**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»**



Факультет 6 «Аэрокосмический»

Кафедра 604 «Системный анализ и управление»

**Отчёт**

по дисциплине «Основы научных исследований»

«Моделирование работы солнечного датчика»

Работу выполнил:

Студент группы М6О-301Б-22

Шуреков Всеволод Александрович \_\_\_\_\_\_\_\_\_\_ (\_\_.\_\_.202\_)

Работу принял:

Ст. преподаватель каф. 604

Розин Пётр Евгеньевич \_\_\_\_\_\_\_\_\_\_

(\_\_.\_\_.202\_)

Москва 202\_

Оглавление

[Постановка задачи: 3](#_Toc199467915)

[Математическая модель 3](#_Toc199467916)

[1. Орбитальное движение 3](#_Toc199467917)

[2. Солнечный датчик 4](#_Toc199467918)

[Реализация программы 7](#_Toc199467919)

[3. Орбитальное движение 7](#_Toc199467920)

[4. Солнечный датчик 7](#_Toc199467921)

[Результаты 9](#_Toc199467922)

[Вывод 9](#_Toc199467923)

[Список литературы 10](#_Toc199467924)

[Листинг 11](#_Toc199467925)

[5. Приложение А. main 11](#_Toc199467926)

[6. Приложение Б. sun\_sensor\_model 11](#_Toc199467927)

[7. Приложение В. orbit\_simulator 13](#_Toc199467928)

[8. Приложение Г. j2\_gravitational\_acceleration 14](#_Toc199467929)

# Постановка задачи:

Разработка математической модели аналогового солнечного датчика и исследование построения солнечной ориентации. Промоделировать процесс работы фотоэлементов

# Математическая модель

Для выполнения задания были реализованы следующие компоненты:  
1. Моделирование орбитального движения КА

2. Моделирование ориентации КА

3. Имитация работы фотоэлектрического солнечного датчика

## Орбитальное движение

Для начала улучшим орбитальную модель движения КА из предыдущего семестра путём изменения метода Эйлера на метод Рунге-Кутты 4-го порядка. Этот метод использует четыре промежуточных вычисления ("коэффициента наклона"), чтобы вычислить приближение решения на новом временном шаге.

Формулы промежуточных коэффициентов и вектора состояния в момент будет:

В случае движения КА по орбите модель описывается ОДУ второго порядка:

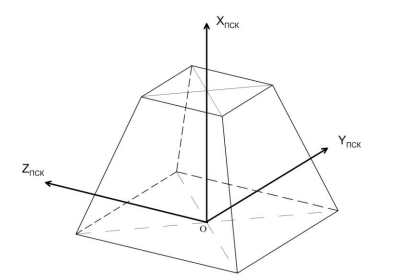
Чтобы свести её к системе первого порядка, мы вводим вектор состояния:

Тогда функция правой части будет выглядеть:

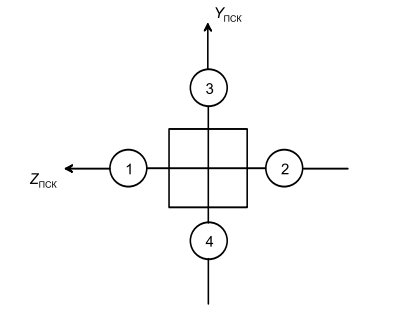
Модель позволяет вычислять текущее положение КА в любой момент времени, что необходимо для дальнейшего расчёта направления на Солнце и его измерения с помощью бортового солнечного датчика.

## Солнечный датчик

Модель аналогового солнечного датчика, основанная на конструкции, описанной в работе Жукова Б. И. Датчик представляет собой усечённую четырёхугольную пирамиду с восемью фотоэлектрическими элементами, расположенными по два на каждой из четырёх боковых граней. Угол наклона граней к основанию составляет 68°.



Выходные сигналы этого датчика — это токи, пропорциональные интенсивности светового потока на каждой грани.



На данном рисунке изображена Приборная система координат (ПСК), где

— перпендикулярен нижнему основанию, направлен от основания к вершине пирамиды.

— лежит в плоскости основания, перпендикулярен одному из ребер.

— дополняет систему до правой тройки.

На вход датчику даются единичный вектор направления на Солнце в ПСК:

и кватернион ориентации КА (поворота ССК → ПСК) относительно оси Y. На выходе будет токи с четырёх граней .

Для того чтобы вычислить сигналы датчика, нужно перевести направление на Солнце из связанной системы координат (ССК) в приборную систему координат (ПСК) — систему координат самого датчика.

Сначала превращаем кватернион в матрицу поворота 3x3, которая используется для преобразования координат. После Перевод направления на Солнце из связанной системы координат (ССК) в приборную (ПСК) происходит с использованием матрицы поворота

Матрица поворота *R*(*q*) вычисляется как:

В нашем случае

**Алгоритм формирования сигнала:**

Сначала задаём датчик.

Эти векторы описывают направления перпендикуляров к граням пирамиды, наклонённым под углом 68° к основанию.

Далее нам ужен Угол падения лучей на грань. Он позволяет вычислить угол между солнечным лучом и нормалью к детектору. Угол падения между направлением на Солнце и нормалью к *n*-й грани:

Потом в коде считается угол визирования Солнца, который определяется как угол между направлением на Солнце и осью ХСПК:

И Азимут Солнца, который вычисляется как арктангенс отношения Z/Y компонент вектора

После этого считается коэффициент засветки, учитывающий затенение блендой.

Если угол визирования Солнца

Если , то

Если , то (Солнце вне поля зрения).

Также учитывается эффект полного отражения

Если угол падения , то (полное отражение).

- Иначе: (формирование тока)

# Реализация программы

## Орбитальное движение

Для начала необходимо добавить вектор состояния.

y0 = np.concatenate((r0, v0))

Далее добавить функцию правой части дифф уравнений.

r\_norm = np.linalg.norm(r)  
a\_central = -mu / r\_norm \*\* 3 \* r  
a\_perturb = a\_grav(dateTime, r)  
a\_total = a\_central + a\_perturb  
return np.concatenate((v, a\_total)) # [dr/dt, dv/dt]

Реализация шага Рунге-Кутты.

def rk4\_step(y, t, dt, dydt\_func):

k1 = dydt\_func(t, y)

k2 = dydt\_func(t + dt / 2, y + dt / 2 \* k1)

k3 = dydt\_func(t + dt / 2, y + dt / 2 \* k2)

k4 = dydt\_func(t + dt, y + dt \* k3)

y\_new = y + dt / 6 \* (k1 + 2 \* k2 + 2 \* k3 + k4)

return y\_new

Далее основной цикл интегрирования, в котором вычисляется текущее время t, обновляется вектор состояния current\_y.

В файле Результат(RK4).txt приведены новые результаты перехода.

## Солнечный датчик

Нормали к граням датчика описаны:

normals = {

'face1': [sin(22), 0, cos(22)],

'face2': [sin(22), 0, -cos(22)],

'face3': [sin(22), cos(22), 0],

'face4': [sin(22), -cos(22), 0]

}

Входные данные

1. Направление на Солнце в связанной системе координат КА

sun\_ort\_real\_ssk = np.array([0.7, 0.7, 0.0])

Это единичный вектор направления на Солнце в связанной системе координат (ССК) — т.е. система координат самого КА.

б) Ориентация КА — кватернион

craft\_orient\_quat = quat\_from\_axis\_angle([0, 1, 0], 50)

Это кватернион поворота , который определяет, как КА ориентирован относительно некоторой базовой системы (например, инерциальной). Здесь: поворот вокруг оси Y на 50°.

Преобразование кватерниона в матрицу поворота происходит в функции:  
def quat\_to\_rotmat(q):

…

Перевод вектора из ССК в ПСК

trans\_matrix = quat\_to\_rotmat(craft\_orient\_quat)

sun\_ort\_real\_psk = np.dot(inst\_matrix, np.dot(trans\_matrix, sun\_ort\_real\_ssk))

sun\_ort\_real\_psk = normalize(sun\_ort\_real\_psk)

Угол падения лучей на грань

def compute\_incidence\_angle(sun\_vec\_psk, normal):

dot = np.dot(sun\_vec\_psk, normal)

dot = np.clip(dot, -1.0, 1.0)

return np.arccos(dot)

Угол визирования Солнца

alpha\_C = np.arccos(sun\_ort\_real\_psk[0])

Азимут Солнца

phi\_s = np.arctan2(sun\_ort\_real\_psk[2], sun\_ort\_real\_psk[1])

Коэффициент засветки

def compute\_Kc(alpha\_C):

…

Проверка на угол падения света:  
def apply\_total\_reflection(angle\_inc, current\_i):

return 0.0 if angle\_inc >= np.radians(83) else current\_i

Формирование тока в коде выглядит так:

for name, n in normals.items():

eps = compute\_incidence\_angle(sun\_ort\_real\_psk, n)

i = np.dot(sun\_ort\_real\_psk, n) \* Kc

i = apply\_total\_reflection(eps, i)

currents[name] = i

# Результаты

Результатом работы программы станут: положение КА в инерциальной системе координат в любой момент времени, Положение Солнца в ИСК

Результат работы модели датчика позволяет формировать выходные сигналы, и использовать их для создания программы восстановления направления на Солнце.

Результат примера с sun\_ort\_real\_ssk = np.array([0.7, 0.7, 0.0]), и craft\_orient\_quat = quat\_from\_axis\_angle([0, 1, 0], 50):

Направление на Солнце в ПСК: [ 0.45451948 0.70710678 -0.54167522]

Измеренные токи: {'face1': 0.0, 'face2': np.float64(0.6724985119639574), 'face3': np.float64(0.8258839841427611), 'face4': 0.0}

Угол визирования α\_C (градусы): 62.965979160175145

Азимут φ\_s (градусы): -37.453719557105146

Коэффициент засветки Kc: 1.0

# Вывод

В данной работе была улучшена модель орбитального движения КА. Также мы создали имитацию солнечного датчика, который в зависимости от направления и ориентации рассчитывает ток, который генерируется каждым фотоэлектрическим элементом датчика при освещении его солнечным светом.

Эта работа составляет основу для дальнейшей разработки алгоритмов управления ориентацией КА на основе данных солнечного датчика, а именно реализация алгоритма восстановления ориентации на Солнце и интегрирование датчика в КА из модели орбитального движения.

# Список литературы

[1] Жуков Б.И. Определение положения Солнца на борту КА по данным фотоэлектрического солнечного датчика // Препринты ИПМ им. М.В. Келдыша. 2019. № 9.

# Листинг

## Приложение А. main

import numpy as np  
from sun\_sensor\_model import sun\_detector\_model, normalize  
  
  
def quat\_from\_axis\_angle(axis, angle\_deg):  
 *"""Создаёт кватернион из оси и угла поворота."""* angle\_rad = np.radians(angle\_deg)  
 axis = normalize(np.array(axis))  
 return np.array([  
 np.cos(angle\_rad / 2),  
 axis[0] \* np.sin(angle\_rad / 2),  
 axis[1] \* np.sin(angle\_rad / 2),  
 axis[2] \* np.sin(angle\_rad / 2)  
 ])  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 *# Направление на Солнце в связанной системе координат (ССК)* sun\_ort\_real\_ssk = np.array([0.7, 0.7, 0.0])  
  
 *# Кватернион ориентации КА: поворот вокруг оси Y на 50 градусов* craft\_orient\_quat = quat\_from\_axis\_angle([0, 1, 0], 50)  
  
 *# Вызов модели датчика* result = sun\_detector\_model(sun\_ort\_real\_ssk, craft\_orient\_quat)  
  
  
 *# Вывод результатов* print("Направление на Солнце в ПСК:", result["sun\_ort\_real\_psk"])  
 print("Измеренные токи:", result["currents"])  
 print("Угол визирования α\_C (градусы):", result["alpha\_C\_deg"])  
 print("Азимут φ\_s (градусы):", result["phi\_s\_deg"])  
 print("Коэффициент засветки Kc:", result["Kc"])

## Приложение Б. sun\_sensor\_model

*# sensor\_model.py*import numpy as np  
  
def normalize(v):  
 norm = np.linalg.norm(v)  
 return v / norm if norm != 0 else v  
  
def quat\_to\_rotmat(q):  
 *"""Преобразует кватернион [w, x, y, z] в матрицу поворота 3x3."""* w, x, y, z = q  
 return np.array([  
 [1 - 2 \* y \*\* 2 - 2 \* z \*\* 2, 2 \* x \* y - 2 \* z \* w, 2 \* x \* z + 2 \* y \* w],  
 [2 \* x \* y + 2 \* z \* w, 1 - 2 \* x \*\* 2 - 2 \* z \*\* 2, 2 \* y \* z - 2 \* x \* w],  
 [2 \* x \* z - 2 \* y \* w, 2 \* y \* z + 2 \* x \* w, 1 - 2 \* x \*\* 2 - 2 \* y \*\* 2]  
 ])  
  
def compute\_Kc(alpha\_C):  
 if alpha\_C >= np.radians(90):  
 return 0.0  
 elif alpha\_C <= np.radians(74):  
 return 1.0  
 else:  
 numerator = np.sin(np.radians(96)) \* np.cos(alpha\_C)  
 denominator = np.sin(np.radians(22) + alpha\_C) \* np.sin(np.radians(16))  
 return numerator / denominator  
  
def compute\_incidence\_angle(sun\_vec\_psk, normal):  
 dot = np.dot(sun\_vec\_psk, normal)  
 dot = np.clip(dot, -1.0, 1.0)  
 return np.arccos(dot)  
  
def apply\_total\_reflection(angle\_inc, current\_i):  
 return 0.0 if angle\_inc >= np.radians(83) else current\_i  
  
def sun\_detector\_model(sun\_ort\_real\_ssk, craft\_orient\_quat, inst\_matrix=np.eye(3)):  
 sin22 = np.sin(np.radians(22))  
 cos22 = np.cos(np.radians(22))  
  
 normals = {  
 'face1': np.array([sin22, 0.0, cos22]),  
 'face2': np.array([sin22, 0.0, -cos22]),  
 'face3': np.array([sin22, cos22, 0.0]),  
 'face4': np.array([sin22, -cos22, 0.0])  
 }  
  
 trans\_matrix = quat\_to\_rotmat(craft\_orient\_quat)  
 sun\_ort\_real\_psk = np.dot(inst\_matrix, np.dot(trans\_matrix, sun\_ort\_real\_ssk))  
 sun\_ort\_real\_psk = normalize(sun\_ort\_real\_psk)  
  
 alpha\_C = np.arccos(sun\_ort\_real\_psk[0])  
 phi\_s = np.arctan2(sun\_ort\_real\_psk[2], sun\_ort\_real\_psk[1])  
  
 Kc = compute\_Kc(alpha\_C)  
  
 currents = {}  
 for name, n in normals.items():  
 eps = compute\_incidence\_angle(sun\_ort\_real\_psk, n)  
 i = np.dot(sun\_ort\_real\_psk, n) \* Kc  
 i = apply\_total\_reflection(eps, i)  
 currents[name] = i  
  
 return {  
 "sun\_ort\_real\_psk": sun\_ort\_real\_psk,  
 "currents": currents,  
 "alpha\_C\_deg": np.degrees(alpha\_C),  
 "phi\_s\_deg": np.degrees(phi\_s),  
 "Kc": Kc  
 }

## Приложение В. orbit\_simulator

import numpy as np  
from datetime import datetime, timedelta  
from j2\_gravitational\_acceleration import a\_grav  
  
t0 = datetime(2024, 10, 1, 11, 0, 0)  
r0 = np.array([5666.282392, 3512.092276, -1780.014521]) *# км*v0 = np.array([2.194685, 0.146478, 7.275306]) *# км/с*endTime = 86400 *# 24 часа в секундах*dt = 1 *# Шаг интегрирования  
  
# Вектор состояния: [rx, ry, rz, vx, vy, vz]*y0 = np.concatenate((r0, v0))  
  
def dydt(t, y):  
 r = y[:3]  
 v = y[3:]  
  
 mu = 398600.4415 *# км^3/с^2 (гравитационная постоянная Земли)* dateTime = t0 + timedelta(seconds=float(t))  
  
 *# Центральное гравитационное ускорение* r\_norm = np.linalg.norm(r)  
 a\_central = -mu / r\_norm \*\* 3 \* r  
  
 *# Возмущение от J2* a\_perturb = a\_grav(dateTime, r)  
  
 a\_total = a\_central + a\_perturb  
  
 return np.concatenate((v, a\_total)) *# [dr/dt, dv/dt]*def rk4\_step(y, t, dt, dydt\_func):  
 k1 = dydt\_func(t, y)  
 k2 = dydt\_func(t + dt / 2, y + dt / 2 \* k1)  
 k3 = dydt\_func(t + dt / 2, y + dt / 2 \* k2)  
 k4 = dydt\_func(t + dt, y + dt \* k3)  
  
 y\_new = y + dt / 6 \* (k1 + 2 \* k2 + 2 \* k3 + k4)  
 return y\_new  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 np.set\_printoptions(suppress=True, formatter={'float\_kind': '{:.6f}'.format})  
 print(f"Старт моделирования: {t0}")  
 current\_y = y0.copy()  
  
 steps = int(endTime // dt) + 1  
 trajectory = []  
 times = []  
  
 for i in range(1, steps):  
 t = (i - 1) \* dt *# текущее время в секундах от начала* current\_time = t0 + timedelta(seconds=t + dt)  
  
 *# Интегрируем на один шаг* current\_y = rk4\_step(current\_y, t, dt, dydt)  
  
 *# Сохраняем результаты* trajectory.append(current\_y.copy())  
 times.append(current\_time)  
  
 *# Выводим вектор состояния* print(f"[{current\_time}] Вектор состояния y = {current\_y}")  
  
 print("Моделирование завершено.")

## Приложение Г. j2\_gravitational\_acceleration

import numpy as np  
import math  
from astropy.time import Time  
  
*# Функция для расчета ускорения с учетом возмущений J2*def a\_grav(dateTime, r0):  
 A = -19089.451590  
 B = 8640184.812866  
 C = 0.093104  
 D = -6.2e-6  
 JD0 = 2451545  
 JDD = 36525  
 DS2R = 7.272205216643039903848712e-5  
 J2 = 1.08262668355e-3 *# Коэффициент второй зональной гармоники* earth\_radius = 6.378137e6 *# Средний радиус Земли в метрах* mu = 3.986004418e14 *# Гравитационная постоянная Земли м^3/с^2  
  
 # Расчёт юлианской даты с использованием astropy* JD = Time(dateTime).jd *# Преобразование datetime в юлианскую дату* t = (JD - JD0) / JDD  
 f = 86400 \* (JD % 1.0)  
 alfa = DS2R \* ((A + (B + (C + D \* t) \* t) \* t) + f)  
 alfa = alfa % (2 \* math.pi)  
 if alfa < 0:  
 alfa += 2 \* math.pi  
  
 *# Матрица перехода от J200 в ГСК* Mj2kGr = np.array([[math.cos(alfa), -math.sin(alfa), 0],  
 [math.sin(alfa), math.cos(alfa), 0],  
 [0 , 0 , 1]])  
  
 *# Радиус-вектор в ГСК* rGsk = np.dot(Mj2kGr, r0 \* 1000)  
  
 *# Расчёт возмущающего ускорения* rMod = np.sqrt(np.dot(rGsk, rGsk))  
 a\_J2 = (-1.5 \* J2 \* (mu / rMod \*\* 2) \* ((earth\_radius / rMod) \*\* 2) \*  
 np.array([(1 - 5 \* (rGsk[2] / rMod) \*\* 2) \* rGsk[0] / rMod,  
 (1 - 5 \* (rGsk[2] / rMod) \*\* 2) \* rGsk[1] / rMod,  
 (3 - 5 \* (rGsk[2] / rMod) \*\* 2) \* rGsk[2] / rMod]))  
  
 *# Ускорение от несфееричности в ИСК* a\_grav = np.dot(Mj2kGr.T, a\_J2) / 1000  
 return a\_grav  
  
 *# Расчёт координат Солнца* JD = Time(dateTime).jd *# Преобразование datetime в юлианскую дату* T = (JD - 2451545.0) / 36525.0 *# Модифицированная Юлианская дата* M = math.radians(357.5226 + 35999.049 \* T) *# Средняя аномалия* lm = om + M + math.radians(6892/3600)\*math.sin(M) + math.radians(72/60)\*math.sin(2\*M)  
 rs = np.array([ math.cos(lm),  
 math.sin(lm)\*math.cos(eps),  
 math.sin(lm)\*math.sin(eps)]) \* R\_orbE  
  
 *# Расчёт углов* dif = rs - r0 *# Угол между вектором КА-Солнце и вектором КА-Земля* phi = math.acos(np.dot(-r0, dif) / (np.linalg.norm(r0) \* np.linalg.norm(dif))) *# Угол между КА и Солнцем*