in = [err_20_in, err_80_in, err_320_in, err_1280_in];
    err_out = [err_20_out, err_80_out, err_320_out, err_1280_out];

    f1 = figure();
    f1.Position = [100 100 1200 500];
    tiledlayout(1,2);
    nexttile;
    scatter(faces,err_in,'filled','red');
    title('Points inside the sphere');
    ylabel('Mean relative error');
    xlabel('Number of faces');
    grid on;
    nexttile;
    scatter(faces,err_out,'filled','red');
    title('Points outside the sphere');
    ylabel('Mean relative error');
    xlabel('Number of faces');
    grid on;

    t_in = [0.97, 3.63, 14.41, 57.71];
    t_out = [1.20, 3.81, 15.22, 58.46];

    f2 = figure();
    f2.Position = [100 100 1200 500];
    tiledlayout(1,2);
    nexttile;
    scatter(faces,t_in,'filled','red');
    title('Points inside the sphere');
    ylabel('Time of calculation [s]');
    xlabel('Number of faces');
    grid on;
    nexttile;
    scatter(faces,t_out,'filled','red');
    title('Points outside the sphere');
    ylabel('Time of calculation [s]');
    xlabel('Number of faces');
    grid on;
```

```
    clear;
    close all;
    addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
    addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data'

```

```
r_fname = 'rand_in.txt';
r_path = fullfile(pwd,"Input_data","Query_points",r_fname);
rq = readmatrix(r_path);
clear r_fname r_path


%% Reading model
model_fname = 'sphere20.stl';
cd_splitted = split(mfilename('fullpath'),'\');
model_path = fullfile(cd_splitted{1:end-2},"model3d",
   model_fname);
model_temp = stlread(model_path);
model20 = triangulation(model_temp.ConnectivityList,model_temp.
   Points);
clear model_fname cd_splitted model_path model_temp centroid

model_fname = 'sphere80.stl';
cd_splitted = split(mfilename('fullpath'),'\');
model_path = fullfile(cd_splitted{1:end-2},"model3d",
   model_fname);
model_temp = stlread(model_path);
model80 = triangulation(model_temp.ConnectivityList,model_temp.
   Points);
clear model_fname cd_splitted model_path model_temp centroid

model_fname = 'sphere320.stl';
cd_splitted = split(mfilename('fullpath'),'\');
model_path = fullfile(cd_splitted{1:end-2},"model3d",
   model_fname);
model_temp = stlread(model_path);
model320 = triangulation(model_temp.ConnectivityList,model_temp
   .Points);
clear model_fname cd_splitted model_path model_temp centroid

model_fname = 'sphere1280.stl';
cd_splitted = split(mfilename('fullpath'),'\');
model_path = fullfile(cd_splitted{1:end-2},"model3d",
   model_fname);
model_temp = stlread(model_path);
model1280 = triangulation(model_temp.ConnectivityList,
   model_temp.Points);
clear model_fname cd_splitted model_path model_temp centroid

%% Creating objects
poly20 = polyhedron_obj(model20);
poly80 = polyhedron_obj(model80);
poly320 = polyhedron_obj(model320);
```

```
    poly1280 = polyhedron_obj(model1280);

    %% Calculating potential
    bulk_density = 1;

    tic
    poly20.calculate(rq,bulk_density);
    t20 = toc
    tic
    poly80.calculate(rq,bulk_density);
    t80 = toc
    tic
    poly320.calculate(rq,bulk_density);
    t320 = toc
    tic
    poly1280.calculate(rq,bulk_density);
    t1280 = toc

    poly20.write_V('rand_in20');
    poly80.write_V('rand_in80');
    poly320.write_V('rand_in320');
    poly1280.write_V('rand_in1280');




    % f1 = figure();
    % scatter3(rq(:,1),rq(:,2),rq(:,3),'filled','red');
    % hold on;
    % [x_s,y_s,z_s] = sphere;
    % surf(x_s,y_s,z_s,'FaceAlpha',0.2);
    % xlim([a,b]);
    % ylim([a,b]);
    % zlim([a,b]);
    % daspect([1 1 1]);
```

```
    clear;
    % close all;
    addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
    addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data'

    %% Reading model
    model_fname = 'sphere20.stl';
    cd_splitted = split(mfilename('fullpath'),'\');
    model_path = fullfile(cd_splitted{1:end-2},"model3d",
       model_fname);
    model_temp = stlread(model_path);
```

```
    model20 = triangulation(model_temp.ConnectivityList,model_temp.
        Points);
    clear model_fname cd_splitted model_path model_temp centroid

    model_fname = 'sphere80.stl';
    cd_splitted = split(mfilename('fullpath'),'\');
    model_path = fullfile(cd_splitted{1:end-2},"model3d",
        model_fname);
    model_temp = stlread(model_path);
    model80 = triangulation(model_temp.ConnectivityList,model_temp.
        Points);
    clear model_fname cd_splitted model_path model_temp centroid

    model_fname = 'sphere320.stl';
    cd_splitted = split(mfilename('fullpath'),'\');
    model_path = fullfile(cd_splitted{1:end-2},"model3d",
        model_fname);
    model_temp = stlread(model_path);
    model320 = triangulation(model_temp.ConnectivityList,model_temp
        .Points);
    clear model_fname cd_splitted model_path model_temp centroid

    model_fname = 'sphere1280.stl';
    cd_splitted = split(mfilename('fullpath'),'\');
    model_path = fullfile(cd_splitted{1:end-2},"model3d",
        model_fname);
    model_temp = stlread(model_path);
    model1280 = triangulation(model_temp.ConnectivityList,
        model_temp.Points);
    clear model_fname cd_splitted model_path model_temp centroid

    %% Creating objects
    poly20 = polyhedron_obj(model20);
    poly80 = polyhedron_obj(model80);
    poly320 = polyhedron_obj(model320);
    poly1280 = polyhedron_obj(model1280);

    %% Creating query points
    x = linspace(-3,3,30); % m
    y = x;
    [X,Y] = meshgrid(x);
    rq = [X(:),Y(:),zeros(length(X(:)),1)];

    %% Calculating potential
    bulk_density = 1;

    poly20.calculate(rq,bulk_density);
```

```
    poly80.calculate(rq,bulk_density);
    poly320.calculate(rq,bulk_density);
    poly1280.calculate(rq,bulk_density);

    poly20.write_V('20');
    poly80.write_V('80');
    poly320.write_V('320');
    poly1280.write_V('1280');


    %
    % % V_scale = 4*10^9;
    % V_scale = 1;
    %
    % x = linspace(-3,3,20); % m
    % y = x;
    % [X,Y] = meshgrid(x);
    % rq = [X(:),Y(:),zeros(length(X(:)),1)];
    %
    % polyhedron_method = polyhedron_obj(model);
    % polyhedron_method.calculate(rq,1);
    % polyhedron_method.plot_V(rq,V_scale,'Units','m');
    %
    %
    % a = 1;
    % M = 4/3 * pi * a^3;
    % G = 6.6743e-11;
    % for i = 1:length(rq(:,1))
    %     r = norm(rq(i,:));
    %     if r > 1
    %         V(i) = -G*M/r;
    %     else
    %         V(i) = -G*M* (3*a^2 - r^2)/(2*a^3);
    %     end
    % end
    % V_sar = reshape(V,[length(x),length(x)]) * V_scale;
    % hold on;
    % surf(X,Y,V_sar,'FaceAlpha',0.5);
```

```
    close all;
    clear;

    %% Initial data (TODO in json file)
    % Asteroid data
    model_fname = "Kleopatra_Ostro.stl";
    bulk_density = 3600; % kg/m^3

```

```
    % Field query points
    x = linspace(-300000,300000,30); % m
    y = x;
    [X,Y] = meshgrid(x);
    r = [X(:),Y(:),zeros(length(X(:)),1)];

    % Polyhedron method configuration
    unit_scale = 1000; % for models provided not in meters

    % Ploting configuration
    plot_unit = 'km'; % units of plots (km or m)
    V_scale = 10^2; % scale of the potential plot

    %% Main section
    % Creating mascons model
    polyhedron_method = polyhedron_obj(model_fname,unit_scale);
    % Calculating gravitational potential
    polyhedron_method.calculate(r,bulk_density);
    % Plotting results
    polyhedron_method.plot_V(r,V_scale,'Units',plot_unit);
```

```
    clear;
    close all;

    unit_scale = 1000;

    model_fname = 'Kleopatra_Ostro.stl';
    cd_splitted = split(mfilename('fullpath'),'\');
    model_path = fullfile(cd_splitted{1:end-2},"model3d",
       model_fname);
    model_temp = stlread(model_path);
    model_temp = triangulation(model_temp.ConnectivityList,
       model_temp.Points * unit_scale);
    centroid = calculate_centroid(model_temp);
    model = triangulation(model_temp.ConnectivityList,model_temp.
       Points - centroid);
    clear model_fname cd_splitted model_path model_temp centroid

    model_new.vertices = model.Points;
    model_new.faces = model.ConnectivityList;
    for i = 1:length(model.Points(:,1))
        dist(i) = norm(model.Points(i,:));
    end
    dist_max = max(dist);
```

```
    clear;
    close all;
```

```
addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data\Coefs'

r_fname = 'rand_kleo.txt';
r_path = fullfile(pwd,"Input_data","Query_points",r_fname);
rq = readmatrix(r_path);
clear r_fname r_path

unit_scale = 1000;

model_fname = 'Kleopatra_Ostro.stl';
cd_splitted = split(mfilename('fullpath'),'\');
model_path = fullfile(cd_splitted{1:end-2},"model3d",model_fname);
model_temp = stlread(model_path);
model_temp = triangulation(model_temp.ConnectivityList,model_temp.Points * unit_scale);
centroid = calculate_centroid(model_temp);
model = triangulation(model_temp.ConnectivityList,model_temp.Points - centroid);
clear model_fname cd_splitted model_path model_temp centroid

n = [4, 10, 50];
bulk_density = 3600;
a = 1.144612198106038e+05;

sh1 = spherical_harmonics_obj(model,bulk_density,n(1),a);
sh2 = spherical_harmonics_obj(model,bulk_density,n(2),a);
sh3 = spherical_harmonics_obj(model,bulk_density,n(3),a);

sh1.read_coefs_werner();
sh2.read_coefs_werner();
sh3.read_coefs_werner();

tic
sh1.calculate(sh1.C_nm,sh1.S_nm,rq);
t10 = toc
tic
sh2.calculate(sh2.C_nm,sh2.S_nm,rq);
t30 = toc
tic
sh3.calculate(sh3.C_nm,sh3.S_nm,rq);
t50 = toc

sh1.write_V('kleo_coef_4');
sh2.write_V('kleo_coef_10');
```

```
46      sh3.write_V('kleo_coef_50');
```

```
1       clear;
2       close all;
3       addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
4       addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data'
5       addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data\
          Potential_data\'
6
7       r_fname = 'rand_out.txt';
8       r_path = fullfile(pwd,"Input_data","Query_points",r_fname);
9       rq_out = readmatrix(r_path);
10      clear r_fname r_path
11
12
13      V_fpath_20 = fullfile(pwd,"Input_data","Potential_data","
          sh_V_rand_mesh_20.txt");
14      V_fpath_80 = fullfile(pwd,"Input_data","Potential_data","
          sh_V_rand_mesh_80.txt");
15      V_fpath_320 = fullfile(pwd,"Input_data","Potential_data","
          sh_V_rand_mesh_320.txt");
16      V_fpath_1280 = fullfile(pwd,"Input_data","Potential_data","
          sh_V_rand_mesh_1280.txt");
17
18      V20 = readmatrix(V_fpath_20);
19      V20 = -V20(:,4);
20      V80 = readmatrix(V_fpath_80);
21      V80 = -V80(:,4);
22      V320 = readmatrix(V_fpath_320);
23      V320 = -V320(:,4);
24      V1280 = readmatrix(V_fpath_1280);
25      V1280 = -V1280(:,4);
26
27      a = 1; % radius
28      M = 4/3 * pi * a^3; % mass of a sphere
29      G = 6.6743e-11; % gravitational constant
30      for i = 1:length(rq_out(:,1))
31          r = norm(rq_out(i,:));
32          if r > 1
33              V_out(i) = -G*M/r;
34          else
35              V_out(i) = -G*M* (3*a^2 - r^2)/(2*a^3);
36          end
37      end
38      V_out = V_out';
39
40      err_20 = mean(abs(V20-V_out)./abs(V_out));
```

```
    err_80 = mean(abs(V80-V_out)./abs(V_out));
    err_320 = mean(abs(V320-V_out)./abs(V_out));
    err_1280 = mean(abs(V1280-V_out)./abs(V_out));

    faces = [20, 80, 320, 1280];
    err_out = [err_20, err_80, err_320, err_1280];

    f1 = figure();
    f1.Position = [100 100 1000 500];
    scatter(faces,err_out,'filled','red');
    title('Points outside the sphere');
    ylabel('Mean relative error');
    xlabel('Number of faces');
    grid on;

    t_out = [0.0357, 0.0344, 0.0347, 0.0349];

    f2 = figure();
    f2.Position = [100 100 1000 500];
    scatter(faces,t_out,'filled','red');
    title('Points outside the sphere');
    ylabel('Time of calculation [s]');
    xlabel('Number of faces');
    grid on;
```

```
    clear;
    close all;
    addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
    addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data'

    r_fname = 'rand_out.txt';
    r_path = fullfile(pwd,"Input_data","Query_points",r_fname);
    rq_out = readmatrix(r_path);
    clear r_fname r_path


    V_fpath_10 = fullfile(pwd,"Input_data","Potential_data","
       sh_V_rand_coef_1.txt");
    V_fpath_20 = fullfile(pwd,"Input_data","Potential_data","
       sh_V_rand_coef_2.txt");
    V_fpath_30 = fullfile(pwd,"Input_data","Potential_data","
       sh_V_rand_coef_3.txt");
    V_fpath_40 = fullfile(pwd,"Input_data","Potential_data","
       sh_V_rand_coef_4.txt");

    V10 = readmatrix(V_fpath_10);
    V10 = -V10(:,4);
```

```
    V20 = readmatrix(V_fpath_20);
    V20 = -V20(:,4);
    V30 = readmatrix(V_fpath_30);
    V30 = -V30(:,4);
    V40 = readmatrix(V_fpath_40);
    V40 = -V40(:,4);

    a = 1; % radius
    M = 4/3 * pi * a^3; % mass of a sphere
    G = 6.6743e-11; % gravitational constant
    for i = 1:length(rq_out(:,1))
        r = norm(rq_out(i,:));
        if r > 1
            V_out(i) = -G*M/r;
        else
            V_out(i) = -G*M* (3*a^2 - r^2)/(2*a^3);
        end
    end
    V_out = V_out';

    err_10 = mean(abs(V10-V_out)./abs(V_out));
    err_20 = mean(abs(V20-V_out)./abs(V_out));
    err_30 = mean(abs(V30-V_out)./abs(V_out));
    err_40 = mean(abs(V40-V_out)./abs(V_out));

    faces = [2, 6, 12, 20];
    err_out = [err_10, err_20, err_30, err_40];

    f1 = figure();
    f1.Position = [100 100 1000 500];
    scatter(faces,err_out,'filled','red');
    title('Points outside the sphere');
    ylabel('Mean relative error');
    xlabel('Degree');
    grid on;

    t_out = [0.0043, 0.016, 0.0455, 0.1096];

    f2 = figure();
    f2.Position = [100 100 1000 500];
    scatter(faces,t_out,'filled','red');
    title('Points outside the sphere');
    ylabel('Time of calculation [s]');
    xlabel('Degree');
    grid on;

    rqn = vecnorm(rq_out,2,2);
```

```
    err_sh = abs(V40-V_out)./abs(V_out);
    f3 = figure();
    scatter(rqn,err_sh);
    grid on;
```

```
    clear;
    close all;
    addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
    addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data'

    r_fname = 'rand_out.txt';
    r_path = fullfile(pwd,"Input_data","Query_points",r_fname);
    rq = readmatrix(r_path);
    clear r_fname r_path


    %% Reading model


    model_fname = 'sphere320.stl';
    cd_splitted = split(mfilename('fullpath'),'\');
    model_path = fullfile(cd_splitted{1:end-2},"model3d",
       model_fname);
    model_temp = stlread(model_path);
    model = triangulation(model_temp.ConnectivityList,model_temp.
       Points);
    clear model_fname cd_splitted model_path model_temp centroid

    n = [2, 6, 12, 20];
    bulk_density = 1;
    a = 1;
    sh1 = spherical_harmonics_obj(model,bulk_density,n(1),a);
    sh1.calculate_coefs();
    sh2 = spherical_harmonics_obj(model,bulk_density,n(2),a);
    sh2.calculate_coefs();
    sh3 = spherical_harmonics_obj(model,bulk_density,n(3),a);
    sh3.calculate_coefs();
    sh4 = spherical_harmonics_obj(model,bulk_density,n(4),a);
    sh4.calculate_coefs();

    %% Calculate V
    tic
    sh1.calculate(sh1.C_nm,sh1.S_nm,rq);
    t10 = toc
    tic
    sh2.calculate(sh2.C_nm,sh2.S_nm,rq);
    t20 = toc
```

```
    tic
    sh3.calculate(sh3.C_nm,sh3.S_nm,rq);
    t30 = toc
    tic
    sh4.calculate(sh4.C_nm,sh4.S_nm,rq);
    t40 = toc

    sh1.write_V('rand_coef_1');
    sh2.write_V('rand_coef_2');
    sh3.write_V('rand_coef_3');
    sh4.write_V('rand_coef_4');
```

```
    clear;
    close all;
    addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
    addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data'

    r_fname = 'rand_out.txt';
    r_path = fullfile(pwd,"Input_data","Query_points",r_fname);
    rq = readmatrix(r_path);
    clear r_fname r_path


    %% Reading model
    model_fname = 'sphere20.stl';
    cd_splitted = split(mfilename('fullpath'),'\');
    model_path = fullfile(cd_splitted{1:end-2},"model3d",
       model_fname);
    model_temp = stlread(model_path);
    model20 = triangulation(model_temp.ConnectivityList,model_temp.
       Points);
    clear model_fname cd_splitted model_path model_temp centroid

    model_fname = 'sphere80.stl';
    cd_splitted = split(mfilename('fullpath'),'\');
    model_path = fullfile(cd_splitted{1:end-2},"model3d",
       model_fname);
    model_temp = stlread(model_path);
    model80 = triangulation(model_temp.ConnectivityList,model_temp.
       Points);
    clear model_fname cd_splitted model_path model_temp centroid

    model_fname = 'sphere320.stl';
    cd_splitted = split(mfilename('fullpath'),'\');
    model_path = fullfile(cd_splitted{1:end-2},"model3d",
       model_fname);
    model_temp = stlread(model_path);
```

```
31      model320 = triangulation(model_temp.ConnectivityList,model_temp
           .Points);
32      clear model_fname cd_splitted model_path model_temp centroid
33
34      model_fname = 'sphere1280.stl';
35      cd_splitted = split(mfilename('fullpath'),'\');
36      model_path = fullfile(cd_splitted{1:end-2},"model3d",
           model_fname);
37      model_temp = stlread(model_path);
38      model1280 = triangulation(model_temp.ConnectivityList,
           model_temp.Points);
39      clear model_fname cd_splitted model_path model_temp centroid
40
41      %% Creating objects
42
43      n = 10;
44      bulk_density = 1;
45      a = 1;
46      sh20 = spherical_harmonics_obj(model20,bulk_density,n,a);
47      sh20.calculate_coefs();
48      sh80 = spherical_harmonics_obj(model80,bulk_density,n,a);
49      sh80.calculate_coefs();
50      sh320 = spherical_harmonics_obj(model320,bulk_density,n,a);
51      sh320.calculate_coefs();
52      sh1280 = spherical_harmonics_obj(model1280,bulk_density,n,a);
53      sh1280.calculate_coefs();
54
55
56      %% Calculating potential
57
58      tic
59      sh20.calculate(sh20.C_nm,sh20.S_nm,rq);
60      t20 = toc
61      tic
62      sh80.calculate(sh80.C_nm,sh80.S_nm,rq);
63      t80 = toc
64      tic
65      sh320.calculate(sh320.C_nm,sh320.S_nm,rq);
66      t320 = toc
67      tic
68      sh1280.calculate(sh1280.C_nm,sh1280.S_nm,rq);
69      t1280 = toc
70
71      sh20.write_V('rand_mesh_20');
72      sh80.write_V('rand_mesh_80');
73      sh320.write_V('rand_mesh_320');
74      sh1280.write_V('rand_mesh_1280');
```

```
75
76
77
78
79      % f1 = figure();
80      % scatter3(rq(:,1),rq(:,2),rq(:,3),'filled','red');
81      % hold on;
82      % [x_s,y_s,z_s] = sphere;
83      % surf(x_s,y_s,z_s,'FaceAlpha',0.2);
84      % xlim([a,b]);
85      % ylim([a,b]);
86      % zlim([a,b]);
87      % daspect([1 1 1]);
```

```
1       clear;
2       % close all;
3       addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
4       addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data'
5
6       %% Reading model
7       model_fname = 'sphere20.stl';
8       cd_splitted = split(mfilename('fullpath'),'\');
9       model_path = fullfile(cd_splitted{1:end-2},"model3d",
           model_fname);
10      model_temp = stlread(model_path);
11      model20 = triangulation(model_temp.ConnectivityList,model_temp.
           Points);
12      clear model_fname cd_splitted model_path model_temp centroid
13
14      model_fname = 'sphere80.stl';
15      cd_splitted = split(mfilename('fullpath'),'\');
16      model_path = fullfile(cd_splitted{1:end-2},"model3d",
           model_fname);
17      model_temp = stlread(model_path);
18      model80 = triangulation(model_temp.ConnectivityList,model_temp.
           Points);
19      clear model_fname cd_splitted model_path model_temp centroid
20
21      model_fname = 'sphere320.stl';
22      cd_splitted = split(mfilename('fullpath'),'\');
23      model_path = fullfile(cd_splitted{1:end-2},"model3d",
           model_fname);
24      model_temp = stlread(model_path);
25      model320 = triangulation(model_temp.ConnectivityList,model_temp
           .Points);
26      clear model_fname cd_splitted model_path model_temp centroid
27
```

```
    model_fname = 'sphere1280.stl';
    cd_splitted = split(mfilename('fullpath'),'\');
    model_path = fullfile(cd_splitted{1:end-2},"model3d",
       model_fname);
    model_temp = stlread(model_path);
    model1280 = triangulation(model_temp.ConnectivityList,
       model_temp.Points);
    clear model_fname cd_splitted model_path model_temp centroid

    %% Creating query points
    x = linspace(-3,3,30); % m
    y = x;
    [X,Y] = meshgrid(x);
    rq = [X(:),Y(:),zeros(length(X(:)),1)];

    %% Creating objects

    n = 10;
    bulk_density = 1;
    a = 1;

    sh_method_20 = spherical_harmonics_obj(model20,bulk_density,n,a
       );
    sh_method_20.calculate_coefs();
    sh_method_20.calculate(sh_method.C_nm,sh_method.S_nm,rq);
    sh_method_20.plot_V(rq,V_scale,'Units','m');


    M = 4/3 * pi * a^3;
    G = 6.6743e-11;
    for i = 1:length(rq(:,1))
        r = norm(rq(i,:));
        if r > 1
            V(i) = -G*M/r;
        else
            V(i) = -G*M* (3*a^2 - r^2)/(2*a^3);
        end
    end
    V_sar = reshape(V,[length(x),length(x)]) * V_scale;
    hold on;
    surf(X,Y,V_sar,'FaceAlpha',0.5);
```

```
    clear;
    close all;
    addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
    addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data'

```

```
    r_fname = 'rand_out.txt';
    r_path = fullfile(pwd,"Input_data","Query_points",r_fname);
    rq_out = readmatrix(r_path);
    clear r_fname r_path

    V_fpath_40 = fullfile(pwd,"Input_data","Potential_data","
       sh_V_rand_coef_4.txt");
    V_fpath_4 = fullfile(pwd,"Input_data","Potential_data","
       mass_V_rand_out4.txt");

    V4_out = readmatrix(V_fpath_4);
    V4_out = V4_out(:,4);


    V40 = readmatrix(V_fpath_40);
    V40 = -V40(:,4);
    a = 1; % radius
    M = 4/3 * pi * a^3; % mass of a sphere
    G = 6.6743e-11; % gravitational constant
    for i = 1:length(rq_out(:,1))
        r = norm(rq_out(i,:));
        if r > 1
            V_out(i) = -G*M/r;
        else
            V_out(i) = -G*M* (3*a^2 - r^2)/(2*a^3);
        end
    end
    V_out = V_out';



    err_sh = abs(V40-V_out)./abs(V_out);

    rqn = vecnorm(rq_out,2,2);
    err_mass = abs(V4_out-V_out)./abs(V_out);
    f3 = figure();
    scatter(rqn,err_mass);
    hold on;
    scatter(rqn,err_sh);
    grid on;
    legend('Point masses','Spherical Harmonics');
```

```
    close all;
    clear;
    addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
    addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data'

```

```
%% Initial data (TODO in json file)
% Asteroid data
model_fname = "sphere20.stl";
bulk_density = 1; % kg/m^3

% Field query points
x = linspace(-300000,300000,30); % m
y = x;
[X,Y] = meshgrid(x);
r = [X(:),Y(:),-6e+4 * ones(length(X(:)),1)];

% Spherical_harmonics method configuration
unit_scale = 1000; % for models provided not in meters

% Ploting configuration
plot_unit = 'm'; % units of plots (km or m)
V_scale = 10^10; % scale of the potential plot

%% Main section
% Creating mascons model
spherical_harmonics_method = spherical_harmonics_obj(
   model_fname,unit_scale,bulk_density,3,1.144612198106038e+05)
   ;
spherical_harmonics_method.calculate_coefs(bulk_density,
   centroid);

% Point masses method
% mas_r = 5000;
% mascons_method = mascons_obj(model_fname,mas_r,unit_scale,
   bulk_density);
% spherical_harmonics_method.calculate_coefs_point_masses(
   centroid,mascons_method.points,mascons_method.M_i);

spherical_harmonics_method.calculate(spherical_harmonics_method
   .C_nm,spherical_harmonics_method.S_nm,r,centroid);

spherical_harmonics_method.plot_V(r,V_scale,'Units',plot_unit);
```

```
close all; clear;

tri1 = make_trinomial(1);
tri2 = make_trinomial(2);

tri3 = tri_mult(tri1, tri2);

integration = integrate_simplex(tri3,2);

```

```
10      function tri = make_trinomial(n)
11          t = (n+2)*(n+1)*0.5;
12          tri = zeros(1,t);
13          for i = n:-1:0
14              for k = 0:(n-i)
15                  j = n-i-k;
16                  index = (j+k)*(j+k+1)/2 + k + 1;
17                  coef = factorial(n)/(factorial(i)*factorial(j)*
                        factorial(k));
18                  tri(index) = coef;
19              end
20          end
21      end
22
23      function tri_result = tri_mult(tri1, tri2)
24          n1 = (-3+sqrt(9-(8*(1-length(tri1)))))/2;
25          n2 = (-3+sqrt(9-(8*(1-length(tri2)))))/2;
26          n = n1 + n2;
27          t_result = (n+2)*(n+1)*0.5;
28          tri_result = zeros(1,t_result);
29          for i = n1:-1:0
30              for k = 0:(n1-i)
31                  j = n1-i-k;
32                  for r = n2:-1:0
33                      for s = 0:(n2-r)
34                          t = n2-r-s;
35                          index = (j+s+k+t)*(j+s+k+t+1)/2 + k + t +
                                1;
36                          index1 = (j+k)*(j+k+1)/2 + k + 1;
37                          index2 = (s+t)*(s+t+1)/2 + t + 1;
38                          tri_result(index) = tri_result(index) +
                                tri1(index1) * tri2(index2);
39                      end
40                  end
41              end
42          end
43      end
44
45      function int_result = integrate_simplex(tri,det_J)
46          n = (-3+sqrt(9-(8*(1-length(tri)))))/2;
47          for i = n:-1:0
48              for k = 0:(n-i)
49                  j = n-i-k;
50                  index = (j+k)*(j+k+1)/2 + k + 1;
51                  mix_factor = factorial(i)*factorial(j)*factorial(k)
                        /factorial(n+3);
52                  tri(index) = tri(index)*mix_factor;
```

```
            end
        end
        int_result = det_J * sum(tri);
    end
```

```
    close all;
    clear;

    C50 = readmatrix('C_werner_50.txt');
    S50 = readmatrix('S_werner_50.txt');

    C4 = C50(1:5,1:5);
    S4 = S50(1:5,1:5);

    C10 = C50(1:11,1:11);
    S10 = S50(1:11,1:11);

    writematrix(C4,'C_werner_4.txt');
    writematrix(S4,'S_werner_4.txt');
    writematrix(C10,'C_werner_10.txt');
    writematrix(S10,'S_werner_10.txt');
```

```
    clear;
    close all;
    addpath 'C:\Projects\personal\przejsciowka\matlab\Functions'
    addpath 'C:\Projects\personal\przejsciowka\matlab\Input_data'

    a = -200000;
    b = 200000;
    n = 100;

    [rq, rqn, i] = gen_points(a,b,n,"out",1.144612198106038e+05);

    writematrix(rq,"rand_kleo.txt");
```

```
    clear;
    close all;

    x = linspace(-300000,300000,30); % m
    y = x;
    [X,Y] = meshgrid(x);
    r = [X(:),Y(:),-6e+4 * ones(length(X(:)),1)];

    writematrix(r,"test.txt");
```

Bibliografia

[1] NASA/JPL/JHUAPL. *Zdjęcie asteroidy (433) Eros*. 2000. URL: https://commons.wikimedia.org/wiki/File:Eros_-_PIA02923_(color).jpg.

[2] VSO Very Large Telescope SPHERE/ZIMPOL team. *Zdjęcie asteroidy (216) Kleopatra*. 2021. URL: https://pl.wikipedia.org/wiki/Plik:216_Kleopatra_VLT_(2021),_deconvolved.pdf.

[3] NASA/JPL. *Model asteroidy (216) Kleopatra*. 2021. URL: https://3d-asteroids.space/asteroids/216-Kleopatra.

[4] Robert Werner i D. Scheeres. "Exterior Gravitation of a Polyhedron Derived and Compared with Harmonic and Mascon Gravitation Representations of Asteroid 4769 Castalia". *Celestial Mechanics and Dynamical Astronomy* 65 (wrz. 1996), s. 313–344. DOI: 10.1007/BF00053511.

[5] Robert A Werner. "Spherical harmonic coefficients for the potential of a constant-density polyhedron". *Computers & Geosciences* 23.10 (1997), s. 1071–1077.

[6] Patrick T Wittick i Ryan P Russell. "Mascon models for small body gravity fields". 162 (2017), s. 17–162.

[7] Georgia Gavriilidou i Dimitrios Tsoulis. "Evaluation Procedures for the Potential Harmonic Coefficients of a Generally Shaped Polyhedron". *Surveys in Geophysics* 45.2 (2024), s. 315–348.

[8] Arunkumar Rathinam i Andrew G Dempster. "Octree-based mascon model for small body gravity fields". *Journal of Guidance, Control, and Dynamics* 42.11 (2019), s. 2557–2567.

[9] Wikipedia contributors. *216 Kleopatra — Wikipedia, The Free Encyclopedia*. [Online; accessed 12-September-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=216_Kleopatra&oldid=1231129526.

[10] Wm. Robert Johnston. *(216) Kleopatra, Alexhelios, and Cleoselene*. 2014. URL: https://www.johnstonsarchive.net/astro/astmoons/am-00216.html.