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.
Вариант 4.

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

Профиль: Математические методы механики полета раке-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:

Название миссии: "Луна-10"

Факты о миссии:

3 апреля 1966 года автоматическая станция "Луна-10" вышла на орбиту Луны и стала ее первым в истории искусственным спутником.

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":

μ(Гравитационная постоянная Луны) = 4902.8000 Н

R(Радиус Луны) = 1737.1 км;

R_a(радиус-вектор апоцентра) = R + 1017 = 2754.1 км

R_p (радиус-вектор перицентра) = R + 350 = 2087.1 км

a(большая полуось) = $\frac{Ra + Rp}{2}$ = 2420.6 км

e(эксцентриситет) = $\frac{Ra + Rp}{2 \cdot a}$ = 0.137776

PI(число Пи) = 3.1415926

T(период обращения) = 2 часа 58 минут 15 секунд

ИЛИ

10695 секунд

Используемые формулы для расчета аномалий:

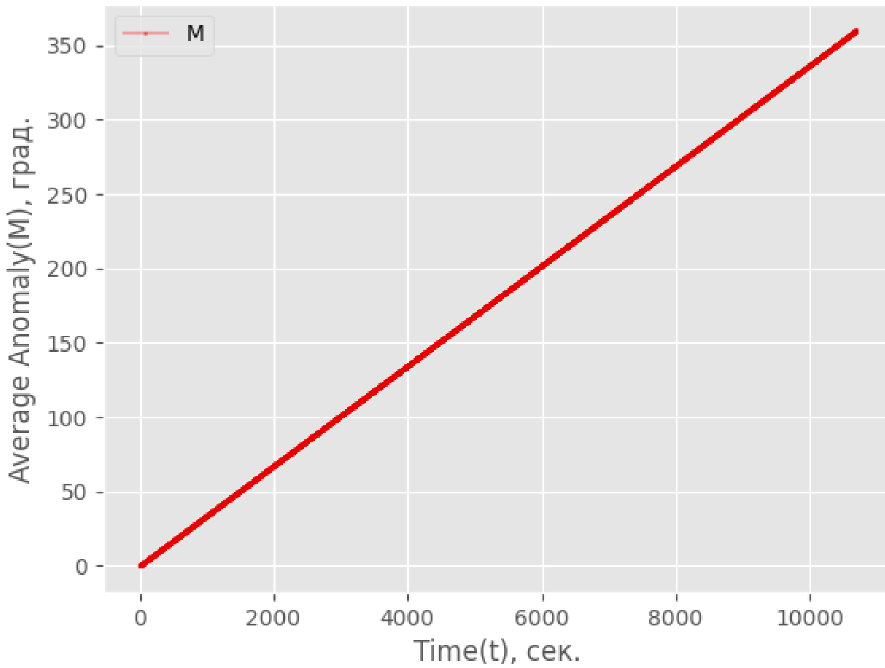
M(t) = $\frac{2 \cdot \pi}{T} \cdot t$

E(t) = $M + e \cdot \sin(pastE)$ - метод последовательных итераций, где past_E - предыдущая итерация, а E - текущая. Начальная итерация равна нулю.

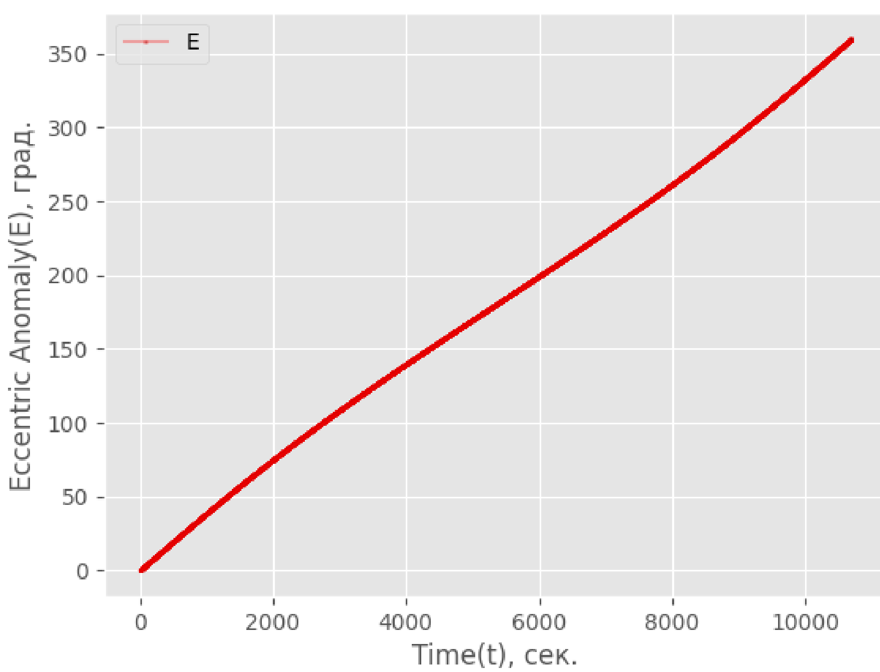
$\vartheta(\mathrm{t}) = \arctan\left(\tan\left(\frac{E}{2}\right) \cdot \sqrt{(1 + e) \cdot (1 - e)}\right) \cdot 2$

Благодаря этим формулам, мы можем построить графики трех аномалий в зависимости от пройденного времени. (M зависит от t; E мы находим, используя M; ϑ мы находим с помощью E.

Графики зависимостей аномалий от времени t:
Средней(M(t)): (рис. 1)



Эксцентрической(E(t)): (рис. 2)



Истинной(ϑ (t)): (рис. 3)

Time(t), сек.

Различные методы нахождения эксцентрической аномалии расписаны в программе METHODS.c, тогда как код для подсчета предыдущих значений и последующих находится в файле task_1.c (см. приложение).

Зная истинную аномалию ϑ и факальный параметр p, можно подсчитать радиус-вектор r, трансверсальную скорость V_n, радиальную скорость V_r и результирующую скорость V с помощью следующих формул:

$$p = a \cdot (1 - e^2) = 2374.652 \text{ км}$$

$$r = \frac{p}{1 + e \cdot \cos(\vartheta)} \text{ км}$$

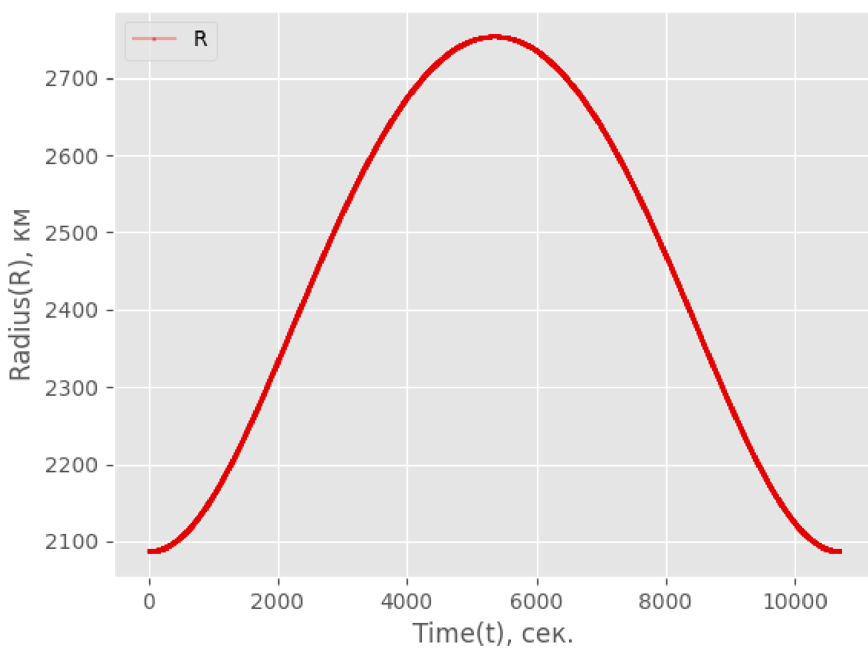
$$V_n = \sqrt{\frac{\mu}{p}} \cdot (1 + e \cdot \cos(\vartheta)) \text{ км/с}$$

$$V_r = \sqrt{\frac{\mu}{p}} \cdot e \cdot \sin(\vartheta) \text{ км/с}$$

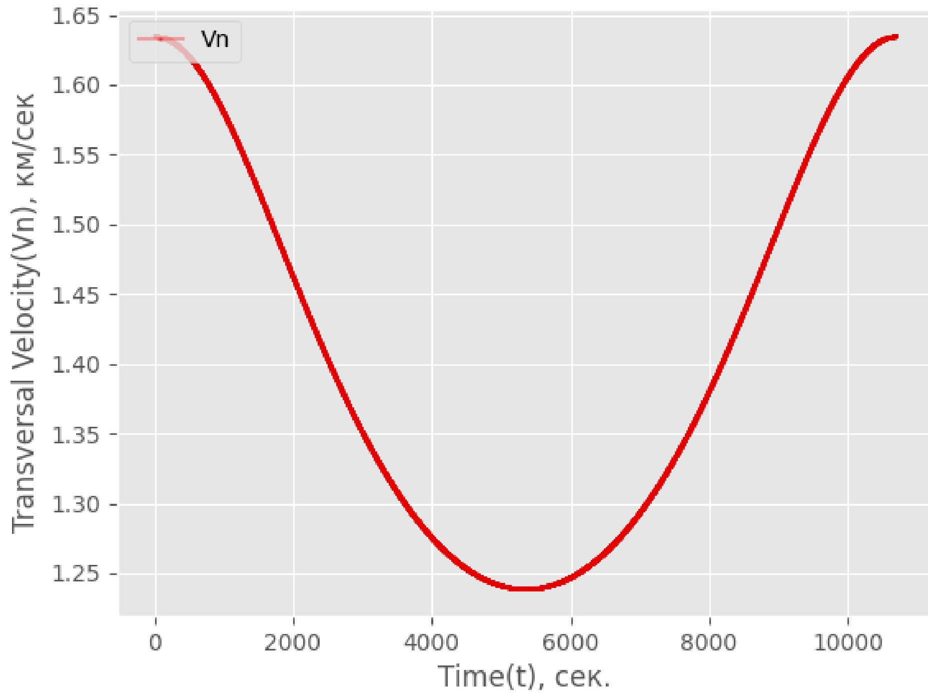
$$V = \sqrt{V_r^2 + V_n^2} \text{ км/с}$$

Графики зависимостей:

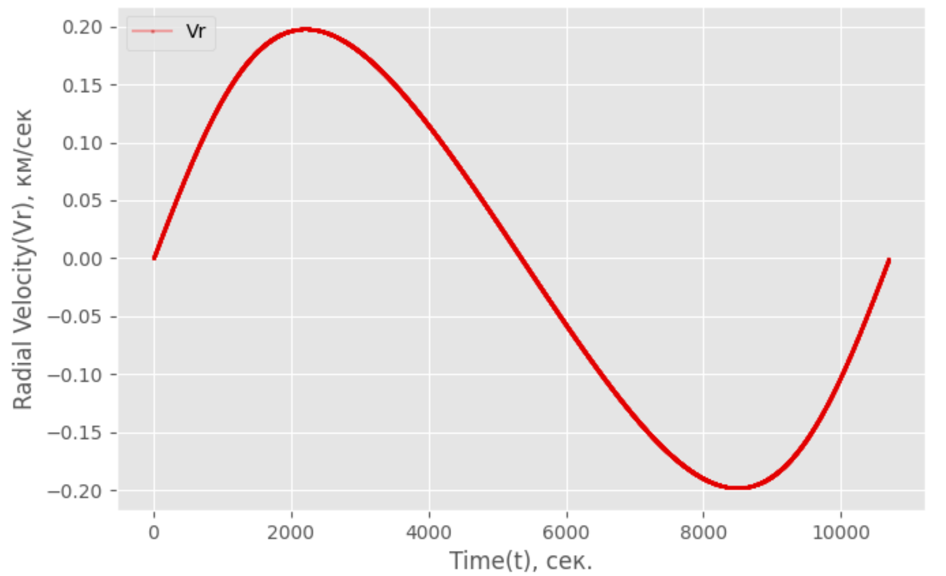
Радиус-вектора от t r(t): (рис. 4)



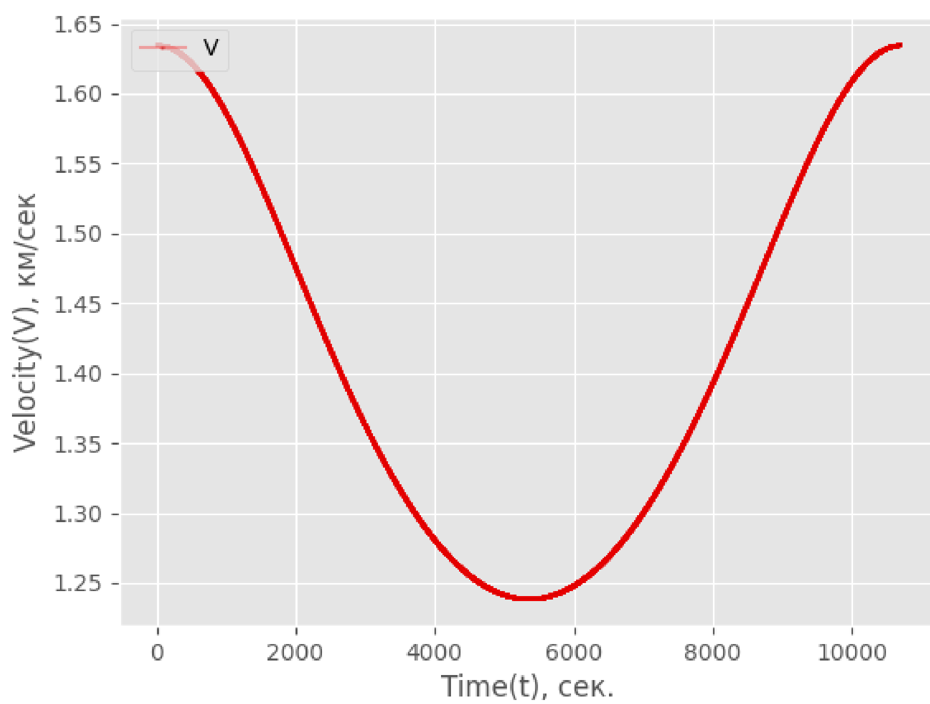
Трансверсальной скорости от t $V_n(t)$: (рис.5)



Радиальной скорости от t $V_r(t)$: (рис. 6)



Результирующей скорости от t V(t): (рис. 7)



Скорости V_r, V_n и V изменяются по закону тригонометрических функций sin и cos, т.к. при их вычислениях используются sin и cos. V_n и V -

косинусоидная, а V_r - синусоидная. Радиальная скорость V_r принимает значения 0 в точках t = 0, $t = \frac{T}{2}$ и t = T => радиальная скорость обнуляется при достижении КА точек апоцентра и перицентра.

Из рис. 5 видно, что V_n принимает значения в пределе [1.238917; 1.634853] км/с. Можно заметить, что V_n принимает максимальное и минимальное значения, когда радиальная скорость V_r равна 0, т.е. углы M, E и ϑ равны 0 град., 180 град. и 360 град.

Из рис. 6 видно, что V_r принимает значения в пределе [-0.197968; 0.197968] км/с. По модулю - [0; 197968]. Эти показатели достигаются, когда:

M = 74.288922 град.
E = 82.107467 град. V_r = 0.197968 км/с (макс. значение)
$\vartheta = 89.992233$ град.

M = 285.778403 град.
E = 277.959819 град. V_r = -0.197968 км/с (мин. значение)
$\vartheta = 270.006407$ град.

Из рис. 7 видно, что V принимает значения в пределе [1.238917; 1.63485] км/с. Также, можно заметить, что график результирующей скорости V имеет схожее поведение, что и график трансверсальной скорости V_n (такие же максимальные и минимальные значения, эти значения получаются при тех же условиях(углы M, E и ϑ равны 0;180;360 град.)). Это же следует из формулы результирующей скорости V. $V = \sqrt{V_r^2 + V_n^2}$(При $\vartheta = 0;180;360$ град. $\sin(\vartheta) = 0 => V_r = 0) => V = \sqrt{0 + V_n^2} = \sqrt{V_n^2} = V_n$.

Радиус-вектор r лежит в отрезке [2087 ; 2754] км. При чем при $t = \frac{T}{2}$ - $r = R_a$, т.е. принимает максимальное значение(апоцентра), а при t = 0 и t = T - $r = R_p$, т.е. принимает минимальное значение(перицентра). Сравнивая графики на рис. 4 и рис. 7, можно заметить, что при минимальном значении радиус-вектора (2087 км) результирующая скорость КА принимает максимальное значение (1.63485 км/с), а при максимальном значении радиус-вектора (2754 км) результирующая скорость КА принимает минимальное значение (1.238917 км/с). Следовательно, при приближении КА к космическому телу(в нашем случае к Луне), его скорость увеличивается и принимает максимально значение в точке перицентра, после начинает уменьшаться по мере отдаления КА от космического тела и достигает минимального значения в точке апоцентра.

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

Код программы:
METHODS.c:

```
#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);



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 + sqrt(hE)));
    }
    return E_ARR;
}
```

task_1.c:

```
#include<stdio.h>

#include<math.h>

#include<stdlib.h>

#define PI 3.1415926

#define μ 4902.8000  //Гравитационный параметр Луны

int DataCollection(float, float, float);



int main() {

    float Ra = 2754.1;

    float Rp = 2087.1;

    float a = (Ra + Rp) / 2.0;

    float e = (Ra - Rp) / (2.0 * a);

    float p = a * (1 - pow(e, 2.0));

    DataCollection(a, e, p);

    return 0;

}

int DataCollection(float a, float e, float p) {

    FILE* T_ = fopen("T.txt", "w");

    FILE* M_dat = fopen("M_Data.txt", "w");

    FILE* E_dat = fopen("E_Data.txt", "w");

    FILE* THETA_dat = fopen("THETA_Data.txt", "w");

    FILE* R_dat = fopen("Radius_Data.txt", "w");

    FILE* Vn_dat = fopen("VelN_Data.txt", "w");

    FILE* Vr_dat = fopen("VelR_Data.txt", "w");

    FILE* V_dat = fopen("Velocity_Data.txt", "w");
```

```
float past_E = 0.0;
float T = 10695.0;
fprintf(T_, "%f", T);
fclose(T_);
for (float i = 0.0; i < T; i++) {
    float M = (2.0 * PI / T) * i;
    float E = e * sin(past_E) + M;
    float THETA = atan(tan(past_E / 2.0) * sqrt((1 + e) / (1 - e))) * 2.0;
    float radius = p / (1 + e * cos(THETA));
    float Vn = sqrt(μ / p) * (1 + e * cos(THETA));
    float Vr = sqrt(μ / p) * e * sin(THETA);
    float V = sqrt(pow(Vn, 2.0) + pow(Vr, 2.0));

    fprintf(M_dat, "%f \n", (M * 180.0 / PI));
    fprintf(E_dat, "%f \n", (E * 180.0 / PI));
    if (THETA < 0) {
        fprintf(THETA_dat, "%f \n", (THETA * 180.0 / PI + 360.0));
    }
    else {
        fprintf(THETA_dat, "%f \n", (THETA * 180.0 / PI));
    }
    fprintf(R_dat, "%f \n", radius);
    fprintf(Vn_dat, "%f \n", Vn);
    fprintf(Vr_dat, "%f \n", Vr);
    fprintf(V_dat, "%f \n", V);

    past_E = E;
}
```

```
    fclose(M_dat);
    fclose(E_dat);
    fclose(THETA_dat);
    fclose(R_dat);
    fclose(Vn_dat);
    fclose(Vr_dat);
    fclose(V_dat);
}
```