МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ СИСТЕМ І ПРОЦЕСІВ

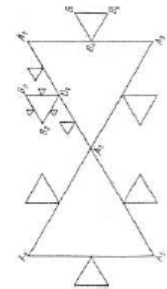
Лекції 4. Моделювання ітераційних процесів. Множини Жюліа та Мандельброда

Київ – 2015

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* $z \notin \mathbf{J}(f_c) \quad \left(\lim\limits_{n\to\infty} f_c^{(n)}(z) = \infty\right)$.

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* $\mathbf{J}(f_c)$

$path = \emptyset$; ✎ *массив точек для отображения*
$f_c(z) = z^2 + c$; ✎ *инициализация итерируемой функции*
для $m = 1, \ldots, p$
  $x = a - \frac{s}{2} + m\frac{s}{p}$;
  для $n = 1, \ldots, p$
    $y = b - \frac{s}{2} + n\frac{s}{p}$;
    $z = x + iy$;
    повторять $ITER$ **раз**
      $z = f_c(z)$;
      если $|z| \geq 2$, **то** выход из цикла;
    если $|z| < 2$, **то** $path = path \cup \{x, y\}$; ✎ *добавить точку в массив для отображения*

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

```
c:%i$
f(z):=expand(z^2+c),float$
[a,b]:[0,0]$
s:3$
p:500$
ITER:10$

path:[]$

for m:1 thru p do
    (x:float(a-s/2+m*s/p),
    for n:1 thru p do
       (y:float(b-s/2+n*s/p),
        z:x+%i*y,
        thru ITER do
            (z:f(z),
            if abs(z)>=2 then return()
            ),
        if abs(z)<2 then path:endcons([x,y],path)
        )
    )$
load(draw)$
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 $J(f_c)$*

```
1:  ITER = 10;                                  ✎ количество итераций для испытания точки
2:  path = ∅;                                   ✎ массив точек для отображения
3:  E = нулевая матрица размера 3 × p;          ✎ матрица-стек множества J(f_c)
4:  f_c(z) = z^2 + c;                           ✎ инициализация итерируемой функции
5:  для n = 1, 2
6:     y = b − s/2 + n s/p;
7:     для m = 1, ..., p
8:        x = a − s/2 + m s/p;
9:        z = x + yi;
10:       повторять ITER раз
11:          z = f_c(z);
12:          если |z| ≥ 2, то выход из цикла;
13:       если |z| < 2, то E_nm = 1;
14: для n = 3, ..., p
15:    y = b − s/2 + n s/p;
16:    для m = 1, ..., p
17:       x = a − s/2 + m s/p;
18:       z = x + yi;
19:       повторять ITER раз
20:          z = f_c(z);
21:          если |z| ≥ 2, то выход из цикла;
22:       если |z| < 2, то E_3m = 1;
23:       иначе E_3m = 0;
```

24: **для** $i = 2, \ldots, p-1$

25: **если** $E_{2i} = 1 \quad \& \quad E_{1i} \cdot E_{2,i-1} \cdot E_{2,i+1} \cdot E_{3i} = 0$, **то**

26: $path = path \cup \{a - \frac{s}{2} + i\frac{s}{p}, b - \frac{s}{2} + (n-1)\frac{s}{p}\}$

27: $E_1 = E_2, E_2 = E_3;$ ✎ *меняем первую строку матрицы на вторую, вторую на третью*

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:

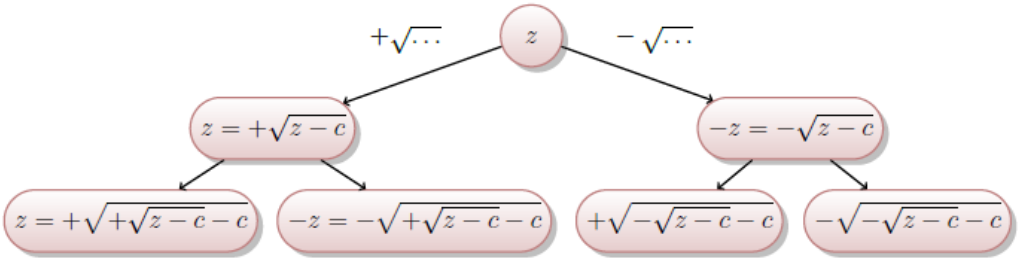
$$A(w) = \{z \in \mathbb{C} : f^{(n)}(z) \to w \text{ при } n \to \infty\}.$$

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 $z = +\sqrt{z-c}$):



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 $J(f_c)$*

$f_c(z) = z^2 + c$; ✎ *итерируемая функция*

$F_c(z) = +\sqrt{z - c} \cup -\sqrt{z - c}$; ✎ *прообраз для f_c, z может быть множеством точек*

Находим z_i — корни уравнения $f_c(z) = z$;

Находим z — максимальный по модулю из этих корней;

$path = \{z\}$; ✎ *инициализируем массив точек для отображения*

повторять $ITER$ **раз**

    $path = F_c(path)$

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`.

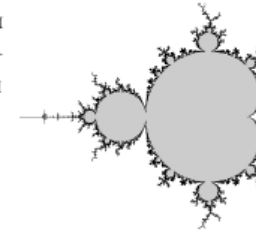
Замечания об использованных командах:

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

Множество Мандельброта $\mathcal{M}$

```
f(z):=expand(z^2+cx+cy*%i),float$
[a,b]:[-0.5,0]$
s:3$
p:400$
ITER:15$

path:[]$
for m:1 thru p do
    (cx:float(a-s/2+m*s/p),
    for n:1 thru p do
        (cy:float(b-s/2+n*s/p),
        z:0,
        thru ITER do
            (z:f(z),
            if abs(z)>=2 then return()
            ),
        if abs(z)<2 then path:endcons([cx,cy],path)
        )
    )$
load(draw)$
wxdraw2d(point_type=dot,points(path))$
```

4.3 Проблема Кэли

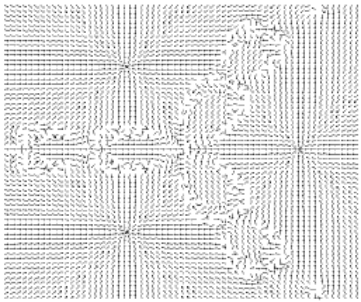
В 1879 г. сэр Артур Кэли в короткой заметке [12], меньше чем на страницу, поставил задачу нахождения областей притяжения корней уравнения $f(z) = 0$ при использовании классического метода Ньютона-Фурье, распространенного на комплексную плоскость. Эта заметка стимулировала вышеупомянутые исследования 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 общем случае не является полиномом, то для этой функции множество Жюлиа можно определить как замыкание множества отталкивающих точек.

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 $Vectors_{ij}$.

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.

Завдання для лабораторної роботи № 4

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 ($f_c(c) = \overline{f_{\overline{c}}(\overline{c})}$). 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.

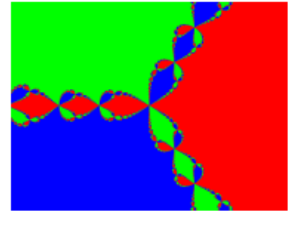
Задача 4.12 Можно существенно ускорить Алгоритм 4.3, привнеся в него элемент случайности: на каждом шаге итерации в строчке 7 вместо отображения F_c применять одно из отображений $+\sqrt{z-c}$ или $-\sqrt{z-c}$, выбранное случайно.

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 $f(z) = z^3 - z$.