МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНА МЕТАЛУРГІЙНА АКАДЕМІЯ УКРАЇНИ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з дисципліни

«ІНФОРМАЦІЙНІ ОСНОВИ СИНЕРГЕТИКИ І НАНОТЕХНОЛОГІЙ»

для студентів спеціальності 122 - «Комп'ютерні науки»
денної форми навчання

Дніпро НМетАУ 2019

УДК 681.3.07

Методичні вказівки до виконання лабораторних робіт з дисципліни «Інформаційні основи синергетики і нанотехнологій» для студентів спеціальності 122 – «Комп'ютерні науки» денної форми навчання / Укл. А.О. Журба. - Дніпро: НМетАУ, 2019 – 53 с.

Містяться робоча програма дисципліни «Інформаційні основи синергетики і нанотехнологій» з коротким викладом змісту її розділів і методичними вказівками до вивчення навчального матеріалу, а також завдання до контрольної роботи, на основі якої студент освоює теоретичні положення з інформаційних основ синергетики і нанотехнологій.

Призначені для студентів спеціальності 122 денної форми навчання, а також для слухачів курсів підвищення кваліфікації, студентів і аспірантів інших спеціальностей.

Укладачі: А.О. Журба, канд. техн. наук, доцент,

Друкується за авторською редакцією.

Затверджено на засіданні кафедри інформаційних технологій і систем, протокол № 9 від 06.03.2019 р.

Відповідальний за випуск О.І. Михальов, д-р техн. наук, проф.

Рецензент : О.І. Дерев'янко, канд. техн. наук, доцент (ДНУ)

Національна металургійна академія України.
49600, Дніпро, пр. Гагаріна, 4

СОДЕРЖАНИЕ

.

ВВЕДЕНИЕ

Синергетика (от греч. συν — «совместно» и греч. εργος — «действующий») — междисциплинарное направление научных исследований, задачей которого является изучение природных явлений и процессов на основе принципов самоорганизации систем (состоящих из подсистем). «…наука, занимающаяся изучением процессов самоорганизации и возникновения, поддержания, устойчивости и распада структур самой различной природы…»[1].

Синергетика изначально заявлялась как междисциплинарный подход, так как принципы, управляющие процессами самоорганизации, представляются одними и теми же, безотносительно природы систем и для их описания должен быть пригоден общий математический аппарат.

С мировоззренческой точки зрения синергетику иногда позиционируют, как «глобальный эволюционизм» или «универсальную теорию эволюции», дающую единую основу для описания механизмов возникновения любых новаций подобно тому, как некогда кибернетика определялась, как «универсальная теория управления», одинаково пригодная для описания любых операций регулирования и оптимизации: в природе, в технике, в обществе и т. п. и т. д.

Существует несколько основных направлений в синергетике, одно из которых – *теория фракталов* – занимается изучением сложных самоподобных структур, часто возникающих в результате самоорганизации.

Фрактал (лат. *fractus* — дробленый, сломанный, разбитый) – термин, означающий геометрическую фигуру, обладающую свойством самоподобия, то есть составленную из нескольких частей, каждая из которых подобна всей фигуре целиком.

Лабораторная работа №1
РЕКУРСИВНЫЙ АЛГОРИТМ ГЕНЕРАЦИИ ФРАКТАЛОВ

Цель: изучение основ построения фрактальных множеств с использованием рекурсивных алгоритмов.

Изучение алгоритма, основанного на использовании рекурсивной функции, проведем на примере построения простого самоподобного фрактала — ковра Серпинского. В рассматриваемом алгоритме используется способ построения, основанный на последовательном удалении из начальной области внутренних подобластей в соответствии с заданными правилами. Выберем в качестве начального множества S_0 равносторонний треугольник вместе с областью, которую он замыкает. Удалим внутренность центральной треугольной области и назовем оставшееся множество S_1 (рис. 1). Затем повторим описанный процесс для каждого из трех оставшихся треугольников и получим приближение S_2. Продолжая таким образом, получим последовательность вложенных множеств S_n, пересечение которых и образует ковер Серпинского *S* (рис. 1). Из построения видно, что ковер является объединением $N = 3$ существенно непересекающихся уменьшенных в два раза копий (коэффициенты подобия по горизонтали и вертикали в данном случае оказываются одинаковыми, $r = 1/2$). Фрактальная размерность ковра Серпинского *d*

$$d = \frac{\log(3)}{\log(2)} \approx 1.5850 \qquad (1)$$

Для построения рассматриваемого фрактала, очевидно, можно использовать следующий алгоритм.

1. Задать порядок ковра *N*.

2. Задать координаты вершин исходного треугольника ABC: (X_A, Y_A), (X_B, Y_B), (X_C, Y_C)

3. Построить равносторонний треугольник *ABC* и залить его синим цветом.

4. Вычислить координаты середин сторон треугольника *ABC*:

$$dx = \frac{X_B - X_A}{2},\ dy = \frac{Y_B - Y_A}{2},$$

$$X_{A'} = X_A + dx,\ Y_{A'} = Y_A + dy,$$

$$X_{B'} = X_B + dx + dx/2,\ Y_{B'} = Y_B + dy,$$

$$X_{C'} = X_C + dx/2,\ Y_C = Y_C + dy$$

5. Построить треугольник $A'B'C'$ и залить его белым цветом.

6. Повторить N раз действия, описанные в пп. 4, 5, для треугольников $AA'C'$, $A'BB'$, $C'B'C$, соответственно.

Наиболее просто описанный выше алгоритм можно реализовать при использовании рекурсивной процедуры, выполняющей последовательность действий, описанных в пп. 4, 5. Для реализации алгоритма в пакете MATLAB следует создать специальную функцию, возвращающую изображение ковра Серпинского, используя для этого встроенный текстовый редактор MATLAB или любой другой текстовый редактор (например, «Блокнот»), и сохранить текст в файле Serpinsky.m.

Для вывода изображения ковра Серпинского, например, пятого порядка, следует ввести в командной строке пакета MATLAB имя функции с соответствующим значением:

```
>> Serpinsky(5);
```

Результат, возвращаемый функцией Serpinsky(), показан на рис. 1 (множество S_5).

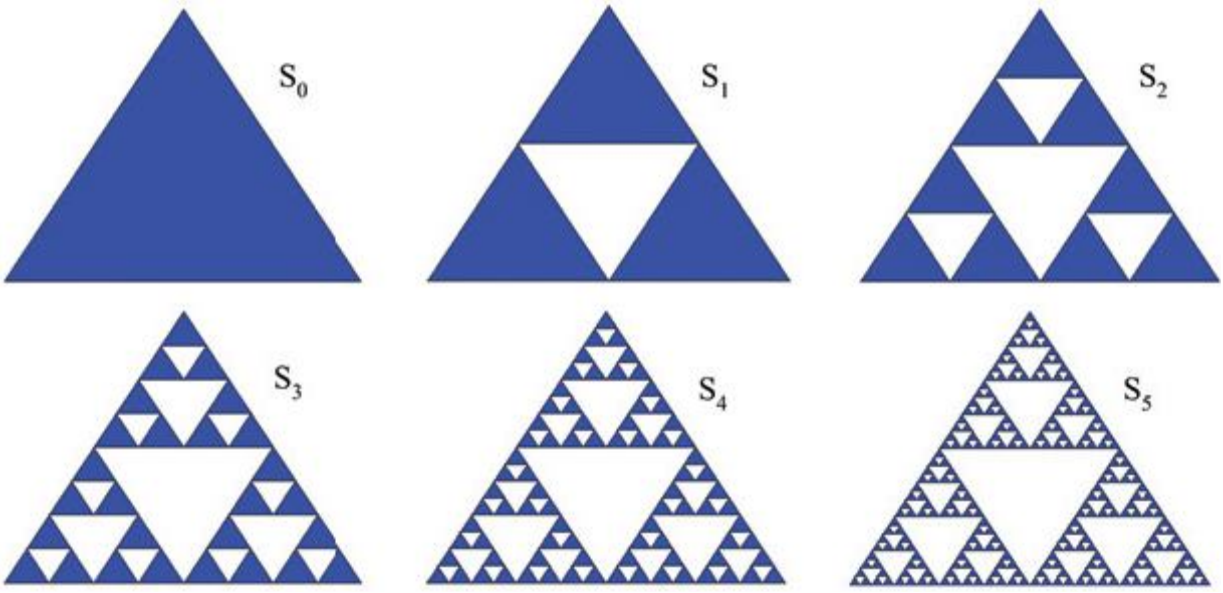

Рис. 1 – Построение ковра Серпинского

Описанный выше алгоритм легко обобщается для фрактальных объектов, правила построения которых аналогичны правилам построения треугольного ковра Серпинского. Например, алгоритм визуализации фрактального объекта,

основные этапы построения которого представлены на рис. 2, реализуется следующей последовательностью действий.

1. Задать порядок ковра *N*.

2. Задать координаты вершин исходного квадрата $ABCD$: $(X_A,Y_A),\ (X_B,Y_B),\ (X_C,Y_C),\ (X_D,Y_D)$

3. Построить квадрат *ABCD* и залить его синим цветом.

4. Вычислить координаты точек, делящих стороны квадрата *ABCD* на три равные части:

$$dx = \frac{X_B - X_A}{3},\ dy = \frac{Y_B - Y_A}{3}$$

$$X_{A'} = X_A + dx,\ Y_{A'} = Y_A + dy,$$

$$X_{B'} = X_A + dx + dx,\ Y_{B'} = Y_A + dy,$$

$$X_{C'} = X_A + dx + dx,\ Y_{C'} = Y_A + dy + dy,$$

$$X_{D'} = X_A + dx,\ Y_{D'} = Y_A + dy + dy$$

5. Построить квадрат *A′B′C′D′* и залить его белым цветом.

6. Повторить *N* раз действия, описанные в пп. 4, 5, для квадратов с вершинами, имеющими следующие координаты:

$$(X_A,Y_A),\ (X_{A'},Y_A),\ (X_{A'},Y_{A'}),\ (X_A,Y_{A'}),$$
$$(X_{A'},Y_A),\ (X_{B'},Y_A),\ (X_{B'},Y_{B'}),\ (X_{A'},Y_{B'});$$
$$(X_{B'},Y_A),\ (X_B,Y_B),\ (X_B,Y_{B'}),\ (X_{B'},Y_{B'}),$$
$$(X_{B'},Y_{B'}),\ (X_B,Y_{B'}),\ (X_B,Y_{C'}),\ (X_{C'},Y_{C'});$$
$$(X_{C'},Y_{C'}),\ (X_B,Y_{C'}),\ (X_C,Y_C),\ (X_{C'},Y_{C'}),$$
$$(X_{D'},Y_{D'}),\ (X_{C'},Y_{C'}),\ (X_{C'},Y_{C'}),\ (X_{D'},Y_{C'});$$
$$(X_A,Y_{D'}),\ (X_{D'},Y_{D'}),\ (X_{D'},Y_C),\ (X_D,Y_D),$$
$$(X_A,Y_{A'}),\ (X_{A'},Y_{D'}),\ (X_{D'},Y_{D'}),\ (X_A,Y_{D'})$$

соответственно.

Ниже приведен листинг файла Serpinsky2.m, который содержит описание функции, возвращающей изображение квадратного ковра Серпинского, представленного на рис. 2.

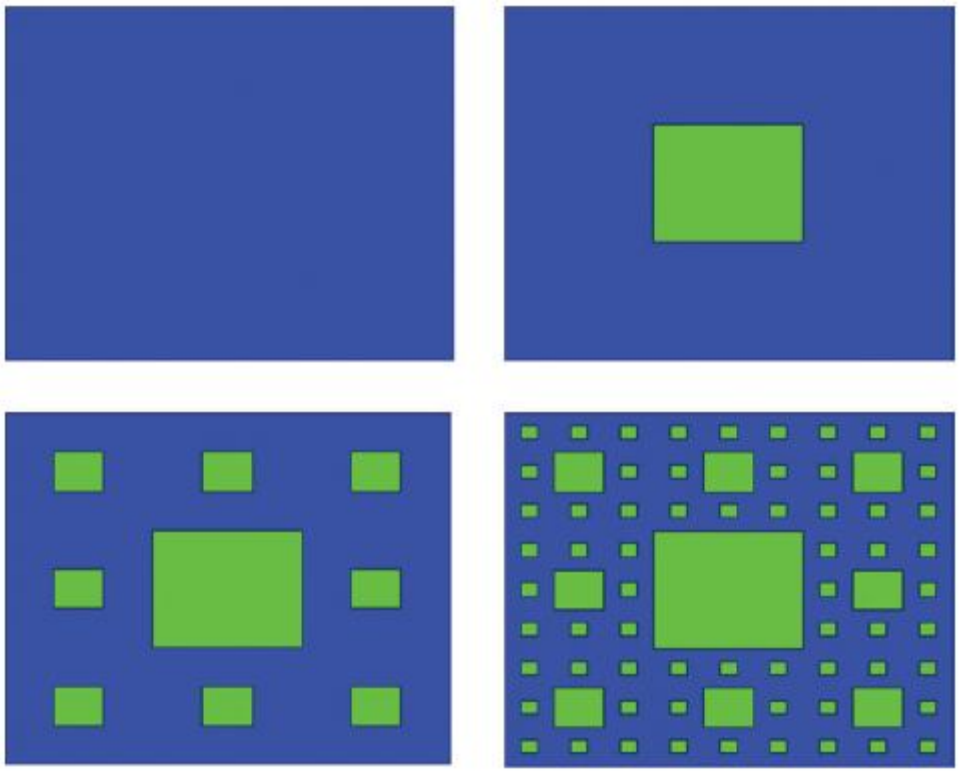

Рис. 2 – Этапы построения квадратного ковра Серпинского

Листинг файла для построения треугольника Серпинского

Serpinsky.m

```
function z = Serpinsky(Lmax)
% функция, возвращающая изображение ковра Серпинского
% Lmax — порядок ковра
% задание координат вершин равнобедренного треугольника
x1=0; y1=0; x2=1; y2=0; x3=0.5; y3=sin(pi/3);
h=figure(1); % инициализация графического окна
hold on; % включение режима рисования фигур в одном
графическом окне
fill([x1 x2 x3],[y1 y2 y3],'k');
% прорисовка равностороннего треугольника
set(gca,'xtick',[],'ytick',[]); % отключение режима
оцифровки осей
set(gca,'XColor','w','YColor','w'); % установка цвета
рисования осей
Simplex(x1,y1,x2,y2,x3,y3,0,Lmax);
% обращение к функции, прорисовывающей равносторонние
треугольники
% белого цвета
hold off % отключение режима рисования фигур в одном
графическом окне
```

```
function z=Simplex(x1,y1,x2,y2,x3,y3,n,Lmax)
% рекурсивная функция, прорисовывающая равносторонние
треугольники
% белого цвета
if n<Lmax
% задание координат вершин текущего равностороннего
треугольника
dx=(x2-x1)/2; dy=(y3-y1)/2; x1n=x1+dx; y1n=y1;
x2n=x1+dx+dx/2;
y2n=y1+dy;x3n=x1+dx/2; y3n=y1+dy;
fill([x1n x2n x3n],[y1n y2n y3n],'w');
% прорисовка текущего равностороннего треугольника
n=n+1;
% рекурсия
Simplex(x1,y1,x1n,y1n,x3n,y3n,n,Lmax);
Simplex(x1n,y1n,x2,y2,x2n,y2n,n,Lmax);
Simplex(x3n,y3n,x2n,y2n,x3,y3,n,Lmax);
end
```

Листинг файла для построения ковра Серпинского

Serpinsky2.m

```
function z=Serpinsky2(Lmax)
% функция, возвращающая изображение квадратного ковра
Серпинского
% Lmax — порядок ковра
% Задание координат вершин исходного квадрата
x1=0; y1=0; x2=1; y2=0; x3=1; y3=1; x4=0; y4=1;
figure(1); hold on; fill([x1 x2 x3 x4],[y1 y2 y3 y4],'b');
set(gca,'xtick',[],'ytick',[]);
set(gca,'XColor','w','YColor','w');
Quadrate(x1,y1,x2,y2,x3,y3,x4,y4,0,Lmax);
hold off

function z=Quadrate(x1,y1,x2,y2,x3,y3,x4,y4,n,Lmax)
% рекурсивная функция, прорисовывающая квадраты белого
цвета
if n<Lmax
dx=(x2-x1)/3; dy=(y3-y1)/3;
x1n=x1+dx; y1n=y1+dy; x2n=x1+dx+dx; y2n=y1+dy;
x3n=x1+dx+dx; y3n=y1+dy+dy; x4n=x1+dx; y4n=y1+dy+dy;
fill([x1n x2n x3n x4n],[y1n y2n y3n y4n],'g');
n=n+1;
```

```
Quadrate(x1,y1,x1+dx,y1,x1+dx,y1+dy,x1,y1+dy,n,Lmax);
Quadrate(x1+dx,y1,x1+2*dx,y1,x1+2*dx,y1+dy,x1+dx,y1+dy,n,L
max);
Quadrate(x1+2*dx,y1,x2,y1,x2,y1+dy,x1+2*dx,y1+dy,n,Lmax);
Quadrate(x1+2*dx,y1+dy,x2,y1+dy,x2,y1+2*dy,x1+2*dx,y1+2*dy
,n,Lmax);
Quadrate(x1+2*dx,y1+2*dy,x2,y1+2*dy,x2,y3,x1+2*dx,y3,n,Lma
x);
Quadrate(x1+dx,y1+2*dy,x1+2*dx,y1+2*dy,x1+2*dx,y4,x1+dx,y4
,n,Lmax);
Quadrate(x1,y1+2*dy,x1+dx,y1+2*dy,x1+dx,y4,x1,y4,n,Lmax);
Quadrate(x1,y1+dy,x1+dx,y1+dy,x1+dx,y1+2*dy,x1,y1+2*dy,n,L
max);
end
```

Еще одним примером фрактального объекта, для построения которого оказывается удобным использовать рекурсивный алгоритм, является кривая Коха. Построение данной кривой начинается с отрезка K_0 единичной длины. Удалим из отрезка K_0 отрезок длины $1/3$ и добавим два новых отрезка такой же длины, как показано на рис. 3.

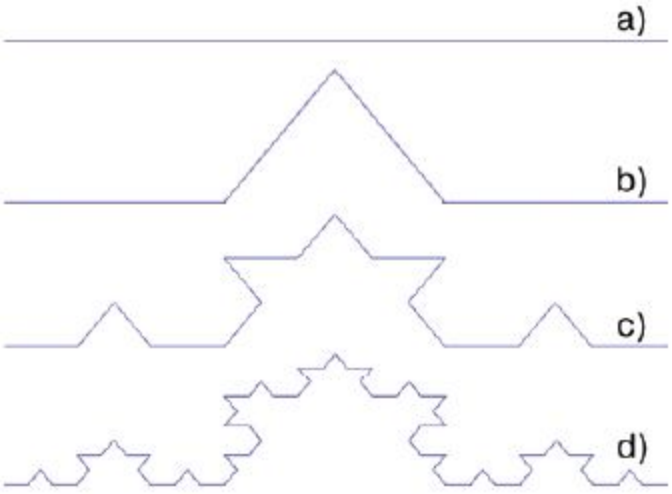

Рис. 3 – Построение кривой Коха: a) K_0, b) K_1, c) K_2, d) K_3

Назовем полученное множество K_1. На **с**ледующем шаге разделим каждый отрезок длины $1/3$ на три части длины $1/9$ и повторим описанную процедуру, заменяя на каждом шаге среднюю ветвь двумя новыми отрезками. Обозначим через K_n фигуру, получившуюся после *n*-го шага. Можно строго

доказать [6], что последовательность кривых $\{K_n\}_1^\infty$ сходится к предельной кривой K бесконечной длины, фрактальная размерность которой равна $d = \frac{\log(4)}{\log(3)} \approx 1,2618$.

Ниже приводится листинг рекурсивной функции, возвращающей изображение кривой Коха.

Листинг рекурсивной функции, возвращающей изображение кривой Коха

```
function z=Koch(N)
% функция, возвращающая изображение кривой Коха
x1=0; y1=0; % левая точка начального отрезка
x2=1; y2=0; % правая точка начального отрезка
figure(1); axis([0 1 0 1]);
set(gca,'XColor','w','YColor','w');
hold on;
Coord(x1,y1,x2,y2,N);
% вызов рекурсивной функции, прорисовывающей кривую Коха
function z=Coord(x1,y1,x2,y2,n)
% рекурсивная функция, прорисовывающая кривую Коха
if n>0
% вычисление координат концов отрезков на очередном шаге
рекурсии
dx=(x2-x1)/3; dy=(y2-y1)/3;
x1n=x1+dx; y1n=y1+dy;
x2n=x1+2*dx; y2n=y1+2*dy;
xmid=dx/2-dy*sin(pi/3)+x1n; ymid=dy/2+dx*sin(pi/3)+y1n;
% рекурсия
Coord(x1,y1,x1n,y1n,n-1);
Coord(x1n,y1n,xmid,ymid,n-1);
Coord(xmid,ymid,x2n,y2n,n-1);
Coord(x2n,y2n,x2,y2,n-1);
else
r1=[x1 y1]; r2=[x2 y2]; R=cat(1,r1,r2);
plot(R(:,1),R(:,2),'Color','k'); % построение кривой Коха
end;
```

Изображение кривой Коха пятого порядка K_5, возвращенное описанной выше функцией, представлено на рис. 4.

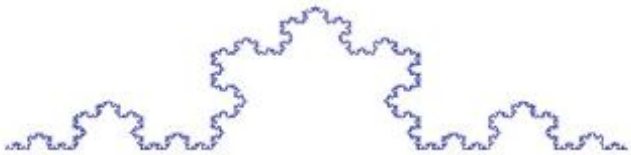

Рис. 4 – Кривая Коха пятого порядка

ЗАДАНИЕ

1. С помощью рекурсивного алгоритма генерации фракталов, построить треугольный ковёр Серпинского, базовым элементом для которого является равносторонний треугольник.

2. С помощью рекурсивного алгоритма генерации фракталов, построить треугольный ковёр Серпинского, базовым элементом для которого является прямоугольный треугольник.

3. С помощью рекурсивного алгоритма генерации фракталов, построить квадратный ковер Серпинского.

4. С помощью рекурсивного алгоритма генерации фракталов, построить кривую Коха, чтобы угол между центральными отрезками составлял 60 градусов.

5. С помощью рекурсивного алгоритма генерации фракталов, построить кривую Коха, чтобы угол между центральными отрезками составлял 90 градусов.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое фрактал? Его основное свойство.

2. Что представляют собой рекурсивные алгоритмы построения фракталов?

3. Как изменяется внешний вид фрактала при увеличении количества итераций?

4. Что такое самоподобие?

5. Приведите примеры фракталов.

Лабораторная работа №2
L-СИСТЕМЫ И ТЕРЛ-ГРАФИКА

Цель: изучение формальной грамматики, используемой в компьютерной графике, для построения фрактальных изображений.

Понятие «L-система» было введено А. Лидермайером в 1968 г. при изучении формальных языков. С их помощью оказывается возможным не только строить многие известные самоподобные фракталы, например, снежинку Коха, ковер Серпинского, кривые Пеано, Гильберта, Серпинского и др., но и создавать бесконечное разнообразие новых фракталов, укладывающихся в данную схему [6].

Терл-графика (от turtle — черепашка) является подсистемой вывода графического представления фрактального объекта. Основным исполнителем данной системы является «черепашка» (точка), которая перемещается по экрану дискретными шагами, прочерчивая или не прочерчивая свой след. «Мгновенное» положение «черепашки» задается тремя параметрами (x, y, α), где (x, y) — координаты «черепашки», α — направление следующего шага (угол, отсчитываемый от положительного направления оси x). Последовательность команд, определяющая направление перемещения и действия «черепашки», задается кодовым словом, буквы которого читаются слева направо. Кодовое слово, представляющее собой результат работы L-системы, может включать в себя следующие буквы:

F — переместиться на один шаг вперед, прорисовывая след;

b — переместиться на один шаг вперед, не прорисовывая след;

[— открыть ветвь;

] — закрыть ветвь;

+ — увеличить угол α на величину Θ;

− — уменьшить угол α на величину Θ;

X , Y — вспомогательные переменные.

Размер шага и величина приращения по углу Θ задаются изначально и остаются неизменными для всех перемещений «черепашки». Формально, детерминированная L-система состоит из алфавита, слова инициализации, называемого аксиомой или инициатором, и набора порождающих правил, указывающих как следует преобразовывать слово при каждой следующей

итерации. Например, L-система, соответствующая снежинке Коха, представленной на рис. 5, задается следующим образом:

Аксиома: **F++F++F**

Порождающее правило: **Newf=F-F++F-F**

Θ:=π/3

Очевидно, что графическим представлением аксиомы F++F++F является равносторонний треугольник. «Черепашка» делает один шаг вперед, затем угол увеличивается на 2π/3 и «черепашка»делает еще один шаг вперед, далее угол вновь увеличивается на 2π/3 и «черепашка» делает еще один шаг.

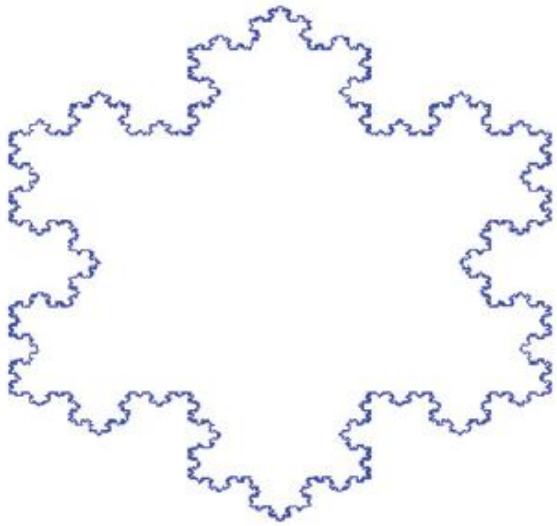

Рис. 5 – Снежинка Коха 4-го порядка

На первом шаге каждая буква *F* в слове-инициаторе заменяется на слово *Newf* :

```
F-F++F-F++F-F++
 F-F++F-F++F-F.
```

Повторяя этот процесс, на втором шаге получим:

```
F-F++F-F-F-F++F-F++
F-F++F-F-F-F++F-F++
F-F++F-F-F-F++F-F++
F-F++F-F-F-F++F-F++
 F-F++F-F-F-F++F-F
```

и т. д.

В пакете MATLAB наиболее просто реализовать L-систему, используя рекурсивную функцию. Ниже приведен пример рекурсивной функции, возвращающей снежинку Коха, представленную на рис. 5.

Листинг функции, вычисляющей путь для построения снежинки Коха

```
function z=RuleKoch(Lmax,Axiom,Newf,n,tmp);
% функция, возвращающая L-систему
% для снежинки Коха
% Входные параметры:
% Lmax — порядок снежинки
% Axiom — строка, содержащая аксиому
% Newf — строка, содержащая порождающее правило
% tmp — входная L-система
while n<=Lmax
if n==1
tmp=Axiom; n=n+1;
else
tmp=strrep(tmp,'F',Newf);
% замена всех букв F на порождающее правило
n=n+1;
tmp=RuleKoch(Lmax,Axiom,Newf,n,tmp);
% рекурсия
end
end
z=tmp;
```

Для вывода изображения снежинки Коха можно использовать следующую m-функцию, реализующую описанный выше алгоритм терл-графики:

Листинг функции, выводящей изображение снежинки Коха

```
function [X,Y]=Snowflake (Lmax)
% функция, возвращающая изображение
% снежинки Коха
% Lmax — порядок снежинки
% порождающие правила
Axiom='F++F++F';
Newf='F-F++F-F';
teta=pi/3;
alpha=0;
p=[0;0]; % стартовая точка
p=Coord(p,Lmax,Axiom,Newf,alpha,teta);
% обращение к функции, возвращающей координаты вершин
M=size(p,2); % число вершин снежинки Коха
X=p(1:1,1:M);
```

```
% создание вектора, содержащего X-е координаты вершин
снежинки
Y=p(2:2,1:M);
% создание вектора, содержащего Y-е координаты вершин
снежинки
figure(1); % инициализация графического окна
plot(X,Y,'Color','k'); % построение снежинки Коха
function z=Coord(p,Lmax,Axiom,Newf,alpha,teta)
% функция, возвращающая координаты вершин снежинки Коха
Rule=RuleKoch(Lmax,Axiom,Newf,1,' '); % задание L-системы
M=length(Rule);
for i=1:M
Tmp=p(1:2,size(p,2):size(p,2));
if Rule(i)=='F' % шаг вперед
R=[cos(alpha);sin(alpha)]; R=R/(4^Lmax);
Tmp=Tmp+R; p=cat(2,p,Tmp);
end
if Rule(i)=='+' % увеличение угла, задающего направление
движения
alpha=alpha+teta;
end
if Rule(i)=='-' % уменьшение угла, задающего направление
движения
alpha=alpha-teta;
end
end
z=p;
```

Однако при построении других фракталов, например, дракона Хартера-Хайтвея (рис. 6), на некоторых шагах возникает необходимость в изменении направления чтения правила (не слева направо, а справа налево). Для решения данной проблемы вводят две дополнительные команды, обозначаемые *X* и *Y*, которые используются для создания соответствующей L-системы, но игнорируются «черепашкой» при перемещении. При использовании этих команд порождающее правило для дракона имеет вид:

Аксиома: **FX**

Порождающие правила:

Newf=FX

Newx=X+YF+

Newy=-FX-Y

В соответствии с данными правилами L-система имеет следующий вид:

1 шаг: `FX+Y+F`

2 шаг: `FX+YF++-FX-YF+`

3 шаг:

```
FX+YF++-FX-YF++
-FX+YF+--FX-YF+
```

4 шаг:

```
 FX+YF++-FX-YF++
-FX+YF+--FX-YF++
-FX+YF++-FX-YF+-
-FX+YF+--FX-YF+
```

Ниже приводится листинг файла Dracon.m, содержащего описание функции, возвращающей изображение дракона, в соответствии с описанной выше L-системой.

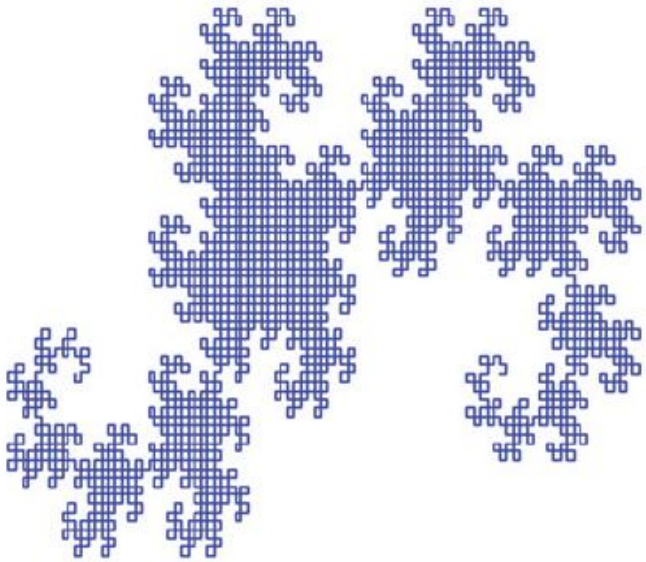

Рис. 6 – Дракон Хаттера-Хайтвея 12-го порядка

Листинг файла Dracon.m для построения дракона Хаттера-Хайтвея

```
function [X,Y]=Dracon(Lmax)
% функция, возвращающая изображение дракона
% Lmax — порядок дракона
% порождающие правила
Axiom='FX';
Newf='F';
Newx='X+YF+';
Newy='-FX-Y';
```

```
teta=pi/2; alpha=0; p=[0;0];
p=Coord(p,Lmax,Axiom,Newf,Newx,Newy,alpha,teta);
% обращение к функции, возвращающей координаты угловых
точек дракона
M=size(p,2);
X=p(1:1,1:M);
% создание вектора, содержащего X-е координаты угловых
точек дракона
Y=p(2:2,1:M);
% создание вектора, содержащего Y-е координаты угловых
точек дракона
figure(1); plot(X,Y,'Color','b');
set(gca,'xtick',[],'ytick',[]);
set(gca,'XColor','w','YColor','w');
function z=Coord(p,Lmax,Axiom,Newf,Newx,Newy,alpha,teta)
% функция, возвращающая координаты угловых точек дракона
Rule=DraconString(Lmax,Axiom,Newf,Newx,Newy,1,''); %
задание L-системы
M=length(Rule);
for i=1:M
Tmp=p(1:2,size(p,2):size(p,2));
if Rule(i)=='F' % шаг вперед
R=[cos(alpha);sin(alpha)]; R=R/(2^Lmax);
Tmp=Tmp+R; p=cat(2,p,Tmp);
end
if Rule(i)=='+' % увеличение угла, задающего направление
движения
alpha=alpha+teta;
end
if Rule(i)=='-' % уменьшение угла, задающего направление
движения
alpha=alpha-teta;
end
end
z=p;
function z=DraconString(Lmax,Axiom,Newf,Newx,Newy,n,tmp);
% функция, возвращающая L-систему
a=1;
if n<=Lmax
if n==1 tmp=Axiom; end
M=length(tmp); tmp1='';
for i=1:M
if tmp(i)=='F' tmp1=strcat(tmp1,Newf); end
```

```
if tmp(i)==’X’ tmp1=strcat(tmp1,Newx); end
if tmp(i)==’Y’ tmp1=strcat(tmp1,Newy); end
if not(tmp(i)==’F’) &not(tmp(i)==’X’) &not(tmp(i)==’Y’)
tmp1=strcat(tmp1,tmp(i)); end
end
end
tmp=tmp1; n=n+1;
tmp=DraconString(Lmax,Axiom,Newf,Newx,Newy,n,tmp);
end;
z=tmp;
```

Заменяя в описанной программе порождающие правила, можно получить и другие фрактальные кривые, например, кривую Гильберта (рис. 7):

Аксиома: **X**

Порождающие правила:

Newf=F

Newx=-YF+XFX+FY-

Newy=+XF-YFY-FX+

$\alpha = 0, \theta = \pi / 2.$

Или кривую Госпера (рис. 8):

Аксиома: XF

Порождающие правила:

Newx=X+YF++YF−FX−−FXFX−YF+

Newy=−FX+YFYF++YF+FX−−FX−Y

$\alpha = 0, \theta = \pi / 2$

Или кривую Серпинского (рис. 9):

Аксиома: F+XF+F+XF

Порождающие правила:

Newf=F

Newx=XF-F+F-XF+F+XF-F+F-X

Newy=””

$\alpha = \pi / 4, \theta = \pi / 2$

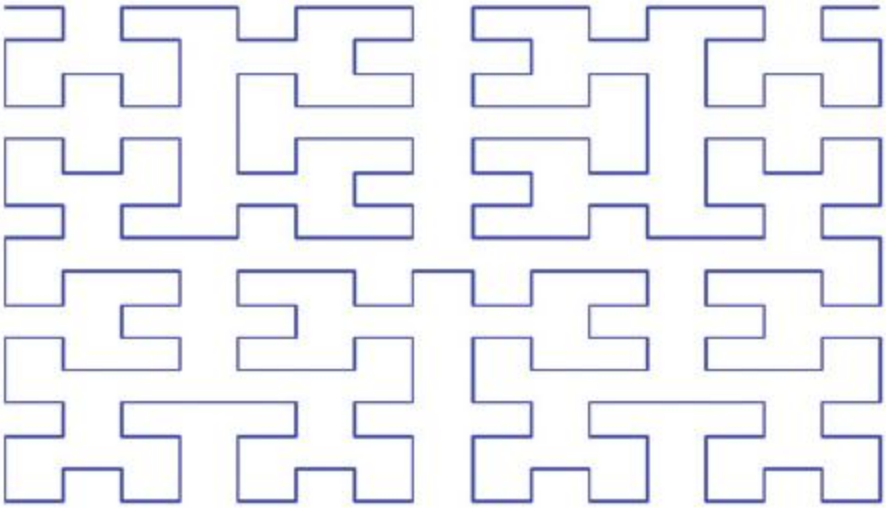

Рис. 7 – Кривая Гильберта 4-го порядка

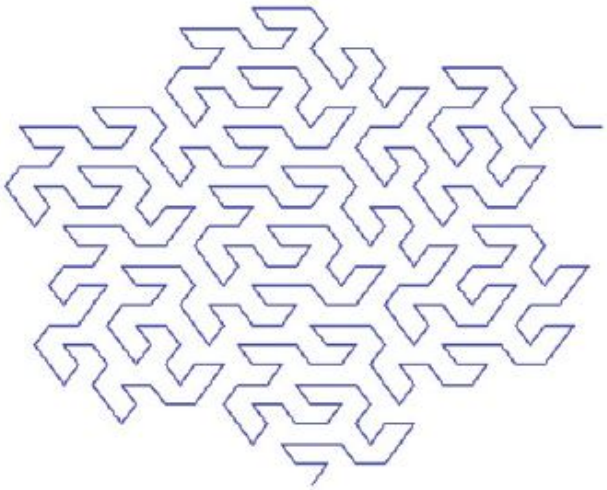

Рис. 8 – Кривая Госпера 3-го порядка

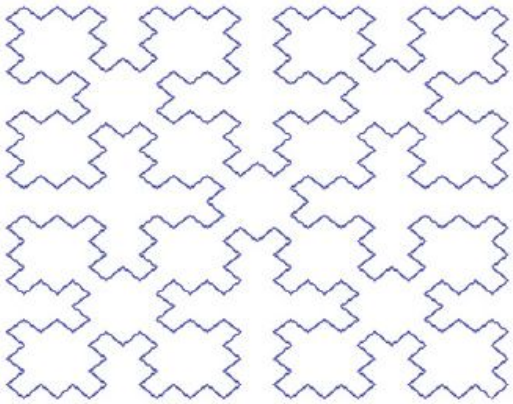

Рис. 9 – Кривая Серпинского 3-го порядка

В заключение остановимся на операции ветвления. Когда в L-системе встречается символ [(открыть ветвь), необходимо запомнить координаты точки нахождения «черепашки» и направление ее движения, т. е. переменные (x,y,α). К сохраненным переменным следует вернуться после обнаружения символа] (закрыть ветвь). Для хранения триплетов (x, y, α) в [6] предлагается использовать стек:

$$\begin{bmatrix} x_1 & y_1 & \alpha_1 \\ x_2 & y_2 & \alpha_2 \\ \vdots & \vdots & \vdots \\ x_n & y_n & \alpha_n \end{bmatrix},$$

в конец которого записываются новые данные. При закрытии ветви переменным (x, y, α) присваиваются значения, считанные из конца стека, затем эти значения из стека удаляются. В пакете MATLAB оказывается более удобным использовать матрицу с переменным числом столбцов:

$$M = \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ \alpha_1 & \alpha_2 & \dots & \alpha_n \end{bmatrix}$$

причем координаты каждой новой точки ветвления добавляются в новый столбец матрицы M. После закрытия ветви переменным (x, y, α) присваиваются значения, считанные из последнего столбца матрицы M, затем этот столбец удаляется.

Таким образом, ветвление задается двумя символами:

[— открыть ветвь: добавить вектор $\begin{bmatrix} x \\ y \\ \alpha \end{bmatrix}$, как новый столбец матрицы M,

] — закрыть ветвь: присвоить переменным (x, y, α) значения координат вектора, являющегося последним столбцом матрицы M.

Пример фрактала, построенного с помощью операции ветвления, представлен на рис. 10. Ниже приводится листинг файла Flower.m, содержащего описание функции, возвращающей изображение цветка, в соответствии с описанной выше L_системой.

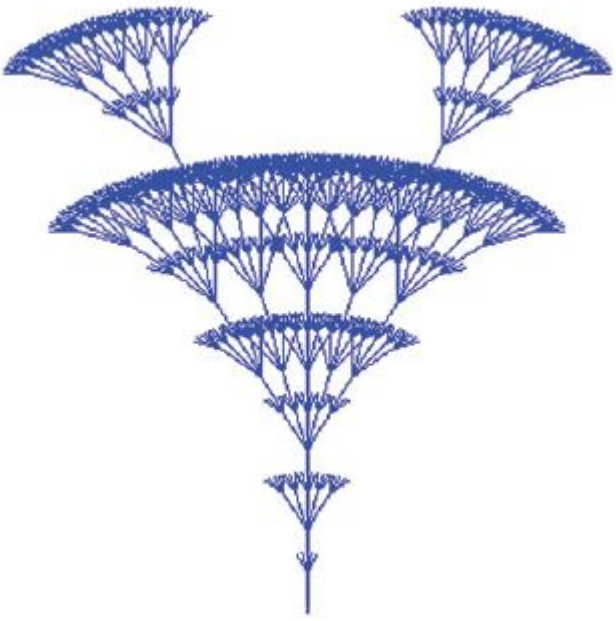

Рис. 10 – Цветок 3-го порядка

Листинг файла Flower.m

```
function [X,Y]=Flower(Lmax)
% функция, возвращающая изображение цветка
% Lmax - порядок цветка
% порождающие правила
Axiom='F[+F+F][-F-F][++F][-F]F';
Newf='FF[++F][+F][F][-F][-F]';
teta=pi/16;
alpha=pi/2;
p=[0;0]; % начальная точка
Coord(p,Lmax,Axiom,Newf,alpha,teta);
% обращение к функции, возвращающей
% изображение цветка
function z=Coord(p,Lmax,Axiom,Newf,alpha,teta)
% функция, возвращающая изображение цветка
Rule=FlowerString(Lmax,Axiom,Newf,1,'');
% задание L-системы
figure(1); hold on;
M=length(Rule);L=0; x0=p(1);y0=p(2);
for i=1:M
if Rule(i)=='F' % шаг вперед
x1=x0+cos(alpha); y1=y0+sin(alpha);
X=[x0,x1]; Y=[y0,y1]; x0=x1; y0=y1;
plot(X,Y,'Color','k'); end
if Rule(i)=='+' % увеличение угла, задающего направление
движения
alpha=alpha+teta;
end
if Rule(i)=='-' % уменьшение угла, задающего направление
движения
alpha=alpha-teta;
end
if Rule(i)=='['; % открыть ветвь
if L==0 St=[x0;y0;alpha]; L=1; else
St=cat(2,St,[x0;y0;alpha]);
end
end
if Rule(i)==']' % закрыть ветвь
M=size(St,2); R=St(1:3,M:M);
x0=R(1); y0=R(2);
alpha=R(3); St=St(1:3,1:M!1);
end
```

```
end
hold off
function z=FlowerString(Lmax,Axiom,Newf,n,tmp)
% функция, возвращающая L-систему
while n<=Lmax
if n==1
tmp=Axiom; n=n+1; else
tmp=strrep(tmp,'F',Newf);
n=n+1; tmp=FlowerString(Lmax,Axiom,Newf,n,tmp); % рекурсия
end
end
z=tmp;
```

Заменяя в описанной программе порождающие правила, можно получить и другие ветвящиеся фрактальные объекты, например, куст (рис. 11):

Аксиома: F

Порождающие правила:

$Newf = -F + F + [+F - F-] - [-F + F + F]$

$\alpha = \pi / 2, \theta = \pi / 8$

Или снежинку (рис. 12):

Аксиома: $[F]+[F]+[F]+[F]+[F]+[F]$

Порождающие правила:

$Newf = F[++F][-FF]FF[+F][-F]FF$

$\alpha = 0, \theta = \pi / 3$

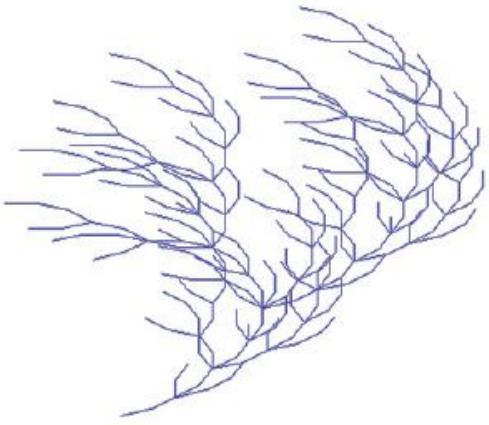

Рис. 11 – Куст 3-го порядка

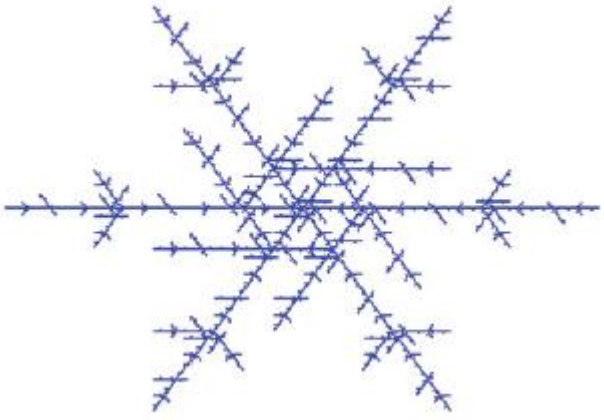

Рис. 12 – Снежинка 3-го порядка

ЗАДАНИЕ

1. Построить снежинку Коха, используя терл-графику.

2. Построить фрактальный объект дракон Хаттера-Хайтвея, используя терл-графику.

3. Построить кривую Гильберта, используя терл-графику.

4. Построить кривую Госпера, используя терл-графику.

5. Построить кривую Серпинского, используя терл-графику.

6. С помощью операции ветвления построить цветок, используя терл-графику.

7. С помощью операции ветвления построить куст, используя терл-графику.

8. С помощью операции ветвления построить снежинку, используя терл-графику.

9. Построить фрактальный объект со следующими параметрами:

Аксиома: F

Порождающие правила:

$Newf = FF - [XY] + [XY]$

$Newx = +FY$

$Newy = -FX$

$\alpha = \pi / 2, \theta = \pi / 8$

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое терл-графика? Для чего используется? Основной принцип.
2. Какие основные команды используются в терл-графике?
3. Что обозначают команды "открыть ветвь" и "закрыть ветвь"?
4. Какие фракталы можно построить при помощи терл-графики?
5. Что такое аксиома и порождающее правило?

Лабораторная работа №3
СИСТЕМЫ ИТЕРИРОВАННЫХ ФУНКЦИЙ

Цель: изучение основных типов аффинных преобразований на множестве действительных чисел и построение фрактальных объектов на их основе.

Напомним, следуя [6], что в общем случае для построения ***системы итерированных функций*** (**СИФ**) в рассмотрение вводится совокупность сжимающих отображений:

T_1, с коэффициентом сжатия s_1,

T_2, с коэффициентом сжатия s_2,

…….

T_m, с коэффициентом сжатия s_m,

действующих в $\mathbb{R}^n$. Эти m отображений используются для построения одного сжимающего отображения T в пространстве Ω всех непустых компактов из $\mathbb{R}^n$. Преобразование Хатчинсона $T:\Omega\to\Omega$ определяется следующим образом:

$$T(E)=T_1(E)\cup T_2(E)\cup\ldots\cup T_m(E),$$
$$E\in\Omega.$$

Данное преобразование ставит в соответствие «точкам» из Ω, под которыми здесь понимаются компактные множества, также «точки» из Ω.

Системой итерированных функций называется совокупность приведенных выше отображений вместе с итерационной схемой

$$E_1=T(E_0),E_2=T(E_1),\ldots,$$
$$E_n=T(E_{n-1}),\ldots,$$

(E_0 — произвольное компактное множество).

Существование предельного множества E $(E=\lim_{n\to\infty}E_n)$ системы итерированных функций в смысле сходимости в метрике Хаусфдорфа:

$$H(E,F)=\min\{\varepsilon>0:E\subset F+\varepsilon\ \&\ F\subset E+\varepsilon\},$$

где E и F — непустые компактные подмножества в $\mathbb{R}^n$, доказывает соответствующая теорема [6].

Например, СИФ при построении ковра Серпинского (рис. 13) задается тремя аффинными преобразованиями, которые в матричной форме имеют следующий вид:

$$T_1\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$T_2\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1/2 \\ 0 \end{bmatrix},$$

$$T_2\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1/4 \\ \sqrt{3}/4 \end{bmatrix}.$$

Различают два подхода к реализации СИФ: ***детерминированный*** (**ДСИФ**), в котором аффинные преобразования применяются последовательно к каждой точке начальной конфигурации, и ***рандомизированный*** (**РСИФ**), в котором случайно выбираемые аффинные преобразования применяют к единственной начальной точке.

Известный детерминированный алгоритм вычисления СИФ, ориентированный на реализацию в виде компьютерной программы на каком-либо языке программирования, допускающем компиляцию, описан в [6]. К недостаткам данного алгоритма можно отнести следующее.

1. Зависимость качества изображения от размера графического окна (удовлетворительное качество изображения достигается для $m \geq 256$).

2. Привязка алгоритма к размеру графического окна и, как следствие, большой объем вычислений (число операций прямо пропорционально числу точек m^2 и числу итераций).

3. Возникновение аварийных остановов программы с сообщением об ошибке «Индекс вышел за пределы» при попадании точки за пределы окна $m \times m$.

Рассмотрим модификацию алгоритма ДСИФ, позволяющую реализовать его в пакете MATLAB, на примере уже рассмотренного в первом разделе ковра Серпинского. Во-первых, заметим, что, как видно из рис. 13, для получения изображения ковра Серпинского *S1* необходимо на каждую точку $(x_i^{(0)}, y_i^{(0)})$, находящуюся внутри исходного треугольника *S*0, раздельно подействовать каждым из аффинных преобразований *T*1, *T2, T*3:

$$T_1\begin{bmatrix} x_i^{(0)} \\ y_i^{(0)} \end{bmatrix} = \begin{bmatrix} x_{i1}^{(1)} \\ y_{i1}^{(1)} \end{bmatrix}, T_2\begin{bmatrix} x_i^{(0)} \\ y_i^{(0)} \end{bmatrix} = \begin{bmatrix} x_{i2}^{(1)} \\ y_{i2}^{(1)} \end{bmatrix},$$

$$T_3\begin{bmatrix} x_i^{(0)} \\ y_i^{(0)} \end{bmatrix} = \begin{bmatrix} x_{i3}^{(1)} \\ y_{i3}^{(1)} \end{bmatrix}.$$

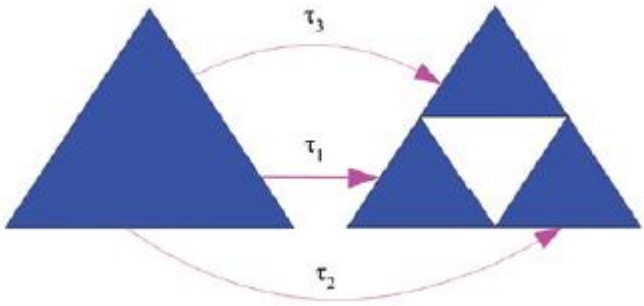

Рис. 13 – Аффинные преобразования для ковра Серпинского

Таким образом, каждая точка $(x_i^{(0)}, y_i^{(0)})$ на первом шаге итерации порождает три новые точки $(x_{i1}^{(0)}, y_{i1}^{(0)}), (x_{i2}^{(0)}, y_{i2}^{(0)}), (x_{i3}^{(0)}, y_{i3}^{(0)})$. На каждую из этих точек на втором шаге итерации вновь следует подействовать аффинными преобразованиями *T*1, *T2, T*3. В результате каждая из трех точек ковра *S*1 вновь породит три точки ковра *S*2 и т. д. Описанный процесс удобно изобразить в виде графа (рис. 14). Из рис. 14 видно, что на втором шаге итерации существует $9 (3^n = 9, где\ n = 2)$ правил, по которым каждой начальной точке $(x_i^{(0)}, y_i^{(0)})$ ставится в соответствие 9 точек ковра Серпинского *S*1. Соответственно, на третьем шаге итерации таких правил будет 27= 3^3:

$T_1T_1T_1\ T_1T_2T_1\ T_1T_3T_1\ T_1T_1T_2$

$T_1T_2T_2\ T_1T_3T_2\ T_1T_1T_3\ T_1T_2T_3\ T_1T_3T_3$

$T_2T_1T_1\ T_2T_2T_1\ T_2T_3T_1\ T_2T_1T_2$

$T_2T_2T_2\ T_2T_3T_2\ T_2T_1T_3\ T_2T_2T_3\ T_2T_3T_3$

$T_3T_1T_1\ T_3T_2T_1\ T_3T_3T_1\ T_3T_1T_2$

$T_3T_2T_2\ T_3T_3T_2\ T_3T_1T_3\ T_3T_2T_3\ T_3T_3T_3$

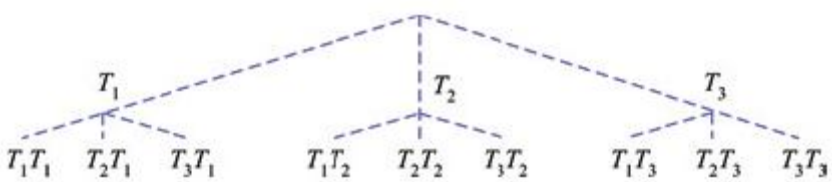

Рис. 14 – Граф аффинных преобразований ковра Серпинского

Таким образом, для построения ковра Серпинского *n*-го уровня с помощью ДСИФ необходимо научиться генерировать 3^n правил, по которым каждой точке $(x_i^{(0)}, y_i^{(0)})$ ставится в соответствие 3^n точек $(x_{ij}^{(0)}, y_{ij}^{(0)}), j = 1,2,...,3^n.$ Введем обозначения: $T_1 \Leftrightarrow 0, T_2 \Leftrightarrow 1, T_3 \Leftrightarrow 2.$ В выбранных обозначениях правила преобразования на третьем шаге итерации имеют вид:

000	010	020	001	011	021	002	012	022
100	110	120	101	111	121	102	112	122
200	210	220	201	211	221	202	212	222

Анализ таблицы закодированных правил преобразований показывает, что названия правил являются множеством натуральных чисел 1, 2, …, 27, записанных в троичной системе счисления. При этом для представления кода каждого правила используется число цифр, совпадающее с порядком ковра *n*. Соответственно, для случая *n*=4 имеем множество, состоящее из $3^4 = 81$ правила, названия которых есть множество чисел 1, 2, …, 81, записанных в троичной системе счисления. При этом для представления каждого числа используются четыре цифры. Очевидно, что для хранения названия правил наиболее удобно использовать массив строковых переменных длиной *n*, число элементов которого равно 3^4.

Таким образом, для построения ковра Серпинского в пакете MATLAB с помощью ДСИФ можно использовать следующий алгоритм.

1. Задать порядок ковра Серпинского *n*.

2. Задать число точек начальной конфигурации *m*.

3. Задать координаты *i* точек $(i = 1,2,...,m)$, заполняющих начальное множество.

4. Перевести каждое из чисел $1,2,...,3^n$ в троичную систему счисления.

5. Сформировать массив, состоящий из 3^n строк, длиной *n* символов.

6. Задать аффинные преобразования.

7. Для *i*-ой точки начальной конфигурации последовательно применить каждое из $j = 1,2,...,3^n$ итерационных правил и отобразить в графическом окне полученные образы каждой начальной точки.

Пример ковра Серпинского, построенного с помощью описанной выше модификации алгоритма ДСИФ, представлен на рис. 15. Ниже приводится листинг файла SerpDSIF.m, содержащего описание соответствующей функции, возвращающей изображение ковра Серпинского.

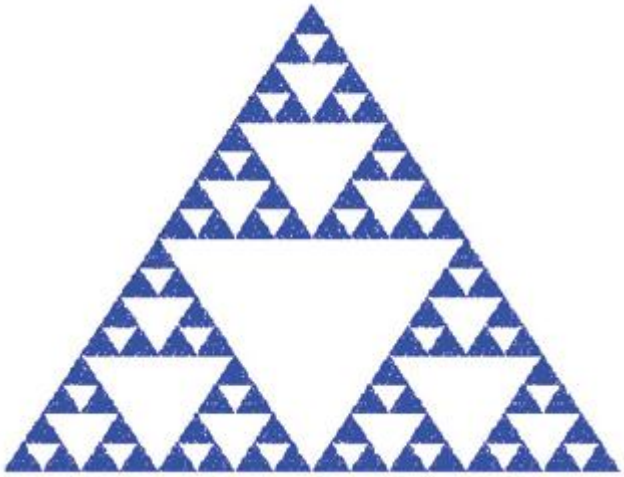

Рис. 15 – Ковер Серпинского 4-го порядка, полученный с помощью модифицированного алгоритма ДСИФ; начальная конфигурация *S0* — 1000 точек, случайным образом размещенных в равностороннем треугольнике

Листинг файла SerpDSIF.m для построения треугольника Серпинского с помощью ДСИФ

```
function z=SerpDSIF(Niter,NPoints)
% функция, возвращающая изображение ковра Серпинского
% Niter - порядок ковра
% NPoints - число точек начальной конфигурации
x=zeros(NPoints,1); y=zeros(NPoints,1);
% задание координат точек начальной конфигурации
x1=0; y1=0; x2=1; y2=0; x3=1/2; y3=sin(pi/3);j=1;
while j<=NPoints
tmpx=rand(1,1);
tmpy=sqrt(3)/2*rand(1,1);
if (-sqrt(3)*tmpx+tmpy<=0)&(sqrt(3)*tmpx+tmpy-sqrt(3)<=0)
x(j)=tmpx; y(j)=tmpy;
j=j+1; end; end
% Формирование массива, содержащего правила итерации
for i=1:3^Niter
Tmp(i)=system3(i!1);
% перевод числа из десятичной в троичную систему счисления
end
n=1; s='0';
```

```
while n<Niter
s=strcat(s,'0'); n=n+1;
end
for i=1:3^Niter
tmp=num2str(Tmp(i)); tmp1=s;
for m=1:length(tmp)
tmp1(Niter!m+1)=tmp(length(tmp)-m+1);
end
Cod(i,1:Niter)=tmp1;
end
a1=[0;0]; a2=[1/2;0]; a3=[1/4;sqrt(3)/4]; A=[1/2,0;0,1/2];
% задание аффинных преобразований
figure(1); hold on; set(gca,'xtick',[],'ytick',[]);
set(gca,'XColor','w','YColor','w'); fill([x1 x2 x3],[y1 y2
y3],'w');
GosperDraw(Niter,NPoints,x,y,A,a1,a2,a3,Cod);
% построение ковра Серпинского
function z=GosperDraw(Niter,NPoints,x,y,A,a1,a2,a3,Cod)
% функция, создающая изображение ковра Серпинского
for m=1:3^Niter
X=x; Y=y; Rule=Cod(m,:);
for i=1:Niter
tmp=Rule(Niter+1-i);
if tmp=='0'
[X Y]=T(NPoints,X,Y,A,a1); % первое аффинное
преобразование
end
if tmp=='1'
[X Y]=T(NPoints,X,Y,A,a2); % второе аффинное
преобразование
end
if tmp=='2'
[X Y]=T(NPoints,X,Y,A,a3); % третье аффинное
преобразование
end
end
plot(X,Y,'.','MarkerSize',1,'MarkerEdgeColor','b');
% отображение результатов итерации
end
function [X,Y]=T(NPoints,x,y,A,a)
% Функция, возвращающая результат аффинного преобразования
X=zeros(NPoints,1); Y=zeros(NPoints,1);
for i=1:NPoints
```

```
R=[x(i);y(i)]; R=A*R+a; X(i)=R(1); Y(i)=R(2);
end
function z=system3(D);
% функция, возвращающая значение целого числа
% в троичной системе счисления
% D - число в десятичной системе счисления
n=1;
while D>=3^n n=n+1; end
if n>1
a=floor(D/3^(n-1))*10^(n-1); b=mod(D,3^(n-1));
if b>=3
b=system3(b); % рекурсия
end
z=a+b;
else z=D;
end
```

С использованием ДСИФ можно построить такие фракталы как «лист» и «ветка папоротника». Листинг соответствующих программ представлен ниже.

Листинг файла для построения фрактала «Лист» с помощью ДСИФ

```
function z=Maple(Niter,NPoints)
x1=0; y1=0;
x2=1; y2=0;
x3=1/2; y3=sin(pi/3);
x=zeros(NPoints,1);
y=zeros(NPoints,1);
j=1;
while j<=NPoints
 tmpx=rand(1,1);
 tmpy=sqrt(3)/2*rand(1,1);
 if(-sqrt(3)*tmpx+tmpy<=0)&&(sqrt(3)*tmpx+tmpy-sqrt(3)<=0)
        x(j)=tmpx;
        y(j)=tmpy;
        j=j+1;
    end
end
for i=1:2^Niter
    Tmp(i)=system2(i-1,2);
end
n=1; s='0';
while n<Niter s=strcat(s,'0'); n=n+1; end
```

```
for i=1:2^Niter
    tmp=num2str(Tmp(i)); tmp1=s;
    for m=1:length(tmp)
        tmp1(Niter-m+1)=tmp(length(tmp)-m+1);
    end
    Cod(i,1:Niter)=tmp1;
end
A1=[0.4,-0.3733;0.0600,0.6000];
A2=[-0.8000,-0.1867;0.1371,0.8000];
a1=[0.3533;0.000];
a2=[1.1000;0.1000];
figure(1); hold on;
set(gca,'xtick',[],'ytick',[]);
set(gca,'XColor','w','YColor','w');
FractalDraw(Niter,NPoints,x,y,A1,A2,a1,a2,Cod);

function z=FractalDraw(Niter,NPoints,x,y,A1,A2,a1,a2,Cod)
for m=1:2^Niter
    X=x; Y=y;
    Rule=Cod(m,:);
    for i=1:Niter
        tmp=Rule(Niter+1-i);
        if tmp=='0' [X Y]=T(NPoints,X,Y,A1,a1); end
        if tmp=='1' [X Y]=T(NPoints,X,Y,A2,a2); end
    end
    plot(X,Y,'.','MarkerSize',1,'MarkerEdgeColor','b');
end

function [X,Y]=T(NPoints,x,y,A,a)
X=zeros(NPoints,1);
Y=zeros(NPoints,1);
for i=1:NPoints
    R=[x(i);y(i)];
    R=A*R+a;
    X(i)=R(1);
    Y(i)=R(2);
end

function z=system2(D,m)
n=1;
while D>=m^n
    n=n+1;
end
```

```
if n>1
    a=floor(D/m^(n-1))*10^(n-1);
    b=mod(D,m^(n-1));
    if b>=m
        b=system2(b,m);
    end
    z=a+b;
else z=D;
end
```

Листинг файла для построения фрактала «Ветка папоротника» с помощью ДСИФ

```
function z=Paporotnic(Niter,NPoints)
x1=0; y1=0; x2=1; y2=0; x3=1/2; y3=sin(pi/3);
x=zeros(NPoints,1); y=zeros(NPoints,1);
j=1;
while j<=NPoints
tmpx=rand(1,1); tmpy=sqrt(3)/2*rand(1,1);
if (-sqrt(3)*tmpx+tmpy<=0)&&(sqrt(3)*tmpx+tmpy-sqrt(3)<=0)
        x(j)=tmpx; y(j)=tmpy; j=j+1;
    end
end
for i=1:4^Niter Tmp(i)=system2(i-1,4); end
n=1; s='0';
while n<Niter s=strcat(s,'0'); n=n+1; end
for i=1:4^Niter
    tmp=num2str(Tmp(i)); tmp1=s;
    for m=1:length(tmp)
        tmp1(Niter-m+1)=tmp(length(tmp)-m+1);
    end
    Cod(i,1:Niter)=tmp1;
end
A1=[0.7000,0;0,0.7000];
A2=[0.1000,-0.4330;0.1732,0.2500];
A3=[0.1000,0.4330;-0.1732,0.2500];
A4=[0,0;0,0.3000];
a1=[0.1496;0.2962]; a2=[0.4478;0.0014];
a3=[0.4450;0.1559]; a4=[0.4987;0.0070];
figure(1); hold on;
set(gca,'xtick',[],'ytick',[]);
set(gca,'XColor','w','YColor','w');
```

```
FractalDraw(Niter,NPoints,x,y,A1,A2,A3,A4,a1,a2,a3,a4,Cod)
;
function
z=Simplex(Niter,NPoints,x,y,A1,A2,A3,A4,a1,a2,a3,a4,Cod)
for m=1:4^Niter
    X=x; Y=y; Rule=Cod(m,:);
    for i=1:Niter
        tmp=Rule(Niter+1-i);
        if tmp=='0' [X Y]=T(NPoints,X,Y,A1,a1); end
        if tmp=='1' [X Y]=T(NPoints,X,Y,A2,a2); end
        if tmp=='2' [X Y]=T(NPoints,X,Y,A3,a3); end
        if tmp=='3' [X Y]=T(NPoints,X,Y,A4,a4); end
    end
    plot(X,Y,'.','MarkerSize',1,'MarkerEdgeColor','k')
end
function [X,Y]=T(NPoints,x,y,A,a)
X=zeros(NPoints,1); Y=zeros(NPoints,1);
for i=1:NPoints
    R=[x(i);y(i)]; R=A*R+a;
    X(i)=R(1); Y(i)=R(2);
end

function
z=FractalDraw(Niter,NPoints,x,y,A1,A2,A3,A4,a1,a2,a3,a4,Co
d)
for m=1:2^ Niter
    X=x; Y=y; Rule=Cod(m,:);
    for i=1:Niter tmp=Rule(Niter+1-i);
        if tmp=='0'
            [X Y]=T(NPoints,X,Y,A1,a1); end
        if tmp=='1'
            [X Y]=T(NPoints,X,Y,A2,a2); end
        if tmp=='2'
            [X Y]=T(NPoints,X,Y,A3,a3); end
        if tmp=='3'
            [X Y]=T(NPoints,X,Y,A4,a4); end
    end
    plot(X,Y,'.','MarkerSize',1,'MarkerEdgeColor','k');
end

function [X,Y]=T(NPoints,x,y,A,a)
X=zeros(NPoints,1); Y=zeros(NPoints,1);
for i=1:NPoints
```

```
    R=[x(i);y(i)]; R=A*R+a;
    X(i)=R(1); Y(i)=R(2);
end

function z=system2(D,m)
n=1;
while D>=m^n n=n+1; end
if n>1
    a=floor(D/m^(n-1))*10^(n-1);
    b=mod(D,m^(n-1));
    if b>=m
        b=system2(b,m);
    end
    z=a+b;
else z=D;
end
```

В отличие от ДСИФ в рандомизированном алгоритме начальное множество $S0$ состоит из одной точки $(x0, y0)$, а правило, по которому точке ставится в соответствие точка (xi, yi), где i — номер правила, выбирается случайным образом из набора, содержащего все возможные правила аффинных преобразований. Например, применительно к ковру Серпинского это означает, что при построении ковра 2-го порядка преобразование должно случайным образом выбираться из следующего множества преобразований:

$$\{T_1, T_2, T_3, T_1T_1, T_1T_2, T_1T_3, T_2T_3, T_3T_1, T_3T_2, T_3T_3\},$$

число элементов N которого равно $N = \sum_{k=1}^{n} m^k$.

Таким образом, для построения ковра Серпинского в пакете MATLAB с помощью РСИФ можно использовать следующий алгоритм.

1. Задать порядок ковра Серпинского n.

2. Задать число испытаний *NTrial*.

3. Задать число аффинных преобразований $m = 3$.

4. Сформировать массив, содержащий набор правил для аффинных преобразований.

5. Задать координаты начальной точки $(x0, y0)$.

6. Перевести каждое из чисел $1, 2, ..., 3^n$ в троичную систему счисления.

7. Задать аффинные преобразования.

8. Для заданного числа испытаний последовательно, начиная с начальной точки, в соответствии с правилами аффинных преобразований, выбираемых случайным образом, вычислить точки итерационной последовательности.

9. Отобразить вычисленное множество точек в графическом окне.

В общем случае для фракталов *n*-го порядка, изображение которых создается с помощью *m* аффинных преобразований, как и при использовании ДСИФ, необходимо переводить числа $1,2,...,m^n$ в *m*-ичную систему счисления.

Пример кристалла, построенного с помощью описанного выше алгоритма РСИФ, представлен на рис. l6. Ниже приводится листинг файла Crystal.m, содержащего описание соответствующей функции.

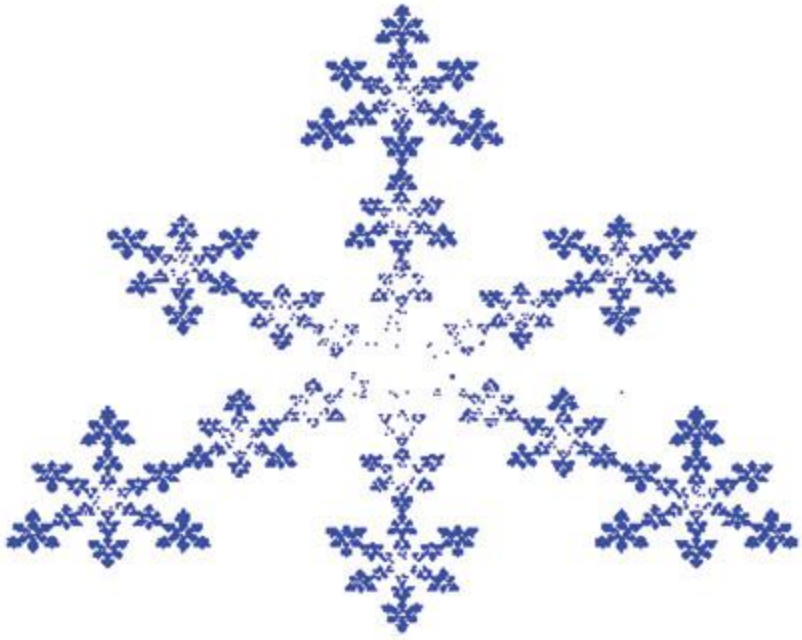

Рис. 16 – Кристалл 4-го порядка, полученный с помощью алгоритма РСИФ после 500000 испытаний

Листинг файла Crystal.m

```
function z=Crystal(Niter,NTrial)
% функция, возвращающая изображение кристалла
% Niter - порядок кристалла
% NTrial - число испытаний
Na=4; % число аффинных преобразований
% Создание массива, содержащего набор
% правил для аффинных преобразований
k=1;
for m=1:Niter
for i=1:4^m Tmp(k)=system3(i-1,Na); k=k+1;
```

```
end
Q(1)=Na;
for m=2:Niter Q(m)=Q(m-1)+Na^m; end
n=1; s='0'; M=1;
while n<=length(Tmp)
m=1;
while n>Q(m) m=m+1; end
if m==1
S(n,1:1)=s;
else
S(n,1:1)=s;
for i=2:m S(n,1:i)=strcat(S(n,:),s); end
end
n=n+1;
end
for i=1:k-1 tmp=num2str(Tmp(i)); m=1;
while i>Q(m) m=m+1; end
tmp1(1:m)=S(i,1:m);
for m=1:length(tmp)
tmp1(length(tmp1)-...
m+1:length(tmp1)-m+1)=...
tmp(length(tmp)-m+1:length(tmp)-m+1);
end
Cod(i,1:length(tmp1))=tmp1;
end
x=0; y=0; % координаты начальной точки
% задание аффинных преобразований
A1=[0.2550,0.0000;0.0000,0.2550];
A2=[0.2550,0.0000;0.0000,0.2550];
A3=[0.2550,0.0000;0.0000,0.2550];
4=[0.3700,-0.6420;0.6420,0.3700];
a1=[0.3726;0.6714]; a2=[0.1146;0.2232];
a3=[0.6306;0.2232]; a4=[0.6356;-0.0061];
figure(1); hold on;
set(gca,'xtick',[],'ytick',[]);
set(gca,'XColor','w','YColor','w');
DrawFractal(Niter,NTrial,x,y,A1,A2,A3,A4,...
a1,a2,a3,a4,Cod); % визуализация фрактала
function z=DrawFractal(Niter,NTrial,x,y,...
A1,A2,A3,A4,a1,a2,a3,a4,Cod)
% функция, возвращающая изображение фрактала
X1=zeros(NTrial,1); Y1=zeros(NTrial,1);
X=x; Y=y;
```

```
for m=1:NTrial
Np=1+round((size(Cod,1)-1)*rand(1,1));
% выбор номера преобразования
Rule=Cod(Np,:);
for i=1:length(Rule)
tmp=Rule(length(Rule)+1-i);
if tmp=='0' [X Y]=T(X,Y,A1,a1); end
if tmp=='1' [X Y]=T(X,Y,A2,a2); end
if tmp=='2' [X Y]=T(X,Y,A3,a3); end
if tmp=='3' [X Y]=T(X,Y,A4,a4); end
end
X1(m)=X; Y1(m)=Y;
end
plot(X1,Y1,'.','MarkerSize',1,...
'MarkerEdgeColor','b');
function [X,Y]=T(x,y,A,a)
% Функция, возвращающая результат
% аффинного преобразования
R=[x;y]; R=A*R+a;
X=R(1); Y=R(2);
function z=system3(D,m);
% функция, возвращающая значение
% числа в четверичной системе координат
n=1;
while D>=m^n n=n+1; end
if n>1
a=floor(D/m^(n-1))*10^(n-1);
b=mod(D,m^(n!1));
if b>=m b=system3(b,m); end
z=a+b;
else
z=D;
end
```

ЗАДАНИЕ

1. Построить ковер Серпинского, используя алгоритм ДСИФ.
2. Построить фрактальный объект «лист», используя алгоритм ДСИФ.
3. Построить фрактальный объект «ветка папоротника», используя алгоритм ДСИФ.

4. Построить кристалл с помощью алгоритма РСИФ, используя 4 афинных преобразования.
5. Построить кристалл с помощью алгоритма РСИФ, используя 5 афинных преобразования.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое аффинные преобразования?
2. Что представляет собой понятие СИФ?
3. В чем разница РСИФ и ДСИФ?
4. Привести примеры фракталов, которые можно построить при помощи ДСИФ и РСИФ.

Лабораторная работа №4
ФРАКТАЛЬНЫЕ КЛАСТЕРЫ

Цель: построение фрактальных кластеров с помощью модели ограниченной диффузией агрегации.

Под *массовыми фракталами* понимают моделируемые и реальные объекты, состоящие из дискретного набора частиц-мономеров, агрегированных в большие кластеры с фрактальной геометрией. Часто при исследовании таких объектов прибегают к анализу пространственного распределения плотности вещества, или, другими словами, распределению массы объекта. Отсюда и название – массовые фракталы.

Модель *ограниченной диффузией агрегации* (ОДА) является наиболее популярным подходом для описания образования массовых фракталов. Эта модель может быть легко реализована в виде компьютерного алгоритма и прекрасно подходит для численного исследования фрактальной геометрии таких объектов.

Существует множество вариантов модели ОДА, различающихся геометрией задачи, особенностями взаимодействия частиц и кластеров и их перемещения в пространстве. В рамках данного пособия мы ограничимся рассмотрением самого простого классического варианта ОДА – модели *частица-кластер*. В этой модели растущий кластер всегда неподвижен в пространстве, а присоединяющиеся к нему частицы-мономеры передвигаются в пространстве по случайным траекториям независимо друг от друга до момента соприкосновения с кластером, после чего они становятся частью кластера.

Рассмотрим эту задачу более подробно. В качестве зародыша будущего кластера выступает частица-мономер, к которой в последующем будут присоединяться другие частицы. Частицы запускаются по одной с произвольной случайно выбранной точки на окружности (в 2D задаче) или сфере (в 3D задаче) заданного радиуса R_0, построенной вокруг центрального зародыша. Далее пошагово моделируется случайное броуновское движение частицы в пространстве – на каждом шагу частица

случайным образом перемещается в один из ближайших соседних узлов сетки, на которой происходит моделирование. В случае, если происходит касание частицей кластера, она становится частью кластера и осуществляется запуск новой частицы. Если в процессе своего движения частица удаляется от центра кластера на расстояние, превышающее $3R_0$, то эта частица исключается из рассмотрения и запускается следующая. Обычно задача останавливается, когда кластер достигает либо определенной заданной массы (общего количества частиц, составляющих кластер), либо когда максимально удаленная точка кластера выходит за некоторый заданный радиус $R < R_0$.

Результат такого моделирования на 2D квадратной решетке представлен на Рис. 17(а). Образовавшийся кластер имеет довольно сложную дендритную структуру и характеризуется фрактальной размерностью $D \approx 1,7$. Необходимо отметить, что фрактальная размерность для данной задачи зависит от типа выбранной решетки. Чтобы избежать этой проблемы, моделирование предпочтительнее проводить "вне решетки", позволяя частицам перемещаться в произвольных направлениях и присоединяться к кластеру в произвольных местах. Такое моделирование, конечно же, является более сложной задачей, но получающиеся кластеры оказываются более реалистичными. Геометрию задачи можно изменить, если рост кластеров производить не из центральной точки, а с некоторой линии (в 2D задаче) или плоскости (в 3D задаче), которые являются дном некоторого сосуда. Частицы при этом запускаются из верхней части сосуда и в процессе своего хаотичного движения отражаются от боковых стенок и крышки сосуда, но оседают на его дне и растущих кластерах. В результате происходит рост "деревьев" на дне сосуда, как показано на Рис. 17(б).

Задачу можно усложнить и сделать более интересной, если ввести вероятность p прилипания частицы к кластеру. При этом каждый раз при соприкосновении свободной частицы с кластером производится проверка условия прилипания – величина p сравнивается со случайным числом $r \in [0,1]$. Если $p \geq r$, то частица присоединяется к кластеру и запускается новая, а иначе частица продолжает свое блуждание. Понятно, что чем

меньше вероятность прилипания, тем больше у частицы шансов проникнуть вглубь кластера между его ветвями, тем более компактным оказывается его структура и тем больше его фрактальная размерность. В пределе, размерность кластера оказывается близка к евклидовой размерности: $D \rightarrow 2$ для 2D задачи и $D \rightarrow 3$ для 3Dзадачи.

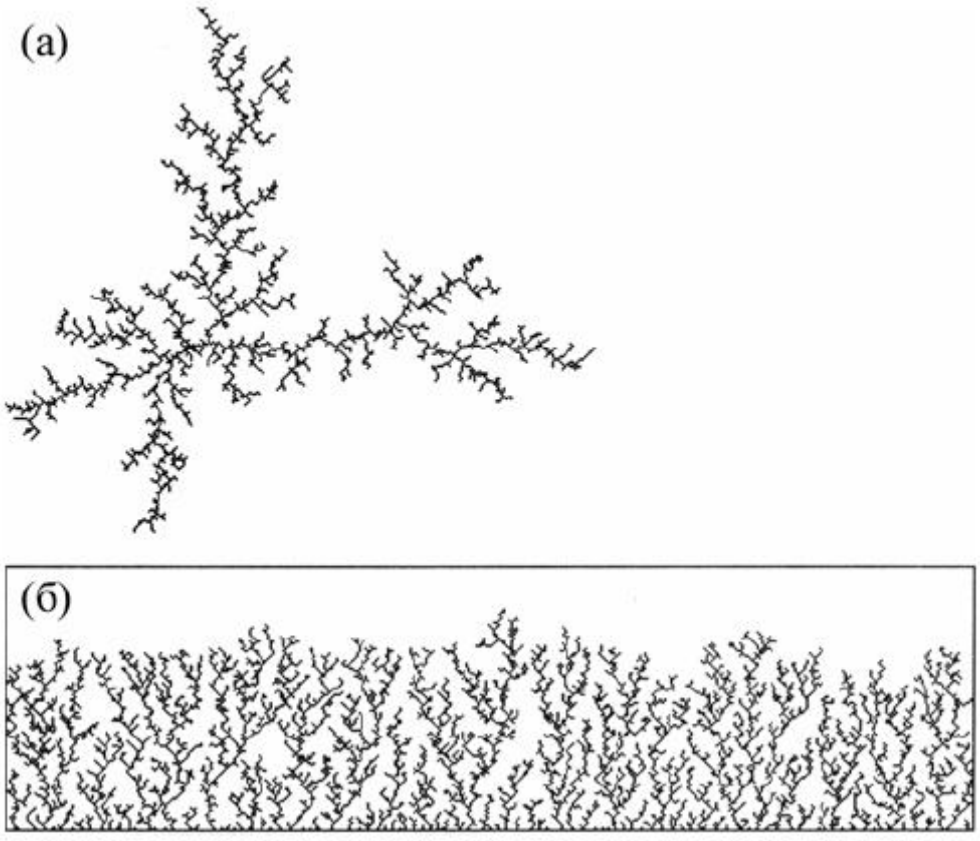

Рис. 17. Моделирование роста фрактальных кластеров с помощью алгоритма ОДА на 2D квадратной решетке: (а) рост из центральной точки, (б) рост от линии

Листинг dla.m для моделирования роста фрактальных кластеров из центральной точки

```
function z = DLA003(filename, dim , N)
% Diffusion Limited Aggregation (DLA) version 0.03
%filename - a string for where to save the file Ex:
'pic10.jpg'
%dim - create a (dim x dim) space
%N - number of particles
%Introduce points at a radius a little bigger than
structure ( moderate speedup )
```

```
%Reseed points if they wander out of radius ( huge
speedup! )
% TODO:  Implement stickiness
started=datestr(now);
step = 3;
randtheta = rand*2*pi;
middle = round(dim/2);
pic = 127*ones(dim,dim);
pic(middle,middle) = 0;
c = 0;
dist = 0;
min_radius = 5;
reseed=0;
num_reseeds=0;
stat_total_sum = 0;
stat_total_avg = 0;
% Let's simulate a drunken sailor
for n=1:N,
% TODO: make this an adjustable parameter
% this makes the initial radius %1 of dim, and linearly
increases it until r=dim/2 at n=N
% TODO: if dim is >=1000, than the initial radius should
be less than %1
radius = round(dim*(1 + 49*(n-1)/N)/100);
  if( radius < min_radius )
    radius = min_radius;
  end
  randtheta = rand*2*pi;
  randx = round( middle +  (radius-2) * cos(randtheta) );
  randy = round( middle +  (radius-2) * sin(randtheta) );
  dist = sqrt((randx-middle)^2 + (randy-middle)^2);
  disp( sprintf('%d: init=(%d,%d) dist=%f radius=%d',
n,randx,randy,dist,radius) );
  % repeat until we hit something
  % if outside radius after 20 steps, RESEED
  % TODO: make # of steps adjustable, or find a better
default
  while ( 1 > 0 ),
    c=c+1;
    newx = randx + (-1)^round(4*rand) * mod( round(
rand*(step*2) ), step ) ;
    newy = randy + (-1)^round(4*rand) * mod( round(
rand*(step*2) ), step ) ;
```

```
    if( mod(c,20) == 0 )
      if( abs( newx - middle ) > radius |  abs( newy -
middle) > radius )
         disp('RESEED');
         reseed=1;
         num_reseeds=num_reseeds+1;
         n=n-1;
         break;
      end
    end
    if( newx > 1 & newx < (dim-1) )
      randx = newx;
    end
    if( newy > 1 & newy < (dim-1) )
      randy = newy;
    end
% Did we walk into something?
    found = 0;
    for j=-1:1,
      for k=-1:1,
        if( randx+j < 1 | randy+k < 1)
          break;
        end
        p = pic(randx+j,randy+k);
% Hit something!
  if ( p == 0 )
  disp(sprintf('%d: hit at (%d,%d)', n,randx+j,randy+k));
          found = 1;
          break;
        end
      end
      if ( found )
        break;
      end
    end
    if( found )
      break;
    end
  end
  % we have hit something if reseed=0
  if( reseed == 1 )
    % reset seed state
    reseed=0;
```

```
  else
    % count total number of iterations
    stat_total_sum = stat_total_sum + c;
    disp(sprintf('%d: took %d attempts to hit',n,c));
    c=0;
    % aggregate
    pic(randx,randy) = 0;
  end
end
finished=datestr(now);
imwrite(pic,filename);
image(pic);
colormap gray;
disp(sprintf('Started  %s', started) );
disp(sprintf('Finished %s', finished) );
disp(sprintf('Avg   # attempts: %d',
round(stat_total_sum/N )) );
disp(sprintf('# of Reseeds    : %d', num_reseeds ) );
```

ЗАДАНИЕ

1. Построить фрактальный кластер с помощью модели ограниченной диффузией агрегации. Рост кластера из центральной точки.

2. Построить фрактальный кластер с помощью модели ограниченной диффузией агрегации. Рост кластера от линии.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое массовый фрактал?
2. Как происходит построение фрактального кластера из центральной точки?
3. Когда частица-мономер становится частью кластера?
4. Когда завершается построение кластера?
5. Что происходит с частицей-мономером в случае несоприкосновения с кластером?
6. Чем отличаются алгоритмы построения фрактального кластера из центральной точки и с линии?

Лабораторная работа №5
ФРАКТАЛЬНЫЕ ВРЕМЕННЫЕ РЯДЫ

Цель: изучение основ построения фрактальных кривых.

Фрактальные временные ряды – целый класс фрактальных кривых, широко используемых при описании и моделировании разнообразнейших явлений в различных областях знаний. С помощью этого подхода описываются такие, казалось бы, не имеющие ничего общего, явления, как движение броуновской частицы, поведение курса обмена валют на финансовых рынках, изменение уровня воды в озерах и реках, и т. д. Необходимо отметить, что рассматриваемые в этом подходе фрактальные кривые не обязательно должны быть зависимостью некоторой величины от времени. Однако важно, что получаемые в результате моделирования кривые являются самоаффинными, а не самоподобными. Кроме того, эти кривые являются однозначными (каждому значению аргумента соответствует одно единственное значение функции).

Алгоритм смещения средней точки (СТТ) является одним из самых популярных методов генерации фрактальных временных рядов, поскольку является наиболее приспособленным и легко реализуемым в качестве компьютерного алгоритма. Для построения берется отрезок единичной длины на оси абсцисс *X*. Этот отрезок разбивается на $N = 2^m$ частей, где m – любое положительное целое число. В результате на отрезке [0,1] имеем N+1 точку, включая концы отрезка. Эти точки x_i будем нумеровать индексами i от 0 до N.

Результатом работы алгоритма будет массив y_i, задающий значения моделируемой функции в точках x_i. Сначала необходимо задать значения функции в граничных точках $x_0 = 0$ и $x_N = 1$. Эти значения y_0 и y_N задаются произвольно из условий конкретной задачи. Далее следует итеративный процесс, в ходе которого исходный единичный отрезок постепенного разбивается на более мелкие части посредством многократного деления пополам. При этом каждый раз вычисляется значение функции y_c в середине нового рабочего отрезка x_c исходя из значений y_{left} и y_{right} в граничных точках этого отрезка x_{left} и x_{right}. Для этого используется следующее выражение:

$$y_c = \left(y_{left} + y_{right} \right) / 2 + h \qquad (2)$$

где h – случайная величина, распределенная по нормальному гауссову закону с нулевым средним и дисперсией σ_h, которая

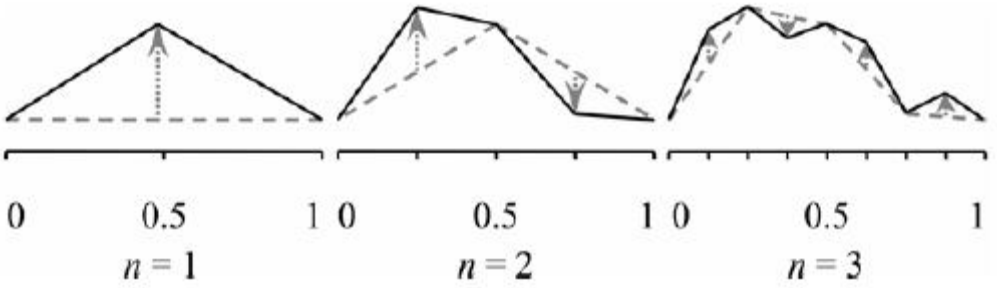

Рис. 18 – Работа метода CCT на первых итерациях

определяется следующим соотношением:

$$\sigma_h = r^H \tag{3}$$

где $r = \left(x_{right} - x_{left} \right) / 2$ – расстояние от средней точки рабочего отрезка x_c, где вычисляется новое значение функции, до концов этого отрезка; H – показатель Херста, который непосредственно связан с фрактальной размерностью моделируемой кривой: $D = 2 - H$. Поскольку величина D для фрактальной кривой должна лежать в диапазоне от 1 до 2, то показатель Херста должен удовлетворять следующему условию:

$$0 \le H \le 1 \tag{4}$$

На рис. 18 представлена работа алгоритма CCT для первых трех итераций. Для первого шага в качестве рабочего отрезка выступает весь исходный единичный отрезок, а значение функции $y_{N/2}$ вычисляется в центре этого отрезка в точке $x_c = x_{N/2} = 1/2$. В этом случае $x_{left} = x_0 = 0$, $x_{right} = x_N = 1$, $r = 1/2$. В результате после выполнения первого шага мы имеем два отрезка с длиной $1/2$, у каждого из которых известны значения функции на концах. Это позволяет применить выражение (2) для нахождения значений функции $y_{N/4}$ и $y_{3N/4}$ в серединах двух новых рабочих отрезков – точках $y_{N/4} = 1/4$ и $y_{3N/4} = 3/4$ соответственно. И так далее, до тех пор, пока значения функции не будут определены во всех $N+1$ точках.

Выражение (2) раскрывает смысл работы алгоритма CCT. Фактически, каждый раз вычисляя значение функции в середине отрезка, мы берем среднее значений функции в концах этого отрезка (что соответствует линейной интерполяции) и сдвигаем полученное значение на некоторую случайную

величину h. Необходимо отметить, что согласно выражению (3), одновременно с уменьшением длины рабочего отрезка, происходит и уменьшение дисперсии этой величины, то есть случайные отклонения становятся все меньше и меньше. Это справедливо всегда за исключением случая, когда величина H близка к 0. В этом случае дисперсия практически не меняется, а получаемая фрактальная кривая оказывается очень сложной и изрезанной с фрактальной размерностью D, близкой к 2.

На рис. 19 для сравнения показаны фрактальные кривые, полученные методом ССТ, для трех разных значений D. При этом для удобства сравнения во всех трех случаях при построении кривой генератор псевдослучайных чисел инициализировался одной и той же величиной, что позволило лучше понять характер изменения кривой при изменении D. Общая тенденция ясна – чем больше D (меньше H), тем кривая более сложная, и наоборот, чем меньше D (больше H), тем кривая более гладкая.

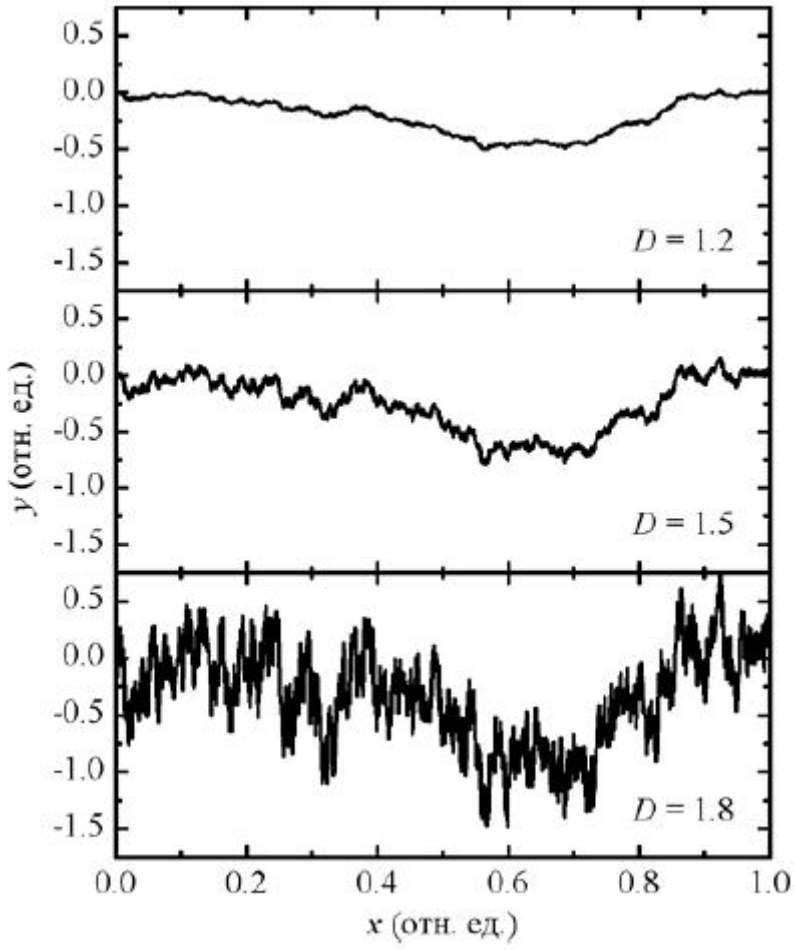

Рис. 19 – Фрактальные кривые, полученные методом ССТ для трех разных значений D

ЗАДАНИЕ

Написать программу, реализующую алгоритм ССТ. Построить фрактальную кривую с помощью разработанной программы с количеством итераций 5.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1.Что такое фрактальные временные ряды?

2. Что позволяют описывать фрактальные временные ряды?

3. Алгоритм смещения средней точки - в чем суть и для чего используется?

4. Что обозначает переменная h в алгоритма ССТ - чему она равна и какое влияние она оказывает на характер кривой?

Лабораторная работа №6
ВЫЧИСЛЕНИЕ ФРАКТАЛЬНОЙ РАЗМЕРНОСТИ

Цель: изучение основных алгоритмов вычисления фрактальной размерности

При описании свойств фрактала важную роль играет такая его характеристика как фрактальная размерность. Дадим общее определение этой величины. Пусть d – обычная Евклидова размерность пространства, в котором находится наш фрактальный объект ($d=1$ – линия, $d=2$ – плоскость, $d=3$ – обычное трехмерное пространство). Покроем теперь этот объект целиком d-мерными "шарами" радиуса r. Предположим, что нам потребовалось для этого не менее**,** чем $N(r)$ шаров. Тогда, если при достаточно малых r величина $N(r)$ меняется с по степенному закону

$$N(r) \sim \frac{1}{r^D}$$

то – D называется размерностью *Хаусдорфа-Безиковича* или *фрактальной размерностью* этого объекта.

Используя понятие фрактальной размерности можно дать более строгое, чем приведенное во введении, определение фрактала. Согласно этому определению фрактал представляет собой объект, размерность *Хаусдорфа-Безиковича* которого больше его топологической размерности (0 – для россыпи точек, 1 – для кривой, 2 – для поверхности и т.д.).

Формулу, представленную выше, можно переписать также в виде

$$D = -\lim_{r \to 0} \frac{\ln N(r)}{\ln r}$$

Это и служит общим определением фрактальной размерности. В соответствии с ним величина является локальной характеристикой данного объекта. Совершенно ясно, что мы получили бы при оценке фрактальной размерности тот же самый результат, если бы использовали процедуру покрытия фрактала кубами (квадратами, если фрактальный объект располагается на плоскости).

Наиболее просто реализовать программно можно алгоритм вычисления фрактальной размерности называемый box-counting. Его идея заключается в том, что евклидово пространство, содержащее изображение объекта, разделяют сеткой с ячейкой размера r и подсчитываются непустые, занятые исследуемым

объектом, квадраты *N(r)*. Затем размер *r* уменьшают, и снова подсчитывают число непустых полей *N(r)*. Наклон графика в логарифмическом масштабе *N(r)* от *1/r* соответствует величине размерности.

Программная реализация алгоритма для бинарных изображений приведена в листинге boxcount.m

Листинг boxcount.m

```
function D = boxcount(c)
warning off
c = logical(squeeze(c));
warning on
dim = ndims(c); % dim is 2 for a vector or a matrix, 3 for
a cube
if dim ~= 2
  error('dimension is not 2');
end
% transpose the vector to a 1-by-n vector
if length(c)==numel(c)
  dim=1;
  if size(c,1)~=1
    c = c';
  end
end
width = max(size(c));    % largest size of the box
p = log(width)/log(2);   % number of generations
% remap the array if the sizes are not all equal,
% or if they are not power of two
% (this slows down the computation!)
if p~=round(p) || any(size(c)~=width)
  p = ceil(p);
  width = 2^p;
  mz = zeros(width, width);
  mz(1:size(c,1), 1:size(c,2)) = c;
  c = mz;
end
n=zeros(1,p+1); % pre-allocate the number of box of size r
n(p+1) = sum(c(:));
for g=(p-1):-1:0
  siz = 2^(p-g);
  siz2 = round(siz/2);
    for i=1:siz:(width-siz+1)
      for j=1:siz:(width-siz+1)
```

```
          c(i,j) = ( c(i,j) || c(i+siz2,j) || c(i,j+siz2)
|| c(i+siz2,j+siz2) );
      end
    end
    n(g+1) = sum(sum(c(1:siz:(width-siz+1),1:siz:(width-
siz+1))));
end
n = n(end:-1:1);
r = 2.^(0:p); % box size (1, 2, 4, 8...)
loglog(r,n,'s-');
xlabel('r, box size'); ylabel('n(r), number of boxes');
title([num2str(dim) 'D box-count']);
koef = polyfit(-log(r), log(n), 1);
D = koef(1);
```

ЗАДАНИЕ

1. Реализовать функцию, которая будет вычислять фрактальную размерность бинарных изображений с помощью алгоритма Box Counting. Продемонстрировать работу данной функции. В каком диапазоне будут значения фрактальной размерности бинарных изображений?

2. Реализовать функцию, которая будет вычислять фрактальную размерность яркостных изображений с помощью алгоритма Box Counting. Продемонстрировать работу данной функции. В каком диапазоне будут значения фрактальной размерности яркостных изображений?

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое фрактальная размерность?
2. Алгоритм Box Counting - суть и для чего необходим?
3. Какие диапазоны значений может принимать фрактальная размерность для бинарных и для цветных изображений?
4. В чем разница в алгоритме Box Counting для бинарных и цветных изображений?

ЛИТЕРАТУРА

1. Данилов Ю. А., Кадомцев Б. Б. Что такое синергетика?//Нелинейные волны. Самоорганизация — М., Наука, 1983.

2. Каток А. Б., Хасселблат Б. Введение в современную теорию динамических систем.— М.: Факториал, 1999.

3. Лихтенберг А., Либерман М. Регулярная и стохастическая динамика.— М.: Меркурий_ПРЕСС, 2000.

4. Шустер Г. Детерминированный хаос.— М.: Мир, 1988.

5. Кренкель Э. Т. Сжатие сигналов с применением теории фракталов.— М.: МТУСИ, 1996.

6. Кроновер Р. М. Фракталы и хаос в динамических системах. Основы теории.— М.: Постмаркер, 2000.