Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Удмуртский государственный университет»
Институт математики, информационных технологий и физики
Кафедра дифференциальных уравнений
Направление 02.03.01 «Математика и компьютерные науки»

Выпускная квалификационная работа
(дипломная работа)

Построение множества поимки в задаче простого преследования

Студент группы ОАБ–02.03.01–42
Логинов Иван Андреевич

____

Научный руководитель: к.ф.-м.н.
Щелчков Кирилл Александрович

____

Заведующий кафедрой: д.ф.-м.н., доцент

______________________________ Попова С.Н.

«_____» ____________________ 2021 г.

Ижевск 2021

Содержание

1 Введение

Цель работы: исследовать задачу простого преследования в случае, когда множество значений управления убегающего – шар, преследователя – выпуклый многогранник.

Задачи

1. Построить алгоритм нахождения управления преследователя для параллельного сближения.
2. Реализовать программно для случая пространства с учетом того, что множество допустимых значений управлений преследователя, выпуклый многогранник, задается как выпуклая оболочка конечного числа точек.
3. Исследовать множество поимки на плоскости.

2 Дифференциальные игры преследования

Изучение оптимальных способов преследования и убегания необходимо и полезно в спорте, военном деле и других областях человеческой деятельности.

Игра простого преследования представляет собой математическую модель реального процесса преследования, где задействованы два игрока – убегающий и преследователь. Задача преследователя – за кратчайшее время поймать убегающего, зная его координаты и направление движения.

3 Постановка задачи

Дана задача простого преследования, где управления игроков задаются дифференциальными уравнениями:

$$\dot{x} = u,\ \ u \in U$$

$$\dot{y} = v,\ \ v \in V,$$

где U – выпуклый многогранник, $V = D_r(0)$ – шар.
Убегающий E в начальный момент времени выбирает фиксированное управление $v \in V$.
Преследователь P на основании своего начального положения $x(0)$, начального положения убегающего $y(0)$ и управления убегающего v выбирает управление $\bar{u}$.
Управление $\bar{u}$ выбирается преследователем P согласно стратегии параллельного сближения. Условие поимки убегающего преследователем:

$$\exists\ T > 0 : x(T) = y(T)$$

4 Стратегия параллельного сближения

Определение

Стратегией параллельного сближения называется следующий способ преследования. Пока убегающий движется по лучу $y(0)A$, преследователь перемещается по лучу $x(0)B$, причем для всех t выполнены соотношения:

а) отрезок $x(t)y(t)$ параллелен отрезку $x(0)y(0)$;

б) $||x(t_2) - y(t_2)|| \leq ||x(t_1) - y(t_1)||$ для всех $t_2 > t_1$.

Таким образом, $B(0)$ — точка на луче $y(0)A$, в которой преследователь P и убегающий E оказываются в один и тот же момент времени t.

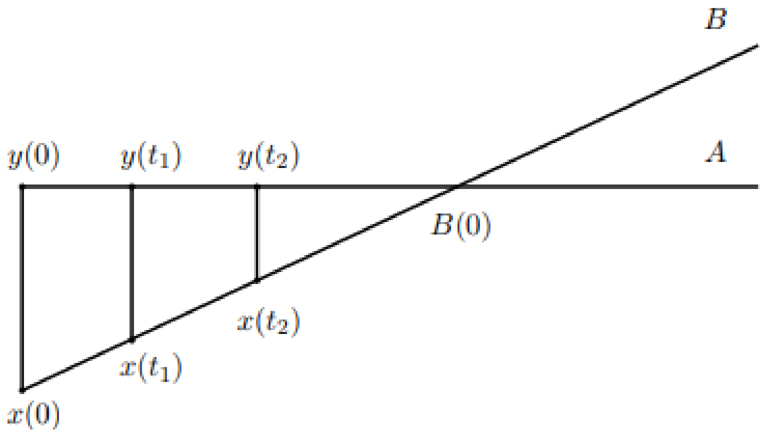

Рис. 1

5 Нахождение управления преследователя

Исходя из стратегии параллельного сближения, будем искать управление преследователя в виде:

$$u = v + \tilde{u},$$

где $u \in \partial U$, $\tilde{u} = \lambda(y(0) - x(0))$. Значит, управление преследователя имеет вид:

$$u = v + \lambda(y(0) - x(0)), \ \lambda \in \mathbb{R}$$

Пусть заданы ненулевые векторы $p_1, ..., p_m$ и числа $\alpha_1, ..., \alpha_m \in \mathbb{R}$. Тогда многогранник U задается следующей системой неравенств:

$$U = \{x \mid (x_i, p_i) \leqslant \alpha_i\}$$

Значит, можно найти λ следующим образом. Подставим $v + \lambda(y(0) - x(0))$ во все уравнения, соответствующие неравенствам из системы:

$$(v + \lambda(y(0) - x(0)), p_i) = \alpha_i$$

Затем выражаем λ:

$$\lambda = \frac{\alpha_i - (v, p_i)}{(y(0) - x(0)), p_i)}$$

И выбираем первое положительное λ, которое будет одновременно удовлетворять всем неравенствам, то есть выбираем $\lambda > 0 : v + \lambda(y(0) - x(0)) \in U$

6 Окружность Аполлония

Известно, что если α и β – максимальные скорости преследователя и убегающего соответственно, $U = D_\alpha(0)$ и $V = D_\beta(0)$ – круги соответствующих радиусов, $x(t)$ и $y(t)$ – положения игроков в момент времени t, то геометрическое место всех точек C – точек поимки, удовлетворяющих условию

$$\frac{|x(t)C|}{\alpha} = \frac{|y(t)C|}{\beta}$$

является окружностью, называемой **окружностью Аполлония** с центром в точке O, лежащей на луче $x(t)y(t)$,

$$|x(t)O| = \frac{\alpha^2}{\alpha^2 - \beta^2}|x(t)y(t)|,$$

и с радиусом

$$R = \frac{\alpha\beta}{\alpha^2 - \beta^2}|x(t)y(t)|.$$

Проще говоря, окружность Аполлония представляет собой множество точек поимки, если множества допустимых управлений игроков – круги.

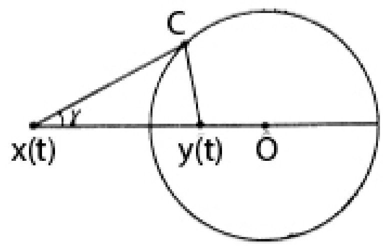

Рис. 2

7 Нахождение множества поимки на плоскости

Преследователь P и убегающий E имеют фиксированные начальные положения. Без ограничения общности считаем, что $x(0)$ распологается в начале координат, а $y(0)$ находится на оси Oy, в ее положительной части.

Множество допустимых управлений убегающего представляет собой шар с центром в начале координат радиуса r:

$$V = D_r(0);$$

Значит управление v убегающего можно представить в виде:

$$v = \begin{pmatrix} \cos\phi \\ \sin\phi \end{pmatrix}$$

Множество допустимых управлений U преследователя представляет собой выпуклый многогранник, который задаётся в виде системы неравенств, соответствующих уравнениям граничных гиперплоскостей многогранника. Кроме того, считаем, что грань, которая находится между преследователем и убегающим, в своей проекции на ось Ox содержит проекцию круга на ось Ox, см. рис. 3.

Данная наклонная прямая в общем случае задается уравнением

$$y = kx + b,$$

где $k, b \in \mathbb{R}$, $b > 0$ такие, что наклонная находится выше шара $V = D_r(0)$ и не пересекает его, $V \subset \text{Int } U$. Соответственно, расстояние от начала координат до наклонной:

$$\frac{|kx_0 + y_0 + b|}{\sqrt{k^2+1}} = \frac{|b|}{\sqrt{k^2+1}} > r.$$

Если начальное положение убегающего $y(0) = \begin{pmatrix} y_1(0) \\ y_2(0) \end{pmatrix}$, тогда управление преследователя будет иметь вид:

$$u = v + \lambda \cdot \begin{pmatrix} 0 \\ y_2(0) \end{pmatrix}$$

Находим λ уже известным нам способом.

Затем рассчитывается время поимки T. Время прохождения преследователя вдоль отрезка, соединяющего точки $x(0)$ и $y(0)$. Вдоль этого направления движение идет посредством вектора $\lambda(y(0) - x(0))$ Следовательно, расстояние делим на скорость и получаем время:

$$T = \frac{\|y(0) - x(0)\|}{\|\lambda(y(0) - x(0))\|} = \frac{1}{\lambda} = \frac{1}{\frac{\alpha_i - (v, p_i)}{(y(0) - x(0)), p_i)}}$$

Таким образом, множество поимки представляет собой геометрическое место точек:

$$y(T) = \begin{pmatrix} \cos \phi \\ \sin \phi \end{pmatrix} \cdot T$$

Игроки движутся со своими максимальными скоростями до момента поимки $T > 0$, не изменяя своё управление.

8 Алгоритм нахождения множества поимки на плоскости

1. Задаются координаты начальных положений $x(0)$ и $y(0)$ преследователя P и убегающего E соответственно.

2. Задаются коэффициенты k и b из уравнения прямой $y = kx + b$. Данная прямая представлет собой ближайшую к начальному положению убегающего E граничную гиперплоскость множества U – выпуклого многогранника и соответствует одному из неравенств системы.

3. Выбирается угол ϕ для управления v убегающего.

4. Рассчитывается время поимки T.

5. Вычисляются координаты точки встречи $y(T) = y(0) + v \cdot T$

6. Пробегая тригонометрическую окружность $[0; 2\pi]$ с заданной величиной разбиения δ, выполняются действия из пунктов 3 - 5. Поточечно строится множество поимки.

9 Реализация программы

Программа была реализована в качестве приложения Windows Forms на языке C# с использованием инструментов графической библиотеки OpenGL. В данном разделе описываются входные данные программы и основные функции, разработанные для реализации алгоритма.

Входные данные

В виде одномерных массивов длины 2 задаются координаты начальных положений игроков P и E, где первый элемент массива соответствует координате по оси Ox, второй элемент – координате по оси Oy.
Коэффициенты k и b из уравнения прямой $y = kx + b$, угол ϕ задаются в виде переменных типа double.
Величина разбиения δ задаётся переменной типа double, которой по умолчанию присваетвается значение 0.00001.
Объявлены переменные $xmin$, $xmax$, $ymin$, $ymax$ типа double – значения наименьших и наибольших координат по осям Ox и Oy, отображающихся на рисунке.

DrawEscaperSet

Данная функция отображает на координатной плоскости множество допустимых управлений убегающего – шар единичного радиуса с центром в начале координат. Данное множество для большей наглядности также изображается с центром в $y(0)$.

```
void DrawEscaperSet()
{Gl.glPointSize(1.0f);
Gl.glColor3f(0.0f, 0.0f, 1.0f);
Gl.glBegin(Gl.GL_POINTS);
for (double i = 0.0; i <= 2.0 * Math.PI; i += delta)
{v[0] = cos(phi);
v[1] = sin(phi);
Gl.glVertex2d(E[0] + v[0], E[1] + v[1]);
Gl.glVertex2d(P[0] + v[0], P[1] + v[1]);
phi += i;}
Gl.glEnd();}
```

Func

Функция принимает на вход значения переменных типа double: x, k и b и возвращает значения выражения $k \cdot x + b$ типа double

```
double Func(double x, double k, double b){return k * x + b;}
```

Norm

Функция принимает на вход значения переменных типа double: $x1$, $y1$, $x2$, $y2$ – координаты точек начала и конца вектора. Функция возвращает норму вектора, образованного этими двумя точками.

```
double Norm(double x1, double y1, double x2, double y2)
{return Math.Sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));}
```

DrawPursuerSet

Функция отображает на координатной плоскости ближайшую к начальному положению $y(0)$ убегающего граничную прямую выпуклого многогранника U – множества допустимых управлений убегающего. Переменные $xmin$ и $xmax$ – наименьшая и наибольшая координата по оси Ox, отображающаяся на рисунке.

```
void DrawPursuerSet()
{Gl.glColor3f(1.0f, 0.0f, 0.0f);
Gl.glBegin(Gl.GL_LINES);
Gl.glVertex2d(xmin, Func(xmin, k, b));
Gl.glVertex2d(xmax, Func(xmax, k, b));
Gl.glEnd();}
```

DrawCapture

Функция принимает на вход значение угла ϕ для выбора управления убегающего.
В функции производятся рассчеты управления преследователя u, времени поимки T; функция отображает на координатной плоскости точку поимки и движение игроков к точке поимки.

```
void DrawCapture(double phi)
{v[0] = cos(phi);
v[1] = sin(phi);
u[0] = v[0];
u[1] = v[0] * k + b;
T = Norm(P[0], P[1], E[0], E[1]) / (u[1] - v[1]);
Gl.glBegin(Gl.GL_LINES);
Gl.glColor3f(0.0f, 0.0f, 1.0f);//движение убегающего
Gl.glVertex2d(E[0], E[1]);
Gl.glVertex2d(E[0] + v[0] * T, E[1] + v[1] * T);
Gl.glColor3f(1.0f, 0.0f, 0.0f);//движение преследователя
Gl.glVertex2d(P[0], P[1]);
Gl.glVertex2d(P[0] + u[0] * T, P[1] + u[1] * T);
```

```
Gl.glEnd();
Gl.glEnable(Gl.GL_LINE_STIPPLE);//управление преследователя
Gl.glLineStipple(1, 0x00FF);
Gl.glBegin(Gl.GL_LINES);
Gl.glVertex2d(P[0], P[1]);
Gl.glVertex2d(P[0] + v[0], P[1] + v[1]);
Gl.glVertex2d(P[0] + v[0], P[1] + v[1]);
Gl.glVertex2d(P[0] + v[0], Func(P[0] + v[0],k , b));
Gl.glEnd();
Gl.glDisable(Gl.GL_LINE_STIPPLE);
Gl.glPointSize(6.0f);
Gl.glBegin(Gl.GL_POINTS);
Gl.glColor3f(1.0f, 0.0f, 1.0f);
Gl.glVertex2d(E[0] + v[0] * T, E[1] + v[1] * T); Gl.glEnd();}
```

DrawCaptureSet

Функция поточечно отображает на координатной плоскости множество поимки с заданной величиной разбиения δ.

```
void DrawCaptureSet()
{Gl.glPointSize(3.0f);
Gl.glColor3f(0.0f, 1.0f, 0.0f);
Gl.glBegin(Gl.GL_POINTS);
for (double i = 0.0; i <= 2.0 * Math.PI; i += delta)
{v[0] = cos(phi);
v[1] = sin(phi);
u[0] = v[0];
u[1] = v[0] * k + b;
T = Norm(P[0], P[1], E[0], E[1]) / (u[1] - v[1]);
Gl.glVertex2d(E[0] + v[0] * T, E[1] + v[1] * T);
phi += i;}
Gl.glEnd();}
```

10 Результат работы программы

Пример 1:

$k = \frac{1}{8},\ b = 2,\ \phi = \frac{\pi}{3},\ y(0) = \begin{pmatrix} 0 \\ 6 \end{pmatrix},\ u = \begin{pmatrix} 0.5 \\ 2.0625 \end{pmatrix}$

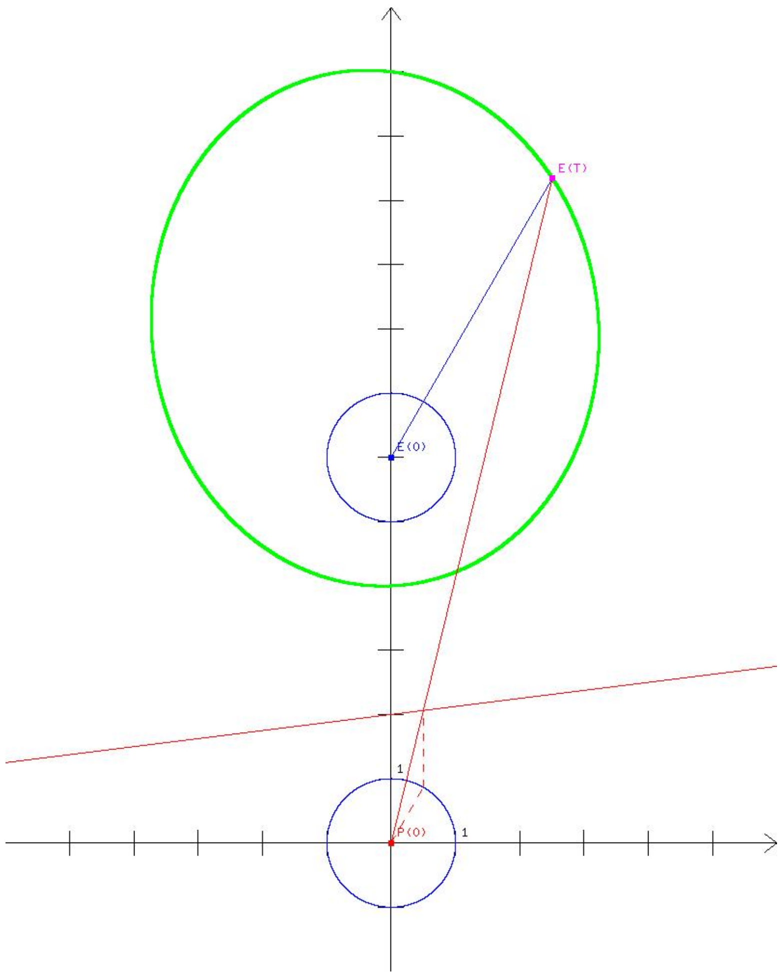

Рис. 3

Пример 2:

$$k = -1.5,\ b = 3.0,\ \phi = \frac{\pi}{4},\ y(0) \approx \begin{pmatrix} 0 \\ 7 \end{pmatrix},\ u = \begin{pmatrix} 0.707 \\ 1.939 \end{pmatrix}$$

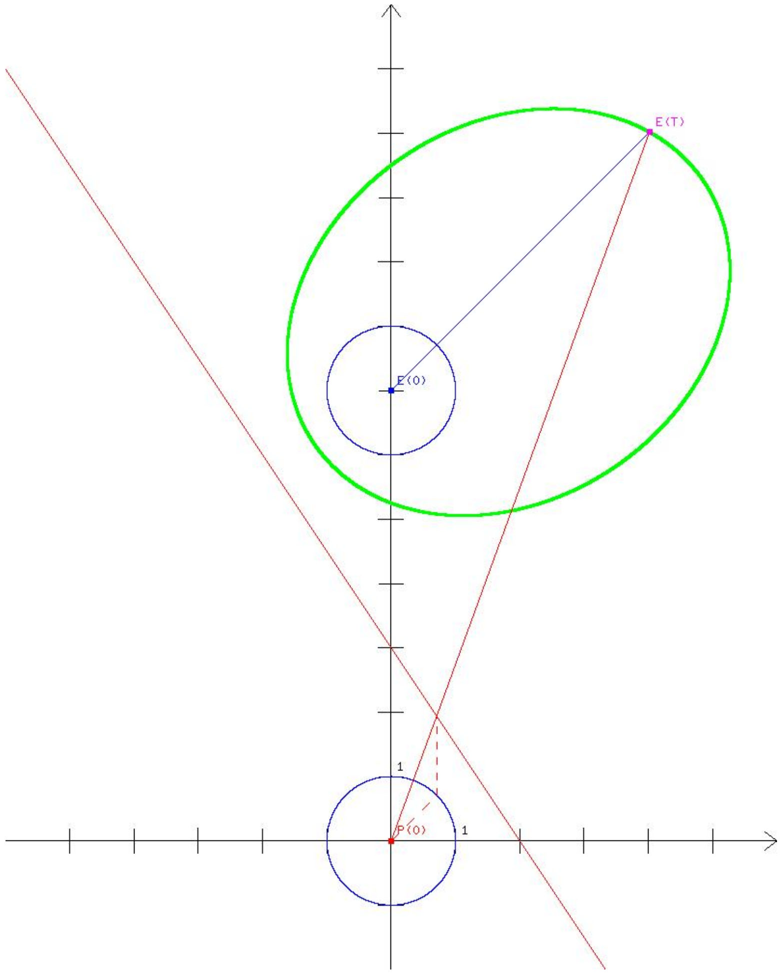

Рис. 4

11 Построение преследования в пространстве

Преследователь P и убегающий E имеют фиксированные начальные положения.

Аналогично задаче на плоскости, множество допустимых управлений убегающего представляет собой шар с центром в начале координат радиуса r.

Множество допустимых управлений преследователя представляет собой выпуклый многогранник, который задаётся как выпуклая оболочка конечного набора точек $a_i(x_i, y_i, z_i)$, то есть

$$U = \mathrm{co}\{a_i\}.$$

12 Алгоритм построения преследования в пространстве

В виде списков типа double задаются координаты начальных положений $x(0)$ и $y(0)$ преследователя P и убегающего E соответственно.

В виде списка одномерных массивов типа double задаётся набор из k точек.

1. Необходимо из заданного набора точек получить систему неравенств, соответствующих уравнениям прямых, задающих грани выпуклого многогранника U. Составляется уравнение плоскости, проходящей через три точки $a_1(x_1, y_1, z_1)$, $a_2(x_2, y_2, z_2)$, $a_3(x_3, y_3, z_3)$:

$$\begin{vmatrix} x - x_1 & y - y_1 & z - z_1 \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{vmatrix} = 0 \qquad (*)$$

2. В данное уравнение плоскости поочерёдно подставляются координаты всех точек из набора, за исключением координат трех уже использованных точек. И получим $k-3$ неравенства.

3. Если все полученные неравенства имеют одинаковый знак, то уравнение плоскости $(*)$ задает одну из граней выпуклого многогранника U. Если же неравенства имеют разные знаки, плоскость не является гранью многогранника.

4. Если уравнение плоскости $(*)$ является уравнением грани, оно преобразуется в соответствующее ему неравенство, сохраняя знак неравенств, полученных из пункта 2.

5. Аналогичные действия производятся для всех возможных троек точек из набора без повторений. Количество таких операций вычисляется как число сочетаний из k по 3:

$$C_k^3 = \frac{k!}{(k-3)! \cdot 3!}$$

Выполнив действия из пунктов 1-4 для всех троек точек, получаем систему неравенств, соответствующую многограннику U

$$U = \{x \mid (x_i, p_i) \leqslant \alpha_i\}$$

6. Выполняется проверка того, что все неравенства имеют знак " $\leqslant$ ". Если неравенство имеет знак " $\geqslant$ ", неравенство умножается на -1, тем самым, знак неравенства заменется на нужный.

Свободный член α_i в каждом неравенстве переносится влево, чтобы в правой части неравенства находился 0.

$$U = \{x \mid (x_i, p_i) - \alpha_i \leqslant 0\}$$

7. Вычисляется λ.

8. Вычисляется время поимки T.

9. Иллюстрируется преследование.

13 Реализация программы

Входные данные

В виде одномерных массивов длины 3 задаются координаты начальных положений игроков P и E, где первый элемент массива соответствует координате по оси Ox, второй элемент – координате по оси Oy, третий – координате по оси Oz.

В виде списка M длины k одномерных массивов длины 3 задается набор точек, выпуклая оболочка которых представляет собой выпуклый многогранник U, где k – количество заданных точек. Элементы каждого массива – координаты соответствующей точки.

Задаётся одномерный массив q длины 3, соответствующий вектору $y(0) - x(0)$. Элементы массива соответствуют координатам данного вектора.

Задаются переменные типа double $theta$ и phi, соответствующие углам θ и ϕ, используемым для выбора управления убегающего.

FuncX, FuncY, FuncZ

Функции принимают на вход значения углов θ и ϕ и возвращают значения x, y и z соответственно из параметрического уравнения сферы единичного радиуса с центром в начале координат:

$$\begin{cases} x = \sin\theta \cdot \cos\phi \\ y = \sin\theta \cdot \sin\phi \\ z = \cos\theta, \end{cases} \tag{**}$$

где $\theta \in [0, \pi]$ и $\phi \in [0, 2\pi)$.

```
double FuncX(double theta, double phi)
{return sin(theta) * cos(phi);}
double FuncY(double theta, double phi)
{return sin(theta) * sin(phi);}
double FuncZ(double theta)
{return cos(theta);}
```

EscaperControl

Функция принимает на вход значения углов θ и ϕ и возвращает одномерный массив, элементы которого – значения x, y и z соответственно из параметрического уравнения (**). Значение функции присваивается одномерному массиву v, который соответствует управлению убегающего.

```
v = EscaperControl(theta, phi);
double[] EscaperControl(double theta, double phi)
{double[] v = { FuncX(theta, phi), FuncY(theta, phi), FuncZ(theta) };
return v;}
```

DrawEscaperSet

Функция отображает в пространстве множество допустимых управлений убегающего – шар единичного радиуса с центром в точке начального положения игрока E. Для наглядности рисунка изображается второй шар – с центром в точке начального положения преследователя.

Переменная h представляет собой малую величину, используемую для отрисовки изображения. Переменная $rate$ по умолчанию принимает значение 20.0.

```
void DrawEscaperSet()
{double h = 2 * Math.PI / rate;
Gl.glBegin(Gl.GL_LINES);
Gl.glColor3f(0.0f, 0.0f, 1.0f);
Gl.glPointSize(3.0f);
for (double i = 0.0; i <= 2 * Math.PI; i += h)
{for (double j = 0.0; j <= Math.PI; j += h)
{Gl.glVertex3d(FuncX(j, i) + P[0], FuncY(j, i) + P[1], FuncZ(j)
+ P[2]);
Gl.glVertex3d(FuncX(j + h, i) + P[0], FuncY(j + h, i) + P[1], FuncZ(j + h)
+ P[2]);
Gl.glVertex3d(FuncX(i, j + h) + P[0], FuncY(i, j + h) + P[1], FuncZ(i)
+ P[2]);
Gl.glVertex3d(FuncX(i, j) + P[0], FuncY(i, j) + P[1], FuncZ(i)
+ P[2]);
Gl.glVertex3d(FuncX(j, i) + E[0], FuncY(j, i) + E[1], FuncZ(j)
+ E[2]);
Gl.glVertex3d(FuncX(j + h, i) + E[0], FuncY(j + h, i) + E[1], FuncZ(j + h)
+ E[2]);
Gl.glVertex3d(FuncX(i, j + h) + E[0], FuncY(i, j + h) + E[1], FuncZ(i)
+ E[2]);
Gl.glVertex3d(FuncX(i, j) + E[0], FuncY(i, j) + E[1], FuncZ(i)
+ E[2]);}}
Gl.glEnd();}
```

EdgeDeterminant

Функция принимает на вход 4 одномерных массива, соответствующих точкам из заданного набора M, затем вычисляет определитель из уравнения $(*)$, где вместо x, y, z, x_i, y_i, z_i подставляются координаты соответствующих точек. Первые три заданные точки образуют плоскость, а координаты четвертой точки подставляются вместо x, y и z

```
double EdgeDeterminant(double[] a0, double[] a1, double[] a2, double[] a4)
{double x = a4[0];
double y = a4[1];
double z = a4[2];
double io = (x - a0[0]) * (a1[1] - a0[1]) * (a2[2] - a0[2]) +
(a1[0] - a0[0]) * (a2[1] - a0[1]) * (z - a0[2]) +
(y - a0[1]) * (a1[2] - a0[2]) * (a2[0] - a0[0]) -
(z - a0[2]) * (a1[1] - a0[1]) * (a2[0] - a0[0]) -
(a1[0] - a0[0]) * (y - a0[1]) * (a2[2] - a0[2]) -
(a2[1] - a0[1]) * (a1[2] - a0[2]) * (x - a0[0]);
return io;}
```

CheckIfEdge

Функция принимает на вход 3 одномерных массива, соответствующих точкам из заданного набора M, а затем проверяет, является ли плоскость, образованная данными точками, гранью выпуклого многогранника U, возвращая значение $true$ или $false$

```
bool CheckIfEdge(double[] a0, double[] a1, double[] a2)
{int more = 0;
int less = 0;
for (int i = 0; i <= n - 1; i++)
{if (EdgeDeterminant(a0, a1, a2, M[i]) > 0)
more++;
if (EdgeDeterminant(a0, a1, a2, M[i]) < 0)
less++;}
if ((more == 0) || (less == 0))
return true;
else if ((more == 0) && (less == 0))
return false;
else return false;}
```

DistanceToZero

Функция принимает на вход 3 одномерных массива, соответствующих точкам из заданного набора M, затем вычисляет расстояние от начала координат до плоскости, проходящей через 3 заданные точки по формуле:

$$\frac{Ax_0 + By_0 + Cz_0 + D}{\sqrt{A^2 + B^2 + C^2}},$$

где $Ax + By + Cz + D = 0$ – уравнение плоскости; $x_0 = y_0 = z_0 = 0$;

```
double DistanceToZero(double[] a0, double[] a1, double[] a2)
{double x = 0.0;
double y = 0.0;
double z = 0.0;
double A = (a0[1] * a1[2] - a0[1] * a2[2] - a0[2] * a1[1] +
a0[2] * a2[1] + a1[1] * a2[2] - a1[2] * a2[1]);
double B = (-a0[0] * a1[2] + a0[0] * a2[2] + a0[2] * a1[0] -
a0[2] * a2[0] - a1[0] * a2[2] + a1[2] * a2[0]);
double C = (a0[0] * a1[1] - a0[0] * a2[1] - a0[1] * a1[0] +
a0[1] * a2[0] + a1[0] * a2[1] - a1[1] * a2[0]);
double D = -a0[0] * a1[1] * a2[2] + a0[0] * a1[2] * a2[1] +
a0[1] * a1[0] * a2[2] - a0[1] * a1[2] * a2[0] -
a0[2] * a1[0] * a2[1] + a0[2] * a1[1] * a2[0];
return (A * x + B * y + C * z + D) / (Math.Sqrt(A * A + B * B + C * C));}
```

CheckBelonging

Функция принимает на вход список одномерных массивов, соответствующий набору точек M, затем проверяет принадлежность $V \subset U$, возвращая значение $true$ или $false$

```
bool CheckBelonging(List<double[]> M)
{for (int i = 0; i <= n - 1; i++)
{for (int j = i; j <= n - 1; j++)
{for (int k = j; k <= n - 1; k++)
{if (CheckIfEdge(M[i], M[j], M[k]) == true)
{if (DistanceToZero(M[i], M[j], M[k]) < 1.0) return false;}}}}
return true;}
```

NumberOfEdges

Функция принимает на вход список одномерных массивов, соответствующий набору точек M и возвращает количество граней многогранника U, образованного данным набором точек.

```
int NumberOfEdges(List<double[]> M1)
{int q0 = 0;
int n = M1.Count;
for (int i = 0; i <= n - 1; i++)
{for (int j = i+1; j <= n - 1; j++)
{for (int k = j+1; k <= n - 1; k++)
{if (CheckIfEdge(M1[i], M1[j], M1[k]))
{q0++;}}}}
return q0;}
```

DrawPursuerSet

Функция отображает в пространстве множество допустимых управлений преследователя – выпуклый многогранник U, используя точки из набора M.
На рисунке отображаются крайние точки множества и граничные гиперплоскости. Внутренние точки не отображаются и через них не проводятся плоскости.

```
void DrawPursuerSet()
{for (int i = 0; i <= n - 1; i++)
{for (int j = i+1; j <= n - 1; j++)
{for (int k = j+1; k <= n - 1; k++)
{if (CheckIfEdge(M[i], M[j], M[k]))
{Gl.glColor3f(1.0f, 0.0f, 0.0f);
Gl.glPointSize(9.0f);
Gl.glBegin(Gl.GL_POINTS);
Gl.glVertex3d(M[i][0], M[i][1], M[i][2]);
Gl.glVertex3d(M[j][0], M[j][1], M[j][2]);
Gl.glVertex3d(M[k][0], M[k][1], M[k][2]);
Gl.glEnd();
Gl.glPointSize(3.0f);
Gl.glBegin(Gl.GL_LINES);
Gl.glVertex3d(M[i][0], M[i][1], M[i][2]);
Gl.glVertex3d(M[j][0], M[j][1], M[j][2]);
Gl.glVertex3d(M[i][0], M[i][1], M[i][2]);
Gl.glVertex3d(M[k][0], M[k][1], M[k][2]);
Gl.glVertex3d(M[j][0], M[j][1], M[j][2]);
```

```
Gl.glVertex3d(M[k][0], M[k][1], M[k][2]);
Gl.glEnd();}
else break;}}}
```

CalculateCoefficients

Функция принимает на вход 3 одномерных массива, соответствующих точкам из заданного набора M, и возвращает в качестве одномерного массива коэффициенты p_i и свободный член α из уравнения плоскости $(x, p_i) = \alpha$.

```
double[] CalculateCoefficients(double[] a0, double[] a1, double[] a2)
{double p0 = (a1[1] - a0[1]) * (a2[2] - a0[2]) -
(a2[1] - a0[1]) * (a1[2] - a0[2]);
double p1 = -(a1[0] - a0[0]) * (a2[2] - a0[2]) -
(a2[0] - a0[0]) * (a1[2] - a0[2]);
double p2 = (a1[0] - a0[0]) * (a2[1] - a0[1]) -
(a2[0] - a0[0]) * (a1[1] - a0[1]);
double alpha = -a0[0] * ((a1[1] - a0[1]) * (a2[2] - a0[2]) -
(a2[1] - a0[1]) * (a1[2] - a0[2])) +
a0[1] * (-(a1[0] - a0[0]) * (a2[2] - a0[2]) -
(a2[0] - a0[0]) * (a1[2] - a0[2])) -
a0[2] * ((a1[0] - a0[0]) * (a2[1] - a0[1]) -
(a2[0] - a0[0]) * (a1[1] - a0[1]));
double[] d = { p0, p1, p2, alpha };
return d;}
```

GetSLAI

Функция принимает на вход список одномерных массивов, соответствующий набору точек M, и возвращает в качестве списка одномерных массивов систему неравенств, задающую выпуклый многогранник U. Элементами массивов являются коэффициенты неравенств p_i и свободный член α.

Выполняется проверка того, что все неравенства имеют знак " $\leqslant$ ". Если неравенство имеет знак " $\geqslant$ ", его коэффициенты домножаются на -1.

Список одномерных массивов $SLAI$ принимает значение данной функции.

```
List<double[]> SLAI = new List<double[]>();
SLAI = GetSLAI(M);
List<double[]> GetSLAI(List<double[]> M)
{List<double[]> SLAI = new List<double[]>();
for (int i = 0; i <= n - 1; i++)
{for (int j = i + 1; j <= n - 1; j++)
```

```
{for (int k = j + 1; k <= n - 1; k++)
{double p0 = CalculateCoefficients(M[i], M[j], M[k])[0];
double p1 = CalculateCoefficients(M[i], M[j], M[k])[1];
double p2 = CalculateCoefficients(M[i], M[j], M[k])[2];
double alpha = CalculateCoefficients(M[i], M[j], M[k])[3];
if (p0 * M[i][0] + p1 * M[i][1] + p2 * M[i][2] + alpha > 0.0)
{p0 = -p0;
p1 = -p1;
p2 = -p2;
alpha = -alpha;}
SLAI.Add(CalculateCoefficients(M[i], M[j], M[k]));}}}
return SLAI;}
```

Lambda

Функция принимает на вход список одномерных массивов, соответствующий набору точек M, и находит коэффициент λ.

```
double Lambda(List<double[]> M)
{double lambda = 0.0;
int counter = 0;
for (int i = 0; i <= SLAI.Count() - 1; i++)
{lambda = (-SLAI[i][3] - (v[0] * SLAI[i][0] +
v[1] * SLAI[i][1] + v[2] * SLAI[i][2])) /
(q[0] * SLAI[i][0] + q[1] * SLAI[i][1] + q[2] * SLAI[i][2]);
for (int j = 0; j <= SLAI.Count() - 1; j++)
{if (lambda <= 0)
break;
if (((v[0] + lambda * q[0]) * SLAI[i][0] + (v[1] + lambda * q[1]) *
SLAI[i][1] + (v[2] + lambda * q[2]) * SLAI[i][2]) + alpha <= 0.0)
counter ++;
if (counter == SLAI.Count())
return lambda;}}
return lambda;}
```

DrawCapture

Функция вычисляет время поимки T и отображает в пространстве точку поимки и движение игроков до точки поимки.

```
void DrawCapture()
{T = 1.0 / Lambda(M);
Gl.glColor3f(0.0f, 0.0f, 1.0f);
Gl.glPointSize(9.0f);
Gl.glBegin(Gl.GL_POINTS);
Gl.glVertex3d(E[0] + v[0] * T, E[1] + v[1] * T, E[2] + v[2] * T);
Gl.glEnd();
Gl.glPointSize(3.0f);
Gl.glBegin(Gl.GL_LINES);
Gl.glVertex3d(E[0], E[1], E[2]);
Gl.glVertex3d(E[0] + v[0] * T, E[1] + v[1] * T, E[2] + v[2] * T);
Gl.glColor3f(1.0f, 0.0f, 0.0f);
Gl.glVertex3d(P[0], P[1], P[2]);
Gl.glVertex3d(E[0] + v[0] * T, E[1] + v[1] * T, E[2] + v[2] * T);
Gl.glEnd();}
```

14 Результат работы программы

Пример 1:

$$U = \text{co } M, M = \{\begin{pmatrix}6\\6\\9\end{pmatrix}, \begin{pmatrix}-5\\5\\5\end{pmatrix}, \begin{pmatrix}7\\-7\\7\end{pmatrix}, \begin{pmatrix}8\\6\\-6\end{pmatrix}, \begin{pmatrix}-5\\-5\\7\end{pmatrix}, \begin{pmatrix}7\\-7\\-5\end{pmatrix}, \begin{pmatrix}-6\\6\\-6\end{pmatrix}, \begin{pmatrix}-7\\-7\\-8\end{pmatrix}\},$$

$$\phi = 2\pi,\ \theta = 2\pi, y(0) = \begin{pmatrix}-5\\7\\7\end{pmatrix},\ u \approx \begin{pmatrix}-4.394\\6.152\\7.152\end{pmatrix};$$

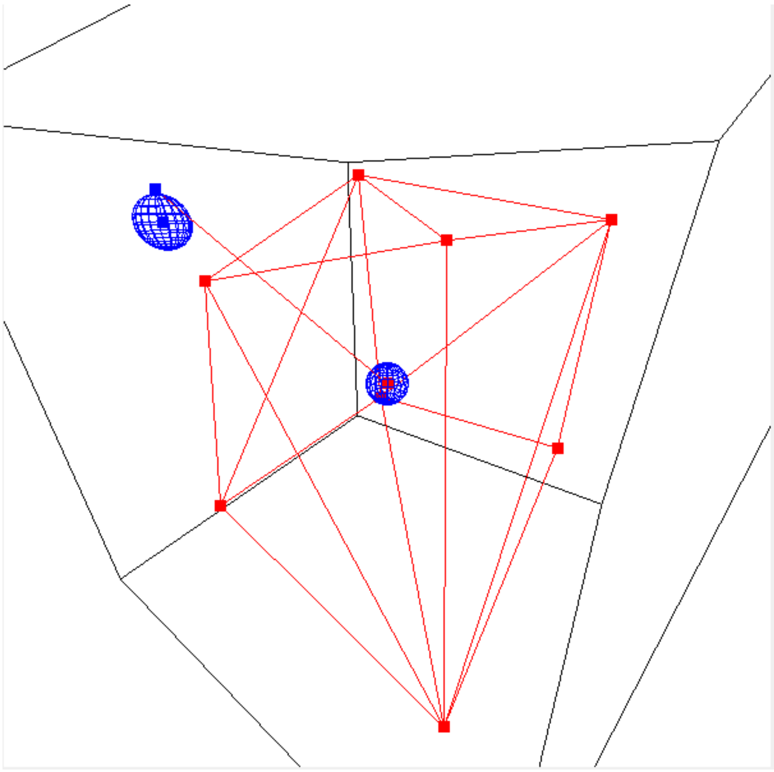

Рис. 5

Пример 2:

$$U = \mathrm{co}M, M = \left\{ \begin{pmatrix} 4 \\ 0 \\ -1 \end{pmatrix}, \begin{pmatrix} -4 \\ -4 \\ -1 \end{pmatrix}, \begin{pmatrix} -4 \\ 4 \\ -1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 4 \end{pmatrix}, \begin{pmatrix} 0.1 \\ 0.1 \\ 0.1 \end{pmatrix} \right\},\ \phi = \pi,\ \theta = \tfrac{\pi}{2},\ y(0) = \begin{pmatrix} 5 \\ 8 \\ 7 \end{pmatrix},$$

$$u \approx \begin{pmatrix} 1.325 \\ 2.12 \\ 0.855 \end{pmatrix};$$

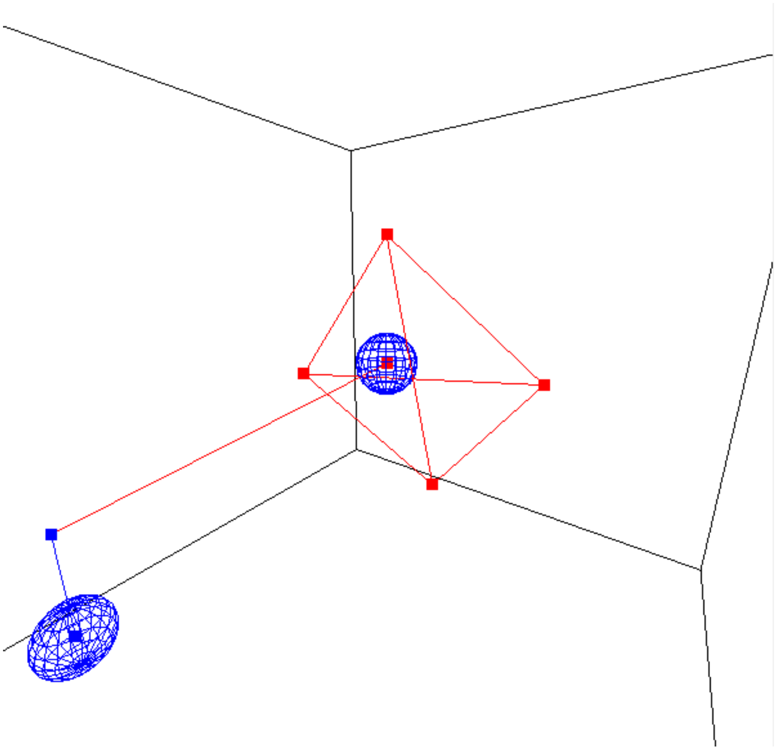

Рис. 6

15 Заключение

Исследование задачи простого преследования, в которой множество значений управления убегающего – шар, преследователя – выпуклый многогранник, потребовало взаимодействия методов различных областей математики – теории дифференциальных игр, выпуклого анализа, аналитической геометрии и математического анализа.

В процессе написания данной работы были проанализированы источники по данной теме, разработаны алгоритмы для решения задач в общем виде, написаны программы для реализации алгоритмов и создания возможности решать подобные задачи с различными входными данными.

С использованием языка C# для реализации алгоритма и библиотеки OpenGL для визуализации, программно реализован алгоритм построения преследования в пространстве.

Для задачи на плоскости в явном виде было найдено множество поимки для постоянного управления убегающего.

Таким образом, в процессе работы были выполнены все поставленные задачи и достигнута основная цель исследования.

16 Список литературы

1. Банников А. С. Введение в дифференциальные игры: учебное пособие / А. С. Банников, Н. Н. Петров, Л. С. Чиркова. – Ижевск: Изд-во «Удмуртский университет», 2013, 48 с.
2. Петров Н.Н. Введение в выпуклый анализ: учеб.-метод. пособие / Н. Н. Петров. – Ижевск: Изд-во УдГУ, 2009, 168 с.
3. Петросян Л.А. Через игры — к творчеству / Л.А. Петросян, Г. В. Томский. – Новосибирск: Изд-во «Наука», 1991, 125 с.