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, time, maximum_height, flight_range] =
ball_flight(initial_speed, angle)

    g = 9.81;
    time = 2 * initial_speed * sin(deg2rad(angle)) / g;
    maximum_height = initial_speed ^2 * sin(deg2rad(angle)) ^2 / (2 * g);
    flight_range = initial_speed ^ 2 * sin(deg2rad(2 * angle)) / g;

    t = linspace(0, time, 100);

    x = initial_speed * cos(deg2rad(angle)) * t;
    y = initial_speed * sin(deg2rad(angle)) .* t - 0.5 * g * 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] = ball_flight(10, 45)
```

Результат выполнения:

```
time =

    1.4416

maximum_height =

    2.5484

flight_range =

   10.1937
```

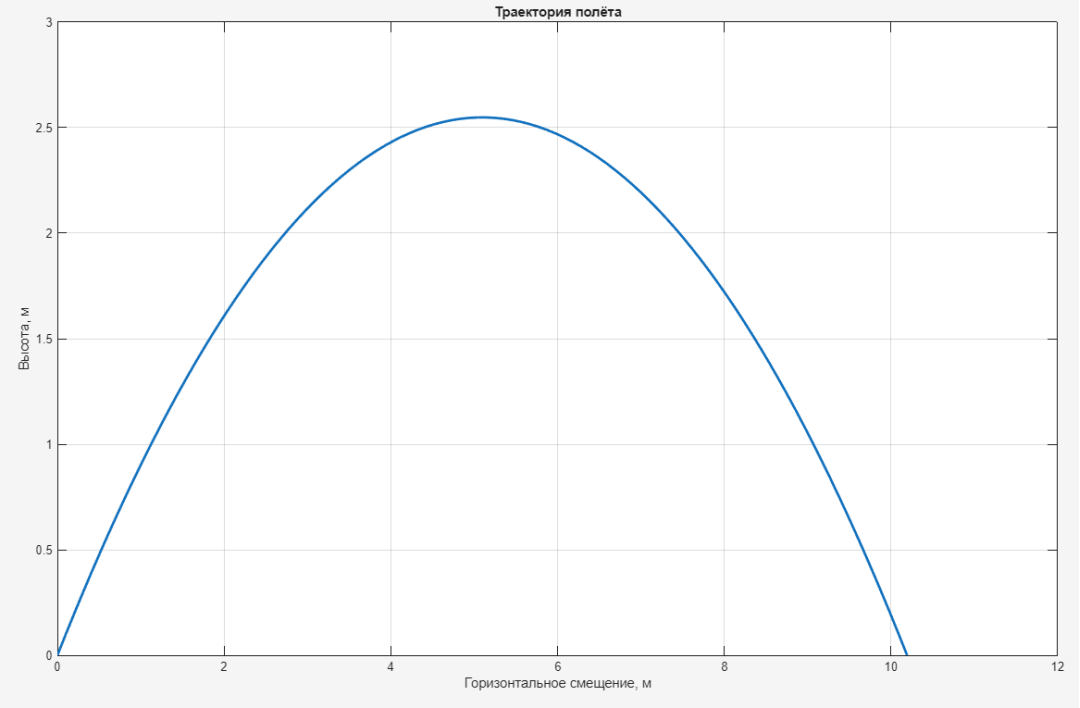

Рис. 1 Траектория полета мяча

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] = force_components(force, angle)
    Fx = force * cos(deg2rad(angle));
    Fy = force * sin(deg2rad(angle));

    % Построение векторов
    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);

    % Оформление графика
    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] = force_components(10, 45)
```

Результат выполнения:

```
Fx =

    7.0711


Fy =

    7.0711
```

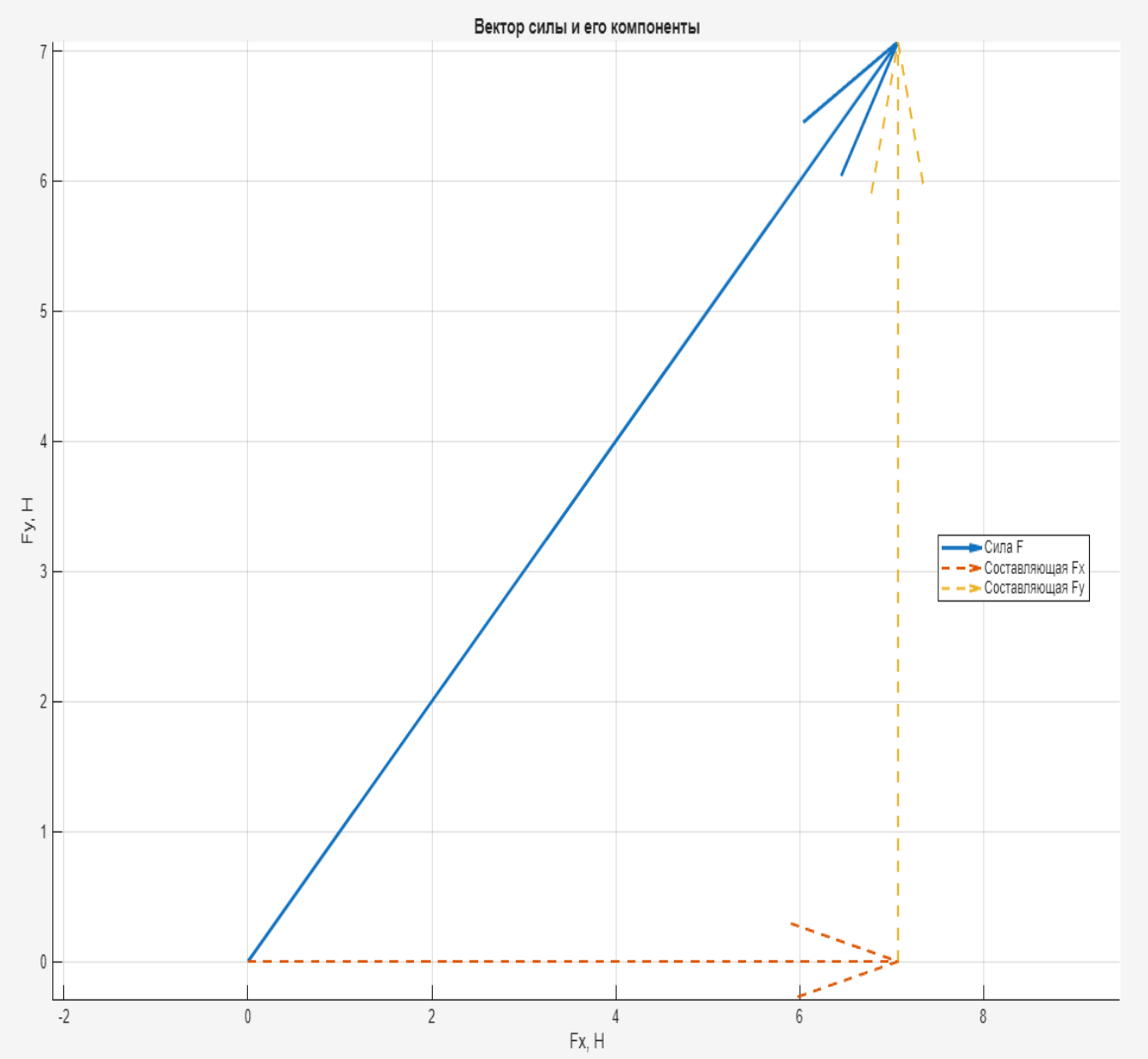

Рис. 2 Вектор силы и его компоненты

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);
    result = [H' round(infelicity.*t, 3)];
    writematrix(result, 'falling_ball.xlsx');
end
```

Результат выполнения:

	A	B	C	D	E	F	G	H	I	J	K
1	200	5.167	6.842	6.651	6.807	6.983	5.223	6.108	6.492	6.972	5.937
2	300	8.002	7.687	7.61	6.324	7.929	7.857	6.483	6.608	7.779	8.193
3	400	8.579	8.898	7.72	8.552	8.58	7.339	9.441	9.011	9.393	7.451
4	500	9.531	10.68	8.804	8.586	9.504	8.294	10.55	9.648	9.452	8.48

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()
    mx = readmatrix('falling_ball.xlsx');
    H = mx(:, 1);
    times = mx(:, 2:end);
    times = sort(times, 2);
    mx2 = [H times];
    writematrix(mx2, 'falling_ball_s.xlsx');

    max_value = times(:, end);
    min_value = times(:, 1);
    mean_value = mean(times, 2);
    mx_stats = [min_value mean(times, 2) max_value];

    mx2 = readmatrix('falling_ball_s.xlsx');
    H = mx2(:,1);
    times = mx2(:,2:end);

    k_vals = 0.5:0.001:2;

    SSE = zeros(size(k_vals));

    g = 9.81;
    for i = 1:length(k_vals)
        k = k_vals(i);
        t_pred = k .* ( sqrt(2 .* H ./ g));  % ваша модель
        SSE(i) = sum( (times - t_pred).^2, "all" );
    end
    [~, idx_min] = min(SSE);
    k_best = k_vals(idx_min);


    % Строим график
    figure; hold on; grid on;
    axis([H(1, 1) *0.9 H(end, end)*1.1 times(1, 1)*0.9 times(end, end)*1.1])
    scatter(H, times, 'filled', "green", 'o');
    scatter(H, mx_stats, 'filled', "black", 'd');
    H_smooth = linspace(min(H), max(H), 200);
    t_fit = k_best * sqrt(2*H_smooth ./ g);
    plot(H_smooth, t_fit, 'b-', 'LineWidth',1.5, 'DisplayName',sprintf('Модель t,
k=%.3f', k_best));

    xlabel('Высота H, м');
    ylabel('Время падения t, с');
    title('Высота vs Время');
end
```

Результат выполнения:

```
max_value =

    6.9830
    8.1930
    9.4410
   10.6800
```

```
min_value =

    5.1670
    6.3240
    7.3390
    8.2940


mean_value =

    6.3182
    7.4472
    8.4964
    9.3532


k_best =

    0.9450
```

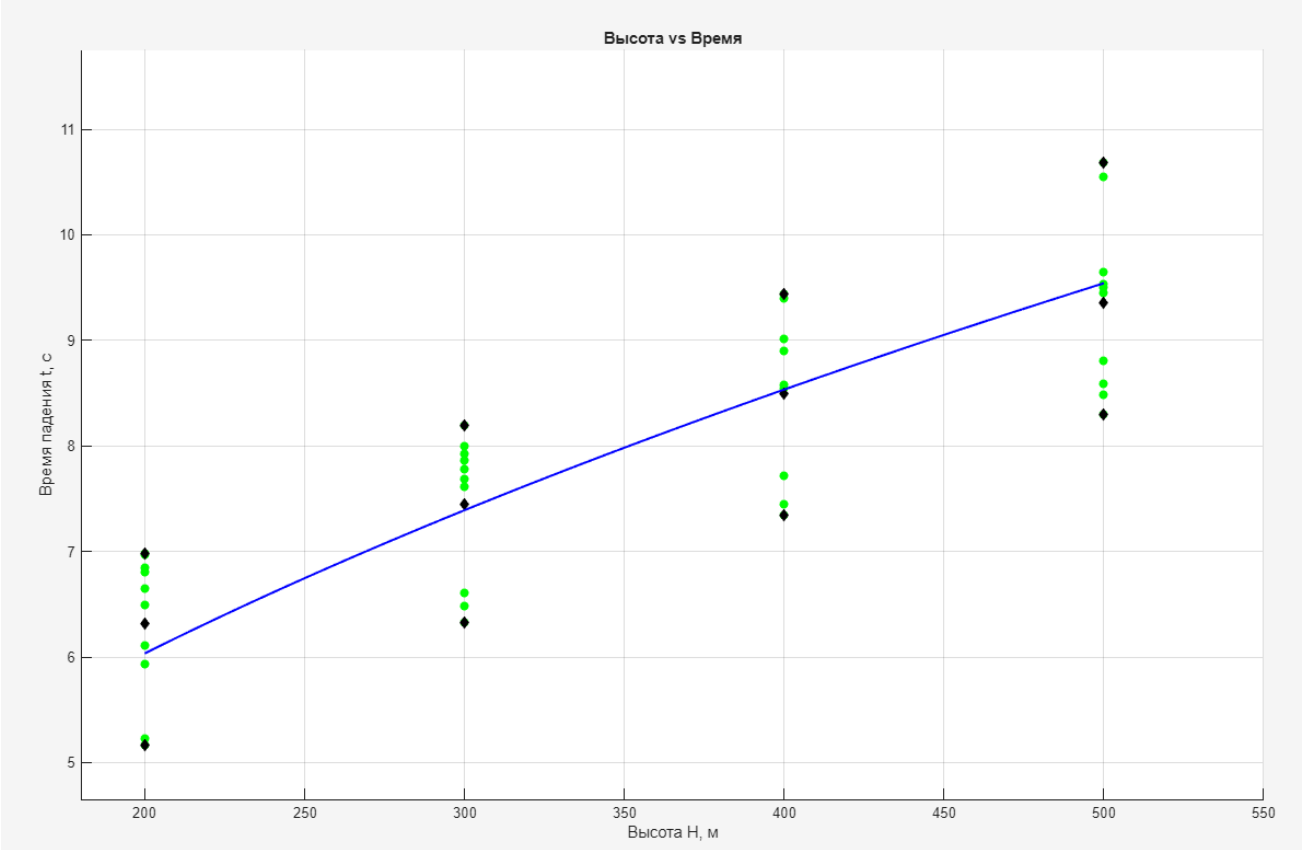

Рис 4. График "Высота vs Время".

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.

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

3.

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

function plotPoints(points)
    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;

end
```

```
function centroid = findCentroid(points)
    xs = arrayfun(@(p) p.x, points);
    ys = arrayfun(@(p) p.y, points);
    centroid = Point(mean(xs), mean(ys));
end
```

4.

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

% Plot points and centroid
plotPoints(points);
centroid = findCentroid(points);
hold on;
plot(centroid.x, centroid.y, 'rx', 'MarkerSize', 12, 'LineWidth', 2);
legend('Points', 'Centroid', 'Location', 'Best');

% Find two most distant points
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 #8  (9.34, 1.30) и #17 (2.29, 9.13). Дистанция -
10.54
>>Точка, ближайшую к центроиду - #5 (3.80, 5.68). Дистанция - 0.27
```

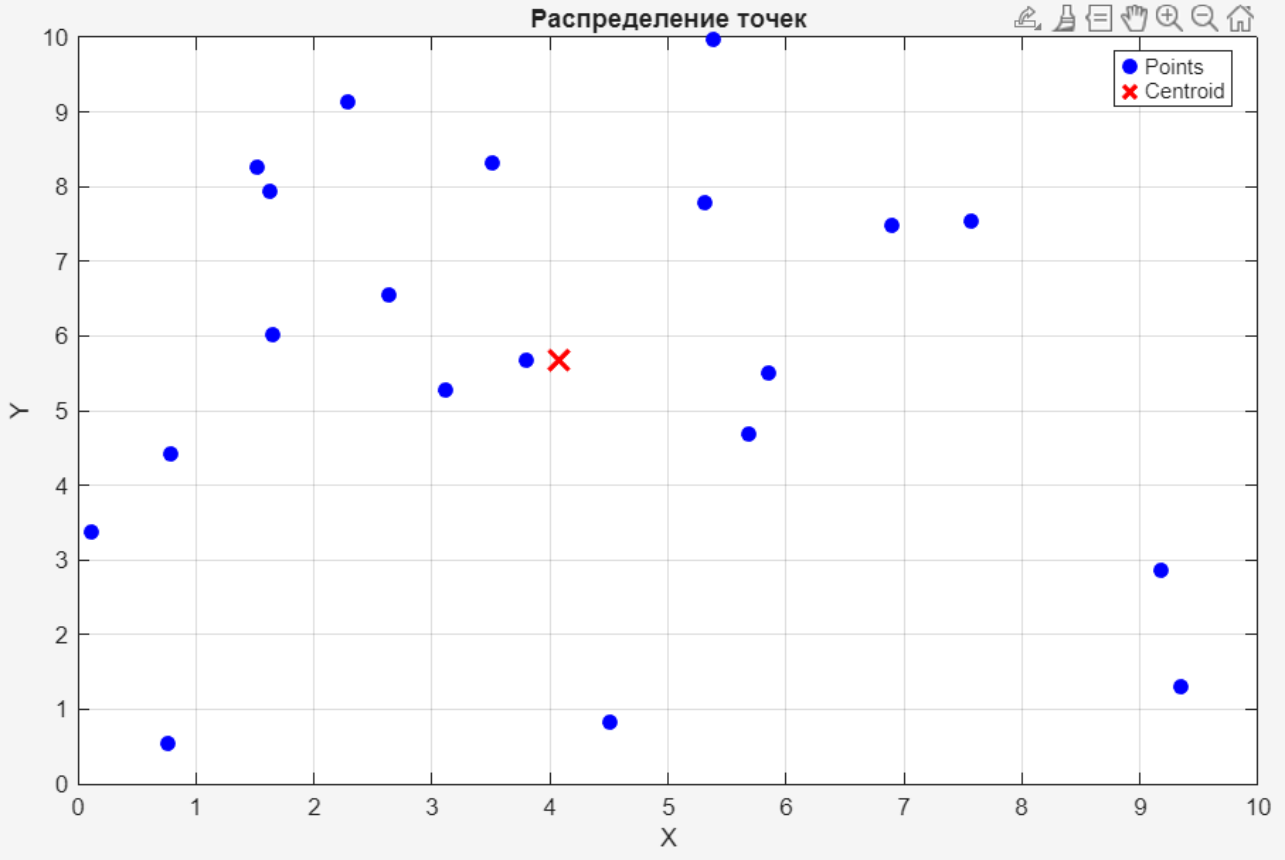

Рис.5

Задание:

Построить замкнутую кривую Гилберта, изображенную на рис. П-11.10. Под замкнутой кривой Гилберта понимается линия, состоящая из двух кривых Гилберта, которые являются зеркальными отражениями друг друга относительно вертикальной оси. Кривая Гилберта первого порядка состоит из трех отрезков: AB, BC и CD, длина которых L = 1. Кривая Гил-берта второго порядка состоит из четырех кривых Гилберта первого порядка, соединенных между собой, как показано на рис. П-11.10. Кривая Гилберта третьего порядка состоит из четырех кривых Гилберта второго порядка и т.д.

Код:

```
function closed_hilbert(n)
    axiom = 'A';
    rules = @(ch) switcher(ch, ...
        'A', '+BF-AFA-FB+', ...
        'B', '-AF+BF B+FA-', ...
        ch,   ch);
    iter_str = axiom;
    for k = 1:n
```

```
        s = '';
        for i = 1:length(iter_str)
            s = [s, rules(iter_str(i))];
        end
        iter_str = s;
    end

    N = length(iter_str);

    L = 1/(2^n - 1);

    x = zeros(N+1,1);
    y = zeros(N+1,1);
    dir = 0;
    for i = 1:N
        c = iter_str(i);
        switch c
            case 'F'
                x(i+1) = x(i) + L*cosd(dir);
                y(i+1) = y(i) + L*sind(dir);
            case '+'
                dir = mod(dir + 90,360);
                x(i+1)=x(i); y(i+1)=y(i);
            case '-'
                dir = mod(dir - 90,360);
                x(i+1)=x(i); y(i+1)=y(i);
            otherwise
                x(i+1)=x(i); y(i+1)=y(i);
        end
    end
    figure; hold on; axis equal off;
    plot(x, y, 'b-');

    xm = 1 - x;
    plot([x(end); xm(end:-1:1)], [y(end); y(end:-1:1)], 'r-');
    title(sprintf('Замкнутая кривая Гилберта порядка %d', n));
end

function out = switcher(ch, patA, repA, patB, repB, patDefault, repDefault)
    if ch == patA
        out = repA;
    elseif ch == patB
        out = repB;
    else
        out = repDefault;
    end
end
```

```
>> closed_hilbert(5)
```

Результат выполнения:

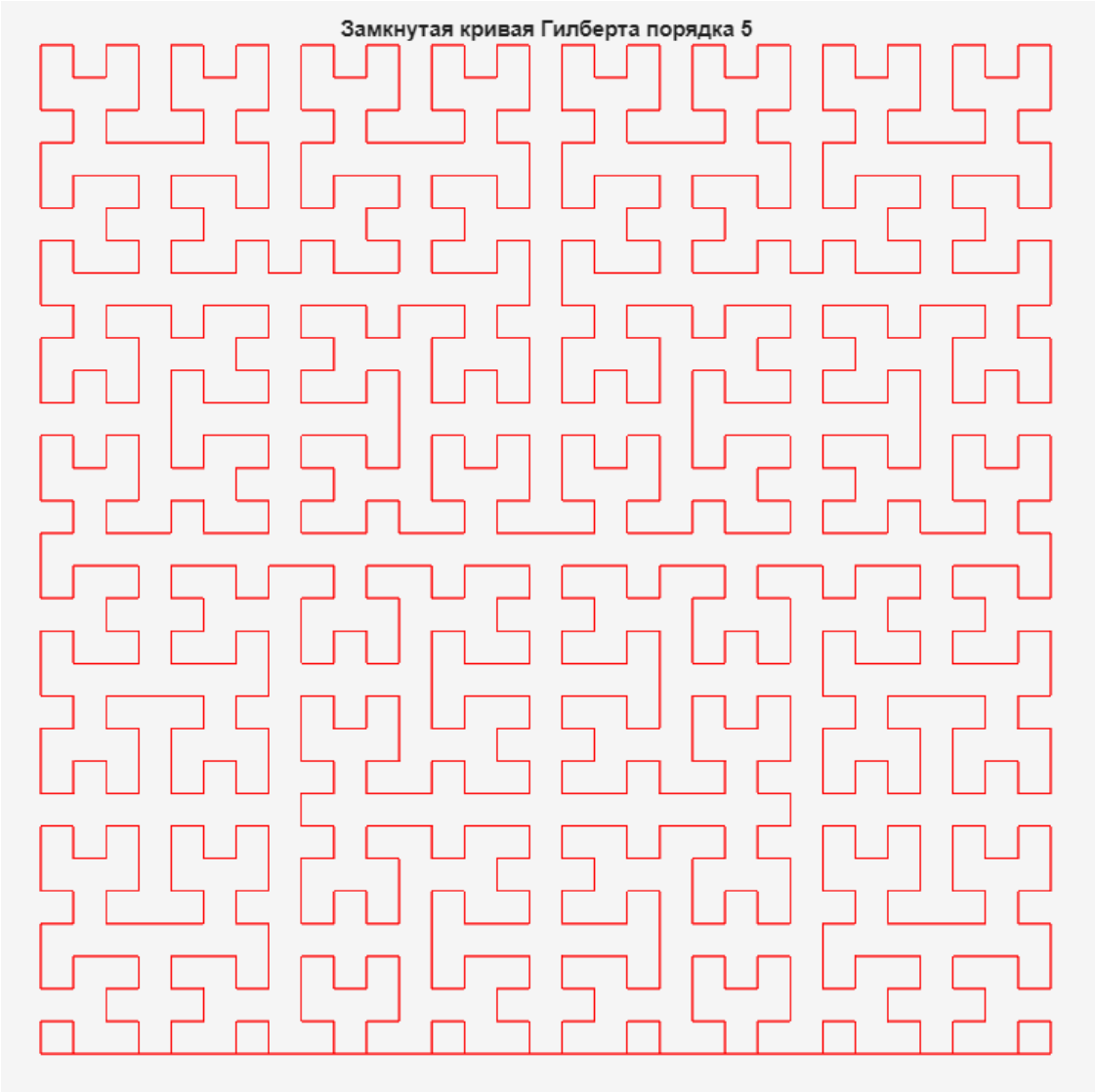

Рис. 6 Кривая Гилберта