МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ СИСТЕМ І ПРОЦЕСІВ

Лекції 2-3. Системи ітерованих функцій. Фрактальна розмірність

Київ – 2015

Мощный способ построения фрактальных множеств, также применяющийся в прикладных областях (например, для сжатия изображений), разработали Дж. Хатчинсон (1981), М. Барнсли (1985) и др. — это применение систем итерированных функций (СИФ).

Рассмотрим конечномерное евклидово пространство[1] E ($\mathbb{R}^1$, $\mathbb{R}^2$ или $\mathbb{R}^3$) и семейство сжимающих отображений $f_j\colon E \mapsto E$ ($j = \overline{1,m}$) с коэффициентами сжатия $s_j \in [0,1)$:

$$|f_j(x) - f_j(y)| \le s_j|x-y| \qquad \forall x, y \in E.$$

Тогда из теоремы о неподвижной точке можно получить, что существует единственное непустое компактнее подмножество $A \subset E$, такое что

$$A = \bigcup_{j=1}^{m} f_j(A).$$

Это подмножество называют *аттрактором СИФ* и оно во многих случаях является фрактальным множеством: оно состоит из уменьшенных копий самого себя. Та же теорема о неподвижной точке сжимающего отображения даёт и способ построения аттрактора:

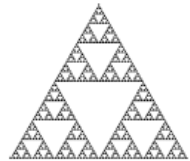
$$A = \lim_{n\to\infty} (f_1 \cup \ldots \cup f_m)^{(n)}(C), \tag{2.1}$$

где C — произвольное непустое компактное множество, а предел берется по метрике Хаусдорфа [1, с. 92]. Верна и более удобная формула:

$$A = \bigcup_{\sigma\in m^\omega} \lim_{n\to\infty} f_{\sigma_0} \circ f_{\sigma_1} \circ \ldots \circ f_{\sigma_n}(x), \tag{2.2}$$

где m^ω — множество бесконечных слов (последовательностей) $\sigma = \sigma_0\sigma_1 \ldots \sigma_n \ldots$, составленных из символов $1, 2, \ldots, m$, $x \in E$ — произвольная точка (в данном случае пределы по метрике Хаусдорфа и по евклидовой метрике для одноточечных множеств совпадают).

Примеры СИФ и их аттракторов.



Если $f_1(x) = \frac{1}{3}x$ и $f_2(x) = \frac{1}{3}x + \frac{2}{3}$, $x \in \mathbb{R}$, то ее аттрактором является пыль Кантора.

Если $f_1(x) = \frac{1}{2}x$, $f_2(x) = \frac{1}{2}x + \frac{1}{2}$ и $f_3(x) = \frac{1}{2}x + (\frac{1}{4}, \frac{\sqrt{3}}{4})$, $x \in \mathbb{R}^2$, то ее аттрактор — треугольник Серпинского.

Для построения фрактальных множеств с помощью СИФ пользуются детерминированным и рандомизированным (предпочтительнее) алгоритмами.

2.1 Детерминированный алгоритм построения

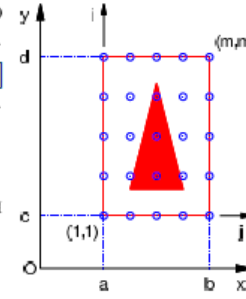
В основе этого алгоритма лежит формула (2.1). Поясним ее: если $F = f_1 \cup \ldots \cup f_m$, то $F(C) = (f_1 \cup \ldots \cup f_m)(C) = \bigcup\limits_{j=1}^{m} f_j(C)$ и $F^{(n)}(C) = \underbrace{F \circ \ldots \circ F}_{n \text{ раз}}(C)$.

Дискретизация вычислений. Для изображения компактного множества A на компьютере надо его дискретизировать: представить конечным набором точек или пикселов. Пусть $A \subset [a,b] \times [c,d]$ и известно, что аттрактор СИФ также лежит в этом прямоугольнике. Разобьём прямоугольник равномерной сеткой $m \times m$ узлов

$$(x_i, y_j) = \left(a + \tfrac{b-a}{m-1}(i-1), c + \tfrac{d-c}{m-1}(j-1)\right), \quad i,j = \overline{1,m}. \qquad (2.3)$$

Произвольную точку (x,y) этого прямоугольника аппроксимируем ближайшим узлом с номерами

$$(i,j) = \left(rnd\left(1 + (m-1)\tfrac{x-a}{b-a}\right), rnd\left(1 + (m-1)\tfrac{y-c}{d-c}\right)\right), \qquad (2.4)$$



rnd — округление числа (`round, ceiling, floor`).

Тогда этой сетке узлов и множеству A будет соответствовать его дискретное приближение — матрица E:

$$E_{ij} = \begin{cases} 1, & \text{если } (x_i, y_j) \in A, \\ 0, & \text{если } (x_i, y_j) \notin A. \end{cases}$$

Образ множества под действием отображения равен объединению образов его точек: $f(A) = \bigcup\limits_{(x,y)\in A} \{f(x,y)\} \approx \bigcup\limits_{(x_i,y_j)\in A} \{f(x_i,y_j)\}$. Точки $f(x_i,y_j)$ могут выйти за границы исходного прямоугольника или не совпасть ни с одним из узлов сетки. В последнем случае аппроксимируем такие точки ближайшими узлами сетки $(x_{\tilde{i}}, y_{\tilde{j}})$ по формуле (2.4).

При отображении матрицы значений яркости в Maxima ее строки отображаются рядами закрашенных прямоугольников одна под другой, начиная с первой строки. Поскольку в записи матрицы E_{ij} обычно i — номер строки, а j — номер столбца, то в матрице нужно изменить порядок всех строк $E_{ij} \mapsto E_{(m+1-i)j}$, а затем транспонировать $E_{(m+1-i)j} \mapsto E_{j(m+1-i)}$ (подумайте, почему). Тогда результат будет соответствовать желаемому расположению фигуры, задающейся этой матрицей.

Алгоритм 2.1: **Детерминированный алгоритм построения СИФ**

Вход: $f_1, \ldots, f_n$; ✎ *итерируемые функции*
m; ✎ *размер матрицы*
level; ✎ *количество итераций*
$[a, b] \times [c, d]$; ✎ *область определения функций*
Выход: E; ✎ *матрица, представляющая аттрактор A*

1: $E =$ случайная матрица $m \times m$; ✎ *инициализация*
2: $S =$ нулевая матрица $m \times m$;
3: $node(i, j) = (a + \frac{b-a}{m-1}(i-1), c + \frac{d-c}{m-1}(j-1))$; ✎ *преобразование координат*
4: $nodeij(x, y) = rnd(1 + \frac{x-a}{b-a}(m-1), 1 + \frac{y-c}{d-c}(m-1))$; ✎ *«обратное» преобразование*
5: **повторять** *level* **раз**
6: **для** $i = 1, \ldots, m$; $j = 1, \ldots, m$
7: **если** $E_{ij} > 0$ **, то**
8: **для** $l = 1, \ldots, n$
9: $(x, y) = f_l(node(i, j))$;
10: **если** $a \le x \le b$ и $c \le y \le d$ **, то**
11: $(p, q) = nodeij(x, y)$;
12: $S_{pq} = 1$;
13: $E = S$;
14: $S = 0$;
15: $E_{ij} = 1 - E_{j,m+1-i}$; ✎ *преобразование матрицы для отображения на экране*

Реализация детерминированного алгоритма построения СИФ

```
1  f[1](x,y):=[0.5*x,0.5*y]$                                  /* итерируемые отображения */
2  f[2](x,y):=[0.5*x+0.5,0.5*y]$
3  f[3](x,y):=[0.5*x+0.25,0.5*y+sqrt(3)/4]$
4  n:3$                                                       /* количество отображений */
5  m:150$          /* размер квадратной матрицы, количество точек разбиения сторон */
6  level:6$                                                   /* количество итераций */
7  [a,b]:[0,1], [c,d]:[0,sqrt(3)/2]$          /* области определения: [a,b]x[c,d] */
8                     /* начальная матрица mxm из случайных элементов для итерации */
9  E: apply(matrix,makelist(makelist(random(2),i,1,m),i,1,m))$
10 S: zeromatrix(m,m)$                          /* матрица для промежуточных вычислений */
11                                                  /* мировые координаты узла (i,j) */
12 node(i,j):=float([a+(b-a)*(i-1)/(m-1),c+(d-c)*(j-1)/(m-1)])$
13                                             /* номера узла, близкого к точке (x,y) */
14 nodeij(x,y):=map(round,[1+(m-1)*(x-a)/(b-a),1+(m-1)*(y-c)/(d-c)])$
```

```
15
16  thru level do
17     (
18     for i:1 thru m do
19        for j:1 thru m do
20          if E[i,j]>0 then for l:1 thru n do
21            (                                  /* применяем отображение к узлам сетки: */
22            [x,y]: apply(f[l],node(i,j)),
23                                    /* если точка оказалась внутри прямоугольника */
24            if a<=x and x<=b and c<=y and y<=d then
25               (                               /* отмечаем узел, близкий к этой точке */
26               [ii,jj]:nodeij(x,y),
27               S[ii,jj]:1
28               )
29            ),
30     E:S,
31     S:zeromatrix(m,m)
32     )$
33                                  /* преобразуем матрицу E для отображения графики */
34  E2:apply(matrix,makelist(makelist(1-E[j,m+1-i],i,1,m),j,1,m))$
35                                                  /* отобразим матрицу на экране */
36  load(draw)$
37  wxdraw2d(image(E2,0,0,size,size),palette=gray,colorbox=false,noframe,
38         user_preamble="set size ratio -1"
39         )$
```

Комментарий.

Замечания об использованных командах:

`map(f,[x1,x2,...])` — применение функции $f(x)$ к массиву поэлементно: получаем новый массив `[f(x1),f(x2),...]`. Для многих функций (например, `float`) такое переопределение команды происходит автоматически (см. строчку 12, там `float` используется, чтобы не хранить в памяти числа в виде обыкновенных дробей);

`apply(f,[x1,x2,...])` — применение функции $f(x_1, x_2, \ldots)$ к массиву своих аргументов: получаем значение `f(x1,x2,...)`;

`makelist(`$f(i)$`,i,1,n)` — создать массив, состоящий из элементов $f(i)$, $i = 1, \ldots, n$;

`makelist(makelist(`$M(i,j)$`,i,1,m),j,1,n)` — типичное создание массива M_{ij} размера $m \times n$;

`apply(matrix,makelist(makelist(`$M(i,j)$`,i,1,m),j,1,n))` — типичное создание матрицы M_{ij} размера $m \times n$;

`random(n)` — генерация случайного числа от 1 до n;

`zeromatrix(m,n)` — создать матрицу $m \times n$ из нулевых элементов;

`and` — логическая операции конъюнкции;

`image(M,x,y,m,n)` — изображение матрицы цветов M (числа от 0 до 255) в виде таблицы размера $m \times n$, левый нижний угол которой имеет координаты (x, y). Самому яркому цвету соответствует 255, самому тёмному — 0, поэтому в строчке 30 мы заменяем все 1 на 0 и наоборот: $E \mapsto 1 - E$;

`palette=gray` — отображение в оттенках серого (попробуйте убрать эту опцию!);
`colorbox=false` — не показывать легенду цветов;
`noframe` — пользовательская опция (см. с. 9), не забудьте ее определить!
Отображение таблицы в исходной системе координат (вместо строчек 34 и 37–39):

```
path:[]$                                          /* объявляем массив точек */
for i:1 thru size do
    for j:1 thru size do  /* добавляем к массиву точки с ненулевым цветом */
        if E[i,j]>0 then path: append(path,[nodeij(i,j)])$
wxdraw2d(point_type=circle,point_size=0.3,user_preamble="set size ratio -1",
     points(path)
)$
```

2.2 Рандомизированный алгоритм

Рандомизированный алгоритм (иногда его называют «игрой в хаос») работает намного быстрее детерминированного за счёт сокращения объёма памяти и количества вычислений и осуществляется наиболее эффективно на компьютерах, в которых есть возможность вывода графического изображения на экран в режиме 1 пиксел за раз[2]. В основе этого алгоритма лежит формула (2.2).

Алгоритм 2.2: **Рандомизированный алгоритм построения СИФ**

Вход: $f_1, \ldots, f_n$; ✎ *итерируемые функции*
$points$; ✎ *количество точек*
$skip$; ✎ *количество пропущенных точек*
Выход: $path$; ✎ *массив координат точек фрактала*

$path = \emptyset$; ✎ *инициализация пустого массива*
$x_0 = 0$; ✎ *начальная точка итераций*
повторять $skip$ **раз**
    $l =$ случайное число от 1 до n;
    $x_0 = f_l(x_0)$;
повторять $points$ **раз**
    $l =$ случайное число от 1 до n;
    $x_0 = f_l(x_0)$;
    $path = path \cup x_0$

Реализация рандомизированного алгоритма построения СИФ

```
1   f[1](x,y):=float([0.5*x,0.5*y])$
2   f[2](x,y):=float([0.5*x+0.5,0.5*y])$
3   f[3](x,y):=float([0.5*x+0.25,0.5*y+sqrt(3)/4])$
4   n:3$
5   pointsnum: 20000$
6   skip: 50$
7
8   p:[0,0]$
9   path:[]$
10
11  thru skip do
12     (l: random(n)+1,
13      p: apply(f[l],p)
14      )$
15  thru pointsnum do
16     (l: random(n)+1,
17      p: apply(f[l],p),
18      path:endcons(p,path)
19      )$
20
21  load(draw)$
22  wxdraw2d(point_type=dot,points(path),user_preamble="set size ratio -1")$
```

Комментарий.

Замечания об использованных командах:
`random(n)` — генерирование псевдослучайного целого числа от 0 до $n-1$.
Заметим, что рандомизированный алгоритм работает еще быстрее, когда нет необходимости хранить в памяти массив точек `path`. Если вы работаете не в ОС Windows, то в строчке 10 можно установить специальный режим `multiplot_mode(screen)$` вывода любой графики на один экран, вместо строчек 21–22 нужно вернуть прежнее состояние вывода на экран: `multiplot_mode(none)$`. Тогда вместо строчки 18 можно непосредственно отображать точку на экране на каждом шаге итерации: `draw2d(points([p]))$`.

См. также Задачу 2.2, где обсуждается равномерность заполнения точками аттрактора.

Задания к лабораторной №3

Задание 1.

Построить аттракторы, задающиеся системами аффинных итерированных отображений. Коэффициенты формулы $f_j([\begin{smallmatrix}x\\y\end{smallmatrix}]) = \left[\begin{smallmatrix}A_j & B_j\\ C_j & D_j\end{smallmatrix}\right]\cdot[\begin{smallmatrix}x\\y\end{smallmatrix}] + \left[\begin{smallmatrix}E_j\\ F_j\end{smallmatrix}\right]$ даны в Таблице 2.1. Для этого удобно ввести матрицу из строк $M = (A_j, B_j, C_j, D_j, E_j, F_j)_{j=1}^{n}$ (её можно заполнять построчно из данных таблицы) и инициализировать отображения:

```
for j:1 thru n do f[j](x,y):=[M[j,1]*x+M[j,2]*y+M[j,5],M[j,3]*x+M[j,4]*y+M[j,6]]$
```

Кристалл

A	B	C	D	E	F
0.2550	0.0000	0.0000	0.2550	0.3726	0.6714
0.2550	0.0000	0.0000	0.2550	0.1146	0.2232
0.2550	0.0000	0.0000	0.2550	0.6306	0.2232
0.3700	-0.6420	0.6420	0.3700	0.6356	-0.0061

Папоротник

A	B	C	D	E	F
0.7000	0.0000	0.0000	0.7000	0.1496	0.2962
0.1000	-0.4330	0.1732	0.2500	0.4478	0.0014
0.1000	0.4330	-0.1732	0.2550	0.4445	0.1559
0.0000	0.0000	0.0000	0.3000	0.4987	-0.0070

Ковер 1

A	B	C	D	E	F
0.5000	0.0000	0.0000	-0.5000	0.5000	0.5000
0.0000	-0.5000	-0.5000	0.0000	0.5000	0.5000
-0.5000	0.0000	0.0000	-0.5000	0.5000	1.0000

Ковер 2

A	B	C	D	E	F
0.5000	0.0000	0.0000	-0.5000	0.0000	1.0000
0.0000	0.5000	0.5000	0.0000	0.0000	0.0000
0.5000	0.0000	0.0000	0.5000	0.5000	0.0000

Лист

A	B	C	D	E	F
0.4000	-0.3733	0.0600	0.6000	0.3533	0.0000
-0.8000	-0.1867	0.1371	0.8000	1.1000	0.1000

Ковер (треугольник) Серпинского

A	B	C	D	E	F
0.5000	0.0000	0.0000	0.5000	0.0000	0.0000
0.5000	0.0000	0.0000	0.5000	0.5000	0.0000
0.5000	0.0000	0.0000	0.5000	0.2500	0.4330

Дерево

A	B	C	D	E	F
0.1950	-0.4880	0.3440	0.4430	0.4431	0.2452
0.4620	0.4140	-0.2520	0.3610	0.2511	0.5692
-0.0580	-0.0700	0.4530	-0.1110	0.5976	0.0969
-0.0350	0.0700	-0.4690	0.0220	0.4884	0.5069
-0.6370	0.0000	0.0000	0.5010	0.8562	0.2513

Таблица 2.1: Коэффициенты для аффинных СИФ.

Существует много формальных определений понятия фрактал. Основоположник этого термина Б. Мандельброт [5], замечая ограниченность своих собственных определений фрактала, до конца своей жизни пытался предложить новые, более универсальные его определения. Но при этом понятие «фрактал» всегда тесно связано с понятиями масштабной инвариантности (в том числе самоподобия) и нецелой размерности.

Существует много видов *фрактальных размерностей* (общий термин), например[1]:

- размерность Хаусдорфа-Безиковича;
- размерность самоподобия (подобия);
- размерность Минковского (размерность Минковского-Булигана, грубая размерность, дробная размерность, фрактальная размерность, емкость по Колмогорову, размерность подсчёта клеток, емкостная размерность, метрическая размерность, логарифмическая плотность и др.);
- упаковочная размерность;
- размерности Реньи (в том числе информационная, энтропийная и корреляционная размерности).

Их выбор диктуется удобством применения и качеством описания особенностей множества. Для многих фрактальных множеств эти размерности совпадают, но существуют примеры множеств, различные виды размерностей которых отличаются.

Мы рассмотрим размерность Минковского, имеющую наиболее простой способ практического вычисления на компьютере[2].

Рис. 3.1: Подсчёт размерности Минковского **границы** побережья Великобритании способом ③ (три рисунка слева). Способы подсчёта ⑤ и ① (два рисунка справа).

3.1 Размерность Минковского

Размерность Минковского множества $A \subset \mathbb{R}^n$, можно определить так[3]:

$$\dim_M(A) = \lim_{\varepsilon \to 0} \frac{\log(N(\varepsilon))}{-\log(\varepsilon)}, \tag{3.1}$$

если этот предел существует (иначе рассматривают нижний или верхний пределы), где $N(\varepsilon)$ выбирается одним из следующих эквивалентных способов[4]:

① наименьшее число замкнутых шаров радиуса ε, покрывающих множество A;
② наименьшее число замкнутых кубов со стороной ε, покрывающих множество A;
③ количество кубов ε-сетки, пересекающихся с множеством A;
④ наименьшее число множеств диаметра не больше ε, покрывающих множество A;
⑤ наибольшее число непересекающихся замкнутых шаров радиуса ε, с центрами в множестве A.

Для практического применения выбирается наиболее удобный из этих способов или придумывается новый. Чаще используется способ ③ («подсчет клеток», «box-counting»).

3.1.1 Клеточный метод подсчёта размерности Минковского

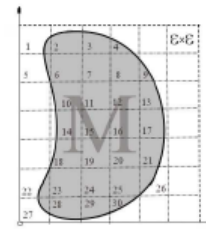
ε-сеткой назовем множество n-мерных кубов вида

$$[m_1\varepsilon, (m_1+1)\varepsilon] \times \ldots \times [m_n\varepsilon, (m_n+1)\varepsilon],$$

где m_i — целые числа. На плоскости $\mathbb{R}^2$ ε-сетка — это разбиение плоскости квадратами со сторонами ε, параллельными осям абсцисс и ординат.

Из формулы (3.1) следует, что $-\dim_M(A)$ — приближённо наклонный коэффициент графика $N = N(\varepsilon)$ в осях $\log(\varepsilon)$–$\log(N(\varepsilon))$[5] при малых ε. Поэтому на практике рассматривают набор масштабов $\varepsilon \in scales = \{s_1, \ldots, s_Q\}$, считают количество $N(\varepsilon)$ кубов ε-сетки, пересекающихся с данным множеством, и находят наклонный коэффициент графика прямой, аппроксимирующей точки $\{(\log(s_p), \log(N(s_p)))\}_{p=1}^{Q}$ методом наименьших квадратов (см. параграф 3.1.3).

При очень малых и при больших ε возникают неточности, связанные с границами диапазона применимости степенного закона (3.1) для описания «фрактальности» данного множества, а также связанные с конечной точностью его дискретизации. Поэтому выбор шкалы масштабов является эмпирической задачей, и поэтому он неизбежно вносит погрешность в вычисление $\dim_M(A)$ (см. Задачу 3.2).



Алгоритм 3.1: **Оценка размерности Минковского. Клеточный метод**

Вход: E; ✎ *квадратная матрица E_{ij}, соответствующая дискретизации множества A*
$scales = \{s_1, \ldots, s_Q\}$; ✎ *массив масштабов (по умолчанию $scales = \{1, 2, \ldots, Q\}$)*
Выход: $\dim_M$ ✎ *оценка размерности Минковского множества A*
и графическое изображение интерполяции методом наименьших квадратов;

1: m, n = размеры матрицы E; ✎ *инициализация*
2: $path = \emptyset$; ✎ *пустой массив точек для последующей интерполяции*
3: **для** $sc \in scales$
4: $N = 0$; ✎ *счётчик кубов размера sc, пересекающих множество A*
5: C_x, C_y = количество кубов со стороной sc, умещающихся в E вдоль соответствующих сторон;
6: **для** $i = 1, \ldots, C_x$; $j = 1, \ldots, C_y$
7: $cnt = \sum_{l=(j-1)\cdot sc+1}^{j\cdot sc} \sum_{k=(i-1)\cdot sc+1}^{i\cdot sc} E_{kl}$; ✎ *число общих точек данного куба и фрактала*
8: **если** $cnt > 0$, **то** $N = N + 1$;
9: $path = path \cup \{\log(sc), \log(N)\}$;
10: методом наименьших квадратов находим прямую $y = Ax + B$, аппроксимирующую набор данных $path$;
11: $\dim_M = -A$;
12: строим на одном графике множество точек $path$ и прямую $y = Ax + B$;

Оценка размерности Минковского. Клеточный метод

```
1  m: length(E)$                    /* размеры матрицы E; матрица известна заранее */
2  n: length(E[1])$
3  scales:[2,4,8,10,16]                        /* пример задания массива масштабов */
4  path:[]$                       /* массив точек для последующей интерполяции */
5
6  for sc in scales do
7      (N:0,
8      Cx:floor(m/sc),
9      Cy:floor(n/sc),
```

```
10      for i:1 thru Cx do
11         for j:1 thru Cy do
12         (     /* суммирование значений пикселов в каждом i-м,j-м кубе разбиения */
13         cnt: sum(sum(E[k,l],l,(j-1)*sc+1,j*sc),k,(i-1)*sc+1,i*sc),
14         if cnt>0 then N:N+1
15           /* если есть хотя бы один пиксел из множества A, засчитываем этот куб */
16         ),              /* составляем массив для интерполяции в формате log-log: */
17      path:endcons(float([log(sc),log(N)]),path)
18      )$
19
20   load (lsquares)$
21   M: apply(matrix,path)$
22   lse:lsquares_estimates(M, [x,y], y=A*x+B, [A,B]),float$
23   [a,b]:map(rhs,[lse[1][1],lse[1][2]]);
24   dim:-a;
25   load(draw)$
26   list:makelist(path[i][1],i,1,length(scales))$
27   [xmin,xmax]:[lmin(list),lmax(list)]$
28   wxdraw2d(points(path),explicit(a*x+b,x,xmin,xmax))$
29   print("Minkowski dimension=",-a)$
```

Комментарий.

Заметим прежде всего, что этот алгоритм работает с матрицей E, представляющей дискретный образ множества A. Эту матрицу можно получить, например, из предыдущей главы, в которой строились аттракторы СИФ. На данный момент Maxima не предусматривает эффективного импорта готовых изображений (есть слабая попытка использовать формат **xpm**, но это чрезвычайно неудобно).

Замечания об использованных командах:

`for i in a do` — цикл по переменной `i`, пробегающей элементы массива `a`;

`floor(x)` — целая часть числа `x` (округление вниз);

`sum(`$f(i)$`,i,`i_1`,`i_2`)` — суммирование $\sum_{i=i_1}^{i_2} f(i)$;

`lmin(list)` и `lmax(list)` — минимум и максимум значений в массиве `list`;

`print(`$a_1,\ldots,a_n$`)` — выполняет команды $a_1,\ldots,a_n$ и выводит результат на экран в одну строку слева направо; кроме того, она возвращает результат последней команды a_n (чтобы этого избежать, мы поставили `$`).

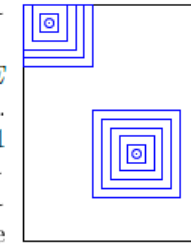
`rhs(x=y)` — возвращает правую часть `y` равенства `x=y` (right-hand-side). Очень полезная команда для «выуживания» ответов из массивов ответов (ее аналог для левых частей равенства — `lhs()`).

Результат выполнения команды `lse:lsquares_estimates(M,[x,y],y=A+B*x,[A,B])`: `[[A=...,B=...]]` — массив равенств, содержащийся в переменной `lse`.

Чтобы использовать эти значения `A` и `B`, нужно «достать» их из массивов `lse[1][1]` и `lse[1][2]`, а затем взять правые части равенств командой `rhs`: `AA: rhs(lse[1][1])` и `BB: rhs(lse[1][2])` (см. строчку 23).

Заметим также, что в этом алгоритме не засчитываются те части множества A, которые попадают в обрезанные клетки (C_x+1, j) и (i, C_y+1). Попробуйте изменить алгоритм так, чтобы учитывать и эти клетки.

3.1.2 Точечный метод подсчёта размерности Минковского



Этот метод основан на следующей оценке $\langle N(L)\rangle$ числа покрывающих клеток $N(L)$ размера $L \times L$.

Пусть дискретное изображение фрактала A — бинарная матрица E размером $I \times J$ — состоит из M точек (то есть в матрице E M единиц). Будем покрывать точки фрактала A квадратами размера $2L+1 \times 2L+1$ (для удобства) со сторонами, параллельными осям координат и с центрами в этих точках. Таким образом, мы получим M квадратов. Подсчитаем число $P_{m,L}$ квадратов, содержащих ровно m (m — натуральное число) точек из A. Тогда $\frac{P_{m,L}}{M}$ — вероятность (доля) того, что в произвольном таком квадрате будет ровно m точек.

Ожидаемое количество квадратов со стороной $2L+1$, содержащих ровно m точек, равно $\frac{M}{m}\frac{P_{m,L}}{M} = \frac{P_{m,L}}{m}$. Тогда при покрытии множества A квадратами ожидаемое их количество

$$\langle N(2L+1)\rangle = \sum_{m=1}^{K} \frac{P_{m,L}}{m}, \tag{3.2}$$

где K — максимальное количество точек A в одном квадрате со стороной $2L+1$ (можно взять $K = (2L+1)^2$).

Далее, из формулы (3.1) получаем, что $\dim_M(A) \approx -\frac{\log(\langle N(2L+1)\rangle)}{\log(2L+1)}$, откуда размерность находится тем же способом, как и в клеточном методе, см. параграф 3.1.1. Также справедливо приведенное там замечание о важности набора масштабов *scales*.

Алгоритм 3.2: **Оценка размерности Минковского. Точечный метод**

Вход: E; ✎ *матрица размера $I \times J$, соответствующая дискретизации множества A*
$scales = \{s_1, \ldots, s_Q\}$; ✎ *массив масштабов (по умолчанию $scales = \{1, \ldots, Q\}$)*
Выход: $\dim_M$ ✎ *оценка размерности Минковского множества A*
и графическое изображение интерполяции методом наименьших квадратов;

$I, J =$ размеры матрицы E; ✎ *инициализация*
$path = \emptyset$; ✎ *пустой массив точек для последующей интерполяции*
$Q =$ длина массива $scales$;
$P =$ нулевая матрица размера $(2s_Q + 1)^2 \times Q$; ✎ *матрица количества клеток размера $2L + 1$ ($L = s_1, \ldots, s_Q$), содержащих m точек ($m \in \{1, \ldots, (2s_Q + 1)^2\}$)*
для $i = 1, \ldots, I$; $j = 1, \ldots, J$
  если $E_{ij} > 0$ **, то**
    ✎ *если это точка фрактала, окружим ее квадратами разных сторон и посчитаем количество точек в них*
    для $L = 1, \ldots, Q$
      $m = \sum_{s=\max(j-s_L,1)}^{\min(j+s_L,J)} \sum_{r=\max(i-s_L,1)}^{\min(i+s_L,I)} E_{rs}$; ✎ *суммируем пикселы внутри квадратов $[i - s_L, i + s_L] \times [j - s_L, j + s_L]$, и если квадрат выходит за границы изображения, то обрезаем его с помощью функций* max *и* min
      $P_{m,L} = P_{m,L} + 1$;
для $L = 1, \ldots, Q$

12: $N = \sum_{m=1}^{(2s_Q+1)^2} \frac{P_{m,L}}{m}$; ✎ *считаем* $\langle N(2L+1)\rangle$ *по формуле* (3.2)

13: $path = path \cup \{\log(2s_L+1), \log(N)\}$;

14: методом наименьших квадратов находим прямую $y = Ax + B$, аппроксимирующую набор данных $path$;

15: $\dim_M = -A$;

16: строим на одном графике множество точек $path$ и прямую $y = Ax + B$;

Оценка размерности Минковского. Точечный метод

```
1   I: 80$                                  /* размеры матрицы E; матрица известна заранее */
2   J: 100$
3   scales:[1,2,3,4,8]$
4   Q:length(scales)$
5   P: zeromatrix((2*scales[Q]+1)^2,Q)$
6
7   path:[]$
8
9   for i:1 thru I do
10     for j: 1 thru J do
11        if E[i,j]>0 then
12           for L:1 thru Q do
13              (m:sum(
14                    sum(E[r,s],r,max(i-scales[L],1),min(i+scales[L],I)),
15                    s,max(j-scales[L],1),min(j+scales[L],J)),
16              P[m,L]:P[m,L]+1
17              )$
18
19  for L:1 thru Q do
20  (N[L]:sum(P[m,L]/m,m,1,(2*scales[Q]+1)^2),
21  path:endcons(float([log(2*scales[L]+1),log(N[L])]),path)
22  )$
```

```
23
24  load (lsquares)$
25  M: apply(matrix,path)$
26  lse:lsquares_estimates (M, [x,y], y= A*x+B, [A,B]),float$
27  [a,b]:map(rhs,[lse[1][1],lse[1][2]]);
28  load(draw)$
29  list:makelist(path[L][1],L,1,Q)$
30  [xmin,xmax]:[lmin(list)-1/2,lmax(list)+1/2]$
31  wxdraw2d(points(path),explicit(a*x+b,x,xmin,xmax))$
32  print("Minkowski dimension=",-a)$
```

3.1.3 Метод наименьших квадратов

Это один из способов аппроксимации ряда численных данных, заключающийся в минимизации среднеквадратичной ошибки. Имеется неизвестная зависимость $y = y(\vec{x})$, $\vec{x} = \{x_1, \ldots, x_n\}$, заданная выборкой значений в m n-мерных точках $y(\vec{x}^0_i) = y^0_i$, $\vec{x} \in \mathbb{R}^n$. Аппроксимируем ее с помощью модели (функции) $y = F(\vec{x}, \vec{\alpha})$, зависящей от вектора параметров $\alpha \in \mathbb{R}^k$. Требуется с помощью выбора параметров свести к минимуму среднеквадратичную ошибку вычисления ответов в данной модели на данной выборке:

$$\mathcal{L}(\vec{\alpha}) = \sum_{i=1}^{m} (F(\vec{x}^0_i, \vec{\alpha}) - y^0_i)^2 \to \min_{\vec{\alpha}}.$$

После нахождения оптимальных параметров, эту модель можно использовать для произвольных данных $\vec{x}$.

В случае гладкого отображения F задача нахождения параметров решается поиском критической точки этого функционала: решается уравнение $\operatorname{grad} \mathcal{L}(\vec{\alpha}) = \vec{0}$. В случае же линейного отображения $y = F(\vec{x}, \vec{\alpha}) = \sum_{i=1}^{n} \alpha_i x_i + \alpha_{n+1}$ эта задача эквивалентна системе линейных алгебраических уравнений.

Например, набор точек $\{(x_i^0, y_i^0)\}_{i=1}^m$ интерполируем прямой $y = Ax + B$ методом наименьших квадратов. Тогда $\overrightarrow{x}_i^0 = x_i^0 \in \mathbb{R}$, $\overrightarrow{\alpha} = \{A, B\}$, $\mathcal{L}(A, B) = \sum_{i=1}^m (Ax_i^0 + B - y_i^0)^2$. Уравнение $\operatorname{grad} \mathcal{L}(\overrightarrow{\alpha}) = \overrightarrow{0}$ эквивалентно системам

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial A} = 0, \\ \frac{\partial \mathcal{L}}{\partial B} = 0, \end{cases} \iff \begin{cases} \sum_{i=1}^m (Ax_i^0 + B - y_i^0) x_i^0 = 0, \\ \sum_{i=1}^m (Ax_i^0 + B - y_i^0) = 0, \end{cases}$$

решение которых можно найти в аналитическом виде по правилу Крамера:

$$A = \frac{\overline{xy} - \overline{x}\,\overline{y}}{\overline{x^2} - \overline{x}^2}, \qquad \overline{x} = \sum_{i=1}^m x_i^0/m, \qquad \overline{y} = \sum_{i=1}^m y_i^0/m,$$

$$B = \overline{y} - \frac{\overline{xy} - \overline{x}\,\overline{y}}{\overline{x^2} - \overline{x}^2}\overline{x}, \qquad \overline{xy} = \sum_{i=1}^m x_i^0 y_i^0/m, \qquad \overline{x^2} = \sum_{i=1}^m (x_i^0)^2/m.$$

Комментарий.

В Maxima метод наименьших квадратов реализован следующей командой пакета `lsquares`:

```
lsquares_estimates(M, [x,y], y=A+B*x, [A,B])$
```

Более общий вариант этой команды:

```
lsquares_estimates(M, [x_1, ..., x_n], F(α_1, ..., α_k, x_1, ..., x_n) = 0, [α_1, ..., α_k])$
```

переменные выборки; модель аппроксимации; искомые параметры

Здесь `M` — матрица $m \times n$, ряды которой — элементы выборки: $M_i = \{x_1, \ldots, x_n\}_i$.

Задания к лабораторной №3

Задание 2.

Испытайте алгоритм оценки размерности Минковского на фракталах, полученных в предыдущем пункте, особенно на тех, у которых размерность можно подсчитать аналитически.