МОДЕЛИРОВАНИЕ ФРАКТАЛОВ В СРЕДЕ MAXIMA

часть I

П.И. ТРОШИН

Казанский федеральный университет

Казань 2012

Оглавление

ВВЕДЕНИЕ

Чем абстрактнее истина, которую ты хочешь преподать, тем сильнее ты должен обольстить ею еще и чувства.

Фридрих Ницше

Красоту в математике так же трудно формально определить, как и красоту в исскустве, но люди, изучающие математику, обычно не имеют затруднений в ее распознавании.

Поль Дирак

Целое столетие для математики прошло впустую, поскольку рисование не играло тогда в науке никакой роли. Рука, карандаш и линейка исчерпали себя.

Бенуа Мандельброт

Не оспаривая удобство использования и интерфейс различного программного обеспечения компьютерной математики, в данном пособии было решено в качестве основного средства компьютерного моделирования использовать систему компьютерной алгебры Maxima. Преимуществами этой системы являются: бесплатная лицензия, кроссплатформенность (реализация в ОС Windows, Linux, Unix, Mac OS X, BSD), открытый код (что понижает количество встроенных ошибок), поддержка языка программирования Lisp и др. Maxima выполняет символьные и численные вычисления, строит различные графики.

Максима имеет несколько графических интерфейсов: XMaxima, wxMaxima и др., а также может работать в режиме командной строки, используя псевдографику. Предполагается, что читатель знаком с основами языка Maxima.

Ввиду сделанного выбора данное пособие может использоваться при изучения применения пакета Maxima в курсе высшей математики. Однако все изложенные алгоритмы читатель легко сможет адаптировать для других языков математических вычислений.

Заметим, что хотя данное пособие посвящено построению фракталов, мы не будем давать строго определения фрактальным множествам и фрактальной размерности, чтобы избежать однобокого или же, наоборот, слишком пространного и техничного определения. Предполагается, что читатель знаком с широким значением понятия, которое мы будем просто называть «фрактал».

Настоящее пособие состоит из двух частей: первая часть посвящена построению фрактальных множеств и их изучению, а вторая часть — изучению динамических систем, обладающих фрактальными аттракторами и свойством хаотичности.

Основное содержание пособия близко по изложению книге Р.М. Кроновера [1] и охватывает широко известные алгоритмы построения фрактальных множеств. Каждая глава снабжена списком Задач. В Приложении даются сведения об использовании готовых специализированных пакетов для изучения фракталов и динамических систем.

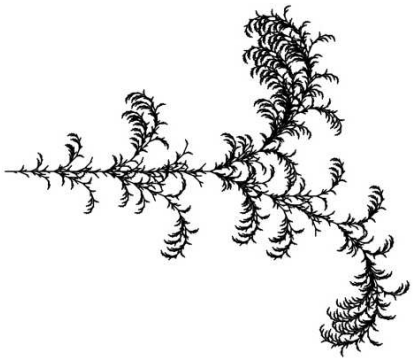
Первое фрактальное множество построим в Maxima прямо на этой странице[1]:

```
plot2d(sum(0.5^n*cos(2*\%pi*4^n*x),n,0,200), [x, 0, 0.5])$
```

Это функция Карла Вейерштрасса $W(x) = \sum_{n=0}^{\infty} w^n \cos(2\pi b^n x)$, всюду непрерывная и недифференцируемая, фрактальная размерность ее графика в нашем случае равна $D = \frac{3}{2}$ ($b > 1$, $0 < w < 1$, $D = 2 + \log_b w$, $1 < D < 2$). О таких функциях Шарль Эрмит писал Томасу Стильтьесу 20 мая 1893 года: «Я с дрожью ужаса отворачиваюсь от ваших несчастных проклятых функций, у которых нет производных». При $D < 1$ $W(t)$ становится дифференцируемой функцией! Первая Задача: используя Приложение, постройте анимацию этого графика при разных значениях w и b.

Рекомендуемая литература:
официальная страница Maxima: [1e],
установочный файл Maxima: [2e],
руководства пользователя Maxima: [5e], [6e], [4e].

[1]Здесь и далее мы не будем ставить точки после формул кода в конце предложений, так как это несомненно вызовет ошибку при их компиляции.



Глава 1

МОДЕЛИРОВАНИЕ L-СИСТЕМ

Красота тесно связана с симметрией.
Герман Вейль

Концепция *L-систем* (Lindenmayer System) была предложена в 1968 году венгерским ботаником Аристидом Линденмайером для изучения развития простых многоклеточных организмов. Позже она была расширена для моделирования сложных ветвящихся структур (деревьев, цветов), формальных языков, биологических моделей селекции, фракталов и др.

1.1 $D0L$-системы

Простейшими из L-систем являются $D0L$-системы («Deterministic 0-sided», «Deterministic Context-Free», «детерминированные контекстно-свободные»), рассматриваемые в этом параграфе.

L-система состоит из построенного итерационным процессом слова и алгоритма его графической интерпретации. Формально слово состоит из символов-команд[1] F, b, $+$, $-$, $[$, $]$ и строится следующим образом. К начальному слову-аксиоме $axiom = W^0 = W^0(F, b, +, -, [,])$ одновременно (параллельно)[2] применяются порождающие правила

$$F \longmapsto newF, \quad b \longmapsto newb,$$

конструируя новое слово $W^1 = W^0(newF, newb, +, -, [,])$. Затем процесс повторяют, получая последовательность слов $W^0, W^1, W^2, \ldots$ со сложной внутренней структурой символов.

[1] Последние два символа [и], строго говоря, принадлежат более сложным *ветвящимся* L-системам. См. параграф 1.2.

[2] Не допускается последовательная замена символов на одном шаге итерационного процесса. Например, нельзя в слове Fb с правилами $F \mapsto b$, $b \mapsto F$ заменить сначала F на b (получив bb), а затем заменить b на F (получив FF)! Результат должен быть bF. Этой особенностью замены L-системы отличаются от формальных грамматик Хомского [2].

Например, из аксиомы $W^0 = F$ и порождающих правил $newF = F + F - -F + F$, $newb = b$ можно получить следующие слова:

$$W^0 = F, \quad W^1 = F + F - -F + F$$

$$W^2 = F + F - -F + F + F + F - -F + F - -F + F - -F + F + F + F - -F + F, \ldots$$

Процесс интерпретации слова в графический объект называют «черепашьей графикой». Представим черепашку как материальную точку на плоскости. Состояние черепашки (x, y, α) определяется ее декартовыми координатами (x, y) и углом α (отмеренным от оси абсцисс), в направлении которого смотрит ее голова. Черепашка находится в начальном положении (x_0, y_0, α_0), она умеет шагать по прямой в направлении своего взгляда (длина одного шага равна $d > 0$) и поворачивать голову на постоянный угол $\theta > 0$. Черепашка читает интерпретируемое слово слева-направо и, в зависимости от прочитанного символа, выполняет следующие команды:
F — проползти на шаг вперед, прочерчивая при этом свой след (отрезок);
b — проползти на шаг вперед, не прочерчивая своего следа;
$+$ — повернуть голову на угол θ;
$-$ — повернуть голову на угол $-\theta$;
[— запомнить свое текущее состояние (x, y, α) (команда ветвления, см. параграф 1.2.1);
] — вернуться в последнее запомненное состояние.

Проиллюстрируем работу черепашки на примере слов W^0, W^1, W^2, полученных в вышеприведённом примере (здесь $\theta = \pi/3$, масштаб длины на всех картинках разный):

Алгоритм 1.1: **Реализация L-системы и черепашьей графики (без символа b)**

Вход: $axiom$; ✎ *слово-аксиома*
$newF$; ✎ *слово-правило $F \mapsto newF$*
$newb$; ✎ *слово-правило $b \mapsto newb$*
$level$; ✎ *количество итераций*
θ; ✎ *угол поворота черепашки*
Выход: W ✎ *результирующее кодовое слово*
и его графическая интерпретация.

1: $W = axiom$; ✎ *инициализация искомого слова*
2: **повторять** $level$ **раз**
3: строим слово T, получающееся из слова W заменами: $F \mapsto newF$, $b \mapsto newb$;
4: $W = T$;
✎ *интерпретация слова W:*
5: $\alpha = 0$; ✎ *инициализируем начальное направление черепашки*
6: $(x_0, y_0) = (0, 0)$; ✎ *инициализируем начальное положение черепашки*
7: $path = \{(x_0, y_0)\}$; ✎ *инициализируем массив точек ломаной пути черепашки*
8: $stack = \emptyset$; ✎ *инициализируем стек ветвления*

для $j = 1, \ldots,$ длина слова W
  если $W[j] = +$, **то** $\alpha = \alpha + \theta$;
  если $W[j] = -$, **то** $\alpha = \alpha - \theta$;
  если $W[j] = F$ **, то**
    $x = x_0 + \cos\alpha$;
    $y = y_0 + \sin\alpha$;
    $path = path \cup (x, y)$;
    $(x_0, y_0) = (x, y)$;
  если $W[j] = [$, **то** $stack = stack \cup (x_0, y_0, \alpha)$;
  если $W[j] =]$ **, то**
    $(x_0, y_0, \alpha) = stack(-1)$; ✎ *достаем последний элемент из стека*
    удаляем последний элемент из стека;
соединяем массив точек $path$ ломаной и выводим на экран;

Программируем на Maxima (листинг № 1.1):

Реализация L-системы и черепашьей графики (без символа b)

```
load(stringproc);                          /* загрузка пакета работы со строками! */
axiom: "F++F++F"$                          /* начальное слово-аксиома */
newF: "F-F++F-F"$                          /* порождающее правило для F */
newb: "b"$                                 /* порождающее правило для b */
level: 4$                                  /* количество итераций */

W: axiom$                                  /* реализация L-системы, результат - слово W */
thru level do
    (T: "",
    for j:1 thru slength(W) do
       (if sequal(charat(W,j),"+") then T:simplode([T,"+"]),
        if sequal(charat(W,j),"-") then T:simplode([T,"-"]),
        if sequal(charat(W,j),"[") then T:simplode([T,"["]),
        if sequal(charat(W,j),"]") then T:simplode([T,"]"]),
        if sequal(charat(W,j),"F") then T:simplode([T,newF]),
        if sequal(charat(W,j),"b") then T:simplode([T,newb])
        ),
    W:T)$
      /* реализация черепашьей графики: интерпретация слова, результат - чертеж */
                                           /* символ b здесь не реализован! */
pict(word,theta):=(
    alpha: 0,                              /* угол начального направления */
    [x0,y0]: [0,0],                        /* начальное положение */
    path: [[x0,y0]],                       /* массив последовательных точек пути черепашки */
    stack: [],                             /* стек ветвлений, пока пустой */

    for j:1 thru slength(word) do
       (if sequal(charat(word,j),"+") then alpha: alpha+theta,
        if sequal(charat(word,j),"-") then alpha: alpha-theta,
        if sequal(charat(word,j),"F") then (
```

```
31                                      x: x0+float(cos(alpha)),
32                                      y: y0+float(sin(alpha)),
33                                      path: endcons([x,y],path),
34                                      x0: x,
35                                      y0: y
36                                      ),
37          if sequal(charat(word,j),"[") then stack: endcons([x0,y0,alpha],stack),
38          if sequal(charat(word,j),"]") then (   /* вызов последнего эл-та стека */
39              [x0,y0,alpha]:last(stack),
40              stack:rest(stack,-1)                        /* его удаление из стека */
41                                              )
42          ),
43       plot2d([discrete,path],[box,false],[plot_format,xmaxima],[xlabel,""],[ylabel,""])
44   )$
45
46   pict(W,%pi/3)$                                /* вызов команды построения чертежа */
```

Комментарий.

Алгоритм состоит из двух частей: в строках 7–18 происходит построение кодового слова, а в строках 21–46 — его интерпретация и графическая реализация.

Напомним некоторые сведения о синтаксисе и командах в Maxima (дальнейшие сведения можно найти непосредственно в руководстве Maxima).

`load(`*имя пакета*`)` — загрузка дополнительного пакета команд;
`a:b` — присваивание переменной `a` значения `b`;
`;` — завершает одиночную команду (ставится в Maxima по умолчанию);
`$` — завершает одиночную команду, запрещая вывод на экран результата выполнения команды;
`/* комментарий */` — запись комментария;
"*символы в двойных кавычках*"— обозначение строк;
`%pi` — константы (например π) начинаются со знака `%`;
`[x1,x2,...]` — массив значений `x1,x2,...`;
`a[i]` — i-й элемент массива `a`;
`thru l do` — упрощенная конструкция цикла `for`: повторять цикл `l` раз; пропущена инициализация переменной — такое сокращение удобно, когда сама переменная цикла не используется внутри него (таким образом, ее можно не определять);
`length(a)` — количество элементов в массиве `a`;
`slength(w)` — количество символов в строке `w`;
`sequal(w1,w2)` — проверка равенства двух слов `w1` и `w2`;
`charat(w1,k)` — `k`-й символ в строке `w1`;
`simplode(w1,w2)` — конкатенация двух строк `w1` и `w2`;
`endcons(elem,list)` — добавить элемент `elem` к концу массива `list`. Аналог этой операции для массивов:
`append(l1,l2)` — присоединение массива `l2` к концу массива `l1`;
`last(a)` — последний элемент массива `a`;
`rest(a,n)` — удалить из массива `a` (а также из матрицы или другого выражения) `n` первых элементов, если `n`> 0, или `n` последних элементов, если `n`< 0.

`plot2d([discrete,[[x1,y1],[x2,y2],...]])` — график дискретного множества точек;

параметры графики команды plot:

`[box,false]` — скрыть обрамляющий графику прямоугольник и оси;
`[plot_format, xmaxima]` — выбор графического формата (`gnuplot` по умолчанию);
`[xlabel,w]` — пометить ось x строкой `w` (пустая строка `""` для скрытия этой пометки, нужно в формате xmaxima).
`[gnuplot_preamble, "set size ratio -1"]` — управление соотношением сторон графики;
Более удобным для отображения графики является пакет **draw**[3], тогда строчку `46` можно заменить командами

```
load(draw)$
draw2d(points(path), point_type=dot, points_joined=true, axis_left=false,
       axis_right=false, axis_top=false,axis_bottom=false, xtics=false,ytics
       =false, user_preamble="set size ratio -1")$
```

Отображение дискретного массива точек задается командой
`draw2d(points([[x1,y1],[x2,y2],...]))`

параметры графики команды draw:

`point_type=dot` — отображать точки «обычными» точками;
`points_joined=true` — соединять последовательные точки отрезками;
`axis_left=false,axis_right=false,axis_top=false,axis_bottom=false` — не отображать соответствующие оси координат;
`xtics=false,ytics=false` — не отображать засечки на осях координат;
`user_preamble="set size ratio -1"` — установить масштаб сторон 1 : 1. Эта команда эквивалентна опции
`proportional_axes=xy`
Заметим, гораздо проще собрать все однотипные опции в пользовательский массив:
`noframe:[axis_left=false,axis_right=false,axis_top=false,axis_bottom=false, xtics=false,ytics=false,user_preamble="set size ratio -1"]$`
и использовать его как опцию команды: `draw2d(...,points(...),noframe)`.
Также, часто используемые опции графики можно сохранить глобально командой `set_draw_defaults(`*опции*`)`. Вернуться к опциям по умолчанию можно «пустой» командой `set_draw_defaults()`.
Заметим, что опции `draw` могут быть глобальными (тогда в команде `draw` они стоят после графического объекта) и локальными (тогда их нужно писать перед графическим объектом, поскольку, если таковых несколько, у каждого могут быть свои опции).
Пользователи wxMaxima могут воспользоваться модифицированными командами `wxplot2d` и `wxdraw2d` для вывода графики в рабочую область приложения, а не в отдельное окно (попробуйте!).
Поскольку итеративные вычисления часто являются ресурсоёмкими, полезно оценивать количество итераций по статистике затраченного времени;

измерение затраченного времени:

`showtime: true` — включает слежение за временем выполнения команд;

`timer(f)` — помещает функцию `f` в таймер на слежение;
`untimer(f)` — убирает функцию `f` из таймера;
`timer_info()` — вызывает статистику вызова функций, помещенных в таймер;
`elapsed_run_time()` — количество секунд в данной сессии Maxima, затраченных на подсчёты.
Также отметим, для отлаживания программы полезна команда
`kill(all)` — стирает все пользовательские определения функций и переменных;
`kill(z)` — стирает определение функции или переменной с именем `z`.

Помните, при осуществлении вывода графики на экран процесс вычислений в Maxima не завершается, пока не закрыты графические окна gnuplot, xmaxima.

1.2 Типы и обобщения L-систем

Приведем общую классификацию L-систем. L-системы принято делить по двум признакам:

- Детерминистические (D) либо стохастические (случайные);
- Контекстно-зависимые ($1, 2$) или контекстно-свободные (0).

В соответствии с этим делением можно классифицировать L-системы следующим образом:

- **D0L-системы.** Детерминистические контекстно-свободные L-системы;
- **D1L-системы.** Детерминистические контекстно-зависимые L-системы;
- **0L-системы.** Стохастические контекстно-свободные системы;
- **1L-системы.** Стохастические контекстно-зависимые L-системы (с односторонней зависимостью);
- **2L-системы.** Стохастические контекстно-зависимые L-системы (с двухсторонней (правой и левой) зависимостью);
- **Параметрические L-системы.**
- **Временные L-системы.**
- **Ветвящиеся L-системы.**

Еще одна классификация L-систем:

- **D0L-системы.** Детерминистические контекстно-свободные системы;
- **Контекстно-зависимые L-системы.** Замена символов происходит в зависимости от их ближайших соседей;
- **Ветвящиеся системы.** Возможность запоминать состояние черепашки и возвращаться к этим состояниям, чертить ветвящиеся структуры;
- **Стохастические системы.** Рандомизация интерпретационных правил, а также самого процесса построения слова (см. задачи 1.1 и 1.2);

[3] Существует надстройка над этим пакетом **qdraw**, обеспечивающая более быстрый и простой доступ к командам и опциям, см. [7e].

- **Временны́е системы.** Эволюция порождающих правил во времени;
- **Параметрические L-системы.** С каждым символом слова связан набор параметров, замена символа происходит только при выполнении некоторых логических условий на эти параметры.

1.2.1 Ветвящиеся L-системы

Для реализации структур, похожих на ветви деревьев, используются парные команды ветвления [и] и массив-стек положений ветвления:

$$\begin{bmatrix} x_1 & y_1 & \alpha_1 \\ \vdots & \vdots & \vdots \\ x_N & y_N & \alpha_N \end{bmatrix}.$$

Каждый раз при открытии ветви символом [в конец этого стека записывается текущее состояние (x_N, y_N, α_N), а при закрытии ветви символом] черепашка возвращается на позицию перед ветвлением и из этого стека удаляется последний элемент. Стек необходим, потому что ветвления могут быть вложенными друг в друга, например `F[+b+[F+F]-b+F]+F`. Примеры ветвящихся L-систем см. в Задачах 1.3, 1.5.

1.2.2 Контекстно-зависимые L-системы

Каждый символ A может меняться в зависимости от своего *контекста* (окружения). В $2L$-системах контекст двухсторонний: от окружающих его слева и справа символов (A_l и A_r) зависит порождающее правило:

$$A_l \ < \ A \ > \ A_r \longmapsto newA.$$

Также используют дополнительные символы 0 и 1, отвечающие за контекст, но не интерпретируемые черепашкой, и указывается, какие символы игнорируются при нахождении контекста: $ignore : + - F$ означает, что, например, в слове

$$F\underline{0}FF\underline{\underline{1}} + F\underline{0} - 1F$$

контекстом символа $A = 1$ в середине слова являются символы $A_l = 0$ и $A_r = 0$. Запись $* < A > * \longmapsto newA$ означает, что символ A меняется независимо от контекста. Как правило, для символа F правила не задаются. Примеры таких $2L$-систем см. в Таблице 1.3.

Если $2L$-система содержит ветвления, то задание контекста несколько усложняется: близкие символы могут быть разделены длинной последовательностью ветвей. В одном из самых простых случаев дочерние ветви не пересекаются с контекстом материнских ветвей. Например, в слове

$$ABC\underline{D}[EF][\underline{G}[HI[JKL]M]\underline{N}OPQ]$$

левым контекстом символа G является D, а правым контекстом — символ N.

В $1L$-системах контекст односторонний: $A_l \ < \ A \longmapsto newA$ или $A \ > \ A_r \longmapsto newA$. Обобщением этих случаев являются IL-системы ((k,l)-системы), в которых контекст образуют k символов слева и l символов справа от A.

1.2.3 Стохастические L-системы

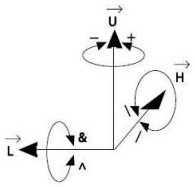
Случайность в построении L-систем можно вводить на двух этапах (см. Задачи 1.1 и 1.2):

- При построении слова вместо одного правила для символа F используется N правил $newF_1, newF_2, \ldots, newF_N$, выбираемых на каждом шаге итерационного процесса с соответствующими вероятностями $p_1, p_2, \ldots, p_N$ ($\sum_{n=1}^{N} p_n = 1$). Обозначение:

$$p_1 \colon F \overset{p_1}{\longmapsto} newF_1,$$
$$\vdots$$
$$p_N \colon F \overset{p_N}{\longmapsto} newF_N.$$

- При интерпретации слова можно менять длину шага черепашки и угол поворота ее головы случайным образом в соответствии с некоторыми распределениями вероятностей.

1.2.4 3D L-системы



Кроме плоского случая ($2D$), рассматриваются и $3D$ реализации L-систем. Ориентация черепашки в пространстве задается ортонормированным правым базисом $\overrightarrow{H}$, $\overrightarrow{L}$ и $\overrightarrow{U}$, в котором $\overrightarrow{H}$ (heading) — направление *взгляда* черепашки, $\overrightarrow{L}$ (left)— направление *налево*, $\overrightarrow{U}$ (up) — *вверх*. Вместо одной пары команд поворота на постоянный угол вводятся три: $+$, $-$ (повороты вокруг оси $\overrightarrow{U}$), ^, & (повороты вокруг оси $\overrightarrow{L}$), /, \ (повороты вокруг оси $\overrightarrow{H}$).

1.2.5 L-системы с двумя символами шага

Для удобства построения некоторых систем вместо одно символа шага F полезно ввести два символа F_x и F_y. Например:

$$axiom = F_x,$$
$$F_x \longmapsto F_x + F_y+,$$
$$F_y \longmapsto -F_x - F_y.$$

Однако тот же результат можно получить, если символ шага F будет один, но добавятся еще два не интерпретируемых черепашкой символа X и Y:

$$axiom = FX,$$
$$X \longmapsto X + YF+,$$
$$Y \longmapsto -FX - Y.$$

Примеры таких L-систем см. в Таблице 1.2. См. также Задачу 1.9.

Задача 1.1 Реализовать алгоритм L-системы со случайным выбором порождающих правил (см. параграф 1.2.3). Испытать алгоритм на примере: `axiom: F`,

`newF1: F[+F]F[-F]F`, `newF2: F[+F]F`, `newF3: F[-F]F`, `p1: 0.33`, `p2: 0.33`, `p3: 0.34`.

Задача 1.2 Реализовать алгоритм рандомизированной L-системы. Угол поворота и длина шага могут быть случайными числами. Подберите их распределение так, чтобы добиться наилучшего (эстетического) результата.

Задача 1.3 Реализовать алгоритм L-системы с символом b, с помощью которого можно рисовать несвязные множества.

Задача 1.4 При выводе на экран Maxima автоматически подгоняет размеры изображения так, чтобы оно полностью уместилось. Заметьте, что, вообще говоря, длина интерпретируемого слова постоянно и неограниченно растет. Соответственно экспоненциально увеличиваются и линейные размеры графики. Как сделать процедуру нормировки так, чтобы картинка всегда была расположена в заданной области (например, квадрате $[-2,2]\times[-2,2]$)?

Задача 1.5 Построить фигуры, задающиеся правилами из Таблицы 1.1.

Задача 1.6 Нарисовать результат слова, полученного после двух итераций следующих правил: `axiom: F`, `newF: FF-[F]+[F]`, `alpha: %pi/2`, `theta: %pi/4`.

Задача 1.7 Найти правила, порождающие ковер Серпинского и пыль Кантора.

Задача 1.8 Записать правила `newF`, порождающие при первой итерации фигуры на рисунке 1.1 из аксиомы `F` (начальную и конечную точку движения черепашки можно выбрать любым образом).

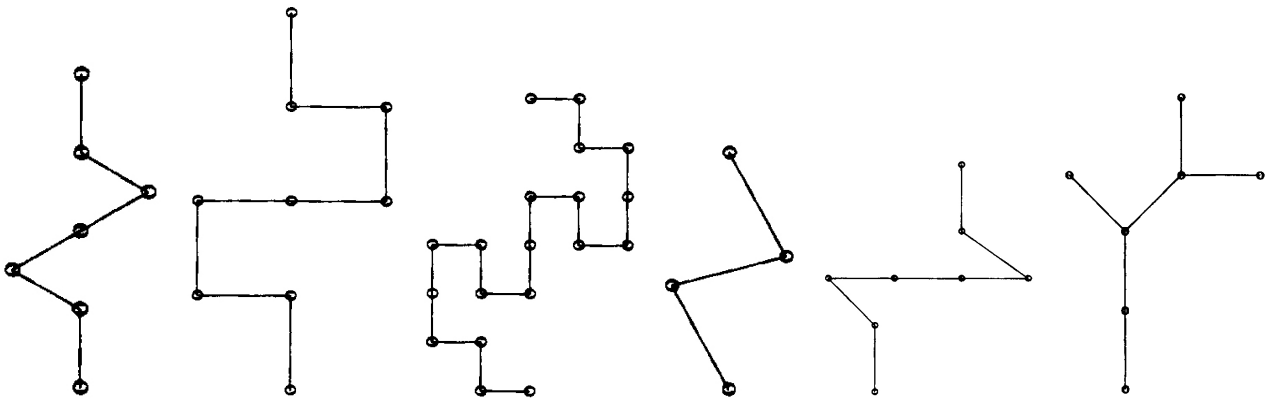


Рис. 1.1: Первые итерации различных L-систем.

Задача 1.9 Реализовать алгоритм L-системы с двумя символами шага (см. параграф 1.2.5) и опробовать его на примерах из Таблицы 1.2. Всегда ли можно перейти от символов F_x и F_y к символам F, X и Y?

Задача 1.10 Реализовать алгоритм $2L$-системы (см. параграф 1.2.2) и опробовать его на примерах из Таблицы 1.3.

Задача 1.11 Реализовать алгоритм $3D$ L-системы (см. параграф 1.2.4) и опробовать его на следующем примере: `axiom: &-XY-XY-XY-XY`, `newX: XY+XY-XY-XYXY+XY+XY-X`, `newY: F^F^F^F^F`, `theta: %pi/2`.

Куст
`axiom: F`
`newF: -F+F+[+F-F-]-[-F+F+F]`
`theta: %pi/8`
	Сорняк
	`axiom: F`
	`newF: F[+F]F[-F]F`
	`theta: %pi/7`

Кривая Пеано
`axiom: F`
`newF: F-F+F+F+F-F-F-F+F`
`alpha: %pi/4`
`theta: %pi/2`
	Цветок
	`axiom: F[+F+F][-F-F][++F][--F]F`
	`newF: FF[++F][+F][F][-F][--F]`
	`alpha: %pi/2`
	`theta: %pi/16`

Мозаика (острова и озера)
`axiom: F-F-F-F`
`newF: F-b++FF-F-FF-Fb-FF+b-FF+F+FF+Fb+FFF`
`newb: bbbbbb`
`theta: %pi/2`
	Цепочка
	`axiom: F+F+F+F`
	`newF: F+b-F-FFF+F+b-F`
	`newb: bbb`
	`theta: %pi/2`

Снежинка
`axiom: [F]+[F]+[F]+[F]+[F]+[F]`
`newF: F[+FF][-FF]FF[+F][-F]FF`
`theta: %pi/3`
	Остров
	`axiom: F+F+F+F`
	`newF: F+F-F-FFF+F+F-F`
	`theta: %pi/2`

Снежинка Коха
`axiom: [F]+[F]+[F]+[F]+[F]+[F]`
`newF: F[+FF][-FF]FF[+F][-F]FF`
`theta: %pi/3`
	Треуголка
	`axiom: F`
	`newF: F+F-F-F+F`
	`theta: %pi/2`

Остров Коха
`axiom: F-F-F-F`
`newF: F-F+F+FF-F-F+F`
`theta: %pi/2`
	Цепочка еще
	`axiom: F-F-F-F`
	`newF: FF-F-F-F-F-F+F`
	`theta: %pi/2`

Коврик
`axiom: F-F-F-F`
`newF: FF-F-F-F-FF`
`theta: %pi/2`
	Кирпичи
	`axiom: F-F-F-F`
	`newF: FF-F+F-F-FF`
	`theta: %pi/2`

Кристаллы
`axiom: F-F-F-F`
`newF: FF-F-F-F`
`theta: %pi/2`
	Заполненный остров
	`axiom: F-F-F-F`
	`newF: F-FF--F-F`
	`theta: %pi/2`

`axiom: AAAA,` `theta: %pi/12`
`newA: X+X+X+X+X+X+`
`newX: [F+F+F+F[--X-Y]+++++F++++++++F-F-F-F]`
`newY: [F+F+F+F[--Y]  +++++F++++++++F-F-F-F]`
	Спиральное покрытие

Таблица 1.1: Правила для черепашьей графики.

Кривая Серпинского	**Кривая Гильберта**	**Кривая Пеано**
axiom: F+X+F+XF	axiom: X	axiom: X
newF: F	newF: F	newF: F
newX: XF-F+F-XF+F+XF-F+F-X	newX: -YF+XFX+FY-	newX: XFYF+F+YFXFY-F-XFYFX
alpha: %pi/4	newY: +XF-YFY-FX+	newY: YFXFY-F-XFYFX+F+YFXFY
theta: %pi/2	theta: %pi/2	theta: %pi/2
Кривая Госпера	**Дракон Хартера-Хайтвея**	**Наконечник Серпинского**
axiom: XF	axiom: FX	axiom: YF
newF: F	newF: F	newF: F
newX: X+YF++YF-FX--FXFX-YF+	newX: X+YF+	newX: YF+XF+Y
newY: -FX+YFYF++YF+FX--FX-Y	newY: -FX-Y	newY: XF-YF-X
theta: %pi/3	theta: %pi/2	theta: %pi/3

Таблица 1.2: Правила для черепашьей графики с двумя символами шага.

Куст 1	**Куст 2**	**Куст 3**
$\theta = 22.5^\circ$	$\theta = 22.5^\circ$	$\theta = 25.75^\circ$
ignore: +-F	ignore: +-F	ignore: +-F
axiom: F1F1F1	axiom: F1F1F1	axiom: F1F1F1
0 < 0 > 0 ⟼ 0	0 < 0 > 0 ⟼ 1	0 < 0 > 0 ⟼ 0
0 < 0 > 1 ⟼ 1[+F1F1]	0 < 0 > 1 ⟼ 1[-F1F1]	0 < 0 > 1 ⟼ 1
0 < 1 > 0 ⟼ 1	0 < 1 > 0 ⟼ 1	0 < 1 > 0 ⟼ 0
0 < 1 > 1 ⟼ 1	0 < 1 > 1 ⟼ 1	0 < 1 > 1 ⟼ 1[+F1F1]
1 < 0 > 0 ⟼ 0	1 < 0 > 0 ⟼ 0	1 < 0 > 0 ⟼ 0
1 < 0 > 1 ⟼ 1F1	1 < 0 > 1 ⟼ 1F1	1 < 0 > 1 ⟼ 1F1
1 < 1 > 0 ⟼ 0	1 < 1 > 0 ⟼ 1	1 < 1 > 0 ⟼ 0
1 < 1 > 1 ⟼ 0	1 < 1 > 1 ⟼ 0	1 < 1 > 1 ⟼ 0
∗ < + > ∗ ⟼ -	∗ < + > ∗ ⟼ -	∗ < + > ∗ ⟼ -
∗ < - > ∗ ⟼ +	∗ < - > ∗ ⟼ +	∗ < - > ∗ ⟼ +
Куст 4	**Куст 5**	**Куст 6**
$\theta = 25.75^\circ$	$\theta = 22.5^\circ$	
ignore: +-F	ignore: +-F	
axiom: F0F1F1	axiom: F1F1F1	
0 < 0 > 0 ⟼ 1	0 < 0 > 0 ⟼ 0	
0 < 0 > 1 ⟼ 0	0 < 0 > 1 ⟼ 1[-F1F1]	
0 < 1 > 0 ⟼ 0	0 < 1 > 0 ⟼ 1	Ваш
0 < 1 > 1 ⟼ 1F1	0 < 1 > 1 ⟼ 1	пример
1 < 0 > 0 ⟼ 1	1 < 0 > 0 ⟼ 0	
1 < 0 > 1 ⟼ 1[+F1F1]	1 < 0 > 1 ⟼ 1F1	
1 < 1 > 0 ⟼ 1	1 < 1 > 0 ⟼ 1	
1 < 1 > 1 ⟼ 0	1 < 1 > 1 ⟼ 0	
∗ < + > ∗ ⟼ -	∗ < + > ∗ ⟼ -	
∗ < - > ∗ ⟼ +	∗ < - > ∗ ⟼ +	

Таблица 1.3: Правила для $2L$-системы (рекомендуется 30 итераций).

Задача 1.12 Реализуйте простейшую параметрическую L-систему: $axiom = A(1)$, $A(s) \longmapsto F(s)[+A(s/1.456)][-A(s/1.456)]$ (здесь команды $F(s)$ и $A(s)$ чертят отрезок длины s). Откуда взялось число 1.456? Что будет, если менять угол $\theta \in [0, \pi]$? Постройте серию картинок при изменении угла θ и сделайте из них анимацию (см. с. 47).

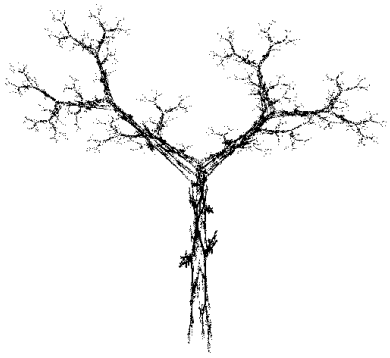
Задача 1.13 Придумать и реализовать на компьютере три новые L-системы, результатом работы которых были бы ваши собственные варианты следующих фигур:

- снежинка или остров (с границей без разрывов);
- мозаика или острова (с разрывной границей);
- куст или сорняк.

Задача 1.14 Самостоятельно рассмотреть какое-либо приложение L-систем, предложенных в книгах [3], [4]. Среди прочего там можно выделить:

- Фрактальную музыку на основе L-систем (в настоящее время в Maxima не поддерживается работа со звуком, однако можно, например, воспользоваться пакетом **sound.lisp** автора Mario Rodriguez Riotorto [11e]);
- Построение клеточных структур на основе L-систем планарных отображений;
- Интерполяция построенной ломаной сплайнами;
- Узоры-«колемы», покрытия плоскости.

Рекомендуемая литература: [1, с. 23], [3], [4]; [14e], [11e], [15e], [16e], [17e], [18e], [19e], [20e], [7e].



Глава 2

СИСТЕМЫ ИТЕРИРОВАННЫХ ФУНКЦИЙ

Восходит солнце, и заходит солнце, и спешит к месту своему, где оно восходит. Идет ветер к югу, и переходит к северу, кружится, кружится на ходу своем, и возвращается ветер на круги свои. Все реки текут в море, но море не переполняется: к тому месту, откуда реки текут, они возвращаются, чтобы опять течь.

Екклезиаст

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:

[1] Вообще говоря, все сказанное здесь верно для полных (а также для компактных) метрических пространств. Но для моделирования фракталов нам понадобятся только евклидовы пространства малых измерений.

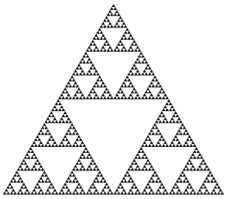
$$A = \lim_{n\to\infty} (f_1 \cup \ldots \cup f_m)^{(n)}(C), \tag{2.1}$$

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.

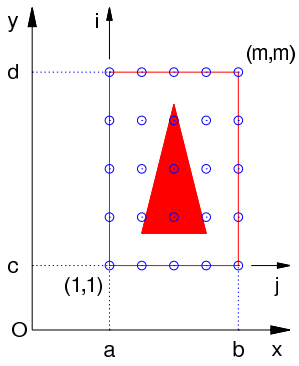
Для построения фрактальных множеств с помощью СИФ пользуются детерминированным и рандомизированным (предпочтительнее) алгоритмами.

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 $F(C) = (f_1 \cup \ldots \cup f_m)(C) = \bigcup_{j=1}^{m} f_j(C)$ 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

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



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

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

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: $f(A) = \bigcup_{(x,y)\in A} \{f(x, y)\} \approx \bigcup_{(x_i,y_j)\in A} \{f(x_i, y_j)\}$. 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*

Программируем на Maxima (листинг № 2.1):

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*

1: $path = \emptyset$; ✎ *инициализация пустого массива*
2: $x_0 = 0$; ✎ *начальная точка итераций*
3: **повторять** $skip$ **раз**
4: $\quad l =$ случайное число от 1 до n;
5: $\quad x_0 = f_l(x_0)$;
6: **повторять** $points$ **раз**
7: $\quad l =$ случайное число от 1 до n;
8: $\quad x_0 = f_l(x_0)$;
9: $\quad path = path \cup x_0$

[2] В Maxima это позволяет делать команда `multi_plot`, но она не работает в Windows.

Программируем на Maxima (листинг № 2.2):

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.

Задача 2.1 Оцените, во сколько раз скорость рандомизированного алгоритма больше скорости детерминированного (оцените время или количество операций).

Задача 2.2 При построении фракталов рандомизированным Алгоритмом 2.2 можно заметить, что фрактал заполняется точками неоднородно (потому что у каждого преобразования f_j свой коэффициент сжатия s_j). Чтобы покрыть аттрактор равномерно, надо выбирать каждое из отображений f_j не с вероятностью $\frac{1}{n}$, а с вероятностью $p_j = \frac{s_j^2}{\sum_{j=1}^{n} s_j^2}$

(вероятности надо брать пропорциональными коэффициентам сжатия площадей). Если все отображения $f_1, \ldots, f_n$ в СИФ являются аффинными: $f_j\left(\begin{bmatrix}x\\y\end{bmatrix}\right) = \begin{bmatrix}A_j & B_j\\C_j & D_j\end{bmatrix}\cdot\begin{bmatrix}x\\y\end{bmatrix} + \begin{bmatrix}E_j\\F_j\end{bmatrix}$, как во многих примерах здесь, то $s_j^2 = |\det \begin{bmatrix}A_j & B_j\\C_j & D_j\end{bmatrix}|$. Реализуйте рандомизированный алгоритм, в котором на входе также подаются значения s_j. В частности, автоматизируйте подсчёт в случае аффинных отображений. Как искать s_j в случае произвольных дифференцируемых отображений?

Задача 2.3 Изменить алгоритм так, чтобы цвет выводимой на экран точки зависел от того, каким из преобразований f_j она получена из предыдущей точки. Одно из возможных решений этой задачи — сохранять в массив `path` вместе с координатами точки еще и цвет (строчка 20):

```
path:append(path,[color=[blue, red, green, magenta][l],points([p])])$
```

(в данном примере цвет выбирается из массива четырех цветов в соответствии с номером отображения). Строчку 24 заменим на

```
wxdraw2d(point_type=dot,path,user_preamble="set size ratio -1")$
```

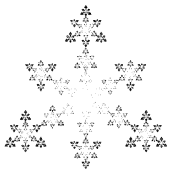
Задача 2.4 Построить аттракторы, задающиеся системами аффинных итерированных отображений. Коэффициенты формулы $f_j\left(\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right]\right) = \left[\begin{smallmatrix}A_j & B_j\\C_j & D_j\end{smallmatrix}\right]\cdot\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right] + \left[\begin{smallmatrix}E_j\\F_j\end{smallmatrix}\right]$ 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]]$
```

Рекомендуемая литература: [1, с. 96], [3], [4]; [14e], [11e], [15e], [16e], [17e], [18e], [19e], [20e].

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

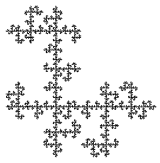


Папоротник

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

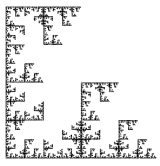
Ковер 1

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

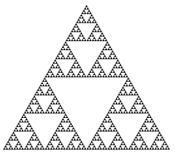


Лист

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

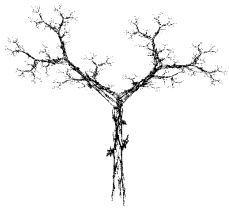
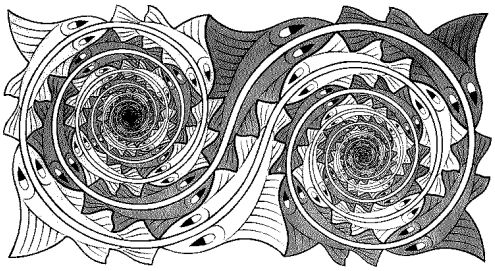


Таблица 2.1: Коэффициенты для аффинных СИФ.



Глава 3

ВЫЧИСЛЕНИЕ РАЗМЕРНОСТИ МИНКОВСКОГО

В 3/9 царстве, в 3/10 государстве все были помешаны на дробях.

Анекдот

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].

[1] Приведённые здесь названия часто путают, поэтому необходимо уточнять, о какой размерности идет речь и как ее подсчитывать.

[2] Существуют и физические способы измерения фрактальной размерности, см. [6, 7].

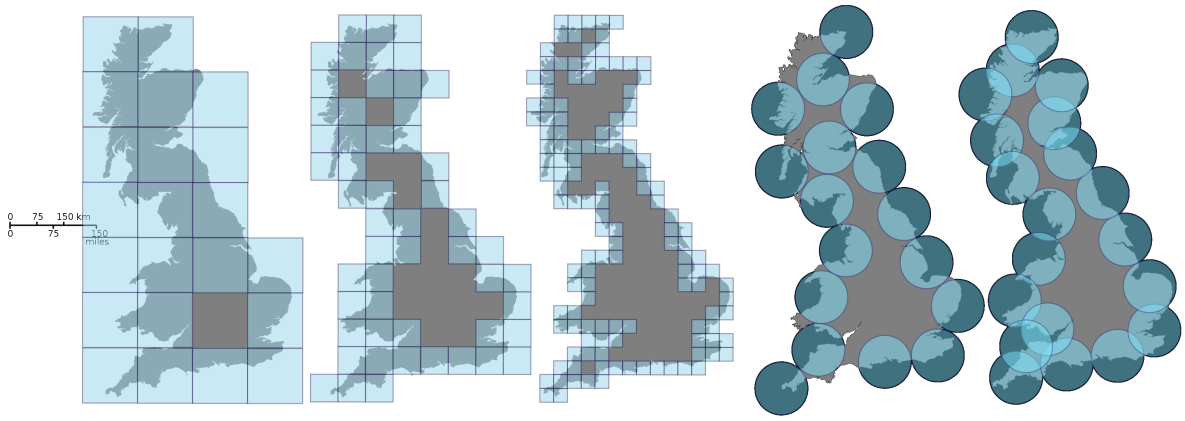


Рис. 3.1: Подсчёт размерности Минковского **границы** побережья Великобритании способом ③ (три рисунка слева). Способы подсчёта ⑤ и ① (два рисунка справа).

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

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

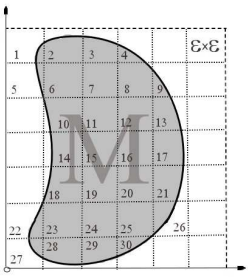
где m_i — целые числа. На плоскости $\mathbb{R}^2$ ε-сетка — это разбиение плоскости квадратами со сторонами ε, параллельными осям абсцисс и ординат.

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 рассматри-

[3] Здесь основание $\log(x)$ несущественно. В Maxima функция `log(x)` выдаёт $\ln(x)$.
[4] Более подробное рассуждение на эту тему см. в [8].
[5] Графики «лог–лог» в Maxima можно строить с помощью опций `[logx],[logy]` в команде `plot`, `logx=true,logy=true` в команде `draw`.

вают 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 $y = Ax + B$;

Программируем на Maxima (листинг № 3.1):

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 `$`).

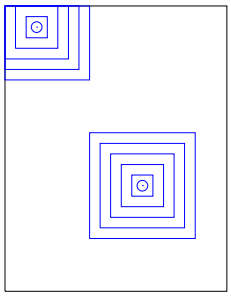
`rhs(x=y)` — возвращает правую часть `y` равенства `x=y` (right-hand-side). Очень полезная команда для «выуживания» ответов из массивов ответов (ее аналог для левых частей равенства — `lhs()`).

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 $y = Ax + B$;

Программируем на Maxima (листинг № 3.2):

Оценка размерности Минковского. Точечный метод

```
I: 80$                                  /* размеры матрицы E; матрица известна заранее */
J: 100$
scales:[1,2,3,4,8]$
Q:length(scales)$
P: zeromatrix((2*scales[Q]+1)^2,Q)$

path:[]$

for i:1 thru I do
   for j: 1 thru J do
      if E[i,j]>0 then
         for L:1 thru Q do
            (m:sum(
                  sum(E[r,s],r,max(i-scales[L],1),min(i+scales[L],I)),
                  s,max(j-scales[L],1),min(j+scales[L],J)),
            P[m,L]:P[m,L]+1
            )$

for L:1 thru Q do
(N[L]:sum(P[m,L]/m,m,1,(2*scales[Q]+1)^2),
path:endcons(float([log(2*scales[L]+1),log(N[L])]),path)
)$

load (lsquares)$
M: apply(matrix,path)$
lse:lsquares_estimates (M, [x,y], y= A*x+B, [A,B]),float$
[a,b]:map(rhs,[lse[1][1],lse[1][2]]);
load(draw)$
list:makelist(path[L][1],L,1,Q)$
[xmin,xmax]:[lmin(list)-1/2,lmax(list)+1/2]$
wxdraw2d(points(path),explicit(a*x+b,x,xmin,xmax))$
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 $y = y(\overrightarrow{x})$, $\overrightarrow{x} = \{x_1, \ldots, x_n\}$, заданная выборкой значений в m n-мерных точках $y(\overrightarrow{x}_i^0) = y_i^0$, $\overrightarrow{x} \in \mathbb{R}^n$. Аппроксимируем ее с помощью модели (функции) $y = F(\overrightarrow{x}, \overrightarrow{\alpha})$, 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:

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

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

В случае гладкого отображения F задача нахождения параметров решается поиском критической точки этого функционала: решается уравнение $\operatorname{grad} \mathcal{L}(\overrightarrow{\alpha}) = \overrightarrow{0}$. В случае же линейного отображения $y = F(\overrightarrow{x}, \overrightarrow{\alpha}) = \sum_{i=1}^{n} \alpha_i x_i + \alpha_{n+1}$ 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: $M_i = \{x_1, \ldots, x_n\}_i$.

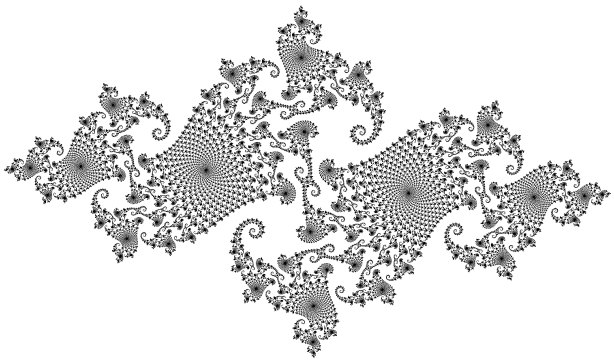
Задача 3.1 Испытайте алгоритм оценки размерности Минковского на фракталах, полученных в предыдущем пункте, особенно на тех, у которых размерность можно под-

считать аналитически.

Задача 3.2 Покажите, что приведённый алгоритм не даёт возможность оценить размерность Минковского точнее третьего знака после запятой. Для этого аппроксимируйте набор точек (x_i, y_i) различными способами: по всем точкам, по первым трем, по последним трем... Даже если на графике все точки расположены визуально на одной линии, отличия таких аппроксимаций могут быть существенны.

Задача 3.3 При оценке размерности Минковского плоского изображения Алгоритмами 3.1 и 3.2 результат может оказаться намного выйти из интервала $[0, 2]$. Подумайте, чем это можно объяснить.

Рекомендуемая литература: [1, с. 127], [8], [5], [6], [7].



Глава 4

КОМПЛЕКСНАЯ ДИНАМИКА

Комплексные числа — это прекрасное и чудесное убежище божественного духа, почти что амфибия бытия с небытием.

Готфрид Вильгельм Лейбниц

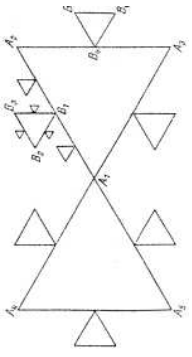
Mandelbrot Set — is the most `complex` *object in mathematics.*

Автор Неизвестен (Б. М.?)

Самым неожиданным, поражающим воображение компьютерным экспериментом, совмещающим в себе простоту алгоритма и загадочную сложность полученного результата, является построение множеств Жюлиа и Мандельброта, а также других аттракторов комплексных динамических систем, существенно связанных с особенностями движения точек внутри аттракторов под действием формирующих их отображений.

Образцом для вдохновения многих людей идеями комплексной динамики является красочная книга Х.-О. Пайтгена и П. Х. Рихтера [9].

4.1 Множества Жюлиа



Французские математики Гастон Жюлиа [10] и Пьер Фату [11] в 1917–1919 одновременно написали основополагающие статьи по итерированию функций комплексного переменного. В 1918 году Жюлиа, вдохновленный проблемой Кэли [12] (1879), описал множество, впоследствии названное его именем. Схематическая иллюстрация была сделана его учеником Хубертом Кремером [13] в 1924 году (год рождения Б. Мандельброта, рисунок справа). Однако увидеть эти множества в полной их красе стало возможным только спустя более пятидесяти лет с наступлением эры ЭВМ. Впервые грубое, но уже характерное изображение множества Жюлиа появилось в работе Роберта Брукса и Питера Мателски [14] в 1978 году.

Орбитой точки $z \in \mathbb{C}$ под действием отображения $f\colon \mathbb{C} \to \mathbb{C}$ назовем множество[1]

$$\{f^{(n)}(z)\}_{n=0}^{\infty}.$$

Множество Жюлиа для *полиномиальной*[2] функции $f\colon \mathbb{C} \to \mathbb{C}$ определяется так:

$$J(f) = \partial\{z \in \mathbb{C}\colon \lim_{n\to\infty} f^{(n)}(z) = \infty\}$$

и представляет собой границу множества точек, орбиты которых не уходят на бесконечность. Дополнение множества Жюлиа $\mathbb{C}\setminus J(f)$ называется *множеством Фату*. Наиболее хорошо исследованы множества Жюлиа для полиномиальных функций.

Визуально более эффектно *заполняющее множество Жюлиа* — множество точек, орбиты которых ограничены:

$$\mathbf{J}(f) = \{z \in \mathbb{C}\colon \exists M_z > 0 \quad \forall n \in \mathbb{N} \quad |f^{(n)}(z)| < M_z\}.$$

Рассмотрим множества Жюлиа, связанные с однопараметрическим семейством квадратичных функций[3]

$$f_c(z) = z^2 + c, \quad c \in \mathbb{C}.$$

4.1.1 Заполняющее множество Жюлиа

Для построения заполняющего множества Жюлиа $\mathbf{J}(f_c)$ полезна следующая теорема.

Теорема 4.1 (См. [1], с. 219) *Пусть* $|c| < 2$ *и для* $z \in \mathbb{C}$ *существует* $n_0 \in \mathbb{N}$ *такое, что* $|{f_c}^{(n_0)}(z)| \geq 2$, *тогда* $z \notin \mathbf{J}(f_c) \quad \left(\lim_{n\to\infty} {f_c}^{(n)}(z) = \infty\right)$.

Таким образом, чтобы построить $\mathbf{J}(f_c)$ в квадратном окне с центром (a, b) и стороной s, разделим окно на p^2 пикселей и проверим каждую точку-пиксел z на убегание ее орбиты на бесконечность. Зададимся предельным числом испытаний $ITER$. Если $|f_c^{(n)}(z)| < 2$, $n = 1, \ldots, ITER$, то будем считать, что $z \in \mathbf{J}(f_c)$.

Алгоритм 4.1: **Заполняющее множество Жюлиа для функции** $f_c = z^2 + c$

Вход: $c \in \mathbb{C}$; ✎ *комплексный параметр функции*
(a, b); ✎ *центр окна*
s; ✎ *размер окна*
p; ✎ *число пикселов в каждой стороне окна (разрешение окна)*
$ITER$; ✎ *количество итераций для испытания точки*
Выход: $path$; ✎ *массив координат точек заполняющего множества Жюлиа* $\mathbf{J}(f_c)$

1: $path = \emptyset$; ✎ *массив точек для отображения*
2: $f_c(z) = z^2 + c$; ✎ *инициализация итерируемой функции*

[1] Здесь $f^{(n)}(z) = \underbrace{f \circ \ldots \circ f}_{n \text{ раз}}(z)$.

[2] Для функций другого вида есть другие определения.

[3] Такой подход не уменьшает общности при рассмотрении квадратичного полинома $f(z) = az^2 + bz + c$, поскольку общий вид может быть сведен к нашему частному виду заменой переменной. См. Задачу 4.9.

3: **для** $m = 1, \ldots, p$
4: $x = a - \frac{s}{2} + m\frac{s}{p}$;
5: **для** $n = 1, \ldots, p$
6: $y = b - \frac{s}{2} + n\frac{s}{p}$;
7: $z = x + iy$;
8: **повторять** $ITER$ **раз**
9: $z = f_c(z)$;
10: **если** $|z| \geq 2$, **то** выход из цикла;
11: **если** $|z| < 2$, **то** $path = path \cup \{x, y\}$; ✎ *добавить точку в массив для отображения*

Программируем на Maxima (листинг № 4.1):

Заполняющее множество Жюлиа для функции $f_c = z^2 + c$

```
1   c:%i$
2   f(z):=expand(z^2+c),float$
3   [a,b]:[0,0]$
4   s:3$
5   p:500$
6   ITER:10$
7
8   path:[]$
9
10  for m:1 thru p do
11      (x:float(a-s/2+m*s/p),
12      for n:1 thru p do
13          (y:float(b-s/2+n*s/p),
14           z:x+%i*y,
15           thru ITER do
16               (z:f(z),
17               if abs(z)>=2 then return()
18               ),
19           if abs(z)<2 then path:endcons([x,y],path)
20           )
21      )$
22  load(draw)$
23  wxdraw2d(point_type=dot,points(path))$
```

Комментарий.

> Прежде всего отметим необходимость проводить вычисления с числами типа `float`, иначе Maxima оперирует простыми дробями, что замедляет вычисления в 10–100 раз. Для этого в строчках 2, 11 и 13 мы используем приведение к типу `float`. Также заметим, что использование символики комплесных чисел замедляет вычисления, но является более наглядным. Так, например, можно было бы функцию `f(z)` в строчке 2 определить иначе: `f(x,y):=[x^2-y^2+cx, 2x*y+cy]`, где $c = c_x + c_y\mathbf{i}$, а число z хранить в виде массива `[x,y]`.

Команда `expand(expression)` раскрывает скобки в `expression` и группирует подобные слагаемые, что также полезно для ускорения вычислений. Эту команду можно заменить на команду

`rectform(z)` — возвращает алгебраический вид $x+y\mathbf{i}$ комплексного числа $z = x+y\mathbf{i}$.

Замечания об использованных командах:

`%i` — обозначение мнимой единицы i;

`return()` — в данном случае — это выход из ближайшего цикла `for` (точнее — из оператора `do`);

`abs(z)` — модуль комплексного числа `z`;

Обратите внимание, что параметры `ITER` и `p` являются антагонистами: чем больше `ITER`, тем меньше точек выдержат испытание; чем больше `p`, тем плотнее сетка испытуемых точек и тем больше вероятность не промахнуться мимо точек из $\mathbf{J}(f_c)$. Варьируя эти параметры, можно добиться приемлемого изображения.

4.1.2 Множество Жюлиа. Метод сканирования границы

Идея построения множества Жюлиа как границы заполняющего множества Жюлиа ($J(f) = \partial\mathbf{J}(f)$) состоит в следующем. Как и в предыдущем алгоритме, разбив экран на p^2 точек, проверить каждую из точек на принадлежность заполняющему множеству Жюлиа (остается ли модуль орбиты точки меньше 2 в течение $ITER$ итераций). Если точка принадлежит $\mathbf{J}(f)$, проверить ее соседей (например, правого, левого, верхнего и нижнего) на предмет убегания их орбит на бесконечность (становится ли модуль точки орбиты больше или равным 2 в течение $ITER$ итераций). Если орбита хотя бы одного из четырёх соседей неограничена, то признаем центральную точку граничной, то есть принадлежащей множеству $J(f)$ — это и будет нашим критерием.

Грубая реализация этой идеи состоит в нахождении матрицы E, представляющей $\mathbf{J}(f)$ ($E_{ij} \leftrightarrow z \in J(f) \Leftrightarrow E_{ij} = 1$), предыдущим алгоритмом и затем, пробегая по этой матрице второй раз, в проверке нашего критерия для соседей каждой точки. Однако при этом необходимо хранить и обрабатывать массив из p^2 точек, что неэффективно при больших p (например, 500).

Модифицируем грубый алгоритм так, чтобы хранить в памяти лишь массив из 3 строк и p столбцов:

- вычисляем первые две строки E_1 и E_2 матрицы E множества $\mathbf{J}(f)$;
- запускаем цикл по оставшимся $p-2$ строкам, в котором:
  - вычисляем следующую строку и записываем ее в E_3;
  - проверяем точки второй строки на принадлежность множеству $J(f)$, используя наш критерий:
    $$E_{2i} \longleftrightarrow z \in J(f) \Longleftrightarrow E_{2j} = 1 \quad \& \quad E_{1i} \cdot E_{2,i-1} \cdot E_{2,i+1} \cdot E_{3i} = 0$$
  - если $z \in J(f)$, то сохраняем точку z;
  - переставляем строчки матрицы E: $E_1 \mapsto E_2$, $E_2 \mapsto E_3$.

Для простоты представления алгоритма мы проверяем только пикселы E_{ij}, у которых $2 \le i, j \le p-1$, то есть только внутреннюю область.

Алгоритм 4.2: **Множество Жюлиа для $f_c = z^2 + c$. Метод сканирования границы**

Вход: $c \in \mathbb{C}$; ✎ *комплексный параметр функции*
(a, b); ✎ *центр окна*
s; ✎ *размер окна*
p; ✎ *число пикселов в каждой стороне окна (разрешение окна)*
$ITER$; ✎ *количество итераций для испытания точки*
Выход: $path$; ✎ *массив координат точек множества Жюлиа $J(f_c)$*

$ITER = 10$; ✎ *количество итераций для испытания точки*
$path = \emptyset$; ✎ *массив точек для отображения*
$E =$ нулевая матрица размера $3 \times p$; ✎ *матрица-стек множества $\mathbf{J}(f_c)$*
$f_c(z) = z^2 + c$; ✎ *инициализация итерируемой функции*
для $n = 1, 2$
  $y = b - \frac{s}{2} + n\frac{s}{p}$;
  для $m = 1, \ldots, p$
    $x = a - \frac{s}{2} + m\frac{s}{p}$;
    $z = x + y\mathbf{i}$;
    повторять $ITER$ **раз**
      $z = f_c(z)$;
      если $|z| \geq 2$, **то** выход из цикла;
    если $|z| < 2$, **то** $E_{nm} = 1$;
для $n = 3, \ldots, p$
  $y = b - \frac{s}{2} + n\frac{s}{p}$;
  для $m = 1, \ldots, p$
    $x = a - \frac{s}{2} + m\frac{s}{p}$;
    $z = x + y\mathbf{i}$;
    повторять $ITER$ **раз**
      $z = f_c(z)$;
      если $|z| \geq 2$, **то** выход из цикла;
    если $|z| < 2$, **то** $E_{3m} = 1$;
    иначе $E_{3m} = 0$;
  для $i = 2, \ldots, p - 1$
    если $E_{2i} = 1 \quad \& \quad E_{1i} \cdot E_{2,i-1} \cdot E_{2,i+1} \cdot E_{3i} = 0$ **, то**
      $path = path \cup \{a - \frac{s}{2} + i\frac{s}{p}, b - \frac{s}{2} + (n-1)\frac{s}{p}\}$
  $E_1 = E_2, E_2 = E_3$; ✎ *меняем первую строку матрицы на вторую, вторую на третью*

Программируем на Maxima (листинг № 4.2):

Множество Жюлиа для $f_c = z^2 + c$. Метод сканирования границы

```
c:%i*0.25+0.52,float$
f(z):=expand(z^2+c),float$
[a,b]:[0,0]$
s:3$
p:300$
ITER:10$

path:[]$
E:zeromatrix(3,p)$
                                        /* Заполняем первые два ряда матрицы E */
for n:1 thru 2 do
   (y:float(b-s/2+n*s/p),
   for m:1 thru p do
      (x:float(a-s/2+m*s/p),
      z:x+%i*y,
      thru ITER do
         (z:f(z),
         if abs(z)>=2 then return()
         ),
      if abs(z)<2 then E[n][m]:1
      )
   )$
                                                  /* Заполняем третий ряд матрицы E */
for n:3 thru p do
   (y:float(b-s/2+n*s/p),
   for m:1 thru p do
      (x:float(a-s/2+m*s/p),
       z:x+%i*y,
       thru ITER do
         (z:f(z),
         if abs(z)>=2 then return()
         ),
       if abs(z)<2 then E[3][m]:1 else E[3][m]:0
      ),   /* Проверяем, обладают ли пикселы второго ряда убегающими соседями */
   for i:2 thru p-1 do
      if E[2][i]=1 and E[1][i]*E[2][i-1]*E[2][i+1]*E[3][i]=0
         then path:endcons(float([a-s/2+i*s/p,b-s/2+(n-1)*s/p]),path),
   E[1]:E[2],                                          /* Меняем строки матрицы E */
   E[2]:E[3]
)$                                /* Показываем общее количество построенных точек */
print("Amount of points:",length(path))$

load(draw)$
wxdraw2d(point_type=dot,points(path))$
```

4.1.3 Множество Жюлиа. Метод обратных итераций

Этот алгоритм построения множества Жюлиа основан на следующих теоретических понятиях. Будем считать, что f — полином степени $n \geq 2$.

Точка $z \in \mathbb{C}$ называется *периодической с периодом* $p \in \mathbb{N}$ (не обязательно наименьшим), если $f^{(p)}(z) = z$. Такая точка называется:
— *сверхпритягивающей*, если $(f^{(p)})' = 0$;
— *притягивающей*, если $|(f^{(p)})'| < 1$;
— *нейтральной*, если $|(f^{(p)})'| = 1$;
— *отталкивающей*, если $|(f^{(p)})'| > 1$.

Если $w \in \mathbb{C}$ — притягивающая или сверхпритягивающая неподвижная точка ($f(w) = w$), то определим *область (бассейн) притяжения для* w так:

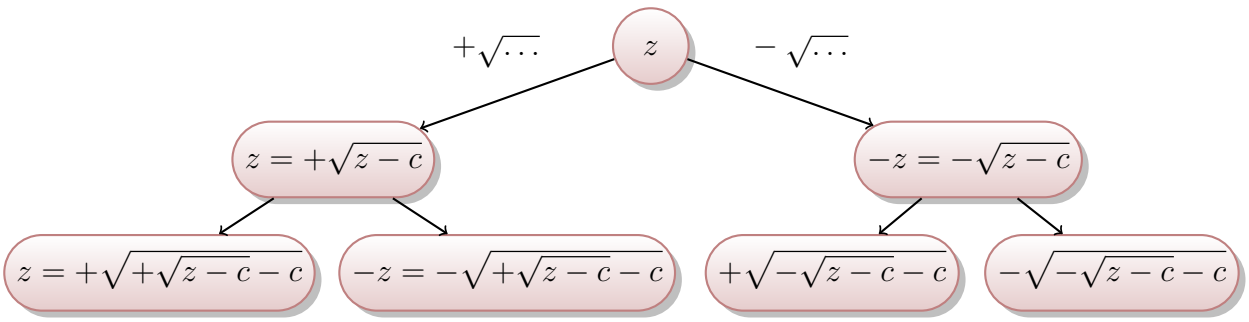
$$A(w) = \{z \in \mathbb{C} : f^{(n)}(z) \to w \text{ при } n \to \infty\}.$$

Теорема 4.2 (См. [1], с. 227; [8], гл. 14) *Пусть f — полином степени $n \geq 2$. Следующие определения множества Жюлиа эквивалентны:*
1. $J(f)$ есть граница области притяжения каждой притягивающей неподвижной точки f, включая ∞.
2. Каждая отталкивающая периодическая точка принадлежит $J(f)$. $J(f)$ является замыканием множества всех отталкивающих периодических точек f.
3. Если $w \in J(f)$, то $J(f)$ есть замыкание множества $\cup_{n=0}^{\infty}(f^{(n)})^{-1}(w)$,
где $(f^{(n)})^{-1}(w) = \{z \in \mathbb{C} : \quad f^{(n)}(z) = w\}$.

Идея алгоритма обратных итераций для построения $J(f_c)$ состоит в следующем. Возьмем точку $z \in J(f_c)$. Как это сделать? Согласно пункту 2. Теоремы 4.2 достаточно найти отталкивающую неподвижную точку: решить уравнение $f_c(z) = z$ и выбрать из его корней такой, для которого $|f'(z)| > 1$. Для функции $f_c(z) = z^2 + c$ эта задача решается аналитически: находим корни квадратного уравнения и выбираем тот из них, который не меньше другого по модулю (см. Задачу 4.10).

Воспользуемся пунктом 3. Теоремы 4.2 и, задавшись необходимым числом точек 2^{ITER}, приблизим $J(f_c)$ множеством $\cup_{n=0}^{ITER}(f_c{}^{(n)})^{-1}(z)$. В нашем случае $(f_c)^{-1}(z) = \sqrt{z-c}$ (здесь под корнем $\sqrt{z}$ понимаются сразу два значения $z_1, z_2 = \pm\sqrt{z}$ корня из комплексного числа z, такие что $z_i^2 = z$). В Maxima функция `sqrt(z)` от комплексного числа `z` возвращает одно значение (см. Задачу 4.11), поэтому мы будем брать его со знаками `+` и `-`.

Итак, возьмем найденное число z и будем применять к нему, а затем и к получающимся множествам, операцию взятия прообраза (пусть $z = +\sqrt{z-c}$):



Рандомизированный вариант этого алгоритма см. в Задаче 4.12.

Заметим, что в результате работы алгоритма обратных итераций некоторые части множества Жюлиа заполняются менее плотно, до этих частей «сложно добраться». Чтобы избежать такой неоднородности, данный метод модифицируют. Существуют и другие методы построения множеств Жюлиа, их сравнение дано в [15] и [16] (см. также [25e], [22e]).

Алгоритм 4.3: **Множество Жюлиа для функции $f_c = z^2+c$. Метод обратных итераций**

Вход: $c \in \mathbb{C}$; ✎ *комплексный параметр функции*
$ITER$; ✎ *количество итераций для испытания точки*
Выход: $path$; ✎ *массив координат точек множества Жюлиа $J(f_c)$*

1: $f_c(z) = z^2 + c$; ✎ *итерируемая функция*
2: $F_c(z) = +\sqrt{z-c} \cup -\sqrt{z-c}$; ✎ *прообраз для f_c, z может быть множеством точек*
3: Находим z_i — корни уравнения $f_c(z) = z$;
4: Находим z — максимальный по модулю из этих корней;
5: $path = \{z\}$; ✎ *инициализируем массив точек для отображения*
6: **повторять** $ITER$ **раз**
7: $path = F_c(path)$

Программируем на Maxima (листинг № 4.3):

Множество Жюлиа для $f_c = z^2 + c$. Метод обратных итераций

```
1  c:0.5,float$
2  f(z):=z^2+c,float$
3  F(z):=append(sqrt(z-c),-sqrt(z-c)),float$
4  ITER:12$
5
6  s:solve(f(z)=z,z)$
7  [s1,s2]:map(rhs,[s[1],s[2]])$
8  if abs(s1)>abs(s2) then z:float(s1) else z:float(s2)$
9  path:[expand(z)]$
10
11 thru ITER do path:F(path)$ /* вот эта строчка выполняет всю работу алгоритма! */
12
13 print("Amount of points:",length(path))$
14 load(draw)$
15 wxdraw2d(point_type=dot,points(realpart(path),imagpart(path)))$
```

Комментарий.

Заметим, что вся работа алгоритма сосредоточена в цикле на строчке `11`.

Замечания об использованных командах:

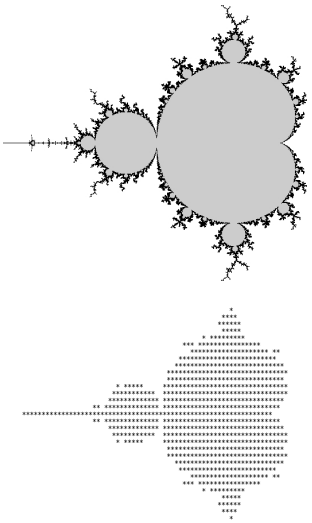
`realpart(z)` — вещественная часть комплексного числа `z`;

`imagpart(z)` — мнимая часть комплексного числа `z` (эти команды переопределяются и на массив комплексных чисел `path`).

В этой программе использовано встроенное в Maxima переопределение функции `sqrt` с чисел на массивы чисел. Таким образом, в строчке `3` сначала получаются два

массива `sqrt(z-c)` и `-sqrt(z-c)`, а потом они объединяются командой `append` (точнее, второй приписывается к первому справа).

4.2 Множество Мандельброта



Множества Жюлиа $J(f_c)$ бывают только двух типов: связные и представляющие собой канторову пыль. *Множество Мандельброта* $\mathcal{M}$ является индикатором этих двух топологических типов и его можно определить следующими эквивалентными способами:
1. $\mathcal{M} = \{c \in \mathbb{C}$: множество Жюлиа $J(f_c)$ связно$\}$.
2. $\mathcal{M} = \{c \in \mathbb{C}$: орбита нуля $\{{f_c}^{(n)}(0)\}_{n=0}^{\infty}$ ограничена$\}$.
3. $\mathcal{M} = \{c \in \mathbb{C}$: ${f_c}^{(n)}(0) \not\to \infty$ при $n \to \infty\}$.

Впервые множество $\mathcal{M}$ рассмотрел французский математик Пьер Фату в 1905 году [17], изучая итерации отображения $z \mapsto z^2 + c$. Однако увидеть это множество во всей его загадочной красоте, как и множества Жюлиа, стало возможным только с появлением компьютеров. В 1978 году Роберт Брукс и Питер Мателски получили и опубликовали первое изображение это множества (см. рис. справа).

В 1980 году французский математик Бенуа Мандельброт [18], ученик Г. Жюлиа, обратил особое внимание на загадки, важность и взаимосвязь множеств $\mathcal{M}$ и $J(f_c)$.

Для построения множества Мандельброта нам нужен критерий, судящий об ограниченности орбит. Для этого воспользуемся теоремой 4.1 и следующей теоремой:

Теорема 4.3 (См. [1], с. 233) *Если $|c| > 2$ и $|z| \geq |c|$, то орбита z устремляется к ∞. В частности, из этого следует, что $c \notin \mathcal{M}$.*

Итак, проверять на принадлежность множеству $\mathcal{M}$ нужно лишь точки $|c| \leq 2$. Более того, известно, что на окружности $|c| = 2$ только точка $c = -2 \in \mathcal{M}$. Для точек $|c| < 2$ критерий принадлежности множеству $\mathcal{M}$: если $|{f_c}^{(n)}(c)| < 2$ на протяжении $ITER$ итераций, то $c \in \mathcal{M}$, иначе $c \notin \mathcal{M}$.

Алгоритм 4.4: **Множество Мандельброта $\mathcal{M}$**

Вход: $c \in \mathbb{C}$; ✎ *комплексный параметр функции*
(a, b); ✎ *центр окна*
s; ✎ *размер окна*
p; ✎ *число пикселов в каждой стороне окна (разрешение окна)*
$ITER = 15$; ✎ *количество итераций для испытания точки*
Выход: $path$; ✎ *массив координат точек множества Мандельброта $\mathcal{M}$*

1: $path = \emptyset$; ✎ *массив точек для отображения*
2: $f_c(z) = z^2 + c$; ✎ *инициализация итерируемой функции*
3: **для** $m = 1, \ldots, p$
4: $\quad c_x = a - \frac{s}{2} + m\frac{s}{p}$;

5: **для** $n = 1, \ldots, p$
6: $c_y = b - \frac{s}{2} + n\frac{s}{p}$;
7: $z = 0$;
8: **повторять** $ITER$ **раз**
9: $z = f_c(z)$;
10: **если** $|z| \geq 2$, **то** выход из цикла;
11: **если** $|z| < 2$, **то** $path = path \cup \{c_x, c_y\}$; ✎ *добавить точку в массив для отображения*

Программируем на Maxima (листинг № 4.4):

Множество Мандельброта $\mathcal{M}$

```
1  f(z):=expand(z^2+cx+cy*%i),float$
2  [a,b]:[-0.5,0]$
3  s:3$
4  p:400$
5  ITER:15$
6
7  path:[]$
8  for m:1 thru p do
9     (cx:float(a-s/2+m*s/p),
10    for n:1 thru p do
11       (cy:float(b-s/2+n*s/p),
12       z:0,
13       thru ITER do
14          (z:f(z),
15          if abs(z)>=2 then return()
16          ),
17       if abs(z)<2 then path:endcons([cx,cy],path)
18       )
19    )$
20 load(draw)$
21 wxdraw2d(point_type=dot,points(path))$
```

4.3 Проблема Кэли

В 1879 г. сэр Артур Кэли в короткой заметке [12], меньше чем на страницу, поставил задачу нахождения областей притяжения корней уравнения $f(z) = 0$ при использовании классического метода Ньютона-Фурье, распространенного на комплексную плоскость. Эта заметка стимулировала вышеупомянутые исследования Г. Жюлиа.

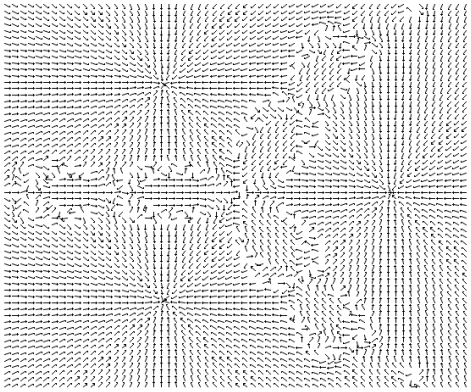
А именно: дано уравнение $f(z) = 0$, где f — дифференцируемая функция, и его корень $\xi \in \mathbb{C}$. Надо найти множество

$$A(\xi) = \left\{ z \in \mathbb{C} \quad | \quad z_n \underset{n\to\infty}{\longrightarrow} \xi, \text{ где } z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}, z_0 = z \right\}.$$

Если рассмотреть функцию $g(z) = z - f(z)/f'(z)$, то нули функции $f(z)$ являются сверхпритягивающими неподвижными точками $g(z)$ (проверьте!). Поскольку $g(z)$ в об-

щем случае не является полиномом, то для этой функции множество Жюлиа можно определить как замыкание множества отталкивающих точек.

Мы предоставим читателю самостоятельно найти способ построения областей притяжения корней и реализовать его в Maxima, см. Задачи 4.16–4.18. А здесь приведем алгоритм построения векторного поля, показывающего, в какую сторону двигаются точки под воздействием многократного применения отображения g. По этому векторному полю можно представить, как расположены области притяжения корней f. На рисунке справа построено векторное поле для уравнения $f(z) = z^3 - 1 = 0$, т.е. для поиска бассейнов притяжения кубических корней из 1.



Идея алгоритма проста: разобьем прямоугольную область, в которой содержатся корни уравнения $f(z) = 0$, на сетку узлов, и в каждом узле z нарисуем вектор $g^{(ITER)}(z) - z$, показывающий направление смещения этого узла.

Алгоритм 4.5: **Векторное поле для проблемы Кэли**

Вход: $f(z)$; ✎ *функций, задающая уравнение $f(z) = 0$*
$[a, b] \times [c, d]$; ✎ *окно, в котором будет построено поле; корни уравнения должны содержаться внутри этой области*
$ITER$; ✎ *количество итераций отображения g*
$n \times m$; ✎ *число точек разбиения сторон окна (разрешение окна)*
Выход: $Vectors$; ✎ *массив векторов, исходящих из соответствующих точек решётки разбиения окна*
и графическое изображение векторного поля

1: $g(z) = z - f(z)/f'(z)$; ✎ *инициализация функции g*
2: $Vectors = \emptyset$; ✎ *инициализация массива векторов*
3: **для** $i = 1, \ldots, n$; $j = 1, \ldots, m$
4: $z_0 = a + (b - a)i/n + (c + (d - c)j/m)\mathbf{i}$; ✎ *пробегаем по узлам решётки*
5: $z = z_0$;
6: **повторять** $ITER$ **раз**
7: $z = g(z)$;
8: $A = \{\mathrm{Re}(z - z_0), \mathrm{Im}(z - z_0)\}$; ✎ *искомый вектор смещения за $ITER$ шагов*
9: $Vectors_{ij} = \alpha \frac{A}{\|A\|}$; ✎ *Подберем число $\alpha \in (0, 1)$ эмпирически так, чтобы векторы на рисунке не пересекались*
10: в точках $\{a + (b - a)i/n, c + (d - c)j/m\}$ окна $[a, b] \times [c, d]$ рисуем векторы $Vectors_{ij}$.

Программируем на Maxima (листинг № 4.5):

Векторное поле для проблемы Кэли (случай $f(z) = z^3 - 1$)

```
n:60$
m:60$
[a,b]:[-1.6,1.5]$
[c,d]:[-1.6,1.5]$
ITER:5$
g(z):=z-(z^3-1)/(3*z^2),float$
Vectors:[]$

for i:1 thru n do
   for j:1 thru m do
       (z0:float(a+(b-a)*i/n+(c+(d-c)*j/m)*%i),
        z:z0,
        thru ITER do z:g(z),
        A:[realpart(z-z0),imagpart(z-z0)],
        len:sqrt(A[1]^2+A[2]^2)/0.04,
        Vectors:append(Vectors,[vector(float([a+(b-a)*i/n,c+(d-c)*j/m]),A/len)])
        )$
load(draw)$
draw2d(color=black,xrange=[a,b],yrange=[c,d],head_length = 0.01,Vectors)$
```

Комментарий.

Заметим, чтобы избежать деления на 0 при вычислении значения $g(0)$, проще всего немного сдвинуть сетку узлов так, чтобы она не проходила через 0.

Замечания об использованных командах:

`vector([x,y],[dx,dy])` — графический объект, представляющий собой координаты точки `[x,y]` и координаты вектора `[dx,dy]`, исходящего из данной точки;

`head_length` — опция команды `draw`, задающая размер стрелки вектора.

Задача 4.1 Покажите, что оба множества $J(f_c)$ и $\mathbf{J}(f_c)$ симметричны относительно точки $z = 0$ ($f_c(z) = f_c(-z)$). Используйте это, чтобы в два раза сократить вычисления в Алгоритмах 4.1 и 4.2.

Задача 4.2 Постройте множества Жюлиа для полиномов степени $n \geq 3$, а также для какой-либо неполиномиальной функции.

Задача 4.3 Сделайте цветную иллюстрацию множества Жюлиа, используя алгоритм «времени убегания»: цвет пиксела экрана зависит от количества шагов, за которые итерации точки выходят из круга радиуса 2.

Задача 4.4 Нарисуйте множество Жюлиа для функции $f(z) = c \sin z$ (возьмите параметр $c = 1 + 0.1\mathbf{i}$), используя следующий критерий: точка $x + y\mathbf{i}$ принадлежит изображению множества Жюлиа, если за $n = 30$ шагов ее итерации остаются внутри круга $|z| < m$,

$m = 100$. Попробуйте менять числа c, n, m, а также сделать цветную иллюстрацию, аналогично Задаче 4.3.

Задача 4.5 Известно, что функция $f_c(z) = z^2 + c$ хаотична на своем множестве Жюлиа $J(f_c)$ при любом $c \in \mathbb{C}$ (см. [1, с. 247]). Проиллюстрируйте это компьютерным экспериментом.

Задача 4.6 Изучите самостоятельно другие алгоритмы построения множеств Жюлиа, см. [15]. Запрограммируйте один из них на Maxima.

Задача 4.7 Покажите, что множество $\mathcal{M}$ симметрично относительно вещественной оси ($f_c(c) = \overline{f_{\bar{c}}(\bar{c})}$). Используйте это, чтобы в два раза сократить вычисления в Алгоритме 4.4. Каким образом можно еще ускорить алгоритм, если мы знаем, что множество $\mathcal{M}$ заполнено внутри соответствующей главной кардиоиды $\rho = \frac{1}{2}(1 - \cos\theta)$?

Задача 4.8 Постройте границу множества Мандельброта $\partial\mathcal{M}$.

Задача 4.9 Найти замену переменной, при которой квадратичный полином $f(z) = az^2 + bz + c$, $a \neq 0$, приобретает вид $f(\tilde{z}) = \tilde{z} + \tilde{c}$.

Задача 4.10 Пусть $c \neq \frac{1}{4}$, а z_1, z_2 — корни уравнения $f_c(z) = 0$, $|z_1| \geq |z_2|$. Тогда $|f_c'(z_1)| > 1$.

Задача 4.11 Выяснить, какое из двух возможных значений корня $\sqrt{a + b\,i}$ возвращает функция `sqrt` в Maxima.

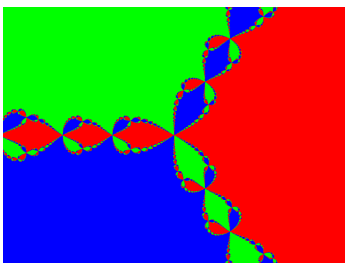
Задача 4.12 Можно существенно ускорить Алгоритм 4.3, привнеся в него элемент случайности: на каждом шаге итерации в строчке 7 вместо отображения F_c применять одно из отображений $+\sqrt{z - c}$ или $-\sqrt{z - c}$, выбранное случайно.

Задача 4.13 Можно ли ускорить вычисления в Листинге 4.3? Например, вычисляя отталкивающую точку z по аналитической формуле. Или изменив прообраз в строчке 3: `F(z):=(Z:float(sqrt(z-c)),append(Z,-Z))$`.

Задача 4.14 Можно ли ускорить вычисления в Алгоритме 4.4, если перебирать не прямоугольную, а радиальную решётку точек внутри круга $|c| < 2$?

Задача 4.15 Постройте множество Мандельброта для отображения $f(z) = z^3 + c$. Покажите, что если $|c| > 2$, то орбита z стремится к ∞.

Задача 4.16 Постройте области притяжения корней уравнения $z^3 - 1 = 0$ в $\mathbb{C}$, раскрасив их в различные цвета.



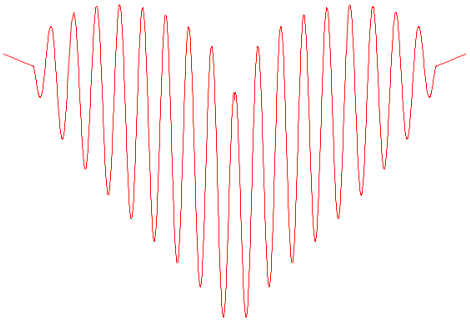
Задача 4.17 Решите Задачу 4.16 для функции $f(z) = z^4 - 1$.

Задача 4.18 Решите Задачу 4.16 для функции $f(z) = z^3 - z$.

Задача 4.19 Подумайте, можно ли использовать команду `plotdf`[4] для построения векторного поля для итераций в проблеме Кэли. В справке по Maxima можно найти обширную информацию и примеры.

Рекомендуемая литература: [9], [1, с. 232], [8], [5], [15], [16]; [23e].

[4] Эта команда используется для изображения векторных полей и их интегральных траекторий, вязанных с решениями дифференциальных уравнений.



ПРИЛОЖЕНИЕ

Разумеется, хорошая математика всегда красива.

Пол Джозеф Коэн

В этой главе собраны дополнительные сведения, полезные для дальнейшего и более глубокого изучения применения программы Maxima для моделирования фракталов: вопросы сохранения графики, анимации, использования дополнительных пакетов.

I Сохранение графики и анимация в Maxima

Как в формате `gnuplot`, так и в `xmaxima` полученное изображение можно сохранить нажатием правой клавиши мыши на нём и выбирая соответствующий пункт меню. Заметим лишь, что изображения, получаемые при помощи команд `wxplot2d` и `wxdraw2d` имеют маленькое разрешение при таком сохранении.

Заметим, что при выводе графики посредственно в рабочую область командами `wxplot` и `wxdraw` опция перенаправления на терминал не работает, потому что этими командами графика выводится на рабочую область программы.

Для сохранения графики можно воспользоваться следующими командами:

пример

```
draw(terminal=png,                                    /* выбор типа файла */
   file_name="c:\\temp\\picture", /* имя целевого файла, без расширения! */
   gr2d(explicit(sin(x),x,0,1)))$                     /* график,с опциями */
```

что эквивалентно коду:

пример

```
draw(terminal=png,
   file_name="c:\\temp\\picture",
   explicit(sin(x),x,0,1))$
```

Команды `gr2d` и `gr3d` задают соответственно дву- и трёхмерные сцены для отрисовки их командой `draw`. Команда `draw` может отрисовать сразу несколько сцен. Для отрисовки одной сцены связка команд `draw`–`grd2` эквивалентна команде `draw2d` (то же самое для `gr3d`).

В качестве параметров для сохранения графики можно использовать параметр `dimensions=`[$xpixels, ypixsels$], указывающий на разрешение целевого файла $xpixels \times$

ypixels. Наиболее распространенные терминалы графических файлов: `png`, `eps`, `jpg`, `eps_color`, `pdf`, `gif`, `animated_gif`, `svg`.

Еще один совет: когда сохраняете много изображений, желательно делать подписи на них (какая функция, какие параметры использовались). Это можно сделать опциями `title` и `key`. При этом удобно использовать команду `string(t)` — она конвертирует числовое значение `t` в строчное. Пример (русские буквы не всегда получается использовать):

пример
```
t:%pi$
draw2d(title="График функции sin(x+t)",
key=concat("Значение параметра t=",string(t)),
explicit(sin(x+t),x,0,1))$
```

Анимацию (видео или анимированное изображение формата GIF) можно создать из серии уже сохранённых изображений. Мы разберем, как это можно сделать с помощью бесплатной программы работы с графикой из командной строки Image Magick [13e]. Вызов ее из командной строки (чтобы не запутаться, обозначим пробел символом ␣):

пример
```
convert␣-delay␣10␣c:\\temp\\*.png␣c:\\temp\\anim.gif)$
```
команда Image Magick; задержка между кадрами; кадры для сборки: все файлы png с таким адресом; файл-результат

пример
```
for t:1 thru 20 do
   draw2d(terminal=png,
          file_name=concat("c:\\temp\\ga",add_zeroes(t)),
          explicit(sin(x+2*%pi/20*t),x,0,2*%pi)
          )$     /* конвертация с помощью Image Magick: */
system("convert -delay 10 c:\\temp\\*.png c:\\anim.gif")$
```

Здесь команда `system("`*cmd*`")` вызывает команду "*cmd*" в командной строке операционной системы. Команда `concat(`x, y`)` присоединяет элемент y справа к элементу x (команда `sconcat` возвращает результат в виде строки). Команда `add_zeroes(`t`)`, где t — строка или число, конвертируемое в строку, — возвращает строку из 10 символов, дополняя строку t слева нулями (например `add_zeroes(abc)`$\mapsto 0000000abc$), а если длина строки t превосходит 9, то возвращается сама строка t. Вместо расширения AVI можно указать MPG — тогда получится файл видео.

Сделаем анимированное изображение семейства графиков функций

$$f(x,t) = \sqrt{\cos x}\cdot\cos\left((200|t|+25)x\right) + \sqrt{|x|} - 0.7(4-x^x)^{0.01}, \qquad x\in[-1.8,1.8],$$

при разных значениях параметра $t\in[-1,1]$. Сохраним его в файл `C:\\temp\\anim.gif`:

пример
```
load(draw)$
f(x,t):=sqrt(cos(x))*cos((abs(t)+25)*x)+sqrt(abs(x))-0.7*(4-x*x)^0.01$
g:makelist(gr2d(explicit(sin(f(x,t)),x,-1.8,1.8),noframe),t,-1,1,0.1)$
draw(terminal=animated_gif, /* тип терминала графики: анимированный GIF */
     delay=20,                  /* Задержка между кадрами в миллисекундах */
     file_name="C:\\temp\\anim",     /* проверьте, существует ли папка! */
     g)$     /* массив графических объектов - графиков семейства f(x,t) */
```

Здесь пользовательская опция `noframe` введена в комментарии на с. 9. Имя файла `anim` должно быть без расширения: оно дополняется выбором терминала выше.

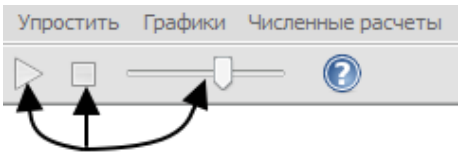
Еще один вариант осуществления анимации возможен в оболочке wxMaxima:

пример

```
with_slider_draw(t,makelist(i,i,1,20),noframe,explicit(f(x,t),x,-1.8,1.8))$
```

t — параметр кадров; опции; кадры графики

После того, как wxMaxima закончила вычисления и отобразила рисунок, выделите этот рисунок мышью — и в верхней панели инструментов активируются три элемента для управления анимацией.



II Обзор пакета `fractals`

С помощью пакета `fractals` автора José Ramírez Labrador можно строить известные фракталы:

- треугольник Серпинского, фракталы «Дерево», «Папоротник»;
- множество Мандельброта и множества Жюлиа;
- снежинки Коха;
- отображения Пеано: кривые Серпинского и Гильберта.

Данный пакет обладает ограниченными возможностями. Однако рекомендуется изучить его исходный код, находящийся в файле «`fractals.mac`». Параметры всех команд этого пакета, приведённых ниже, можно изменить непосредственно в тексте этого пакета или скопировать соответствующий кусок кода в свою рабочую область и изменить его. Рассмотрим функции этого пакета.

☆ `sierpinskiale(`n`)` — возвращает массив из координат $n-1$ случайной точки, принадлежащей треугольнику Серпинского, получающемуся рандомизированным Алгоритмом (см. секцию 2.2, с. 21). Функции этой СИФ:

$$f_1\begin{bmatrix}x\\y\end{bmatrix}=0.5\begin{bmatrix}x\\y\end{bmatrix},\ f_2\begin{bmatrix}x\\y\end{bmatrix}=0.5\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}1\\0\end{bmatrix},\ f_3\begin{bmatrix}x\\y\end{bmatrix}=0.5\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}0\\1\end{bmatrix}.$$

пример

```
load(fractals)$
plot2d([discrete,sierpinskiale(10000)], [style,dots])$
```

☆ `treefale(`n`)` — возвращает массив из координат $n-1$ случайной точки, принадлежащей древовидному аттрактору СИФ, получающемуся рандомизированным алгоритмом. Функции этой СИФ:

$$f_1\begin{bmatrix}x\\y\end{bmatrix}=\tfrac{3}{4}\begin{bmatrix}0\\y\end{bmatrix},\ f_2\begin{bmatrix}x\\y\end{bmatrix}=\tfrac{3}{4}\begin{bmatrix}\cos\frac{\pi}{6} & -\sin\frac{\pi}{6}\\ \sin\frac{\pi}{6} & \cos\frac{\pi}{6}\end{bmatrix}\cdot\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}0\\1\end{bmatrix},\ f_3\begin{bmatrix}x\\y\end{bmatrix}=\tfrac{3}{4}\begin{bmatrix}\cos\frac{\pi}{6} & \sin\frac{\pi}{6}\\ -\sin\frac{\pi}{6} & \cos\frac{\pi}{6}\end{bmatrix}\cdot\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}0\\1\end{bmatrix}.$$

☆ `fernfale(`n`)` — возвращает массив из координат $n-1$ случайной точки, принадлежащей древовидному аттрактору СИФ, получающемуся рандомизированным алгоритмом (здесь вероятности выбора функции зависят от коэффициентов сжатия). Функции этой СИФ:

$$f_1\begin{bmatrix}x\\y\end{bmatrix}=\begin{bmatrix}0\\0.16y\end{bmatrix},\ f_2\begin{bmatrix}x\\y\end{bmatrix}=\begin{bmatrix}0.85 & 0.04\\-0.04 & 0.85\end{bmatrix}\cdot\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}0\\1.6\end{bmatrix},\ f_3\begin{bmatrix}x\\y\end{bmatrix}=\begin{bmatrix}0.2 & -0.26\\0.23 & 0.22\end{bmatrix}\cdot\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}0\\1.6\end{bmatrix},$$
$$f_4\begin{bmatrix}x\\y\end{bmatrix}=\begin{bmatrix}-0.15 & 0.28\\0.26 & 0.24\end{bmatrix}\cdot\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}0\\0.44\end{bmatrix}.$$

☆ `mandelbrot_set(`x, y`)` — возвращает число $n \le 29$ итераций, для которого $|f_{x+y\mathbf{i}}{}^{(n)}(0)| \ge 100$, либо число 30, если за 29 итераций орбита точки 0 осталась внутри круга $|z| < 100$. Напомним, множеству Мандельброта $\mathcal{M}$ принадлежат точки $c = x + y\mathbf{i}$, для которых орбита нуля ограничена (см. секцию 4.2, с. 41).

Эту функцию можно использовать для раскрашивания областей по скоростям убегания точек, например, так:

пример
```
load(fractals)$
plot3d(mandelbrot_set, [x, -2.5, 1], [y, -1.5, 1.5],
[gnuplot_preamble, "set view map"],[grid, 150, 150])$
```

Здесь опция `grid` указывает на то, что область изменения координат $[-2.5, 1] \times [-1.5, 1.5]$ должна быть разделена на 150×150 точек (x, y), в которых будет вычисляться функция `mandelbrot_set(x,y)`. Опция `gnuplot_preamble` используется для интерпретации значения функции `mandelbrot_set(x,y)` как цвета точки `(x,y)` (попробуйте убрать эту опцию).

☆ `julia_set(`x, y`)` — возвращает число $n \le 29$ итераций, для которого $|f_c{}^{(n)}(x+y\mathbf{i})| \ge 100$, либо число 30, если за 29 итераций орбита точки $x + y\mathbf{i}$ осталась внутри круга $|z| < 100$. Здесь число c по умолчанию задается переменной `julia_parameter:%i`. Напомним, заполняющему множеству Жюлиа принадлежат точки $x+y\mathbf{i}$, орбита которых ограничена (см. секцию 4.1, с. 33). Применять эту команду можно так:

пример
```
load(fractals)$
plot3d(julia_set, [x, -2, 1], [y, -1.5, 1.5],
[gnuplot_preamble, "set view map"],[grid, 150, 150])$
```

Автор данного пакета советует опробовать этот код на следующих числах `julia_parameter`: $-0.745 + 0.113002\mathbf{i}$, $-0.39054 - 0.58679\mathbf{i}$, $-0.15652 + 1.03225\mathbf{i}$, $-0.194 + 0.6557\mathbf{i}$, $0.011031 - 0.67037\mathbf{i}$.

☆ `julia_sin(`x, y`)` — то же, что и предыдущая команда, только для функции $f(z) = c\sin z$. Автор рекомендует параметр `julia_parameter:1+0.1*%i$`.

☆ `snowmap(`$vert, iter$`)` — возвращает массив вершин снежинки Коха после $iter$ итераций, построенной на сторонах ломаной линий, заданной массивом вершин $vert$ в виде комплексных чисел. Если первая и последняя вершины совпадают, то ломаная линия замкнута. Результирующее множество расположено слева от ломаной линии, считая в направлении от начала к ее концу. Для построения используется рекурсивный алгоритм. Пример использования (подумайте, в чем разница):

пример
```
plot2d([discrete,snowmap([1,exp(%i*%pi*2/3),exp(-%i*%pi*2/3),1],4)])$
```

пример
```
plot2d([discrete,snowmap([1,exp(-%i*%pi*2/3),exp(%i*%pi*2/3),1],4)])$
```

☆ `hilbertmap(`*iter*`)` — возвращает массив вершин непрерывной кривой Гильберта, плотно заполняющей плоскую область, после *iter* итераций.

пример
```
plot2d([discrete,hilbertmap(4)])$
```

☆ `sierpinskimap(`*iter*`)` — возвращает массив вершин непрерывной кривой Серпинского, плотно заполняющей плоскую область, после *iter* итераций.

пример
```
plot2d([discrete,sierpinskimap(4)])$
```

III Обзор пакета `dynamics`

С помощью пакета `dynamics` можно строить различные графические представления динамических систем и фракталов:

- паутинная диаграмма;
- бифуркационная диаграмма;
- эволюция орбиты одно- и двумерного отображений;
- «игра в хаос»;
- система итерированных функций, заданная аффинными преобразованиями;
- множества Жюлиа, Мандельброта;

а также, в нём реализован метод Рунге-Кутты 4го порядка для решения систем дифференциальных уравнений.

Данный пакет обладает ограниченными возможностями. Однако рекомендуется изучить его исходный код, находящийся в файле «`dynamics.mac`». Параметры всех команд этого пакета, приведённых ниже, можно изменить непосредственно в тексте этого пакета или скопировать соответствующий кусок кода в свою рабочую область и изменить его. Рассмотрим функции этого пакета.

Напомним, для использования команд данного пакета нужно его загрузить:

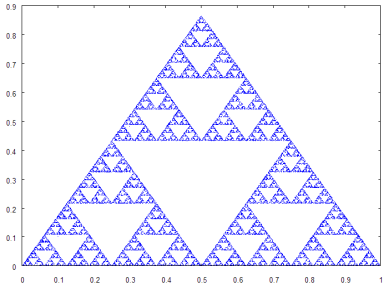
```
load(dynamics)$
```

Для вывода графики команды пакета `dynamics` используют команду `plot2d`, поэтому все опции *options* этой команды можно передавать в команды пакета `dynamics`, например, можно менять графические интерфейсы, различные стили графиков, цвета, менять масштаб осей на логарифмический и т.д. Смотрите список опций команды `plot2d` в справке по Maxima.

☆ `chaosgame([[`x_1`,`y_1`],...,[`x_m`,`y_m`]],[`x_0`,`y_0`],b,n,options)` — реализует «игру в Хаос»: отмечается точка (x_0, y_0),

пример

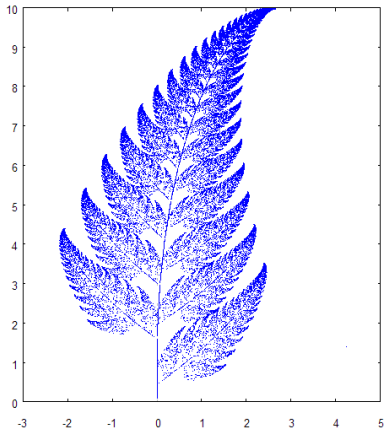
```
chaosgame([[0,0],[1,0],[0.5,sqrt(3)/2]],
          [0.1,0.1],1/2,30000,[style,dots])$
```



☆ `ifs([`r_1`,...,`r_m`],[`A_1`,...,`A_m`],[[`x_1`,`y_1`],...,[`x_m`,`y_m`]],[`x_0`,`y_0`],n,options)` — реализует построение аттрактора системы итерированных функций (x_0, y_0),

пример

```
a1:matrix([0.85,0.04],[-0.04,0.85])$
a2:matrix([0.2,-0.26],[0.23,0.22])$
a3:matrix([-0.15,0.28],[0.26,0.24])$
a4:matrix([0,0],[0,0.16])$
p1:[0,1.6]$
p2:[0,1.6]$
p3:[0,0.44]$
p4:[0,0]$
r: [85,92,99,100]$
ifs(r,[a1,a2,a3,a4],[p1,p2,p3,p4],
          [5,0],50000,[style,dots])$
```



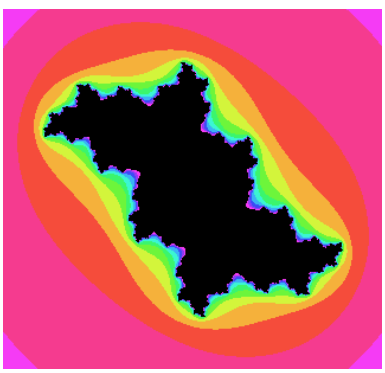
☆ `julia(`x`,`y`,options)` — создаёт изображение множества Жюлиа с параметром $c = x + iy$, $x, y \in \mathbb{R}$, и сохраняет в файле в формате XPM в директорию установки Maxima (см. ниже опцию `filename`).

Точки вне множества Жюлиа окрашиваются в различные цвета в зависимости от количества итераций, после которых орбита выходит из круга радиуса 2. Максимальное количество итераций определяется опцией `levels`. Если после заданного количества итераций орбита точки остается внутри круга радиуса 2, точка окрашивается в цвет, заданный опцией `color`. Все цвета, в которые раскрашиваются точки вне множества Жюлиа, имеют одинаковую насыщенность (*saturation*) и яркость (*value*), но разные углы оттенка (*hue*), равномерно распределённые между *hue* и *hue* + *huerange*.

Доступные опции, дополнительные к опциям пакета `plot2d`, опишем в конце секции.

пример

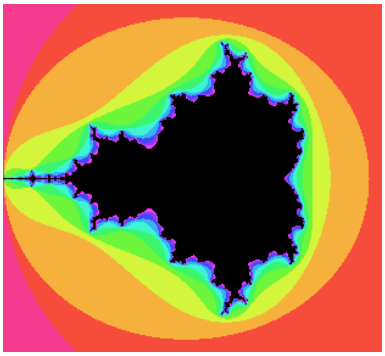
```
julia(-0.11,0.65569999)$
```



☆ `mandelbrot(options)` — создаёт изображение множества Мандельброта и сохраняет в файле в формате XPM. Окрашивание точек происходит по тому же правилу, что и в команде `julia`.

пример

```
mandelbrot()$
```



Дополнительные опции команд `julia` и `mandelbrot`:

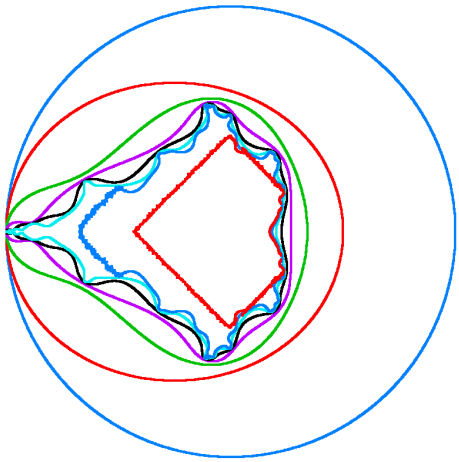
- `size` — задаёт размер изображения в пикселах. При этом `size=a` (или `[size,a]`) задает квадратное изображение `a`×`a`, а `[size,a,b]`. Если `a`≠`b`, изображение будет выглядеть искаженным. По умолчанию размеры `400`×`400`.
- `levels` — определяет максимальное число итераций, это число также равно количеству используемых цветов для раскраски точек вне множества. Значение по умолчанию 12. Большие значения требуют большего времени вычислений.
- `huerange` — задает область углов оттенка (*hue*) для раскраски точек вне множества. Значение по умолчанию 360 (это означает, что цвета будут всех оттенков). Если это значение больше 360, область углов оттенка повторяется. Если значение меньше нуля, то углы оттенка будут убывать с количеством итераций.
- `hue` — задаёт значение оттенка в градусах для первого цвета для окраски точек вне множеств. Значение по умолчанию 300, что отвечает пурпурному цвету (*magenta*). Стандартные значения для других цветов: 0 (красный цвет), 45 (оранжевый), 60 (желтый), 120 (зелёный), 180 (голубой, *cyan*), 240 (синий).
- `saturation` — задаёт насыщенность цвета точек вне множества (в пределах от 0 до 1). По умолчанию 0.46.
- `value` — задаёт яркость цвета точек вне множества (в пределах от 0 до 1). По умолчанию 0.96.
- `color` — задаёт значение цвета точек множества тремя параметрами: `[color,`*hue*`,`*sturation*`,`*value*`]` (оттенок, насыщенность, яркость). По умолчанию все параметры равны 0, что соответствует чёрному цвету.
- `center` — задает координаты точки $c = x+iy$ комплексной плоскости, приходящейся на центр изображения: `[center,`x`,,`y`]`. Значение по умолчанию $x = y = 0$ (начало координат находится в середине изображения).
- `radius` — задаёт радиус наибольшей окружности, умещающейся в изображаемой квадратной области (определяет масштаб изображения). По умолчанию 2.
- `filename` — задаёт имя файла изображения без расширения. По умолчанию это имя `"julia"` или `"mandelbrot"` и файл сохраняется в директорию, в которой установлена Maxima. Если необходимо сохранять файлы в другой директории, следует указать полное имя файла с путём, но без расширения.

Мы опускаем описание следующих команд:
☆`evolution`,
☆`evolution2d`,
☆`orbits`,
☆`staircase`,
☆`rk`
— они будут описаны в Приложении части II настоящего пособия.

IV Дополнительные примеры листингов программ

Здесь собраны листинги программ разных авторов (в основном, отсюда [10e],[27e],[28e]). Мы не останавливаемся на подробном обсуждении реализации этих алгоритмов. Предполагается, что необходимые сведения читатель найдет в интернете. Рекомендуется изучить приведенные листинги, запустить их в Maxima, изучить неизвестные вам команды и опции и подумать, почему автор именно так реализовал свой алгоритм. Попробуйте изменить параметры программ, чтобы лучше их понять.

☆ Вычисление и отображение лемнискат множества Мандельброта (авторы алгоритма на Maxima A. Majewski, J. Villate и A. Vodopivec), см. [24e].



Программа выводит результат в файл `lemniscates.png` в папке программы Maxima.

Программируем на Maxima (листинг № 5.6):

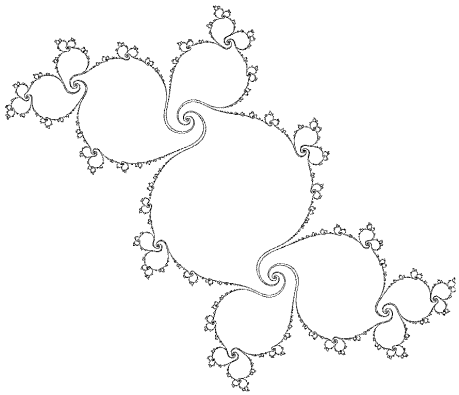
Лемнискаты множества Мандельброта

```
load(implicit_plot)$
c: x+%i*y$
ER:2$                                   /* Радиус убегания, должен быть >=2 */
f[n](c) := if n=1 then c else (f[n-1](c)^2 + c)$
ip_grid:[100,100]$          /* устанавливаем разрешение первичной сетки узлов для
                                        implicit_plot, по умолчанию: [50, 50] */
ip_grid_in:[15,15]$         /* устанавливаем разрешение вторичной сетки узлов для
                                        implicit_plot, по умолчанию: [5, 5] */
my_preamble: "set zeroaxis; set title 'Boundaries of level sets of escape time
```

```
                of Mandelbrot set'; set xlabel 'Re(c)'; set ylabel 'Im(c)'"$
implicit_plot(makelist(abs(ev(f[n](c)))=ER,n,1,7), [x,-2.5,2.5],[y,-2.5,2.5],
              [gnuplot_preamble, my_preamble],[gnuplot_term,"png size 1000,1000"],
              [gnuplot_out_file, "lemniscates.png"])$
```

Заметим, что программе нужно много времени для вычислений. Если вы не видите результат, уменьшите количество выводимых лемнискат (параметр `n`). Приведенная программа не может корректно строить лемнискаты №7–8 из-за вычислительных трудностей при работе с числами с плавающей точкой (floating point).

☆ Построение множества Жюлиа с помощью модифицированного метода обратных итераций MIIM (автор алгоритма на Maxima A. Majewski), см. [25e].



Программа выводит результат в файл `miim.png` в папке программы Maxima. Модифицированный метод обратных итераций: останавливаем обратные итерации, если пиксел был посещён больше, чем `HitLimit` раз.

Программируем на Maxima (листинг № 5.7):

Множество Жюлиа, алгоритм MIIM

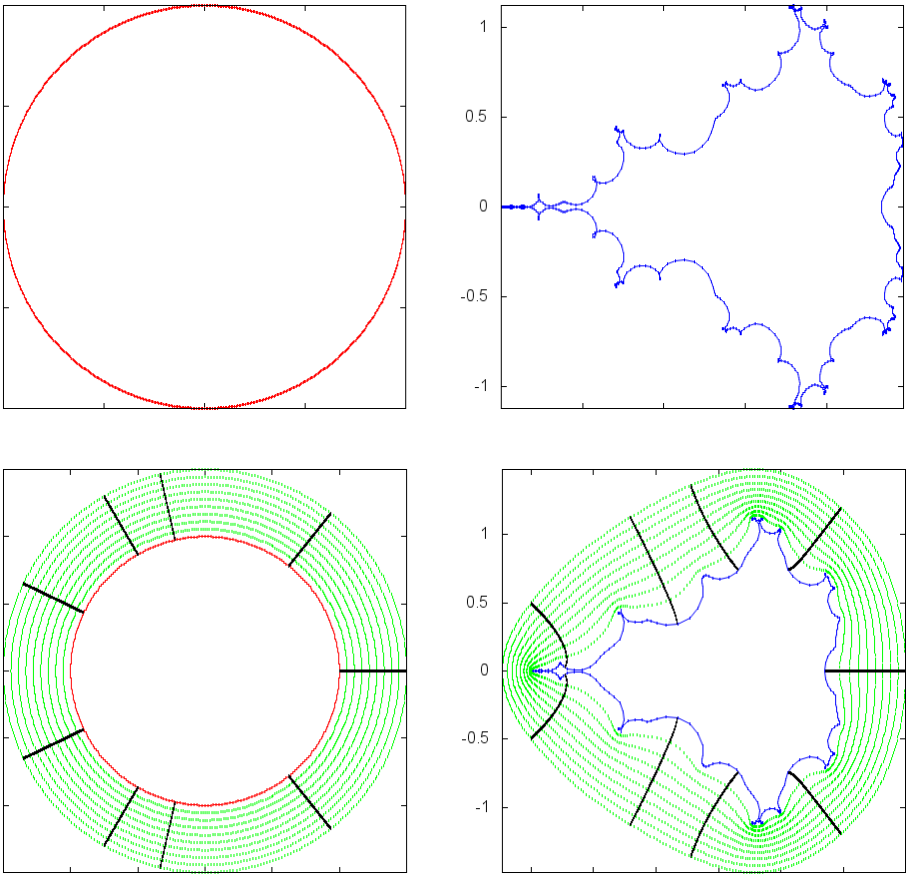
```
c:-0.11+0.65569999*%i$                                  /* параметр множества Жюлиа */
HitLimit:14$       /* пропорционален степени детализации и затраченному времени */
[iXmax,iYmax]:[500,500]$   /* разрешение, пропорционально детализации и времени */
                   /* размер рисунка будет : ширина:iXmax-0+1; высота:iYmax-0+1 */
start:elapsed_run_time ();
NumberOfPoints:0;
f(z,c):=z*z+c;
finverseplus(z,c):=sqrt(z-c);
finverseminus(z,c):=-sqrt(z-c);
[zxMin,zxMax]:[-2.0,2.0]$          /* определяем область комплексной плоскости, */
[zyMin,zyMax]:[-2.0,2.0]$                        /* в которой ищем множество Жюлиа */
PixelWidth:(zxMax-zxMin)/iXmax$
PixelHeight:(zyMax-zyMin)/iYmax$
array(Hits,iXmax,iYmax);  /* 2D массив, считающий количество посещений пикселов */
                                                 /* решётки точками нашей орбиты */
    /* Hits>0 означает, что точка принадлежит орбите. Вначале таких точек нет: */
for iX:0 thru iXmax step 1 do
```

```
   for iY:0 thru iYmax step 1 do Hits[iX,iY]:0$
                    /* считаем неподвижные точки отображения f(z,c): z=f(z,c) */
fixed:float(rectform(solve([z*z+c=z],[z])));
                                    /* выделяем из решения отталкивающую точку */
if (abs(2*rhs(fixed[1]))<1)
   then block(beta:rhs(fixed[1]),alfa:rhs(fixed[2]))
   else block(alfa:rhs(fixed[1]),beta:rhs(fixed[2]))$
                          /* выбираем отталкивающую точку в качестве начальной */
stack:[beta]$                                        /* сохраняем beta в стеке */
                        /* создаём 2 списка точек и копируем beta в эти списки */
xx:makelist (realpart(beta), i, 1, 1)$                          /* список Re(z) */
yy:makelist (imagpart(beta), i, 1, 1)$                          /* список Im(z) */
NumberOfPoints:1$
                                                    /* обратные итерации beta: */
block(
   loop,
   z:last(stack),
   stack:delete(z,stack),
   z:finverseplus(z,c),        /* обратная итерация - первый прообраз (корень) */
   iX:fix((realpart(z)-zxMin)/PixelWidth),   /* переводим из мировых координат */
   iY:fix((imagpart(z)-zyMin)/PixelHeight),                    /* в экранные */
   hit:Hits[iX,iY],
   if hit<HitLimit
      then block(
         Hits[iX,iY]:hit+1,
         stack:endcons(z,stack),          /* добавляем z в конец списка stack */
         if hit=0 then block(xx:cons(realpart(z),xx),yy:cons(imagpart(z),yy)),
         NumberOfPoints:NumberOfPoints+1
         ),
   z:-z,                       /* обратная итерация - второй прообраз (корень) */
   iX:fix((realpart(z)-zxMin)/PixelWidth),   /* переводим из мировых координат */
   iY:fix((imagpart(z)-zyMin)/PixelHeight),                    /* в экранные */
   hit:Hits[iX,iY],
   if hit<HitLimit
      then block(
         Hits[iX,iY]:hit+1,
         stack:endcons(z,stack),          /* добавляем z в конец списка stack */
         if hit=0 then block(xx:cons(realpart(z),xx),yy:cons(imagpart(z),yy)),
         NumberOfPoints:NumberOfPoints+1
         ),
   if is(not emptyp(stack)) then go(loop)
)$
stop:elapsed_run_time()$
t:fix(stop-start);

load(draw)$      /* отображаем обратные орбиты точки beta с помощью пакета draw */
draw2d(file_name="miim",terminal='png,dimensions=[iXmax,iYmax],
      yrange=[zyMin,zyMax],xrange=[zxMin,zyMax],
```

```
        title="Julia set in dynamical plane for f(z,c):=z*z+c drawn using MIIM",
        key=concat("c=",string(c),". There are ",string(NumberOfPoints)," points"),
        label([concat("HitLimit=",string(HitLimit)," Time=",string(t)),-1.2,-1.8]),
        xlabel="Z.re ", ylabel="Z.im",point_type=dot,point_size=8,color=black,
        points(xx,yy)
)$
```

☆ Построение отображения окружности на множество Мандельброта (автор алгоритма на Maxima A. Majewski, R.Fateman, G. A. Edgar), см. [26e].



Как показали А. Дуади и Дж. Хаббард, существует конформное преобразование, отображающее в расширенной комплексной плоскости $\hat{\mathbb{C}}$ внешность множества Мандельброта $\mathcal{M}$ на внешность единичного диска $\overline{\mathbb{D}}$ (функция Л. Бетхера) $\Phi_M\colon \hat{\mathbb{C}} \setminus \mathcal{M} \to \hat{\mathbb{C}} \setminus \overline{\mathbb{D}}$.

Обратное к нему отображение $\Psi_M\colon \hat{\mathbb{C}} \setminus \overline{\mathbb{D}} \to \hat{\mathbb{C}} \setminus \mathcal{M}$ называется отображением И. Юнграйза, точнее, алгоритмом Юнграйза, предложившего конструктивное построение этой функции (см. [19, 20, 21]):

$$\Psi_M(\omega) = \omega + \sum_{m=0}^{\infty} b_m \omega^{-m} = \omega - \frac{1}{2} + \frac{1}{8\omega} - \frac{1}{4\omega^2} + \frac{15}{128\omega^3} + \dots$$

Программируем на Maxima (листинг № 5.8):

Отображение Юнграйза для множества Мандельброта

```
/* Используем симметрию вещественной оси, Psi_M функцию и алгоритм Юнграйза */
start:elapsed_run_time()$
jMax:50$          /* точность, пропорциональна детализации и времени вычислений */
iMax:300$                                    /* количество отрисовываемых точек */
iMaxBig:400$
                                  /* вычисление коэффициентов b функции Юнграйза */
betaF[n,m]:=block
    ([nnn:2^(n+1)-1],
    if m=0 then 1.0 else if ((n>0) and (m < nnn)) then 0.0 else
       (betaF[n+1,m]-sum(betaF[n,k]*betaF[n,m-k],k,nnn,m-nnn)-betaF[0,m-nnn])/2.0
    )$
b[m]:=betaF[0,m+1]$
wn[w,j]:= if j=0 then 1 else w*wn[w,j-1]$                          /* степени w^j */
Psi_M(w):=w + sum(b[j]/wn[w,j],j,0,jMax)$       /* функция Юнграйза; c=Psi_M(w) */
            /* показательная форма комплексного числа, t - доля дуги окружности */
                    /* возвращает точку на единичной окружности для доли угла t */
GiveCirclePoint(t):=R*%e^(%i*t*2*%pi)$
               /* возвращает точку внешнего луча с радиусом R и долей угла tRay */
GiveWRayPoint(R):=R*%e^(%i*tRay*2*%pi)$
NmbrOfCurves:9$
                           /* координаты на w-плоскости (а не на c-плоскости!) */
[R_min,R_max]:[1,1.5]$
dR:(R_max-R_min)/NmbrOfCurves$                                  /* для окружностей */
dRR:(R_max-R_min)/iMax$                                            /* для лучей */
                                                 /* f_0 плоскость = w-плоскость */
R:1$                                                    /* единичная окружность */
circle_angles:makelist(i/iMax,i,0,iMax/2)$
CirclePoints:map(GiveCirclePoint,circle_angles)$
                                                        /*  внешние окружности */
circle_radii: makelist(R_min+i*dR,i,1,NmbrOfCurves)$
circle_angles:makelist(i/iMaxBig,i,0,iMaxBig/2)$
WCirclesPoints:[]$
for R in circle_radii do
    WCirclesPoints:append(WCirclesPoints,map(GiveCirclePoint,circle_angles))$
                                                                  /*  внешние w-лучи */
ray_radii:makelist(R_min+dRR*i,i,1,iMax)$
ray_angles:[0,1/3,1/7,2/7,3/7]$                 /* список углов < 1/2 для лучей */
WRaysPoints:[]$
for tRay in ray_angles do
    WRaysPoints:append(WRaysPoints,map(GiveWRayPoint,ray_radii))$
                                              /* плоскость параметра = c плоскость */
MPoints:map(Psi_M,CirclePoints)$                    /* точки множества Мандельброта */
CRaysPoints:map(Psi_M,WRaysPoints)$                            /* внешние z лучи */
Equipotentials:map(Psi_M,WCirclesPoints)$
                                     /* добавляем точки ниже вещественной оси */
```

```
for w in CirclePoints do CirclePoints:cons(conjugate(w),CirclePoints)$
for w in WRaysPoints do WRaysPoints:cons(conjugate(w),WRaysPoints)$
for w in WCirclesPoints do WCirclesPoints:cons(conjugate(w),WCirclesPoints)$
for c in MPoints do MPoints:cons(conjugate(c),MPoints)$
for c in CRaysPoints do CRaysPoints:cons(conjugate(c),CRaysPoints)$
for c in Equipotentials do Equipotentials:cons(conjugate(c),Equipotentials)$
                                                        /* затраченное время */
stop:elapsed_run_time ()$
time:fix(stop-start);
load(draw)$                                                          /* рисуем */
draw(file_name="jung", dimensions=[1000,500],
    terminal='png, columns=2,
    gr2d(title="unit circle with external rays & circles",
        point_type=filled_circle, points_joined=true,
        point_size=0.34, color=red,
        points(map(realpart, CirclePoints),map(imagpart, CirclePoints)),
        points_joined =false, color=green,
        points(map(realpart, WCirclesPoints),map(imagpart, WCirclesPoints)),
        color=black,
        points(map(realpart, WRaysPoints),map(imagpart, WRaysPoints))
        ),
    gr2d(title="Parameter plane : Image under Psi_M(w)",
        points_joined=true, point_type=filled_circle,
        point_size=0.34, color=blue,
        points(map(realpart, MPoints),map(imagpart, MPoints)),
        points_joined=false, color=green,
        points(map(realpart, Equipotentials),map(imagpart, Equipotentials)),
        color=black,
        points(map(realpart, CRaysPoints),map(imagpart, CRaysPoints))
        )
);
```

РЕСУРСЫ ИНТЕРНЕТА

[1e] http://maxima.sourceforge.net/ru/
Официальная страница проекта Maxima на русском языке (однако, она редко обновляется, рекомендуется пользоваться английской версией страницы).

[2e] http://sourceforge.net/projects/maxima/files/
Здесь можно скачать установочный файл для Maxima.

[3e] http://maxima.sourceforge.net/ru/documentation.html
Подборка руководств по Maxima с официального сайта, в том числе и на русском языке.

[4e] http://andrejv.github.com/wxmaxima/help.html
Коллекция руководств по Maxima.

[5e] http://www.uneex.ru/static/MethodBooks_Maxima/Maxima.pdf
Пособие Н.А. Стахина «Основы работы с системой аналитических (символьных) вычислений Maxima» на русском языке.

[6e] http://www3.msiu.ru/~lao4/maxima.pdf
Методическое пособие с неофициального сайта МГИУ `www3.msiu.ru`.

[7e] http://euler.us.es/~renato/clases/maxima/manualesPDF/MaximaByExamples/mbe5qdraw.pdf
Документация пакета Edwin L. Woollett **qdraw**.

[8e] http://www.csulb.edu/~woollett/
Сайт профессора Edwin L. Woollett с очень полезной информацией о Maxima.

[9e] http://www.csulb.edu/~woollett/mbe2plotfit.pdf
Туториал от профессора Edwin L. Woollett о выводе графике и работе с файлами (см. также другие его туториалы по предыдущей ссылке).

[10e] http://en.wikibooks.org/wiki/Maxima
Полезная информация о Maxima в открытой энциклопедии. Есть примеры программирования фракталов.

[11e] http://www.telefonica.net/web2/biomates/maxima/sound/index.html
Пакет работы со звуком **sound.lisp**.

[12e] http://maxima.cvs.sourceforge.net/viewvc/maxima/maxima/share/contrib/fractals/
Пакет для построения простейших фракталов **fractals.mac**. На этом же сайте есть много других пакетов для Maxima.

[13e] http://www.imagemagick.org/script/download.php
Отличная бесплатная программа Image Magick для работы с изображениями из командной строки.

[14e] http://www.biologie.uni-hamburg.de/b-online/e28_3/lsys.html
L-системы.

[15e] http://algorithmicbotany.org/
L-системы. Сайт профессора Першемыслава Прузинкевича.

[16e] http://algorithmicbotany.org/papers/#abop
Книга Прузинкевича в открытом доступе.

[17e] http://mathworld.wolfram.com/LindenmayerSystem.html
Черпаем вдохновение в готовых демонстрациях с сайта `mathworld.wolfram.com`.

[18e] http://www.austromath.at/daten/maxima/zusatz/Graphics_with_Maxima.pdf
Хорошая документация по использованию графики в Maxima.

[19e] http://riotorto.users.sourceforge.net/gnuplot/animation/
Примеры создания анимации средствами Maxima.

[20e] http://riotorto.users.sourceforge.net/gnuplot/
Хорошая подборка примеров использования графики от создателя пакета **draw** Mario Rodríguez Riotorto.

[21e] http://en.wikibooks.org/wiki/Fractals/Iterations_in_the_complex_plane/Julia_set
Алгоритмы построения и раскрашивания Множеств Жюлиа.

[22e] http://www.fraktal.republika.pl/iim.html
Реализации метода обратных итераций для построения множеств Жюлиа.

[23e] http://www.fraktal.republika.pl
Различные алгоритмы для изучения множеств Жюлиа, множества Мандельброта и др., а также полезные ссылки.

[24e] http://commons.wikimedia.org/wiki/File:Lemniscates5.png
Подробнее о Листинге 5.6.

[25e] http://commons.wikimedia.org/wiki/File:Miim.jpg
Подробнее о Листинге 5.7.

[26e] http://en.wikipedia.org/wiki/File:Jung50e.png
Подробнее о Листинге 5.8.

[27e] http://en.wikibooks.org/wiki/Fractals/Iterations_in_the_complex_plane/Julia_set#BSM.2FJ
Много различной информации об алгоритмах построения фрактальных множеств на комплексной плоскости.

[28e] http://en.wikibooks.org/wiki/Fractals/Computer_graphic_techniques/2D
Много различной информации об алгоритмах построения фрактальных множеств на комплексной плоскости.

СПИСОК ЛИТЕРАТУРЫ

(После источников указаны страницы

настоящего пособия, на которых они упоминаются)

[1] *Кроновер, Р. М.* Фракталы и хаос в динамических системах. Основы теории / Р. М. Кроновер. — М. : Постмаркет, 2000. — 352 с. с. 4, с. 16, с. 18, с. 23, с. 32, с. 34, с. 39, с. 41, с. 45, с. 46

[2] *Хомский, Н.* Введение в формальный анализ естественных языков / Н. Хомский, Д. Миллер; Под ред. А. А. Ляпунова, О. Б. Лупанова. Кибернетический сборник. — М. : Мир, 1965. с. 5

[3] *Prusinkiewicz, P.* Lindenmayer Systems, Fractals, and Plants / P. Prusinkiewicz, J. Hanan. Lecture Notes in Biomathematics no. 79. — New York : Springer-Verlag, 1989. — 120 p. с. 16, с. 23

[4] *Lindenmayer, A.* The Algorithmic Beauty of Plants / A. Lindenmayer, P. Prusinkiewicz. — New York ; Berlin ; Heidelberg : Springer-Verlag, 1996. — 228 p. (http://algorithmicbotany.org/papers/#abop). с. 16, с. 23

[5] *Мандельброт, Б.* Фрактальная геометрия природы / Б. Мандельброт. — М. : Институт компьютерных исследований, 2002. — 656 с. с. 25, с. 32, с. 46

[6] *Циллис, К.* Об измерении фрактальных размерностей по физическим свойствам / К. Циллис // В сб. статей «Фракталы в физике». — М. : МИР, 1988. с. 25, с. 32

[7] *Встовский, Г. В.* Введение в мультифрактальную параметризацию структур материалов / Г. В. Встовский, А. Г. Колмаков, И. Ж. Бунин. — М. ; Ижевск : НИЦ «Регулярная и хаотическая динамика», 2001. — 116 с. с. 25, с. 32

[8] *Falconer, K. J.* Fractal Geometry: Mathematical Foundations and Applications / K. J. Falconer. — 2nd edition. — New York : John Wiley & Sons, 2003. — 328 p. с. 26, с. 32, с. 39, с. 46

[9] *Пайтген, Х.-О.* Красота фракталов. Образы комплексных динамических систем / Х.-О. Пайтген, П. Х. Рихтер. — М. : МИР, 1993. — 176 с. с. 33, с. 46

[10] *Julia, G.* Mémoire sur l'itération des fonctions rationnelles / G. Julia // *Journal de mathématiques pures et appliquées* 8^e *série.* — 1918. — Vol. 1. — Pp. 47–246. с. 33

[11] *Fatou, P.* Sur les equations fonctionelles / P. Fatou // *Bulletin de la Société Mathématique de France.* — 1919–1920. — Vol. 47–48. с. 33

[12] *Caley, A.* The Newton-Fourier imaginary problem / A. Caley // *American Journal of Mathematics.* — 1879. — Vol. 2. — 97 p. с. 33, с. 42

[13] *Cremer, H.* Über die iteration rationaler funktionen / H. Cremer // *Jahresberichte der Deutschen Mathematischen Vereinigung.* — 1925. — Vol. 33. — Pp. 185–210. с. 33

[14] *Brooks, R.* The dynamics of 2-generator of subgroups of PSL(2,$\mathbf{c}$) / R. Brooks, J. P. Matelski // Proceedings of the 1978 Stony Brook Conference of «Riemann surfaces and related topics» / Ed. by Kra, Maskit. — 1978. — Pp. 65–71. с. 33

[15] *Drakopoulos, V.* Comparing rendering methods for julia sets / V. Drakopoulos // WSCG'02. — 2002. — Pp. 155–162. с. 40, с. 45, с. 46

[16] *Saupe, D.* Efficient computation of Julia sets and their fractal dimension / D. Saupe // *Physica D.* — 1987. — Vol. 28. — Pp. 358–370. с. 40, с. 46

[17] *Fatou, P.* Sur les solutions uniformes de certaines equations fonctionnelles / P. Fatou // *Comptes Rendus.* — 1906. — Vol. 143. — Pp. 546–548. с. 41

[18] *Mandelbrot, B. B.* Fractal aspects of the iteration of $z \to \lambda z(1-z)$ for complex λ and z / B. B. Mandelbrot // Nonlinear Dynamics, Annals New York Acad. Sciences. — 1980. — Pp. 249–259. с. 41

[19] *Douady, A.* Itération des polynômes quabratiques complexes / A. Douady, J. H. Hubbard // *C. R. Acad. Sci.* — 1982. — Vol. 294, no. 3. — Pp. 123–126. с. 57

[20] *Еременко, А. Э.* Динамика аналитических преобразований / А. Э. Еременко, М. И. Любич // *Алгебра и анализ.* — 1989. — Т. 1, № 3. — С. 1–70. с. 57

[21] *Jungreis, I.* The uniformization of the complement of the Mandelbrot set / I. Jungreis // *Duke Math. J.* — 1985. — Vol. 52, no. 4. — Pp. 935–938. с. 57

[22] *Barnsley, M. F.* Fractals everywhere, second edition / M. F. Barnsley. — Boston : Academic Press, 1993. — 550 p.

[23] *Barnsley, M. F.* Superfractals / M. F. Barnsley. — Cambridge : Cambridge University Press, 2006. — 453 p.

УКАЗАТЕЛЬ КОМАНД

Замечания и пожелания можно отправлять автору по адресу Paul.Troshin@gmail.com