Отчет по задаче практикума «Задачи вариационного исчисления и оптимального управления»

Лобзин Фёдор, 407 группа

14 мая 2021 г.

1 Постановка задачи

Задача. Найти решение задачи

$$\begin{cases} B_0 = \int_0^T (\ddot{x}^2 - \dot{x}^2 - x^2)dt \to inf, \\ x(0) = x(T) = 0,\ \dot{x}(T) = 1, \\ T = \{0.1;, 1.0; 2.0; 3.0; 3.5; 10.0; 20.0\}. \end{cases}$$

2 Формализация

Обозначим $x_1(t) = x(t),\ \ x_2(t) = \dot{x}(t),\ \ u(t) = \ddot{x}(t)$. Тогда:

$$B_0(x_1(\cdot), x_2(\cdot), u(\cdot)) = \int_0^T (u(t)^2 - x_2(t)^2 - x_1(t)^2)dt \to inf. \tag{1}$$

$$\dot{x_1}(t) - x_2(t) = 0,\ \dot{x_2}(t) - u(t) = 0,\ \forall t \in [0, 1]. \tag{2}$$

$$x_1(0) = 0,\ x_1(T) = 0,\ x_2(T) = 1. \tag{3}$$

3 Необходимые условия

1) Функция Понтрягина: $H = p_1x_2 + p_2u - \lambda_0\left(u(t)^2 - x_2(t)^2 - x_1(t)^2\right)$
Терминант: $l = \lambda_{10}x_1(0) + \lambda_{01}x_1(T) + \lambda_{11}x_2(T)$.
Где $\lambda_0,\ \lambda_{ij}$- числа, $p_1(t), p_2(t)$- функции.

2) Система условий принципа максимума в задаче (1-3):

   a) уравнение Эйлера-Лагранжа:

$$\dot{\widehat{p_i}} = -\frac{\partial \widehat{H}}{\partial x_i} \Rightarrow \dot{p_1} = -2\widehat{\lambda_0}x_1,\ \dot{p_2} = -p_1 - 2\widehat{\lambda_0}x_2.$$

   b) условия трансверсальности:

$$\widehat{p_i}(t_k) = (-1)^k\frac{\partial \widehat{l}}{\partial x_i(t_k)} \Rightarrow p_1(0) = \lambda_{10},\ p_1(T) = -\lambda_{01},\ p_2(0) = 0,\ p_2(T) = -\lambda_{11}$$

   c) условие оптимальности:

$$\widehat{u} = argabsmax\,(H) = argabsmax\left(p_2u - \widehat{\lambda_0}\left(u^2\right)\right)$$

Если $\lambda_0 = 0$, то $\frac{\partial^2 \widehat{H}}{\partial u^2} = 0$, что нас не интересует, следовательно $\lambda_0 \neq 0$, тогда из условий $\frac{\partial \widehat{H}}{\partial u} = 0$ и $\frac{\partial^2 \widehat{H}}{\partial u^2} < 0$ получаем:

$$p_2 - 2\widehat{\lambda_0}\widehat{u} = 0,\ \widehat{\lambda_0} > 0$$

d) условия стационарности: не существенны, тк отрезок стационарен.

e) условия не жесткости отсутствуют, так как нет ограничений типа неравенств.

f) условие неотрицательности: $\widehat{\lambda_0} \geq 0$

g) $\lambda_0,\ \lambda_{ij}$- числа, $p_1(t), p_2(t)$- функции - не равны одновременно нулю.

3) Так как $\lambda_0 \neq 0$ из условия оптимальности можно выбрать (без ограничения общности) $u = p_2$, тогда, после подстановки в (1) и (2) получается система (с данного момента все рассуждения будут касательно экстремали):

$$\begin{cases} \dot{x_1} = x_2, \\ \dot{x_2} = p_2, \\ \dot{p_1} = -x_1, \\ \dot{p_2} = -p_1 - x_2. \\ x_1(0) = 0,\ x_1(T) = 0, \\ x_2(T) = 1,\ p_2(0) = 0. \end{cases}$$

Пусть:

$$\begin{cases} x_1 = x_{1c} + \alpha x_{1\alpha} + \beta x_{1\beta}, \\ x_2 = x_{2c} + \alpha x_{2\alpha} + \beta x_{2\beta}, \\ p_1 = p_{1c} + \alpha p_{1\alpha} + \beta p_{1\beta}, \\ p_2 = p_{2c} + \alpha p_{2\alpha} + \beta p_{2\beta}, \end{cases}$$

Тогда 3), при условии, что $x_2(0) = \alpha$ и $p_1(0) = \beta$, α, β определим по правым граничным условиям, и систему дифференциальных уравнений:

$$\begin{cases} \dot{x_{1c}} = x_{2c}, \\ \dot{x_{1\alpha}} = x_{2\alpha}, \\ \dot{x_{1\beta}} = x_{2\beta}, \\ \dot{x_{2c}} = p_{2c}, \\ \dot{x_{2\alpha}} = p_{2\alpha}, \\ \dot{x_{2\beta}} = p_{2\beta}, \\ \dot{p_{1c}} = -x_{1c}, \\ \dot{p_{1\alpha}} = -x_{1\alpha}, \\ \dot{p_{1\beta}} = -x_{1\beta}, \\ \dot{p_{2c}} = -p_{1c} - x_{2c}, \\ \dot{p_{2\alpha}} = -p_{1\alpha} - x_{2\alpha},, \\ \dot{p_{2\beta}} = -p_{1\beta} - x_{2\beta}, \\ x_{1c}(0) = 0,\ x_{1\alpha}(0) = 0,\ x_{1\beta}(0) = 0, \\ x_{2c}(0) = 0,\ x_{2\alpha}(0) = 1,\ x_{2\beta}(0) = 0, \\ p_{1c}(0) = 0,\ p_{1\alpha}(0) = 0,\ p_{1\beta}(0) = 1, \\ p_{1c}(0) = 0,\ p_{1\alpha}(0) = 0,\ p_{1\beta}(0) = 0. \end{cases}$$

Будем решать эту задачу методом Рунге-Кутты 5-ого порядка с использованием расчетных формул Дормана-Принса 5(4) DDOPRI5 с автоматическим выбором шага (то есть с контролем относительной локальной погрешности на шаге по правилу Рунге). Для вычисления глобальных погрешностей будем

использовать оценку с использованием максимального собственного значения симметрической матрицы $A' = \frac{1}{2}(A + A^T)$, где матрица A - матрица производных исходной системы дифференциальных уравнений. Вычислим матрицу A:

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 \end{pmatrix}$$

Тогда A':

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \end{pmatrix}$$

Матрица $A^{'}$- блочная, а именно:

$$A^{'} = \begin{pmatrix} 0 & E & -E & 0 \\ E & 0 & 0 & 0 \\ -E & 0 & 0 & -E \\ 0 & 0 & -E & 0 \end{pmatrix}$$

Следовательно достаточно найти максимальное по модулю собственное значение матрицы:

$$B = \begin{pmatrix} 0 & 1 & -1 & 0 \\ 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Все собственные значение этой матрицы: $\lambda_{1,2} = \pm\frac{1-\sqrt{5}}{2},\ \lambda_{3,4} = \frac{1\pm\sqrt{5}}{2}$,откуда максимально по модулю: $\lambda = \frac{1+\sqrt{5}}{2}$.

После определения α, β (с точность до $\varepsilon = 10^{-9}$) Экстремальные значение интеграла определим

как $x_3(T)$ в решении дифференциального уравнения:

$$\begin{cases} \dot{x_1} = x_2, \\ \dot{x_2} = p_2, \\ \dot{p_1} = -x_1, \\ \dot{p_2} = -p_1 - x_2. \\ \dot{x_3} = (p_2)^2 - (x_2)^2 - (x_1)^2. \\ x_1(0) = 0,\ x_2(0) = \alpha,\ x_3(0) = 0, \\ p_1(0) = \beta,\ p_2(0) = 0. \end{cases}$$

Аналогично решим эту задачу методом Рунге-Кутты 5-ого порядка с использованием расчетных формул Дормана-Принса 5(4) DDOPRI5 с автоматическим выбором шага (то есть с контролем относительной локальной погрешности на шаге по правилу Рунге).

4 Достаточные условия

. По принципу максимума (минимума) условие Лежандра выполняется автоматически, поэтому остается проверить Условие Якоби. Рассмотрим первую вариацию функционала, приравняем ее к нулю:

$$\begin{cases} \Delta\dot{x}_1 = \Delta x_2, \\ \Delta\dot{x}_2 = \Delta u, \\ \Delta x_1(0) = 0,\ \Delta x_1(T) = 0,\ \Delta x_2(T) = 0, \end{cases}$$

Тогда положительная определенность второй вариации функционала: $\int_0^T (\Delta u)^2 - (\Delta x_2)^2 - (\Delta x_1)^2 dt$, при условии, равенства нулю первой его вариации, (по усиленному условию Якоби) следует из того, что: $det \begin{pmatrix} \Delta x_1^1 & \Delta x_1^2 \\ \Delta x_2^1 & \Delta x_2^2 \end{pmatrix} \neq 0$ — на отрезке $[0, \tau]$ где Δx_i^j- решения уравнений (уравнения аналогичны, так как вторая вариация функционала совпадает с изначальным функционалом с точности до замены переменных):

$$\begin{cases} \Delta\dot{x}_1 = \Delta x_2, \\ \Delta\dot{x}_2 = q_2, \\ \dot{q_1} = -\Delta x_1, \\ \dot{q_2} = -q_1 - \Delta x_2. \\ x_1(0) = 0,\ x_2(0) = 1, \\ q_1(0) = 0,\ q_2(0) = 0. \end{cases} \qquad \begin{cases} \Delta\dot{x}_1 = \Delta x_2, \\ \Delta\dot{x}_2 = q_2, \\ \dot{q_1} = -\Delta x_1, \\ \dot{q_2} = -q_1 - \Delta x_2. \\ x_1(0) = 0,\ x_2(0) = 0, \\ q_1(0) = 1,\ q_2(0) = 0. \end{cases}$$

Решим параллельно два этих дифференциальных уравнения, следя за знаком выражения:

$$det \left(\begin{pmatrix} \Delta x_1^1(\varepsilon) & \Delta x_1^2(\varepsilon) \\ \Delta x_2^1(\varepsilon) & \Delta x_2^2(\varepsilon) \end{pmatrix} \cdot \begin{pmatrix} \Delta x_1^1(\tau) & \Delta x_1^2(\tau) \\ \Delta x_2^1(\tau) & \Delta x_2^2(\tau) \end{pmatrix} \right)$$

Перемена знака на отрезке $T \in [0.1, T]$ происходит только при $\widehat{\tau} \approx 3.278982474066 \Rightarrow$ по теореме Коши о промежуточном значении:

$$det \begin{pmatrix} \Delta x_1^1(\widehat{\tau}) & \Delta x_1^2(\widehat{\tau}) \\ \Delta x_2^1(\widehat{\tau}) & \Delta x_2^2(\widehat{\tau}) \end{pmatrix} \approx 0$$

И в окрестности $\widehat{\tau}$ это выражение принимает значение ноль. Откуда следует, что усиленное условие Якоби выполняется, при $T \in \{0.1, 1, 2, 3\}$, и, как следствие, полученная экстремаль доставляет сильный минимум. При $T \in \{3.5, 10\}$ не выполняется условие Якоби $\Rightarrow$ полученная экстремаль не доставялет ни слабый, ни сильный минимум.

5 Вычислительный эксперимент

В данном разделе будем обозначать за ε требуемое значение точности вычислений (условие остановки метода),за T- правый конец исходного отрезка, за $Integ$-значение интеграла . Приведем результаты работы метода, реализованного на языке C для значений $T \in \{0.1, 1, 2, 3, 3.5, 10\}$, точностей $\varepsilon \in \{10^{-5}, 10^{-7}, 10^{-9}\}$:

T	α	β	$Integ(\varepsilon = 10^{-5})$	$Integ(\varepsilon = 10^{-7})$	$Integ(\varepsilon = 10^{-9})$
$T = 0.1$	-0.500249978	-299.800245261	29.980213697	29.979963180	29.979957171
$T = 1$	-0.529658917	-2.834528011	2.773546375	2.773533707	2.773533551
$T = 2$	-0.704254202	-0.715004208	0.820260153	0.820260876	0.820260894
$T = 3$	-2.766260068	-1.285304259	-2.828364430	-2.828385179	-2.828385420
$T = 3.5$	3.123090349	1.433593811	4.927900332	4.927835132	4.927834492
$T = 10$	1.119116341	0.691834552	-0.301452020	-0.301448745	-0.301448713

Изучим главный член аппроксимации интеграла, ниже представлены значения $\Delta Integ$ и $\frac{\Delta Integ - \Delta_1 Integ}{\Delta_1 Integ - \Delta_2 Integ}$, при разных значениях T:

T	$Integ(e^{-5}) - Integ(e^{-7})$	$Integ(e^{-7}) - Integ(e^{-9})$	$\frac{Integ(e^{-5}) - Integ(e^{-7})}{Integ(e^{-7}) - Integ(e^{-9})}$
$T = 0.1$	0.000000020	0.000000001	14.667620040
$T = 1$	0.000002096	0.000000030	70.163193205
$T = 2$	0.000002216	0.000000023	98.196723355
$T = 3$	-0.000004676	-0.000000062	75.776092572
$T = 3.5$	0.000009675	0.000000106	91.599496980
$T = 10$	-0.000000035	-0.000000006	6.291720241

6 Листинг программы

```
#include <math.h>
#include <stdio.h>

#define max(x,y) ( (x) < (y) ? (y) : (x) )
#define min(x,y) ( (x) < (y) ? (x) : (y) )

#define ATTEMPTS 10
#define MIN_SCALE_FACTOR 0.125
#define MAX_SCALE_FACTOR 4.0
#define _USE_MATH_DEFINES

#define n 5

static double Runge_Kutta(double a, void(*f)(double, double, double*, double*),
double y[][n], double x,
double h);

void f(double a, double x, double *y, double *ans)
{
    ans[0] = y[1];
```

```
    ans[1] = y[3];
    ans[2] = -y[0];
    ans[3] = -y[2]-y[1];
    ans[4] = y[3]*y[3]-y[1]*y[1]-y[0]*y[0];

}
double lambda(double a, double x)
{

    return fabs(1+sqrt(5))/2;
}
double Prince_Dormand(double a, double alpha, double beta, void(*f)(double, double, double*
,double x, double h, double xmax, double *h_next, double tolerance) {
    double scale,integ[6],integr;
    double temp_y[2][n];
    double err = 0, global_err = 0;
    double l0 = 0, l1 = 0;
    double yy = 0;
    int i, j;
    alpha=2;
    int last_interval = 0;
    if (xmax < x || h <= 0.0) return -2;
    *h_next = h;
    for(i = 0; i < n; i++)
        y[1][i] = y[0][i];
    if (xmax == x) return 0;
    h = min(h, xmax - x);
    tolerance /= (xmax - x);
    for (i = 0; i < n; i++)
        temp_y[0][i] = y[0][i];
    while (x < xmax) {
        scale = 1.0;
        for (i = 0; i < ATTEMPTS; i++) {
            yy = 0;
            err = fabs(Runge_Kutta(a, f, temp_y, x, h));
            if (err == 0.0) { scale = MAX_SCALE_FACTOR; break; }
            for (j = 0; j < n; j++) yy += (temp_y[0][j] == 0.0) ? tolerance:
            fabs(temp_y[0][j]);
            scale = 0.8 * sqrt(sqrt(tolerance * yy / err));
            scale = min(max(scale, MIN_SCALE_FACTOR), MAX_SCALE_FACTOR);
            if (err < (tolerance * yy)) break;
            h *= scale;
            if (x + h > xmax) h = xmax - x;
            else if (x + h + 0.5 * h > xmax) h = 0.5 * h;
        }

        if (i >= ATTEMPTS) { *h_next = h * scale; return -1; };
        for (j = 0; j < n; j++) temp_y[0][j] = temp_y[1][j];
        x += h;
        l1 = lambda(a, x);
        global_err = err + global_err * h * max(l0, l1);
        l0 = l1;
        h *= scale;
        *h_next = h;
```

```
        if (last_interval) break;
        if (x + h > xmax) { last_interval = 1; h = xmax - x; }
        else if (x + h + 0.5 * h > xmax) h = 0.5 * h;
    }
    for (j = 0; j < n; j++) y[1][j] = temp_y[1][j];


    //printf("%.6f, %.6f \n ", alpha, beta);




    return global_err;
}


static double Runge_Kutta(double a, void(*f)(double, double, double*, double*)
, double y[][n], double x0, double h) {
    static const double r_45 = 1.0 / 45.0;
    static const double r_8_9 = 8.0 / 9.0;
    static const double r_6561 = 1.0 / 6561.0;
    static const double r_167904 = 1.0 / 167904.0;
    static const double r_142464 = 1.0 / 142464.0;
    static const double r_21369600 = 1.0 / 21369600.0;
    double y_tmp[n];
    double err = 0;
    double k1[n], k2[n], k3[n], k4[n], k5[n], k6[n], k7[n];
    double h5 = 0.2 * h;
    for (int i = 0; i < n; i++)
        y_tmp[i] = y[0][i];
    (*f)(a, x0, y_tmp, k1);
    for (int i = 0; i < n; i++)
        y_tmp[i] = y[0][i] + h5 * k1[i];
        (*f)(a, x0 + h5, y_tmp, k2);
    for (int i = 0; i < n; i++)
        y_tmp[i] = y[0][i] + h * (0.075 * k1[i] + 0.225 * k2[i]);
    (*f)(a, x0 + 0.3*h, y_tmp, k3);
    for (int i = 0; i < n; i++)
        y_tmp[i] = y[0][i] + h * r_45 * (44.0 * k1[i] - 168.0 * k2[i] + 160 * k3[i]);
    (*f)(a, x0 + 0.8*h, y_tmp, k4);
    for (int i = 0; i < n; i++)
        y_tmp[i] = y[0][i] + r_6561 * h * (19372.0 * k1[i]
        - 76080.0 * k2[i] + 64448.0 * k3[i] - 1908.0 * k4[i]);
    (*f)(a, x0 + r_8_9 * h, y_tmp, k5);
    for (int i = 0; i < n; i++)
        y_tmp[i] = y[0][i] + r_167904 * h * (477901.0 * k1[i] - 1806240.0 * k2[i]
        + 1495424.0 * k3[i] + 46746.0 * k4[i] - 45927.0 * k5[i]);
    (*f)(a, x0 + h, y_tmp, k6);
    for (int i = 0; i < n; i++)
        y_tmp[i] = y[0][i] + r_142464 * h * (12985.0 * k1[i] + 64000.0 * k3[i]
        + 92750.0 * k4[i] - 45927.0 * k5[i] + 18656.0 * k6[i]);
    (*f)(a, x0 + h, y_tmp, k7);
    for (int i = 0; i < n; i++)
    {
```

```
        y[1][i] = y[0][i] + r_21369600 * h * (1921409.0 * k1[i] + 9690880.0 * k3[i]
        + 13122270.0 * k4[i] - 5802111.0 * k5[i] + 1902912.0 * k6[i] + 534240.0 * k7[i]);
        err += fabs(r_21369600 * (26341.0 * k1[i] - 90880.0 * k3[i] + 790230.0 * k4[i]
        - 1086939.0 * k5[i] + 895488.0 * k6[i] - 534240.0 * k7[i]));
    }
    return err;
}

int main()
{
    double y[2][n];
    double y1[2][n];
        double y2[2][n];
        double err1, err2;
    double h = 0.000000085, eps;
    double a = 0.5, alpha,beta;
    double h_next, err;
    double x_start = 0, x_end  = 10;
    double pogr;
    double m= 1,b,c,x=0;
    //printf("%.6f", lambda(a,1));


    eps =1e-5;
    x_end=0.1;

    y[0][0] = 0;
    y[0][1] = -0.500249978275693;
    y[0][2] = -299.800245260743395;
    y[0][3] = 0;
    y[0][4] = 0;

    y1[0][0] = 0;
    y1[0][1] = -0.500249978275693;
    y1[0][2] = -299.800245260743395;
    y1[0][3] = 0;
    y1[0][4] = 0;

    y2[0][0] = 0;
    y2[0][1] = -0.500249978275693;
    y2[0][2] = -299.800245260743395;
    y2[0][3] = 0;
    y2[0][4] = 0;

    err = Prince_Dormand(a, alpha, beta, &f, &lambda, y, x_start, h, x_end, &h_next, eps);
    err1 = Prince_Dormand(a, alpha, beta,  &f, &lambda, y1, x_start, h, x_end, &h_next, eps
    err2 = Prince_Dormand(a, alpha, beta, &f, &lambda, y2, x_start, h, x_end, &h_next, eps=
    printf("$T=0.1$ & %.9f & %.9f & %.9f ", y[0][1], y[0][2], y[1][4]);
    printf("& %.9f ", y1[1][4]);
     printf("& %.9f ", y2[1][4]);
    printf("\\\\ \n");
     // printf("$T=0.1$ & %.9f & %.9f & %.9f \\\\ \n ", y[1][1]-y1[1][1], y1[1][1]-y2[1][1
```

```
x_end=1;

y[0][0] = 0;
y[0][1] = -0.529658916500840;
y[0][2] =-2.834528010763926 ;
y[0][3] = 0;
y[0][4] = 0;


y1[0][0] = 0;
y1[0][1] = -0.529658916500840;
y1[0][2] =-2.834528010763926 ;
y1[0][3] = 0;
y1[0][4] = 0;

y2[0][0] = 0;
y2[0][1] = -0.529658916500840;
y2[0][2] =-2.834528010763926 ;
y2[0][3] = 0;
y2[0][4] = 0;


err = Prince_Dormand(a, alpha, beta, &f, &lambda, y, x_start, h, x_end, &h_next, eps);
err1 = Prince_Dormand(a, alpha, beta, &f, &lambda, y1, x_start, h, x_end, &h_next, eps
err2 = Prince_Dormand(a, alpha, beta, &f, &lambda, y2, x_start, h, x_end, &h_next, eps
printf("$T=1$ & %.9f & %.9f & %.9f ", y[0][1], y[0][2], y[1][4]);
printf("& %.9f ", y1[1][4]);
printf("& %.9f ", y2[1][4]);
printf("\\\\ \n");
// printf("$T=1$ & %.9f & %.9f & %.9f \\\\ \n", y[1][1]-y1[1][1], y1[1][1]-y2[1][1], (


x_end=2;
y[0][0] = 0;
y[0][1] = -0.704254201907537;
y[0][2] =  -0.715004207651468 ;
y[0][3] = 0;
y[0][4] = 0;

y1[0][0] = 0;
y1[0][1] = -0.704254201907537;
y1[0][2] =  -0.715004207651468 ;
y1[0][3] = 0;
y1[0][4] = 0;

y2[0][0] = 0;
y2[0][1] = -0.704254201907537;
y2[0][2] =  -0.715004207651468 ;
y2[0][3] = 0;
y2[0][4] = 0;

err = Prince_Dormand(a, alpha, beta, &f, &lambda, y, x_start, h, x_end, &h_next, eps);
```

```
err1 = Prince_Dormand(a, alpha, beta,  &f, &lambda, y1, x_start, h, x_end, &h_next, eps
err2 = Prince_Dormand(a, alpha, beta, &f, &lambda, y2, x_start, h, x_end, &h_next, eps=

printf("$T=2$ & %.9f & %.9f & %.9f ", y[0][1], y[0][2], y[1][4]);
printf("& %.9f ", y1[1][4]);
printf("& %.9f ", y2[1][4]);
printf("\\\\ \n");
//printf("$T=2$ & %.9f & %.9f & %.9f \\\\ \n", y[1][1]-y1[1][1], y1[1][1]-y2[1][1], (y



x_end=3;
y[0][0] = 0;
y[0][1] = -2.766260068479832;
y[0][2] = -1.285304259409433;
y[0][3] = 0;
y[0][4] = 0;

y1[0][0] = 0;
y1[0][1] = -2.766260068479832;
y1[0][2] = -1.285304259409433;
y1[0][3] = 0;
y1[0][4] = 0;

y2[0][0] = 0;
y2[0][1] = -2.766260068479832;
y2[0][2] = -1.285304259409433;
y2[0][3] = 0;
y2[0][4] = 0;
err = Prince_Dormand(a, alpha, beta, &f, &lambda, y, x_start, h, x_end, &h_next, eps);
err1 = Prince_Dormand(a, alpha, beta,  &f, &lambda, y1, x_start, h, x_end, &h_next, eps
err2 = Prince_Dormand(a, alpha, beta, &f, &lambda, y2, x_start, h, x_end, &h_next, eps=

printf("$T=3$ & %.9f & %.9f & %.9f ", y[0][1], y[0][2], y[1][4]);
printf("& %.9f ", y1[1][4]);
printf("& %.9f ", y2[1][4]);
printf("\\\\ \n");
//printf("$T=3$ & %.9f & %.9f & %.9f \\\\ \n", y[1][1]-y1[1][1], y1[1][1]-y2[1][1], (y

x_end=3.5;
y[0][0] = 0;
y[0][1] = 3.123090348841166;
y[0][2] = 1.433593811023259;
y[0][3] = 0;
y[0][4] = 0;

y1[0][0] = 0;
y1[0][1] = 3.123090348841166;
y1[0][2] = 1.433593811023259 ;
y1[0][3] = 0;
y1[0][4] = 0;

y2[0][0] = 0;
y2[0][1] = 3.123090348841166;
```

```
y2[0][2] = 1.433593811023259 ;
y2[0][3] = 0;
y2[0][4] = 0;

err = Prince_Dormand(a, alpha, beta, &f, &lambda, y, x_start, h, x_end, &h_next, eps);
err1 = Prince_Dormand(a, alpha, beta,  &f, &lambda, y1, x_start, h, x_end, &h_next, eps
err2 = Prince_Dormand(a, alpha, beta, &f, &lambda, y2, x_start, h, x_end, &h_next, eps=

printf("$T=3.5$ & %.9f & %.9f & %.9f ", y[0][1], y[0][2], y[1][4]);
printf("& %.9f ", y1[1][4]);
printf("& %.9f ", y2[1][4]);
printf("\\\\ \n");
//printf("$T=3.5$ & %.9f & %.9f & %.9f \\\\ \n", y[1][1]-y1[1][1], y1[1][1]-y2[1][1],

x_end=10;
y[0][0] = 0;
y[0][1] = 1.119116341222183;
y[0][2] = 0.691834551735015;
y[0][3] = 0;
y[0][4] = 0;

y1[0][0] = 0;
y1[0][1] = 1.119116341222183;
y1[0][2] = 0.691834551735015;
y1[0][3] = 0;
y1[0][4] = 0;

y2[0][0] = 0;
y2[0][1] = 1.119116341222183;
y2[0][2] = 0.691834551735015;
y2[0][3] = 0;
y2[0][4] = 0;
err = Prince_Dormand(a, alpha, beta, &f, &lambda, y, x_start, h, x_end, &h_next, eps);
err1 = Prince_Dormand(a, alpha, beta,  &f, &lambda, y1, x_start, h, x_end, &h_next, eps
err2 = Prince_Dormand(a, alpha, beta, &f, &lambda, y2, x_start, h, x_end, &h_next, eps=

printf("$T=10$ & %.9f & %.9f & %.9f ", y[0][1], y[0][2], y[1][4]);
printf("& %.9f ", y1[1][4]);
printf("& %.9f ", y2[1][4]);
printf("\\\\ \n");
//printf("$T=10$ & %.9f & %.9f & %.9f \\\\ \n ", y[1][1]-y1[1][1], y1[1][1]-y2[1][1],




/*for (int i = 0; i < 1; i++)
    {
            eps = 1e-7;
            err = Prince_Dormand(a, alpha, beta, &f, &lambda, y, x_start, h, x_end, &h
            err1 = Prince_Dormand(a, alpha, beta,  &f, &lambda, y1, x_start, h, x_end,
            err2 = Prince_Dormand(a, alpha, beta, &f, &lambda, y2, x_start, h, x_end, &
```

```
            printf("$T = %.4f$", x_end);

            printf(" & $||Delta x = %e$", y[1][0] - y1[1][0]);
            printf(" & $||Delta x = %e$", y1[1][0] - y2[1][0]);
            printf(" &  %.3f", (y[1][0] - y1[1][0]) / (y1[1][0] - y2[1][0]));

            printf(" \\\\ \\hline \n ");
            x_end *= 2;
        }

        for (int i = 0; i < 4; i++)
            printf(" %.12f & %.12f & %.12f\n " , y[1][i], y1[1][i], y2[1][i]);
        printf("d = %.12f\n", err);*/


}
```