МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

(МАИ)

Кафедра «Системный анализ и управление»

ОТЧЕТ

По курсовой работе

По курсу «Динамическое проектирование систем управления ЛА»

Задание 2. Исследование работы системы ориентации и стабилизации КА на этапе переориентации с использованием двигателей-маховиков

Выполнил:

Бельянинов А.Д.

Группа М6О-407Б-19

Работу принял:

Розин П.Е.

Москва 2022

Содержание

[Исходные данные 3](#_Toc121930457)

[Цели работы 3](#_Toc121930458)

[Постановка задачи 3](#_Toc121930459)

[Алгоритм решения 8](#_Toc121930460)

[Выводы 12](#_Toc121930461)

[Листинг 13](#_Toc121930462)

[Main.py 13](#_Toc121930463)

[Math\_def.py 21](#_Toc121930464)

[constants.py 23](#_Toc121930465)

[Start\_data.py 24](#_Toc121930466)

# **Исходные данные**

Задание 2. Вариант 1:

рад/с,

,

,

,

,

Орбита №1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вариант | эпоха, ДМВ |  |  |  |
| 1 | 01.07.2022 00:00:00.000 | 5666.282392 | 3512.092276 | -1780.014521 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 2.194685 | 0.146478 | 7.275306 |

# **Цели работы**

Научится моделировать управляемое вращательное движение КА, оснащённого ДМ, и закрепить понимания работы системы управления угловым движением КА с двигателями-маховиками (ДМ).

# **Постановка задачи**

Рассмотрим задачу переориентации с последующей стабилизацией КА в инерциальном пространстве при использовании в качестве управляющих органов ДМ и в качестве измерительного прибора – датчика угловых скоростей.

Будем полагать, что датчик угловых скоростей перед началом работы прошел калибровку систематической составляющей дрейфа нулевого сигнала, масштабных коэффициентов и не ортогональности осей. Т.е. основной составляющей ошибок ориентации и стабилизации является шумовая составляющая измерений. Будем полагать, что шум измерений является некоррелированным белым шумом, который моделируется как нормальный закон распределения случайной величины с заданной дисперсией и нулевым математическим ожиданием.

**Методика решения**

Векторное дифференциальное уравнение динамики Эйлера для КА как твердого тела в форме Коши выглядит в виде (1.1).

|  |  |
| --- | --- |
|  | (1.1) |

Управление вращательным движением КА будет осуществляться за счет изменения вектора кинетического момента системы ДМ , которое будет возникать за счет подачи управляющего напряжения каждого из приборов. Итоговая динамическая система вращающегося КА, оснащенного ДМ, для случая идеального ДМ (отработка сигнала управления без запаздывания, перерегулирования и т.п.) будет выглядеть как соотношения (1.2).

|  |  |
| --- | --- |
|  | (1.2) |

где − вектор угловой скорости вращения КА вокруг центра масс; 𝐽 – тензор инерции КА; − вектор суммарного момента сил, действующих на КА; –вектор угловых скоростей вращения ДМ; диагональная матрица, содержащая значения величин моментов инерции роторов ДМ, установленных по соответствующей оси ССК КА. Соответственно, для управления вращательным движением КА требуется формировать на каждый из ДМ вектор управляющих сигналов , прямо пропорциональный .

Уравнения (1.1) и (1.2) описывают динамику вращательного движения КА. То есть их решение позволяет определить вектор угловой̆ скорости относительно инерциального пространства. В тоже время, угловое положение ССК КА относительно известного вектора угловой̆ скорости, описывается нелинейными соотношениями. Наиболее широкое распространение получили кинематические уравнения в кватернионной форме (уравнения Пуассона) или представленные в виде матрицы направляющих косинусов. В данном варианте рассматриваются кинематические соотношения в виде уравнений Пуассона (1.3).

|  |  |
| --- | --- |
|  | (1.3) |

где Λ – кватернион ориентации КА в пространстве, ⊗ - оператор кватернионного перемножения (см. приложение Г).

Геометрический̆ смысл кватерниона можно определить так:

Где 𝜑 – угол поворота вокруг орта 𝑒̅ =. То есть кватернион характеризует орт оси пространственного поворота и угол, на который нужно вокруг этой оси повернуть КА, чтобы оси его связанной системы координат заняли определенное положение относительно осей инерциальной системы координат.

При интегрировании уравнения (1.3) после каждого шага интегрирования осуществлять нормировку получаемого кватерниона, то есть обеспечивать выполнение равенства:

Комбинация уравнений кинематики и динамики полностью описывают вращательное движение КА как твердого тела.

Рассмотрим модель орбитального движения спутника, которая получается в результате интегрирования дифференциальных уравнений движения с учетом следующих внешних возмущений:

* несферичности гравитационного поля Земли (учитывая влияние только второй зональной гармоники),
* гравитационного воздействия Солнца.

В геоцентрической инерциальной̆ системе координат J2000 уравнении

возмущенного орбитального движения можно записать в виде (1.4).

|  |  |
| --- | --- |
|  | (1.4) |

радиус-вектор КА в геоцентрической инерциальной системе координат J2000; = 398600.4415 − гравитационный параметр Земли; − ускорение за счёт влияния несферичности гравитационного поля; − ускорение, приобретаемое за счёт гравитационного влияния Солнца.

Ускорение за счёт влияния несферичности гравитационного поля можно представить в виде (1.5).

|  |  |
| --- | --- |
|  | (1.5) |

где - матрица перехода от J2000 в гринвичскую СК, которая определяется из соотношения (1.6); - возмущающее ускорение в гринвичской вращающейся системе координат определяется из соотношения (1.7).

|  |  |
| --- | --- |
|  | (1.6) |
|  | (1.7) |

Гравитационное ускорение, обусловленное влиянием Солнца, будет определяться из соотношения (1.8).

|  |  |
| --- | --- |
|  | (1.8) |

где: радиус-вектор Солнца, радиус вектор КА, расстояние между КА и Солнцем [км]. - гравитационный параметр Солнца= 132712517951.

Выражение для гравитационного момента может быть представлено в следующем виде:

|  |  |
| --- | --- |
|  | (1.9) |

Где,

𝜂 – единичный̆ вектор от центра масс Земли до центра масс КА, в проекции на оси ССК КА

𝐽 – тензор инерции КА.

𝜇 – гравитационный̆ параметр Земли, 𝜇 = 3,986 × 1014 [м3/с2].

𝑅 – расстояние от центра масс Земли до центра масс КА (м).

Для вычисления проекции вектора 𝜂 на оси ССК КА достаточно выполнить следующие преобразования:

, где

Λ𝑡 – текущий кватернион ориентации КА в момент времени t.

– нормированный̆ вектор из центра Земли в центре масс КА в инерциальной системе координат в момент времени t.

# **Алгоритм решения**

Для решения задачи требуется реализовать программное обеспечение позволяющее моделировать следующую последовательность действий:

1. Осуществляется интегрирование уравнений (2) и (3) методом Ругне- Кутта 4-го порядка с заданным шагом h с учетом возмущений. Решаются уравнения (4).
2. Обозначим −h̅̇ из уравнения (8) как . Тогда линейный, асимптотически устойчивый закон управления ДМ можно представить в виде:

Где Λ1:3 = [Λ1 Λ2 Λ3]𝑇 – вектор из элементов кватерниона ориентации.

Следует также ввести две нелинейности типа «насыщение» для и для величины следующего вида.

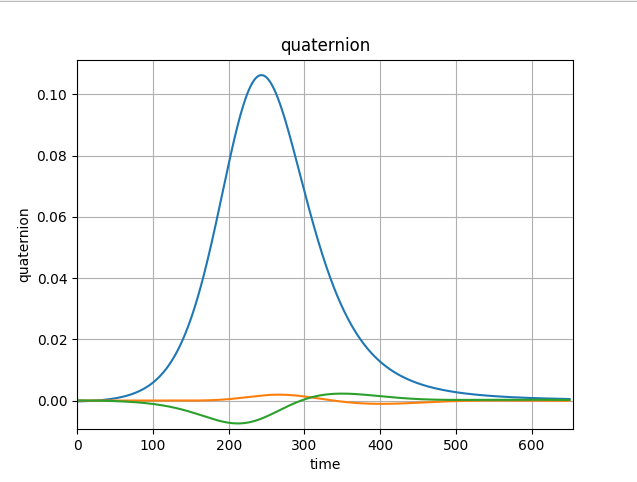
,

Первая нелинейность характеризует существование предельного ускорения ДМ. Вторая – существование предельной скорости вращения ДМ.

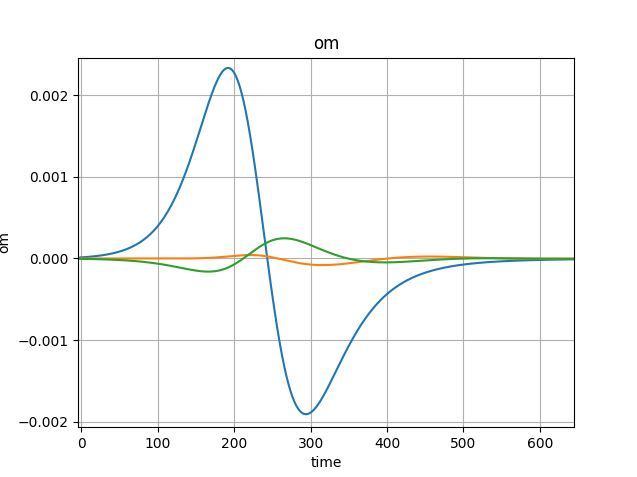
Для анализа результатов, каждые 100 мс счета программы требуется выводить в файл следующие величины: Λ, 𝜔̅, Ω̅, 𝑟̅. Моделирование необходимо провести с учетом влияния гравитационного возмещающего момента (уравнение (1.9)).

**Полученные результаты**

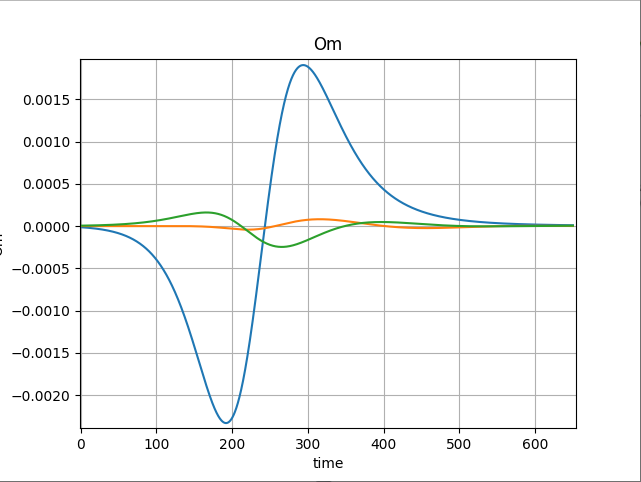
Результаты при и



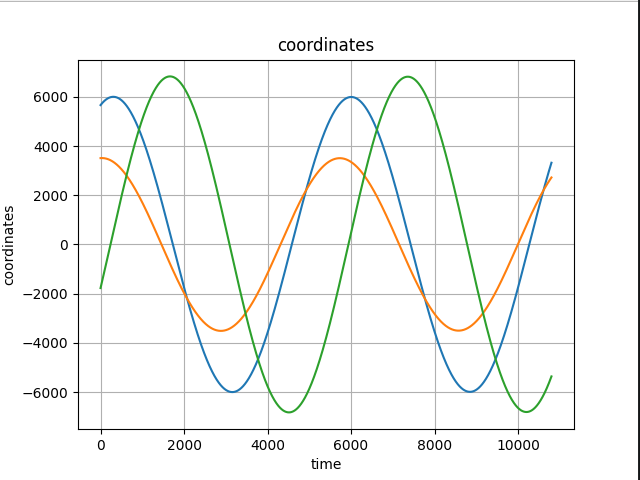
*Рисунок 1, Изменение компонент кватерниона ориентации КА в пространстве.*

*.*

*Рисунок 2. Изменение компонент угловой скорости вращения КА.*

*.*

*Рисунок 3. Изменение компонент угловой скорости вращения маховика.*



*Рисунок 4. Изменение компонент радиус-вектора КА*

# **Выводы**

В результате выполнения лабораторной работы разработан алгоритм решения задачи приведения ориентации летательного аппарата, от текущего направления к заданному с использованием в качестве управляющих органов реактивных двигателей стабилизации. Был изучен процесс моделирования управляемого вращательного движения КА, оснащённого ДМ, закреплено понимания работы системы управления угловым движением КА с ДМ. Во время решения задачи было учтено влияние Солнца и не сферичности гравитационного поля Земли.

* Увеличение и уменьшение приводит к уменьшению времени переходного процесса, но переходный процесс ухудшается.
* Уменьшение и увеличение приводит к увеличению времени переходного процесса.
* При сильном изменении коэффициентов усиления система теряет устойчивость и сопровождается колебаниями с недопустимой амплитудой.

# **Листинг**

## Main.py

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346  347  348  349  350  351  352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367  368  369  370  371  372  373  374  375  376  377  378  379  380  381  382  383  384  385  386  387  388  389  390  391  392  393  394  395  396  397  398  399  400  401  402  403  404  405  406  407  408  409  410  411  412  413  414  415  416  417  418  419  420  421  422  423  424  425  426  427  428  429  430  431  432  433  434  435  436  437  438  439  440  441  442  443  444  445  446  447  448  449  450  451  452  453  454  455  456  457  458  459  460  461  462  463  464  465  466  467 | **from** **math** **import** cos, sin, sqrt  **import** **matplotlib.pyplot** **as** **plt**  **from** **constants** **import** \*  **from** **start\_data** **import** \*  **from** **Math\_def** **import** (  matr3\_vect3\_cross,  vect\_cross,  quat\_cross,  JData,  matrix\_transposition,  norm\_quaternion,  module\_vect,  sign  )  **def** **get\_AGR**(t, coordinates):  """Ускорение за счет влияния несферичности гравитационного поля."""    rj2000 = [  coordinates['x'] \* METR\_IN\_KM,  coordinates['y'] \* METR\_IN\_KM,  coordinates['z'] \* METR\_IN\_KM  ];  time = (JData(t) - JD0) / JDD  f = **86400** \* (JData(t) % **1**)  alfa = (DS2R \* ((A + (B + (C + D \* time) \* time) \* time) + f)) % (**2** \* PI)    **if** alfa < **0**:  alfa += **2** \* PI    # матрица перехода из ск  Mj2gr = [  [cos(alfa), sin(alfa), **0**],  [-sin(alfa), cos(alfa), **0**],  [**0**, **0**, **1**]  ]    # вектор в ск j2000  r = matr3\_vect3\_cross(Mj2gr, rj2000)  absr = module\_vect(r)  ag\_grav = [] # вектор aj2    # посчитали ag\_grav  ag\_grav.append(-**1.5** \* GEOPOTONTIAL\_COEFF \* (MU\_M / absr\*\***2**) \*  (RADIUS\_EARTH / absr)\*\***2** \*  ((**1** - **5** \* (r[**2**] / absr)\*\***2**) \* (r[**0**] / absr)))  ag\_grav.append(-**1.5** \* GEOPOTONTIAL\_COEFF \* (MU\_M / absr\*\***2**) \*  (RADIUS\_EARTH / absr)\*\***2** \*  (**1** - **5** \* (r[**2**] / absr)\*\***2**) \* (r[**1**] / absr))  ag\_grav.append(-**1.5** \* GEOPOTONTIAL\_COEFF \* (MU\_M / absr\*\***2**) \*  (RADIUS\_EARTH / absr)\*\***2** \*  (**3** - **5** \* (r[**2**] / absr)\*\***2**) \* (r[**2**] / absr))      **for** i **in** range(**3**):  ag\_grav[i] /= **1000**    Mj2grT = matrix\_transposition(Mj2gr)  Agrav = matr3\_vect3\_cross(Mj2grT, ag\_grav)    #print('ag\_grav =', ag\_grav)  #print('Agrav =', Agrav)    **return** Agrav  **def** **get\_ASUN**(t, coordinates):  """Гравитационное ускорение, за счет влияния солнца."""    rj2000 = [coordinates['x'], coordinates['y'], coordinates['z']]    # модифицированное Юлианское время  T = (JData(t) - **2451545.5**) / **36525**  M = radians(**357.5226** + **35999.049** \* T)    # лямбда с точкой в кружочке  lambda\_tchk = (SUM\_DVU\_ARG\_PER + M +  **6892** \* PI \* sin(M) / (**648000**) +  **72** \* PI \* sin(**2** \* M) / (**648000**))  # модуль вектора с точкой в кружочке  absR = (**149.619** - **2.499** \* cos(M) - **0.021** \* cos(**2** \* M)) \* **10**\*\***6**    # считаем вектор на солнце  rs = [**0**, **0**, **0**]  rs[**0**] = absR \* cos(lambda\_tchk)  rs[**1**] = absR \* sin(lambda\_tchk) \* cos(EPS)  rs[**2**] = absR \* sin(lambda\_tchk) \* sin(EPS)    Asun = []  **for** i **in** range(**3**):  Asun.append(  MU\_SUN \*  ((rs[i] - rj2000[i]) /  (sqrt(  (rs[**0**] - rj2000[**0**])\*\***2** +  (rs[**1**] - rj2000[**1**])\*\***2** +  (rs[**2**] - rj2000[**2**])\*\***2**  )  )\*\***3**  -  rs[i] /  (module\_vect(rs)\*\***3**)  )  )  **return** Asun  **def** **get\_h**(quaternion, om):  """Вектор кинетического момента систем."""    h =[]  q = sign(quaternion[**0**])    h.append(-kp\*q \* quaternion[**1**] - kd \* om[**0**])  h.append(-kp\*q \* quaternion[**2**] - kd \* om[**1**])  h.append(-kp\*q \* quaternion[**3**] - kd \* om[**2**])    **for** i **in** range(len(h)):  **if** abs(h[i]) >= h\_max:  h[i] = h\_max \* sign(h[i])  **elif** abs(Om[i]) >= Om\_max:  h[i] = **0**    **return** h  **def** **get\_gravitational\_moment**(coordinates, quaternion):  """Гравитационный момент.  M = ( 3\*mu / 2\* R^(3) ) \* (nu x J\*nu); nu = Q\_ x rj2000 x Q  """    quaternion\_ = [  quaternion[**0**],  -quaternion[**1**],  -quaternion[**2**],  -quaternion[**3**]  ]  rj2000 = [  **0**,  coordinates['x'] \* METR\_IN\_KM,  coordinates['y'] \* METR\_IN\_KM,  coordinates['z'] \* METR\_IN\_KM  ]  # расстояние от центра масс Земли до центра масс КА  m\_rj2000 = module\_vect(rj2000)    # нормированный вектор r0j2000  r0j2000 = []  **for** i **in** range(**4**):  r0j2000.append(rj2000[i] / m\_rj2000)    #Получили кватарнион ню  nu = quat\_cross(quaternion\_, r0j2000)  nu = quat\_cross(quaternion, nu)    Jnu = matr3\_vect3\_cross(J, nu[**1**:])  nJn = vect\_cross(nu[**1**:], Jnu)    Mvozm = []  **for** i **in** range(**3**):  Mvozm.append((**3** \* MU\_M / (**2** \* m\_rj2000\*\***3**)) \* nJn[i])    **return** Mvozm    **def** **get\_new\_quaternion**(quaternion, om):  """L` = 1/2 \* L (x) Lw"""    Lw = [**0**, om[**0**], om[**1**], om[**2**]]  quaternion\_res = quat\_cross(quaternion, Lw)  **for** i **in** range(len(quaternion\_res)):  quaternion\_res[i] \*= **0.5**    **return** quaternion\_res  **def** **get\_om**(om, Om, Mvozm, h):  """Уравнение динамики Эйлера в форме Коши для КА оснащенного ДМ.  w` = Jinv\*(Msum - w x(J\*w+I\*Om)-h)"""  Jw = matr3\_vect3\_cross(J, om)    temp = []  **for** i **in** range(**3**):  IOm = Idm \* Om[i]  # J\*w+I\*Om  temp.append(Jw[i] + IOm)  # w\*(J\*w+Idm\*Om)  temp = vect\_cross(om, temp)  temp2 = []  **for** i **in** range(**3**):  #M-w\*(J\*w+Idm\*Om)-h  temp2.append(Mvozm[i] - temp[i] + h[i])  **return** matr3\_vect3\_cross(Jinv, temp2)  **def** **get\_Om**(h):  Om\_res = []  **for** i **in** range(len(h)):  Om\_res.append(Idm\*\*(-**1**) \* h[i])    **return** Om\_res  **def** **get\_perturbed\_orbital\_motion**(coordinates, agrav, asun):  """Уравнение возмущенного орбитального движения."""    orbital\_motion = []  coordinats = [  coordinates['x'],  coordinates['y'],  coordinates['z']  ]  **for** i **in** range(len(coordinats)):  orbital\_motion.append(-MU\_KM \* coordinats[i] /  (module\_vect(coordinates)\*\***3**) + agrav[i] + asun[i])    **return** orbital\_motion  **def** **RP**(t, main\_data):  new\_main\_data = {}    h = get\_h(main\_data['quaternion'], main\_data['om'])  Mvozm = get\_gravitational\_moment(  main\_data['coordinates'],  main\_data['quaternion']  )  quaternion\_res = get\_new\_quaternion(  main\_data['quaternion'],  main\_data['om']  )  om\_res = get\_om(  main\_data['om'],  main\_data['Om'],  Mvozm,  h  )  Om\_res = get\_Om(h)    agrav = get\_AGR(t, main\_data['coordinates'])  asun = get\_ASUN(t, main\_data['coordinates'])    orbital\_motion = get\_perturbed\_orbital\_motion(  main\_data['coordinates'],  agrav,  asun)  new\_main\_data['coordinates'] = {  'x': main\_data['speeds']['Vx'],  'y': main\_data['speeds']['Vy'],  'z': main\_data['speeds']['Vz'],  }  new\_main\_data['speeds'] = {  'Vx': orbital\_motion[**0**],  'Vy': orbital\_motion[**1**],  'Vz': orbital\_motion[**2**],  }  new\_main\_data['quaternion'] = quaternion\_res  new\_main\_data['om'] = om\_res  new\_main\_data['Om'] = Om\_res    **return** new\_main\_data  **def** **main\_data\_in\_list**(main\_data):  main\_data\_list = []  coordinates = main\_data['coordinates']  main\_data\_list.append(coordinates['x'])  main\_data\_list.append(coordinates['y'])  main\_data\_list.append(coordinates['z'])    speeds = main\_data['speeds']  main\_data\_list.append(speeds['Vx'])  main\_data\_list.append(speeds['Vy'])  main\_data\_list.append(speeds['Vz'])    quaternion = main\_data['quaternion']  **for** i **in** quaternion:  main\_data\_list.append(i)    om = main\_data['om']  **for** i **in** om:  main\_data\_list.append(i)    Om = main\_data['Om']  **for** i **in** Om:  main\_data\_list.append(i)    **return** main\_data\_list  **def** **main\_data\_in\_dict**(main\_data\_list):  main\_data = {}    main\_data['coordinates'] = {  'x': main\_data\_list[**0**],  'y': main\_data\_list[**1**],  'z': main\_data\_list[**2**]  }    main\_data['speeds'] = {  'Vx': main\_data\_list[**3**],  'Vy': main\_data\_list[**4**],  'Vz': main\_data\_list[**5**]  }  main\_data['quaternion'] = [  main\_data\_list[**6**],  main\_data\_list[**7**],  main\_data\_list[**8**],  main\_data\_list[**9**]  ]    main\_data['om'] = [  main\_data\_list[**10**],  main\_data\_list[**11**],  main\_data\_list[**12**]  ]  main\_data['Om'] = [  main\_data\_list[**13**],  main\_data\_list[**14**],  main\_data\_list[**15**]  ]    **return** main\_data    **def** **runge\_kutta4**(f, t, main\_data):  step2 = step / **2**  t1 = t + step2    data = [**0** **for** \_ **in** range(NUMBER\_EQUATIONS\_IN\_SYSTEM)]    data1 = f(t, main\_data)  main\_data\_list = main\_data\_in\_list(main\_data)  data1\_list = main\_data\_in\_list(data1)    **for** i **in** range(NUMBER\_EQUATIONS\_IN\_SYSTEM):  data[i] = main\_data\_list[i] + step2 \* data1\_list[i]    data\_dict = main\_data\_in\_dict(data)  data2 = f(t1, data\_dict)  data2\_list = main\_data\_in\_list(data2)    **for** i **in** range(NUMBER\_EQUATIONS\_IN\_SYSTEM):  data[i] = main\_data\_list[i] + step2 \* data2\_list[i]    data\_dict = main\_data\_in\_dict(data)  data3 = f(t1, data\_dict)  data3\_list = main\_data\_in\_list(data3)    **for** i **in** range(NUMBER\_EQUATIONS\_IN\_SYSTEM):  data[i] = main\_data\_list[i] + step \* data3\_list[i]    data\_dict = main\_data\_in\_dict(data)  data4 = f(t + step, data\_dict)  data4\_list = main\_data\_in\_list(data4)  new\_data\_list = []    **for** i **in** range(NUMBER\_EQUATIONS\_IN\_SYSTEM):  new\_data\_list.append(  main\_data\_list[i] + step \*  (data1\_list[i] + data4\_list[i] +  **2** \* (data2\_list[i] + data3\_list[i])  ) / **6**  )    **return** main\_data\_in\_dict(new\_data\_list)  **def** **print\_graph**(  x,  y,  xlabel = 'x',  ylabel = 'y',  ):    plt.plot(x, y)  plt.xlabel(xlabel)  plt.ylabel(ylabel)  plt.title(ylabel)  plt.grid(True)  plt.show()    **def** **print\_group\_graph**(  x,  y,  xlabel = 'x',  ylabel = 'y'  ):    **for** i **in** range(len(y)):  plt.plot(x, y[i])    plt.xlabel(xlabel)  plt.ylabel(ylabel)  plt.title(ylabel)  plt.grid(True)  plt.show()  **def** **output\_graphich**(time, \*\*kwarg):  **for** key **in** kwarg.keys():  **for** i **in** range(len(kwarg[key])):  print\_group\_graph(time, kwarg[key], 'time', f'{key}')  #print\_graph(time, kwarg[key][i], 'time', f'{key} {i}')  **def** **main**():  main\_data = {  'coordinates': coordinates,  'speeds': speeds,  'quaternion': quaternion,  'om': om,  'Om': Om  }  t = **0**  time\_graph = []  quaternion\_graph = [[], [], [], []]  om\_graph = [[], [], []]  Om\_graph = [[], [], []]  coordinates\_graph = [[], [], []]  **while** (t <= t\_kon):  new\_data = runge\_kutta4(RP, t, main\_data) #Рунге-Кутт  **for** key **in** main\_data.keys():  main\_data[key] = new\_data[key]    **for** i **in** range(len(main\_data['quaternion'])):  quaternion\_graph[i].append(main\_data['quaternion'][i])    main\_data['quaternion'] = norm\_quaternion(main\_data['quaternion'])    time\_graph.append(t)    #for i in range(len(main\_data['quaternion'])):  #quaternion\_graph[i].append(main\_data['quaternion'][i])    **for** i **in** range(len(main\_data['om'])):  om\_graph[i].append(main\_data['om'][i])  Om\_graph[i].append(-main\_data['om'][i])  coordinates\_graph[**0**].append(main\_data['coordinates']['x'])  coordinates\_graph[**1**].append(main\_data['coordinates']['y'])  coordinates\_graph[**2**].append(main\_data['coordinates']['z'])  #print(f'{t} {main\_data["coordinates"]} {main\_data["speeds"]} {main\_data["quaternion"]}'  # f'{main\_data["om"]} {main\_data["Om"]}')  t += step  #Графики  output\_graphich(  time\_graph,  quaternion = quaternion\_graph[**1**:],  om = om\_graph,  Om = Om\_graph,  coordinates = coordinates\_graph  )  **if** \_\_name\_\_ == '\_\_main\_\_':  main() |

## Math\_def.py

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97 | **from** **math** **import** copysign, sqrt  **from** **start\_data** **import** nach\_era  sign = **lambda** x: copysign(**1**, x)  **def** **matr3\_vect3\_cross**(matr, vect):  """Перемножение матрицы 3 на 3 на вектор 3 на 1.  matr - матрица, vect - вектор."""    res\_vec = []  res\_vec.append(matr[**0**][**0**] \* vect[**0**] +  matr[**0**][**1**] \* vect[**1**] +  matr[**0**][**2**] \* vect[**2**])    res\_vec.append(matr[**1**][**0**] \* vect[**0**] +  matr[**1**][**1**] \* vect[**1**] +  matr[**1**][**2**] \* vect[**2**])    res\_vec.append(matr[**2**][**0**] \* vect[**0**] +  matr[**2**][**1**] \* vect[**1**] +  matr[**2**][**2**] \* vect[**2**])    **return** res\_vec  **def** **vect\_cross**(vect1, vect2):  """Векторное произведение."""    res\_vec = []  res\_vec.append(vect1[**1**] \* vect2[**2**] - vect1[**2**] \* vect2[**1**])  res\_vec.append(vect1[**2**] \* vect2[**0**] - vect1[**0**] \* vect2[**2**])  res\_vec.append(vect1[**0**] \* vect2[**1**] - vect1[**1**] \* vect2[**0**])    **return** res\_vec  **def** **quat\_cross**(quat1, quat2):  """Перемножение кватарнионов."""    res\_quat = []    res\_quat.append(quat1[**0**] \* quat2[**0**] -  quat1[**1**] \* quat2[**1**] -  quat1[**2**] \* quat2[**2**] -  quat1[**3**] \* quat2[**3**])    res\_quat.append(quat1[**0**] \* quat2[**1**] +  quat1[**1**] \* quat2[**0**] +  quat1[**3**] \* quat2[**2**] -  quat1[**2**] \* quat2[**3**])    res\_quat.append(quat1[**0**] \* quat2[**2**] +  quat1[**2**] \* quat2[**0**] +  quat1[**1**] \* quat2[**3**] -  quat1[**3**] \* quat2[**1**])    res\_quat.append(quat1[**0**] \* quat2[**3**] +  quat1[**3**] \* quat2[**0**] +  quat1[**2**] \* quat2[**1**] -  quat1[**1**] \* quat2[**2**])    **return** res\_quat  **def** **JData**(t):  """Вычисление юлианской даты."""    a = (**14** - nach\_era['month']) // **12**;  y = nach\_era['year'] + **4800** - a;  m = nach\_era['month'] + **12** \* a - **3**;  JD = nach\_era['day'] + (**153** \* m + **2**) // **5** + **365** \* y + y // **4** - y // **100** + y // **400** - **32045**;    **return** (JD + (nach\_era['hours'] - **12**) / **24** +  nach\_era['minut'] / **1440** +  (nach\_era['sec'] + t) / **86400**)  **def** **matrix\_transposition**(matrix):  transposition\_matrix = matrix    **for** i **in** range(**3**):  **for** j **in** range(**3**):  transposition\_matrix[i][j] = matrix[j][i]    **return** transposition\_matrix  **def** **norm\_quaternion**(quaternion):  """Нормировка кватерниона."""    **for** i **in** range(**4**):  quaternion[i] /= sqrt(quaternion[**0**]\*\***2** +  quaternion[**1**]\*\***2** +  quaternion[**2**]\*\***2** +  quaternion[**3**]\*\***2**)    **return** quaternion  **def** **module\_vect**(vect):  **if** type(vect) == dict:  vect = list(vect.values())  **return** sqrt(vect[**0**]\*\***2** + vect[**1**]\*\***2** + vect[**2**]\*\***2**) |

## constants.py

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | **from** **math** **import** radians  PI = **3.141592653589793**  MU\_KM = **398600.4415**  MU\_M = **3.986004418** \* **10**\*\***14**  MU\_SUN = **132712517951**  RADIUS\_EARTH = **6378137**  GEOPOTONTIAL\_COEFF = **1.08262668355** \* **10**\*\*(-**3**)  EPS = radians(**23.43929111**)  SUM\_DVU\_ARG\_PER = radians(**282.94**)  A = -**19089.451590**  B = **8640184.812866**  C = **0.093104**  D = -**6.2** \* **1e-6**  JD0 = **2451545**  JDD = **36525**  DS2R = **7.272205216643039903848712** \* **10**\*\*(-**5**)  METR\_IN\_KM = **1000**  NUMBER\_EQUATIONS\_IN\_SYSTEM = **16** |

## Start\_data.py

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | nach\_era = {  'year': **2022**,  'month': **6**,  'day': **30**,  'hours': **21**,  'minut': **0**,  'sec': **0**,  'msec': **0**,  }  del\_phi = **0.01**  t = **0**  t\_kon = **10800**  step = **1** #0.1  coordinates = {  'x': **5666.282392**,  'y': **3512.092276**,  'z': -**1780.014521**,  }  speeds = {  'Vx': **2.194685**,  'Vy': **0.146478**,  'Vz': **7.275306**,  }  quaternion = [**1**, **0**, **0**, **0**] # кватернион  om = [**10**\*\*(-**5**), **10**\*\*(-**6**), **10**\*\*(-**5**)]  Om = [**0**, **0**, **0**]  h\_max = **0.01**  Om\_max = **628**  J = [[**1**, **0**, **0**], # тензор инерции  [**0**, **2**, **0**],  [**0**, **0**, **1**]]  Jinv = [[**1**, **0** ,**0**],  [**0**, **0.5**, **0**],  [**0**, **0**, **1**],]  Idm = **0.00016**  kp = **0.05** #0.05  kd = **0.4** #0.1 |