Отчет по задаче практикума
«Уравнения с частными производными»

Тиунова Анастасия, 407 группа

15 мая 2021 г.

1 Постановка задачи

Задача. Найти решение задачи

$$\begin{cases} B_0 = \int_0^T (\ddot{x}^2 - x^2)dt \to inf, \\ x(0) = x(T) = 0,\ \dot{x}(T) = 1, \\ T \in [0.1; 10]. \end{cases}$$

2 Формализация

Обозначим $x_1(t) = x(t),\ x_2(t) = \dot{x}(t),\ u(t) = \ddot{x}(t)$. Тогда:
(1) $B_0(x_1(\cdot), x_2(\cdot), u(\cdot)) = \int_0^T (u^2(t) - x_1^2(t))dt \to inf,$
(2) $\dot{x}_1(t) - x_2(t) = 0, \dot{x}_2(t) - u(t) = 0,\ \forall t \in [0, 1],$
(3) $x_1(0) = 0,\ x_1(T) = 0,\ x_2(T) = 1.$

3 Необходимые условия

1) Функция Понтрягина: $H = p_1 x_2 + p_2 u - \lambda_0(u^2(t) - x_1^2(t))$
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: $\dot{\widehat{p}}_i = -\frac{\partial \widehat{H}}{\partial x_i} \Rightarrow\ \dot{p_1} = -2\widehat{\lambda_0} x_1,\ \dot{p_2} = -p_1$

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

c) условие оптимальности: $\widehat{u} = argabsmax(H) = argabsmax\left(p_2 u - \widehat{\lambda_0}(u^2)\right)$
Если $\lambda_0 = 0$, пусть $p_2 \neq 0$ тогда существует точка x_0, в которой $p(x_0) = C \neq 0$. По теореме отделимости в окрестности точки x_0: $p(x) \neq 0$ и без ограничения общности $p(x) > 0$. Рассмотрим u_k такие, что в данной окрестности они равны k, а вне её равны 0. Следовательно максимум не достигается, а значит $p_2 = 0$. Из условия a) $p_1 = 0$, что противоречит тому, что множители Лагранжа одновременно не равны нулю $\Rightarrow\ \lambda_0 \neq 0$. 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 существует, тк отрезок стационарен

e) условия не жесткости нет, тк 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 равные одновременно нулю.

3) Так как $\lambda_0 \neq 0$ из условияоптимальности можно выбрать (без оганичения общности) $u = p_2$, тогда, после подстановки в (1) и (2) получается система:

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

Решать задачу будем методом стрельбы. Пусть $x_2(0) = \alpha,\ p_1(0) = \beta$ -параметры пристрелки. Для того, чтобы их определеить, надо решить две дополнительные задачи:

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

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 матри-цы $A' = \frac{1}{2}(A + A^T)$, где матрица A- матрица производных исходной системы дифференциальныхуравнений. Вычислим матрицу A':

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

Все собственные значения этой матрицы: $\lambda_{1,2} = 0,\ \lambda_{3,4} = \pm 2$, откуда максиамльное по модулю: $\lambda = 2$. После определения α, β (с точностью до $\epsilon = 10^{-9}$) экстремальное значение мнтеграла определим как $x_3(T)$ в решении дифф уравнения:

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

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:

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

Тогда положительная определенность второй вариации функционала: $\int_0^T (\Delta u)^2 - (\Delta x_1)^2 dt$, 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):

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

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 $\widehat{\tau} \approx 3.931849978933 \Rightarrow$ 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 $T \in [0.1, \widehat{\tau})$, и, как следствие, полученная экстремаль доставляет сильный минимум. При $T = \widehat{\tau}$ выполнено условие Якоби $\Rightarrow$ достигается слабый минимум. При $T \in (\widehat{\tau}, 10]$ 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	$Integ(\varepsilon = 10^{-5})$	$Integ(\varepsilon = 10^{-7})$	$Integ(\varepsilon = 10^{-9})$
$T = 1$	-0.502989291	-3.039474645	2.980921850	2.980880895	2.980880617
$T = 2$	-0.551200594	-0.920064377	1.337824527	1.337817525	1.337817452
$T = \widehat{\tau}$	138.557303706	131.042445101	191.627711835	191.560793206	191.559881720
$T = 5$	0.815115894	0.794317272	1.543333680	1.543324621	1.543324530
$T = 10$	-3.389418706	-3.389083868	-3.687597709	-3.687648012	-3.687648517

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	$Integ(e^{-5}) - Integ(e^{-7})$	$Integ(e^{-7}) - Integ(e^{-9})$	$\frac{Integ(e^{-5})-Integ(e^{-7})}{Integ(e^{-7})-Integ(e^{-9})}$
$T = 1$	0.000005376	0.000000066	81.985586669
$T = 2$	0.000003384	0.000000039	85.842556569
$T = \widehat{\tau}$	0.000273333	0.000004445	61.498066269
$T = 5$	0.000004954	0.000000053	93.388827273
$T = 10$	-0.000011035	-0.000000131	84.087300208

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];
    ans[4] = y[3]*y[3]-y[0]*y[0];

}
double lambda(double a, double x)
{

    return 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=1;

y[0][0] = 0;
y[0][1] = -0.502989290781072;
y[0][2] = -3.039474645049093;
y[0][3] = 0;
y[0][4] = 0;

y1[0][0] = 0;
y1[0][1] = -0.502989290781072;
y1[0][2] =-3.039474645049093;
y1[0][3] = 0;
y1[0][4] = 0;

y2[0][0] = 0;
y2[0][1] =-0.502989290781072;
y2[0][2] = -3.039474645049093;
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] = -0.551200593688885;
y[0][2] = -0.920064376872628 ;
y[0][3] = 0;
y[0][4] = 0;

y1[0][0] = 0;
y1[0][1] =-0.551200593688885;
y1[0][2] = -0.920064376872628;
y1[0][3] = 0;
y1[0][4] = 0;

y2[0][0] = 0;
y2[0][1] = -0.551200593688885;
y2[0][2] =  -0.920064376872628;
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= 3.931849978933;
y[0][0] = 0;
y[0][1] = 138.557303705548094;
y[0][2] = 131.042445100575378;
y[0][3] = 0;
y[0][4] = 0;

y1[0][0] = 0;
y1[0][1] = 138.557303705548094;
y1[0][2] = 131.042445100575378;
y1[0][3] = 0;
y1[0][4] = 0;

y2[0][0] = 0;
y2[0][1] =138.557303705548094;
y2[0][2] = 131.042445100575378;
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("$T= 3.931849978933$ & %.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("$T= 3.931849978933$ & %.9f & %.9f & %.9f \\\\ \n", y[1][1]-y1[1][1], y1[1][1]

x_end=5;
y[0][0] = 0;
y[0][1] = 0.815115893599342;
y[0][2] =  0.794317271575859;
y[0][3] = 0;
y[0][4] = 0;

y1[0][0] = 0;
y1[0][1] = 0.815115893599342;
y1[0][2] = 0.794317271575859 ;
y1[0][3] = 0;
y1[0][4] = 0;

y2[0][0] = 0;
y2[0][1] = 0.815115893599342;
```

```
    y2[0][2] = 0.794317271575859 ;
    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("$T=5$ & %.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("$T=5$ & %.9f & %.9f & %.9f \\\\ \n", y[1][1]-y1[1][1], y1[1][1]-y2[1][1], (y


    x_end=10;
    y[0][0] = 0;
    y[0][1] =-3.389418705820504;
    y[0][2] = -3.389083867852695;
    y[0][3] = 0;
    y[0][4] = 0;

    y1[0][0] = 0;
    y1[0][1] = -3.389418705820504;
    y1[0][2] = -3.389083867852695;
    y1[0][3] = 0;
    y1[0][4] = 0;

    y2[0][0] = 0;
    y2[0][1] = -3.389418705820504;
    y2[0][2] = -3.389083867852695;
    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],

}
```