ЛАБОРАТОРНА РОБОТА №4

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 I^n . Эти m отображений используются для построения одного сжимающего отображения T в пространстве Ω всех непустых компактов из I^n .

Преобразование Хатчинсона $T : \Omega \to \Omega$ определяется следующим образом:

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

$$E \in \Omega.$$

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

$$E_1 = T(E_0), E_2 = T(E_1), ...,$$

$$E_n = T(E_{n-1}), ...,$$

(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 I^n, доказывает соответствующая теорема [6].

Например, СИФ при построении ковра Серпинского (рис. 1) 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:

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

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

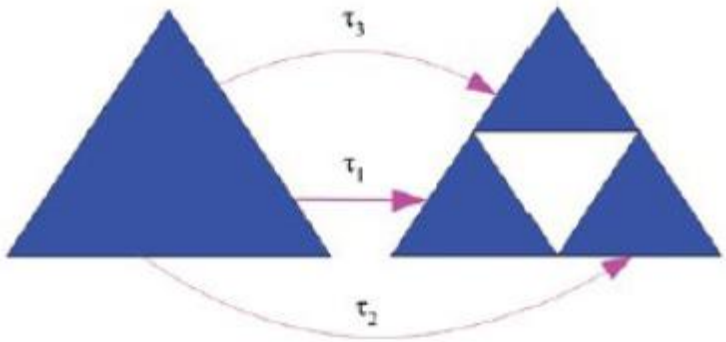

Рис. 13 – Аффинные преобразования для ковра Серпинского

Таким образом, каждая точка ($x_i^{(0)}, y_i^{(0)}$) на первом шаге итерации порождает три новые точки $\left(x_{i1}^{(0)}, y_{i1}^{(0)} \right), \left(x_{i2}^{(0)}, y_{i2}^{(0)} \right), (x_{i3}^{(0)}, y_{i3}^{(0)})$.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:

$T_1T_1T_1\ T_1T_2T_1\ T_1T_3T_1\ T_1T_1T_2$

$T_1T_2T_2\ T_1T_3T_2\ T_1T_1T_3\ T_1T_2T_3\ T_1T_3T_3$

$T_2T_1T_1\ T_2T_2T_1\ T_2T_3T_1\ T_2T_1T_2$

$T_2T_2T_2\ T_2T_3T_2\ T_2T_1T_3\ T_2T_2T_3\ T_2T_3T_3$

$T_3T_1T_1\ T_3T_2T_1\ T_3T_3T_1\ T_3T_1T_2$

$T_3T_2T_2\ T_3T_3T_2\ T_3T_1T_3\ T_3T_2T_3\ T_3T_3T_3$

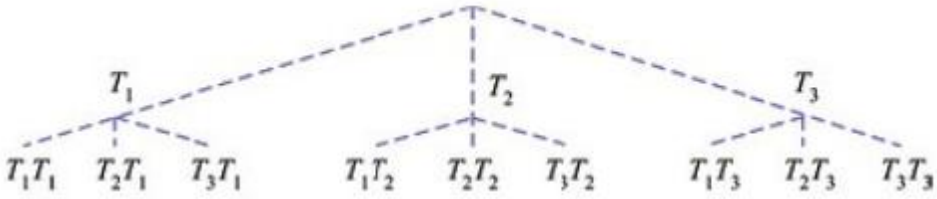

Рис. 14 – Граф аффинных преобразований ковра Серпинского

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 $\left(x_{ij}^{(0)}, y_{ij}^{(0)} \right)$, 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.

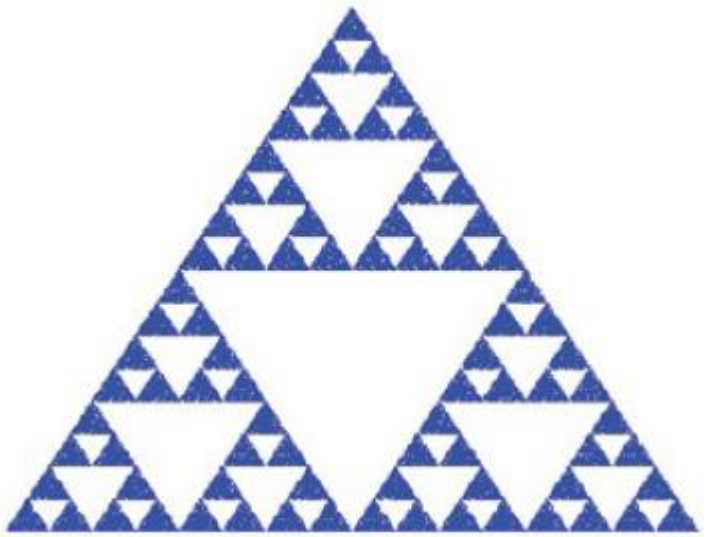

Рис. 15 – Ковер Серпинского 4-го порядка, полученный с помощью модифицированного алгоритма ДСИФ; начальная конфигурация *S0* — 1000 точек, случайным образом размещенных в равностороннем треугольнике

Листинг файла SerpDSIF.m

```
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+tmpysqrt(
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,:);
else
z=D;
end
```

Пример фрактала, называемого лист, построенного с использованием ДСИФ, и листинг соответствующей программы представлен в Приложении 1.

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. 16. Ниже приводится листинг файла Crystal.m, содержащего описание соответствующей функции.

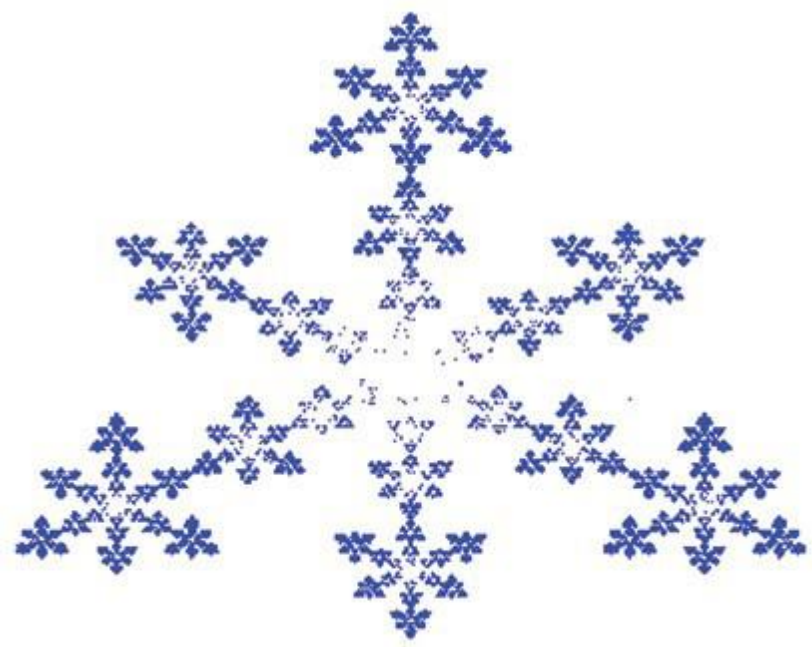

Рис. 16 – Кристалл 4-го порядка, полученный с помощью алгоритма РСИФ после 500000 испытаний

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. Изучить основные типы аффинных преобразований на множестве действительных чисел.
2. Модифицировать аффинные преобразования для программы crystal.m для получения изображения кристалла другой формы.