6. Отчет обучающего по практике

1. Задание:

1)Смоделировать полет мяча, брошенного под углом к горизонту (без учета сопротивления воздуха). Реализовать через функцию, где пользователь вводит начальную скорость и угол. Рассчитать траекторию (x(t), y(t)), время полета, максимальную высоту, дальность полета. Построить траекторию в графическом окне.

Код на MATLAB:

```
function [x, y, vremia, max_visota, dlina_poleta] =
polet_mecha(nach_skorost, ugol)

    vremia = 2 * nach_skorost * sin(deg2rad(ugol)) / 9.81;
    max_visota = nach_skorost ^2 * sin(deg2rad(ugol)) ^2 / (2 * 9.81);
    dlina_poleta = nach_skorost ^ 2 * sin(deg2rad(2 * ugol)) / 9.81;

    t = linspace(0, vremia, 100);

    x = nach_skorost * cos(deg2rad(ugol)) * t;
    y = nach_skorost * sin(deg2rad(ugol)) .* t - 0.5 * 9.81 * t.^2;

    figure;
    plot(x, y, 'LineWidth', 2);
    title('Траектория полёта');
    xlabel('Горизонтальное смещение, м');
    ylabel('Высота, м');
    grid on;
end
```

```
>> [x, y, time, maximum_height, flight_range] = polet_mecha (10, 45)
```

Результат выполнения:

```
time =

    1.4416

maximum_height =

    2.5484

flight_range =

   10.1937
```

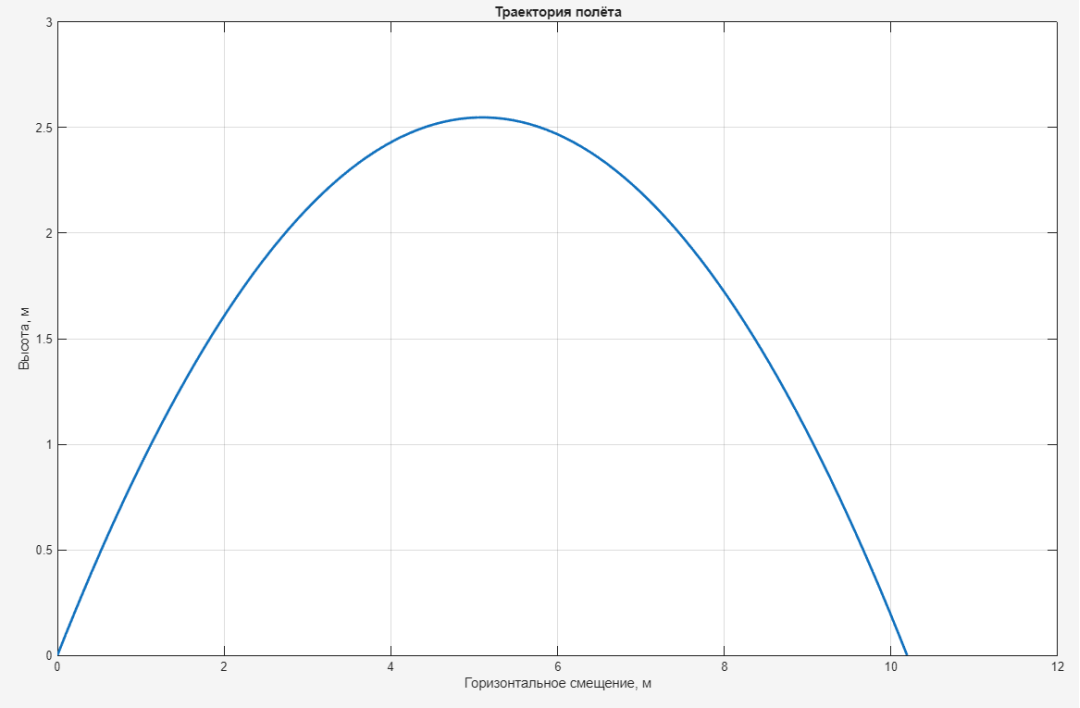

Рис. 1 Траектория полета мяча

2. Задание:

Сделать функцию, в которую пользователь вводит силу (величину и угол к горизонту). Программа разлагает силу на компоненты (Fx, Fy) с помощью тригонометрии (sind, cosd). Построить вектор силы и его компоненты на графике (использовать quiver или plot со стрелками).

Код на MATLAB:

```
function [Fx, Fy] = vectori_sily(sila, ugol)
    Fx = sila * cos(deg2rad(ugol));
    Fy = sila * sin(deg2rad(ugol));
    figure;
    hold on;
    axis equal;
    grid on;
    quiver(0, 0, Fx, Fy, 0, 'LineWidth', 2, 'MaxHeadSize', 0.5);
    quiver(0, 0, Fx, 0, 0, '--', 'LineWidth', 1.5, 'MaxHeadSize', 0.5);
    quiver(Fx, 0, 0, Fy, 0, '--', 'LineWidth', 1.5, 'MaxHeadSize', 0.5);
    xlabel('Fx, H');
    ylabel('Fy, H');
    title('Вектор силы и его компоненты');
    legend({'Сила F', 'Составляющая Fx', 'Составляющая Fy'}, 'Location',
'best');

    hold off; end
```

```
>> [Fx, Fy] = vectori_sily(10, 60)
```

Результат выполнения:

```
Fx =

    7.0711


Fy =

    7.0711
```

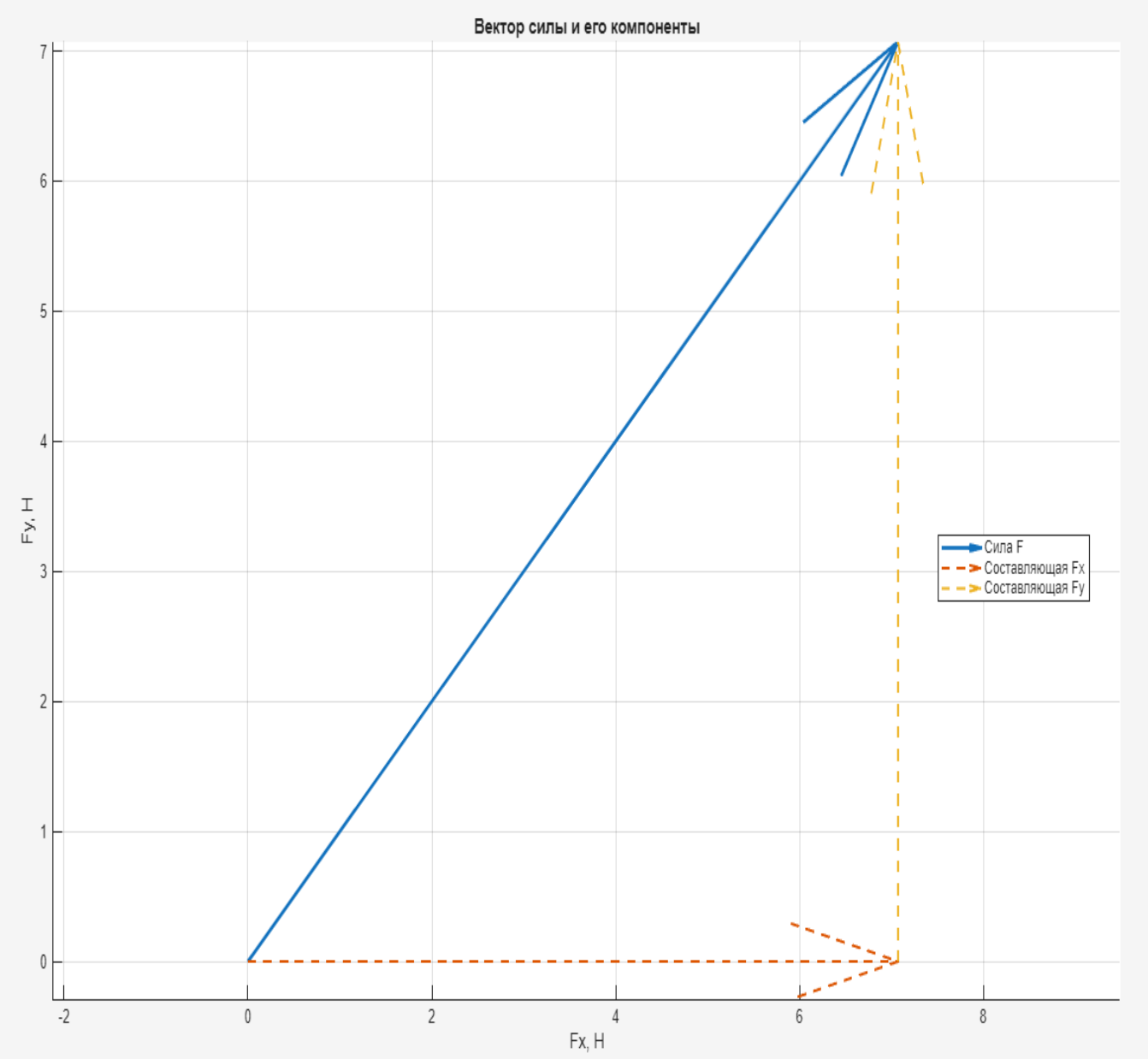

Рис. 2 Вектор силы и его компоненты

Задание 3:

1. Задание:
   Создать файл, имитирующий эксперимент по изменению времени при падении шарика с разной высоты. Добавить шумы и отклонения, имитирующие реальный эксперимент. Получить минимум 10 значений для одной высоты.

   Полученные результаты выгрузить в таблицу Excel.

Код выполнения задания:

```
function falling_time(H)
    t = sqrt(2 * H / 9.81)' * ones(1, 10);
    infelicity = 0.8 + (1.1 - 0.8) * rand(4, 10);
    infelicity_time = round(infelicity .* t, 3);
    result = [ repelem(H', 10, 1), infelicity_time(:) ];
    writematrix(result, 'falling_ball.xlsx');
end
```

Результат выполнения:

	A	B
1	100	4.02
2	100	6.011
3	100	6.797
4	100	9.512
5	100	8.667
6	100	3.918
7	100	5.435
8	100	6.791
9	100	8.405
10	100	9.019

Рис. 3 Excel file со значениями высоты и времени падения мяча

2. Создать отдельный файл, загружающий из таблицы массив с результатами измерений. Выполнить:

2.1. Отсортировать данные по времени (sort).

2.2.Найти минимальное, максимальное, среднее время для одной высоты.

2.3. Построить график "Высота vs Время".

2.4. Попытаться подобрать коэффициент k для формулы времени падения t методом наименьших квадратов в лоб (перебором k в разумном диапазоне, расчет суммы квадратов отклонений для каждого k, поиск минимума). Построить подобранную кривую на том же графике со значениями эксперимента.

Код выполнения задания:

```
function [max_value, min_value, mean_value, k_best] = falling_ball_grath()

    T = readtable('falling_ball.xlsx', 'ReadVariableNames', false);
    T.Properties.VariableNames = {'Height', 'Time'};

    T = sortrows(T, 'Height');

    stats = groupsummary(T, 'Height', {'min','mean','max'}, 'Time');

    stats.Properties.VariableNames{'min_Time'} = 'MinTime';
    stats.Properties.VariableNames{'mean_Time'} = 'MeanTime';
    stats.Properties.VariableNames{'max_Time'} = 'MaxTime';

    min_value  = stats.MinTime
    mean_value = stats.MeanTime
    max_value  = stats.MaxTime
    disp(stats);

    M = readmatrix('falling_ball.xlsx');
    H = M(:,1);
    T = M(:,2);

    k_vals = 0.1 : 0.001 : 1.0;

    SSE = zeros(size(k_vals));

    for i = 1:length(k_vals)
        k = k_vals(i);
        T_model = k * sqrt(H);
        residuals = T - T_model;
        SSE(i) = sum(residuals .^ 2);
    end

    [~, idx_min] = min(SSE);
    k_best = k_vals(idx_min);
    SSE_best = SSE(idx_min);

    fprintf('Лучший k = %.4f (минимальная SSE = %.5f)\n', k_best, SSE_best);


    figure;
    scatter(H, T, 50, 'filled');
    xlabel('Высота, м');
    ylabel('Время падения, с');
```

```
    title('Все измерения: высота vs время');
    grid on;

    hold on;
    H_model = linspace(min(H), max(H), 100);
    T_model = sqrt(2 * H_model / 9.81);
    plot(H_model, T_model, 'r--', 'LineWidth', 1.5);
    legend('Эксперименты','Теория t = k*\surd(H)','Location','best');
    hold off;

end
```

Результат выполнения:

```
>> falling_ball_grath


    Высота    MinTime    MeanTime    MaxTime

     100       3.918      6.8575      9.512
     200       4.207      7.2808     11.045
     300       3.967      6.8334        8.6
     400       3.968      7.3731     10.994
     500       4.353      7.1385      9.967

Лучший k = 0.3980
```

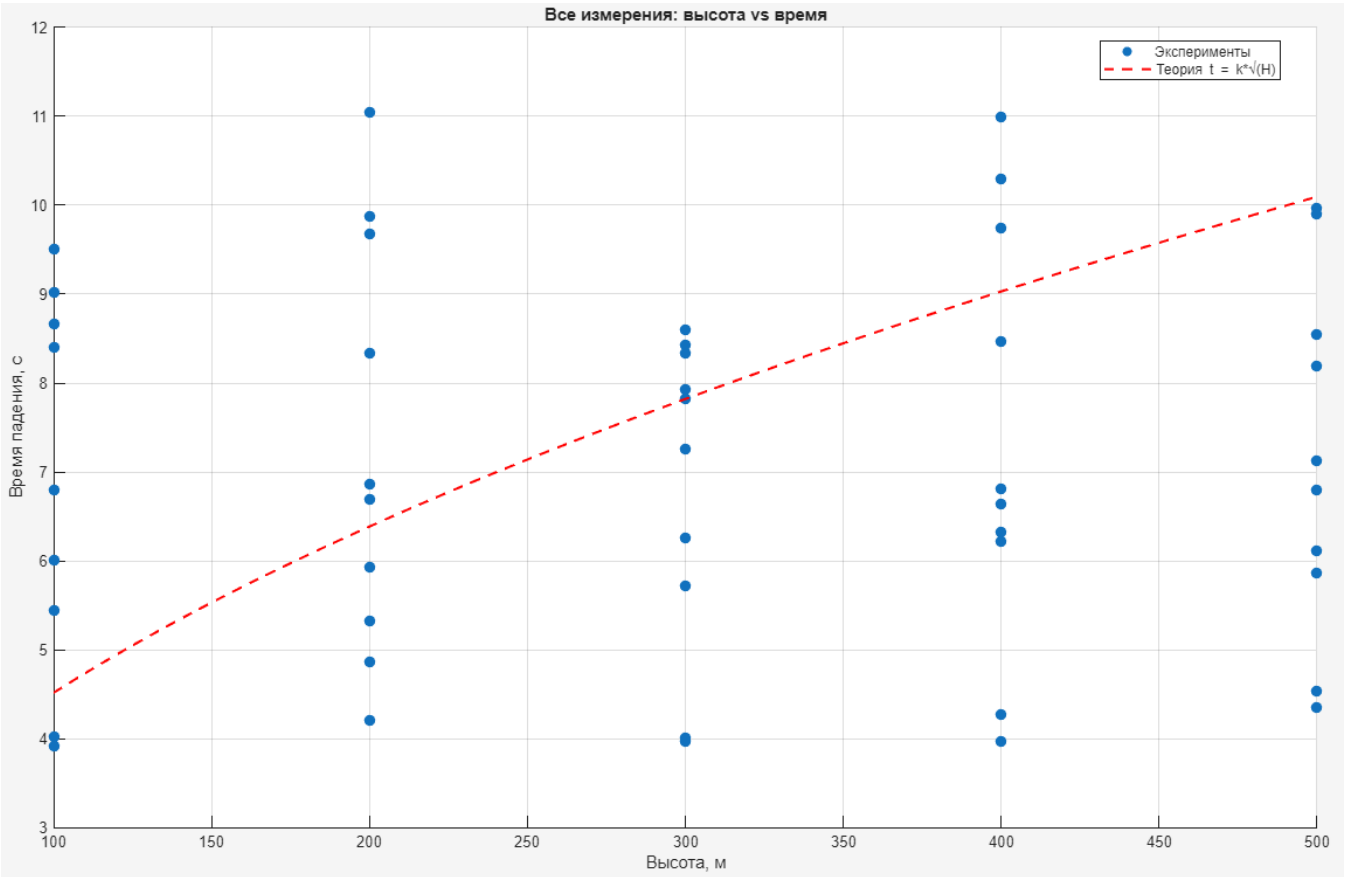

Рис 4. График "Высота vs Время".

Задание:

1. Создать структуру (struct) или класс (classdef) Point с полями x, y.
2. Создать массив/вектор из объектов Point (задать координаты вручную или случайно).
3. Написать функции: dist = distance(p1, p2) - расстояние между двумя точками;plotPoints(points) - отобразить все точки на графике; centroid = findCentroid(points) - найти центр масс точек (среднее по x и y).
4. Найти две самые удаленные точки (max расстояний) или точку, ближайшую к центроиду.

   Примечание. Должно быть задано минимум 20 точек.

1.

```
classdef Point
    properties
        x
        y
    end
    methods
        function obj = Point(x, y)
            if nargin > 0
                obj.x = x;
                obj.y = y;
            else
                obj.x = 0;
                obj.y = 0;
            end
        end
    end
end
```

2.

```
function d = distance(p1, p2)
    d = sqrt((p1.x - p2.x)^2 + (p1.y - p2.y)^2);
end
```

3.

```
numPoints = 20;
points = arrayfun(@(i) Point(rand()*10, rand()*10), 1:numPoints);

xs = arrayfun(@(p) p.x, points);
ys = arrayfun(@(p) p.y, points);
figure;
plot(xs, ys, 'bo', 'MarkerFaceColor', 'b');
xlabel('X');
ylabel('Y');
title('Распределение точек');
grid on;
```

```
xs = arrayfun(@(p) p.x, points);
ys = arrayfun(@(p) p.y, points);
centroid = Point(mean(xs), mean(ys));
hold on;
plot(centroid.x, centroid.y, 'rx', 'MarkerSize', 12, 'LineWidth', 2);
legend('Points', 'Centroid', 'Location', 'Best');

maxDist = 0;
pairIdx = [1, 2];
for i = 1:numPoints-1
    for j = i+1:numPoints
        d = distance(points(i), points(j));
        if d > maxDist
            maxDist = d;
            pairIdx = [i, j];
        end
    end
end
fprintf('Самые удаленные точки #%d  (%.2f, %.2f) и #%d (%.2f, %.2f). Дистанция -
%.2f\n', ...
    pairIdx(1), points(pairIdx(1)).x, points(pairIdx(1)).y, ...
    pairIdx(2), points(pairIdx(2)).x, points(pairIdx(2)).y, maxDist);

minDist = inf;
closestIdx = 1;
for i = 1:numPoints
    d = distance(points(i), centroid);
    if d < minDist
        minDist = d;
        closestIdx = i;
    end
end
fprintf('Точка, ближайшую к центроиду - #%d (%.2f, %.2f). Дистанция - %.2f\n', ...
    closestIdx, points(closestIdx).x, points(closestIdx).y, minDist);
```

Результат выполнения:

```
Самые удаленные точки #1  (8.85, 9.13) и #10 (1.98, 0.31). Дистанция -
11.19
Точка, ближайшую к центроиду - #6 (6.54, 4.94). Дистанция - 1.01
```

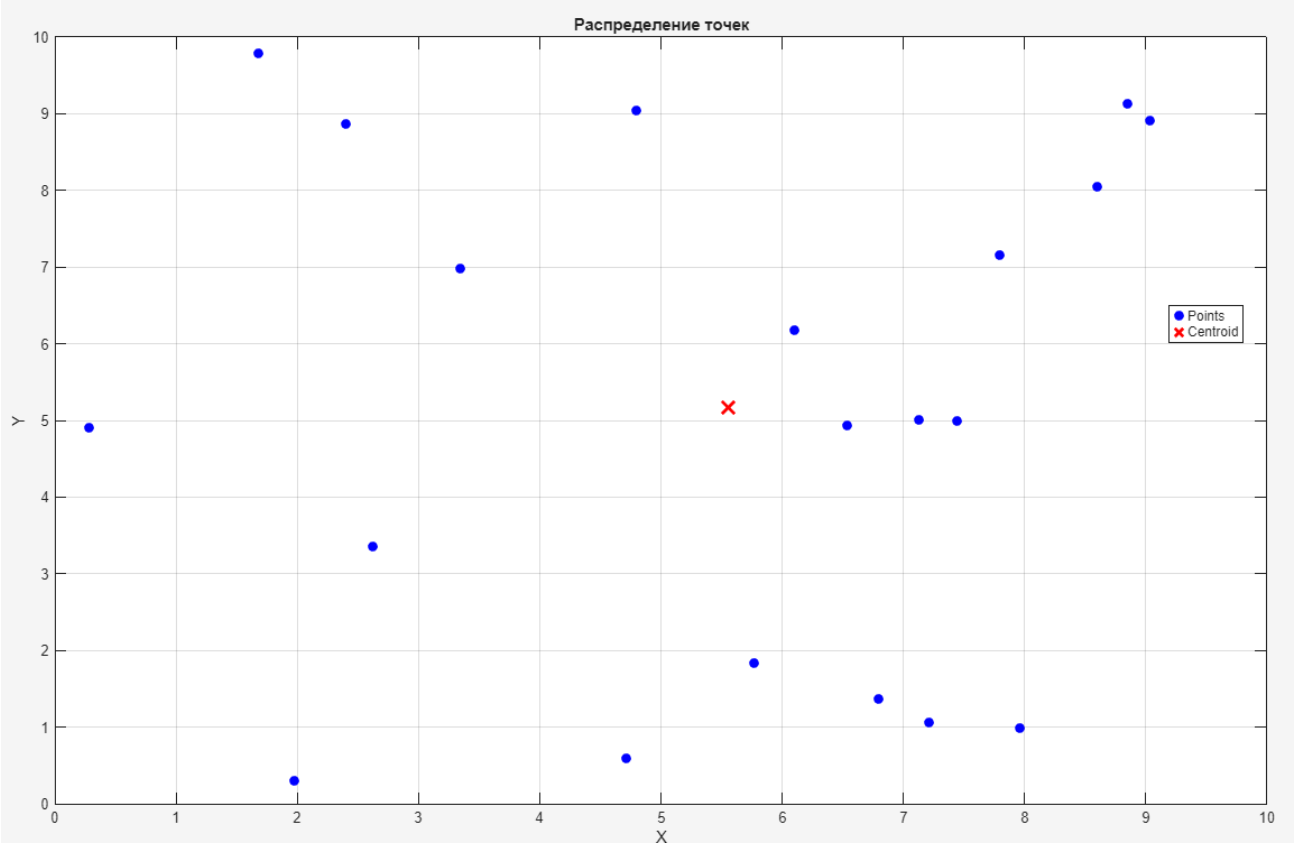

Рис.5

Задание:

Построить «снежинку» Коха, показанную на рис. П-11.11. Базисом «снежинки» является равносторонний треугольник ABO, длина сторо-ны которого L = 1. При построении «снежинки» следующего уровня сторона треугольника разбивается на три отрезка (AC, CD и DB) и вместо среднего от-резка выполняется построение двух отрезков (CE и ED), которые являются бо-ковыми сторонами равностороннего треугольника CED. Следовательно, сторона исходного треугольника заменяется на четыре линии, длина которых в 3 раза меньше ее длины. Минимальная длина линии LMIN = 0.01.

Код:

```
L = 1;
Lmin = 0.01;

P = [0, 0; 0.5, sqrt(3)/2; 1, 0; 0, 0];

while true
    newP = [];
    N = size(P,1);
    subdivide = false;
    for i = 1:N-1
        A = P(i, :);
```

```
        B = P(i+1, :);
        segLen = norm(B - A);
        if segLen > Lmin
            subdivide = true;
            C = A + (B - A)/3;
            D = A + 2*(B - A)/3;
            angle = pi/3;
            v = D - C;
            R = [cos(angle), -sin(angle); sin(angle), cos(angle)];
            E = C + (v * R');
            newP = [newP; A; C; E; D];
        else
            newP = [newP; A];
        end
    end
    newP = [newP; P(end, :)];
    P = newP;
    if ~subdivide
        break;
    end
end

figure;
plot(P(:,1), P(:,2), '-b', 'LineWidth', 1);
axis equal off;
title('Koch Snowflake');

>> closed_hilbert(5)
```

Результат выполнения:

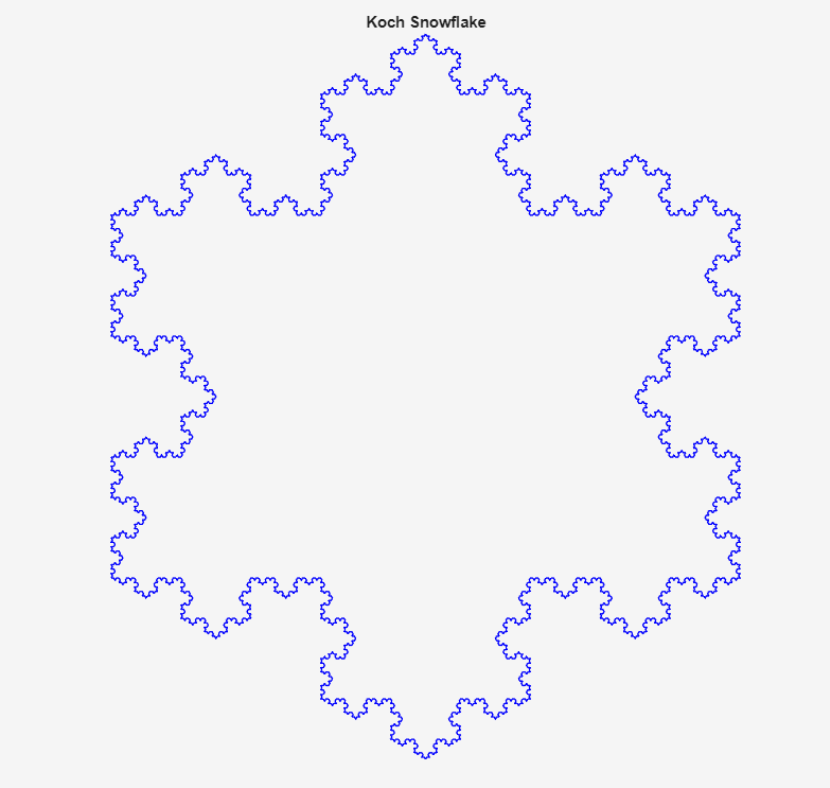

Рис. 6 Снежинка Коха