Федеральное государственное автономное образовательное учреждение высшего образования
«Российский университет дружбы народов имени Патриса Лумумбы»
Инженерная академия
Департамент механики и процессов управления

ОТЧЕТ

По Лабораторной работе №1 по Механике Космического Полета.
4 Вариант.

Направление: 01.03.02 Прикладная математика и информатика.
(код направления / название направления)

Профиль: Математические методы механики полёта ракет-носителей и космических аппаратов.
(название профиля)

Тема: Уравнение Кеплера для эллиптической орбиты.
(название лабораторной / курсовой)

Выполнено студентом: Критским Матвеем Димитриевичем
(ФИО)

Группа: ИПМбд-02-22

№ студенческого: 1132226149

Москва, 2023

Теоретическая часть:

Благодаря миссии "Луна-10" ученые установили несферичность гравитационного потенциала Луны и впервые получили его точную модель. Также было уточнено значение массы Луны.

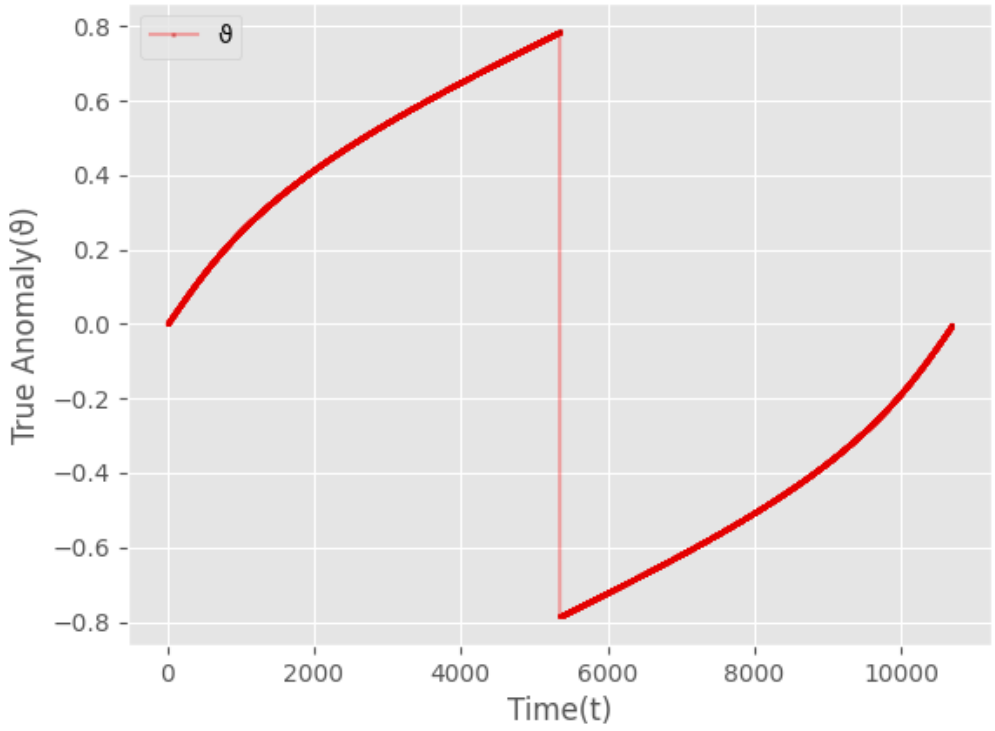
Данные миссии "Луна-10":

Ra(радиус-вектор апоцентра) = 1017 км
Rp(радиус-вектор перицентра) = 350 км
a(большая полуось) = (Ra + Rp) / 2 = 683.5 км
e(эксцентриситет) = (Ra - Rp) / (Ra + Rp) = 0.487930
T(период обращения) = 2 часа 58 минут 15 секунд
ИЛИ
10695 секунд

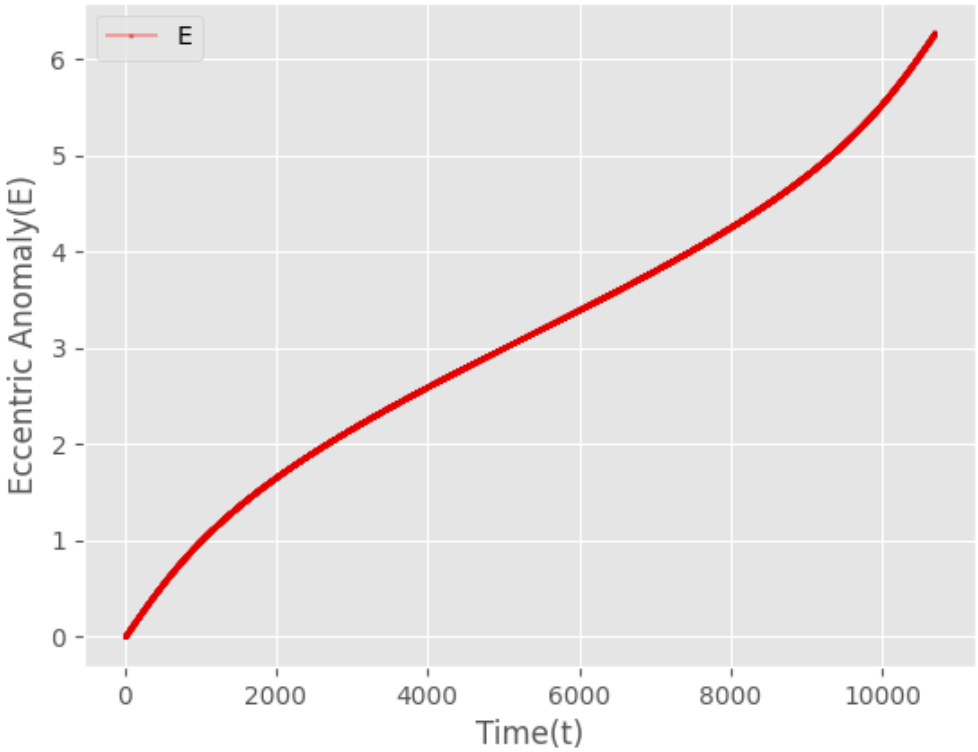
Практическая часть:

Графики зависимости аномалий от времени t:

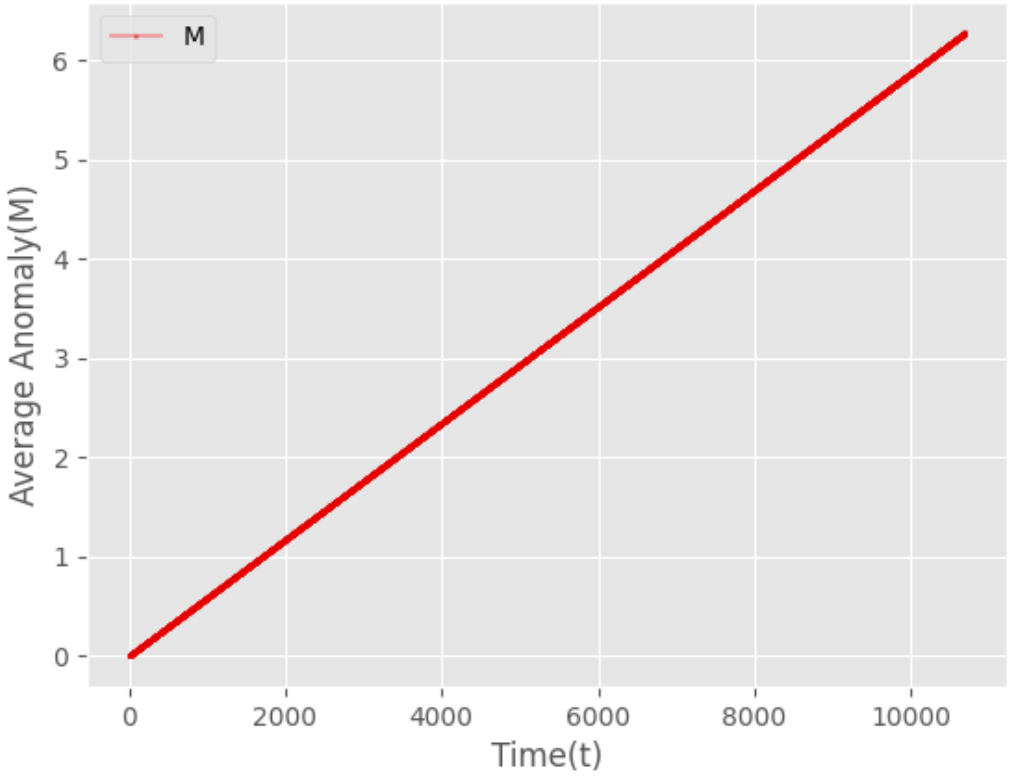
Истинной (ϑ(t)):



Эксцентрической E(t):



Средней M(t):



Приложение:
Ссылка на код в GitHub: https://github.com/rudnmk/MSF/tree/main/1

Код основной программы :

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#define SIGMA 0.001

float* IterationMethod(float, int);
float* HalfDivMethod(float, float, float, int);
float* GoldenRatioMethod(float, float, float, int);
float* NewtonMethod(float, int);
float* LaguerreMethod(float, int);

int DataForM(float, int);
int DataForE(float, int);
int DataForTHETA(float, int);
```

```
int main() {
    float Ra = 1017.0;
    float Rp = 350.0;
    float a = (Ra + Rp) / 2;
    float e = (Ra - Rp) / (Ra + Rp);
    float T = 10695.0;
    DataForM(e, T);
    DataForE(e, T);
    DataForTHETA(e, T);
    return 0;
}


float* IterationMethod(float e, int T) {
    float* E_ARR = (float*)malloc(T * sizeof(float));
    E_ARR[0] = 0;
    for(int i = 1; i < T; i++) {
        float M = (2 * 3.14 / T) * i;
        E_ARR[i] = e * sin(E_ARR[i - 1]) + M;
    }

    return E_ARR;
}

float* HalfDivMethod(float e, float A, float B, int T) {
    int i = 0;
    float* E_ARR = (float*)malloc(T * sizeof(float));
    int flag = 0;
    float C;

    while ((i < T) && ((B - A) > SIGMA)) {
        float M = (2 * 3.14 / T) * i;
        if (flag == 0) {
            C = (B + A) / 2;
        }

        else {
            C = (B - A) / 2;
        }
```

```
        float Ea = A - e * sin(A) - M;
        float Eb = B - e * sin(B) - M;
        float Ec = C - e * sin(C) - M;

        if (Ea * Ec <= 0.0) {
            B = C;
        }
        else {
            A = C;
        }

        if ((A < 0.0 && B < 0.0) || (A > 0.0 && B > 0.0)) {
            flag = 1;
        }

        E_ARR[i] = Ec;
        i++;

        if (E_ARR[i] == E_ARR[i - 1]) {
            break;
        }
    }

    return E_ARR;
}

float* GoldenRatioMethod(float e, float A, float B, int T) {
    int i = 0;
    float* E_ARR = (float*)malloc(T * sizeof(float));
    int flag = 0;
    float C;

    while ((i < T) && ((B - A) > SIGMA)) {
        int M = (2 * 3.14 / T) * i;
        if (flag == 0) {
            C = (B + A) / 1.618;
        }

        else {
            C = (B - A) / 1.618;
```

```
        }

        float Ea = A - e * sin(A) - M;
        float Eb = B - e * sin(B) - M;
        float Ec = C - e * sin(C) - M;

        if (Ea * Ec <= 0) {
            B = C;
        }
        else {
            A = C;
        }

        if ((A < 0 && B < 0) || (A > 0 && B > 0)) {
            flag = 1;
        }

        E_ARR[i] = Ec;
        i++;
    }

    return E_ARR;
}

float* NewtonMethod(float e, int T) {
    float* E_ARR = (float*)malloc(T * sizeof(float));
    E_ARR[0] = 0;
    for (int i = 1; i < T; i++) {
        float M = (2 * 3.14 / T) * i;
        float fE = E_ARR[i - 1] - e * sin(E_ARR[i - 1]) - M;
        float DfE = 1 - e * cos(E_ARR[i - 1]);
        E_ARR[i] = E_ARR[i - 1] - (fE / DfE);
    }
    return E_ARR;
}

float* LaguerreMethod(float e, int T) {
    float* E_ARR = (float*)malloc(T * sizeof(float));
    int n = 3;
    E_ARR[0] = 0;
```

```
    for (int i = 1; i < T; i++) {
        float M = (2 * 3.14 / T) * i;
        float fE = E_ARR[i - 1] - e * sin(E_ARR[i - 1]) - M;
        float DfE = 1 - e * cos(E_ARR[i - 1]);
        float DDfE = e * sin(E_ARR[i - 1]);
        float hE = fabs((n - 1) * ((n - 1) * pow(DfE, 2) - n * fE * DDfE));
        E_ARR[i] = E_ARR[i - 1] - ((fE * n) / (DfE + pow(hE, (1 / 2))));
    }
    return E_ARR;
}

int DataForM(float e, int T) {
    FILE *file = fopen("M_Data.txt", "w");
    for (int i = 0; i < T; i++) {
        float M = (2 * 3.14 / T) * i;
        fprintf(file, "%f \n", M);
    }
    fclose(file);
    return 0;
}

int DataForE(float e, int T) {
    FILE *file = fopen("E_Data.txt", "w");
    int iterations = T;
    float* E_ARR = LaguerreMethod(e, T);
    fprintf(file, "%f \n", E_ARR[0]);
    for (int i = 1; i < T; i++) {
        fprintf(file, "%f \n", E_ARR[i]);
    }
    fclose(file);
    return 0;
}

int DataForTHETA(float e, int T) {
    FILE *file = fopen("THETA_Data.txt", "w");
    int iterations = T;
    float* E_ARR = LaguerreMethod(e, T);
    fprintf(file, "%f \n", 0);
    for (int i = 1; i < T; i++) {
        float THETA = atan(tan(E_ARR[i] / 2) * pow((1 + e) / (1 - e), (1 / 2))) / 2;
```

```
        fprintf(file, "%f \n", THETA);
    }
    fclose(file);
    return 0;
}
```