Министерство науки и высшего образования РФ
ФГБОУ ВО «Удмуртский государственный университет»
Институт математики, информационных технологий и физики

Лабораторная работа

"Построение конструктивных фракталов"

Выполнил: студент 4 курса
группы ОАБ-02.03.01-42
Лаврентьев Михаил Андреевич

Ижевск
2022

Задание

Написать программу для построения конструктивных фракталов.

Конструктивные фракталы

Конструктивные или геометрические фракталы являются самыми наглядными, в них самоподобие видно сразу. Для построения геометрических фракталов характерно задание "основы" и "фрагмента", повторяющегося при каждом уменьшении масштаба. Методика: сначала изображается основа. Затем некоторые части основы заменяются на фрагмент. На каждом следующем этапе части уже построенной фигуры, аналогичные замененным частям основы, вновь заменяются на фрагмент, взятый в подходящем масштабе. Всякий раз масштаб уменьшается. Для получения самого фрактала нужно бесконечное число этапов. Меняя основу и фрагмент, можно получать разные геометрические фракталы. Рассмотрим построение классических фракталов.

Кривая Леви

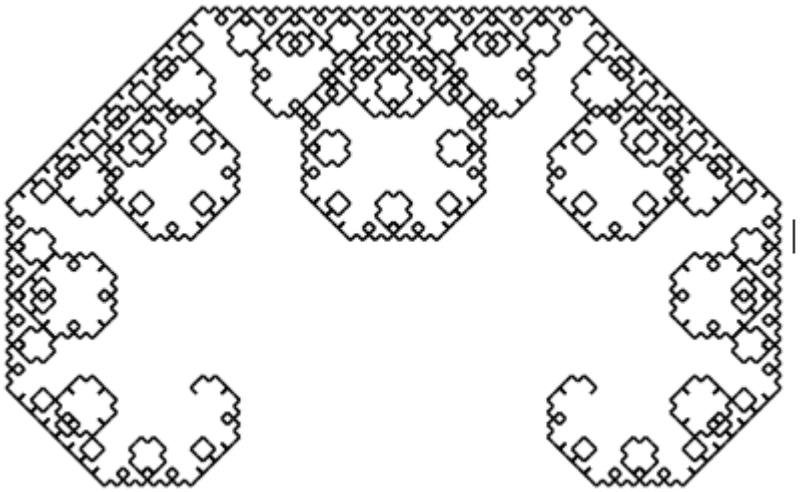
Кривая Леви — фрактал. Впервые изучено итальянцем Эрнесто Чезаро в 1906 году. Однако его самоподобие и фрактальные свойства исследовал французский математик П. Леви в 1930-х годах.

За сходство с буквой «С», написанной витиеватым шрифтом, ее еще называют С-кривой Леви.

```
> restart; with(plots) : with(plottools) :
```

Кривая Леви

```
> _Levy := proc(lst, n)
    local shifted, temp, result, i :
      if n = 0 then
          lst :
      else
          temp := _Levy(lst, n - 1) :
          result := [temp[1]] :
          for i from 2 to nops(temp) do
              shifted := temp[i - 1] + 1/2 · (temp[i] - temp[i - 1]) + 1/2 · [-(temp[i] - temp[i - 1])[2], (temp[i]
    - temp[i - 1])[1]] :
              result := [op(result), shifted, temp[i]] :
          end do:
          result :
      end if:
  end proc:
> _Points := [[0, 0], [1, 0]] :
  result := _Levy(_Points, 11) :
> display([seq(line(result[i], result[i + 1], color = black), i = 1 ..nops(result) - 1)], scaling = constrained, axes = none);
```



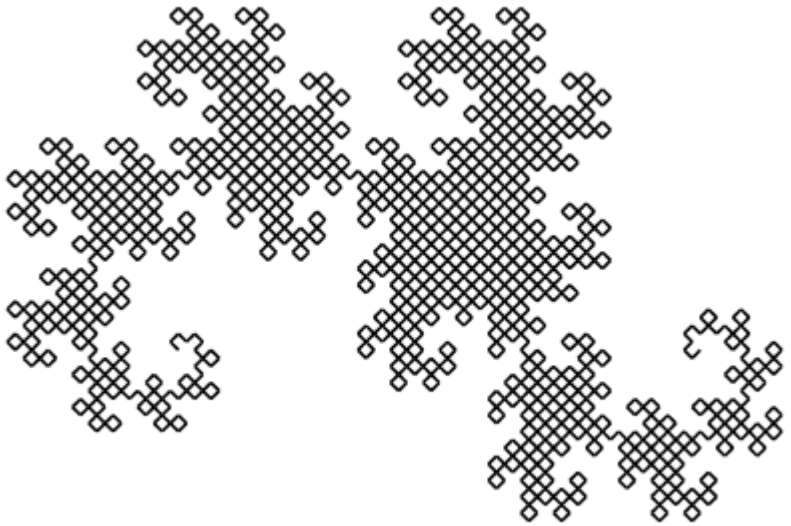
Кривая Дракон Хартера-Хейтуэя

Дракон Хартера, также известный как дракон Хартера — Хейтуэя, был впервые исследован физиками NASA — John Heighway, Bruce Banks, и William Harter. Кривая дракона принадлежит к семейству некоторых фрактальных кривых, которые могут быть получены рекурсивными методами. Дракон Хартера был описан в 1967 году Мартином Гарднером (Martin Gardner) в колонке «Математические игры» журнала «Scientific American». Многие свойства фрактала были описаны Чандлером Девисом и Дональдом Кнутом.

```
> restart; with(plots) : with(plottools) :
```

Кривая Дракон Хартера-Хейтуэя

```
> _Levy := proc(lst, n)
    local shifted, temp, result, i :
      if n = 0 then
          lst :
      else
          temp := _Levy(lst, n - 1) :
          result := [temp[1]] :
          for i from 2 to nops(temp) do
              shifted := temp[i - 1] + 1/2 · (temp[i] - temp[i - 1]) + (-1)^i/2 · [-(temp[i] - temp[i - 1])[2], (temp[i]
    - temp[i - 1])[1]] :
              result := [op(result), shifted, temp[i]] :
          end do:
          result :
      end if:
  end proc:
> _Points := [[0, 0], [1, 0]] :
  result := _Levy(_Points, 11) :
> display([seq(line(result[i], result[i + 1], color = black), i = 1 ..nops(result) - 1)], scaling = constrained, axes = none);
```



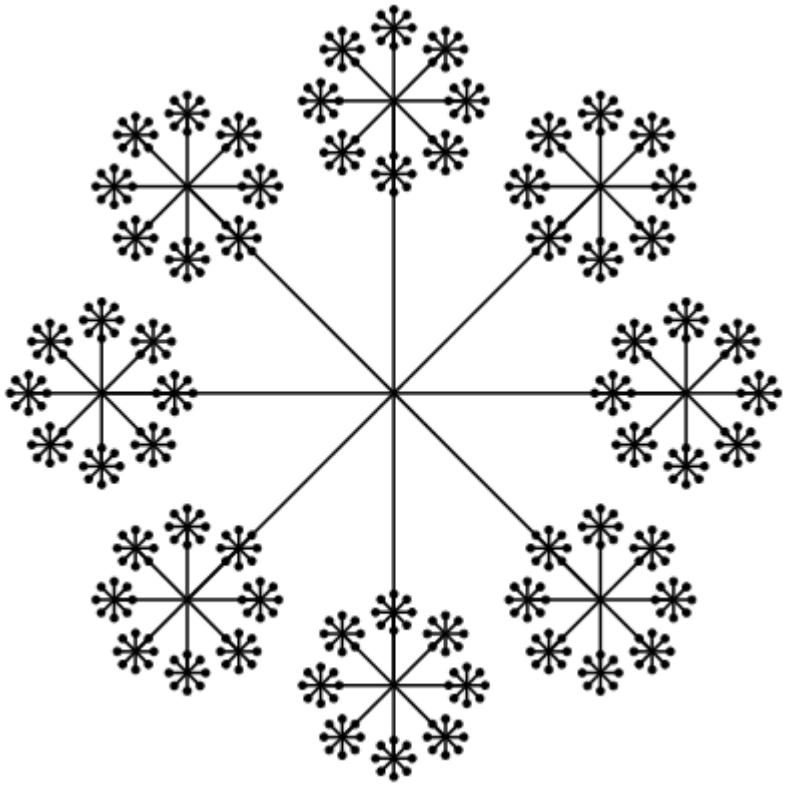
Снежинка

Для построения фрактальной снежинки необходимо построить k-лучевую звезду с центром в заданной точке (список cntr в программном коде) и длинной лучей равной r. Далее, конец каждого луча используется в качестве центра звезд следующего поколения, длина лучей которых меньше родительской. Получаем k штук звезд второго поколения. Далее на концах всех лучей всех звезд первого поколения строим звезды второго поколения и т.д. Ниже приведен программный код генерации фрактальной снежинки и пример трех итераций фрактала.

```
> restart; with(plots) : with(plottools) :
```

Фрактальная снежинка

```
> _snowflake := proc(cntr, r, k, n)
    global graph :
    local t, i, temp :
    t := 2 · Pi / k;
    for i from 1 to k do
    temp := [cntr[1] + r·cos(i·t), cntr[2] + r·sin(i·t)] :
    graph := [op(graph), line(cntr, temp)] :
    if n ≥ 1 then _snowflake(temp, 0.25 · r, k, n − 1)
    end if:
    end do:
    end proc:
> cntr := [0, 0] :
  graph := [ ] :
  _snowflake(cntr, 10., 8, 3) :
> display(graph, scaling = constrained, axes = none);
```

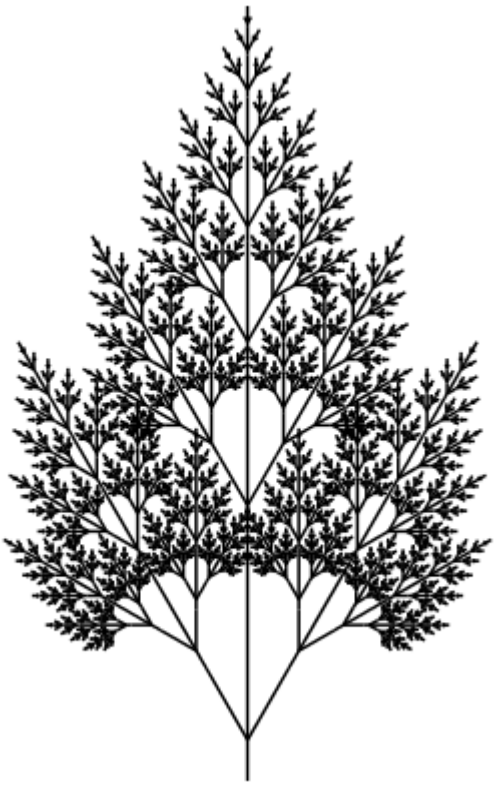


Фрактальная ветка

```
> restart; with(plots) : with(plottools) :
```

Фрактальная ветка

```
> _branch := proc(root, side, angle, n)
   global graph :
   local end1, end2, end3, root2, root3 :
   end1 := root + side·[cos(angle), sin(angle)] :
   root2 := root + 1/6·side·[cos(angle), sin(angle)] :
   end2 := root2 + 1/3·side·[cos(angle + Pi/6.), sin(angle + Pi/6)] :
   root3 := root + 1/6·side·[cos(angle), sin(angle)] :
   end3 := root3 + 1/3·side·[cos(angle−Pi/6.), sin(angle − Pi/6)] :
   graph := [op(graph), line(root, end1), line(root2, end2), line(root3, end3)] :
   if n ≥ 1 then
   _branch(end1, 0.7·side, angle, n − 1) :
   _branch(end2, 0.5·side, angle + Pi/6, n − 1) :
   _branch(end3, 0.5·side, angle − Pi/6, n − 1) :
   end if:
   end proc:
> cntr := [0, 0] :
   graph := [ ] :
   _branch(cntr, 100, Pi/2, 7) :
> display(graph, scaling = constrained, axes = none);
```

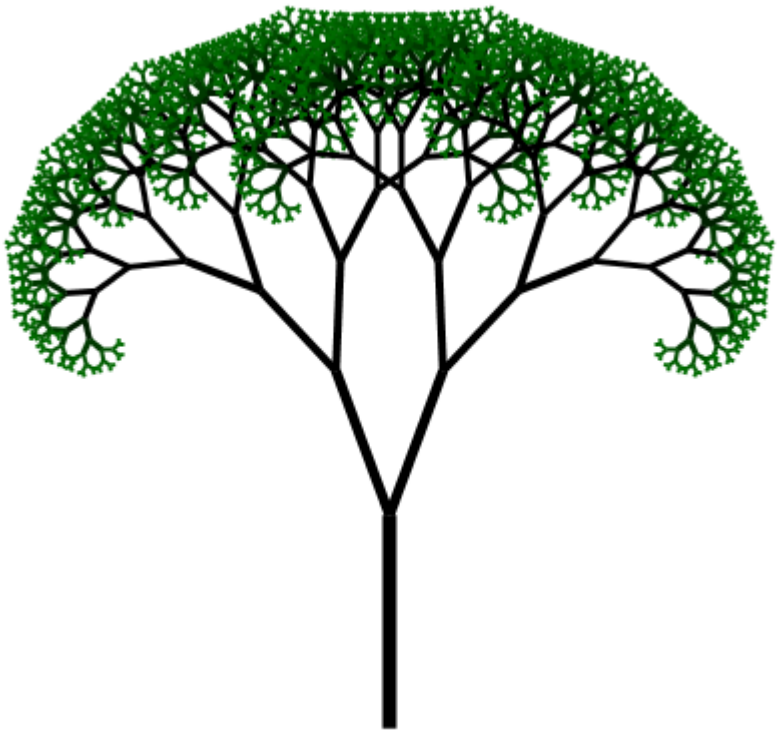


Фрактальное дерево

```
> restart; with(plottools) : with(plots) :
```

Фрактальное дерево

```
> Deflection := 600/1000·Pi/3. :
  Scale := 400/1000·0.3 + 0.6 :
  _tree := proc(root, side, angle, n)
  global graph :
   local temp :
  if side > 1.8 then
  temp := root + side·[cos(angle), sin(angle)] :
  graph := [op(graph), line(root, temp, color = COLOR(RGB, 0, 1 - 7/n, 0), thickness = trunc(1 + 6/sqrt(n)))] :
  _tree(temp, Scale( )·side, angle + (Pi/10 + Deflection( )·n/15), n + 1) :
  _tree(temp, Scale( )·side, angle - (Pi/10 + Deflection( )·n/15), n + 1) :
  end if:
   end proc:
  cntr := [0, 0] :
   graph := [ ] :
   _tree(cntr, 80., Pi/2., 1) :
   display(graph, scaling = constrained, axes = none);
```



Множество Кантора

В 1883 году немецкий математик Георг Кантор описал множество, которое теперь называют его именем. Оно конструируется следующим образом. Берется единичный отрезок. Этот отрезок делится на три части, средняя удаляется. Получается два отрезка длиной ⅓ каждый. Далее с каждым из отрезков проделывается та же процедура (удаление серединной части); и так до бесконечности. Полученное множество состоит из точек отрезков не удаленных алгоритмом и является самоподобным (его фрагмент выглядит как все множество в целом.

Построение множества Кантора приведем с помощью Maple

```
> restart; with(plots) :
```

Множество Кантора

Процедура ***basic*** получает на входе координаты концов исходного отрезка и возвращает концевые точки двух производных отрезков

```
> n := 3;
  basic := proc(p1, p2)
    local dx, p3, p4, p5, p6 :
    dx := (p2[1] - p1[1]) / n :
    p3 := [p1[1] + dx, p1[2]] :
    p4 := [p2[1] - dx, p2[2]] :
    p1, p3, p4, p2;
  end proc:
```

$$n := 3 \qquad (1.1)$$

```
> pt := basic([0, 0], [1, 0]);
```

$$pt := [0, 0], \left[\frac{1}{3}, 0\right], \left[\frac{2}{3}, 0\right], [1, 0] \qquad (1.2)$$

Процедура ***m_k*** берет список точек, которые представляют собой концы отрезков исходного множества, и возвращает список, в котором каждый отрезок заменен двумя, посредством процедуры ***basic***

```
> m_k := proc( )
    local i, lin, k_0, j :
    k_0 := NULL :
  for i by 2 to nargs - 1 do
      k_0 := k_0, basic(args[i], args[i + 1]) :
   end do:
  k_0 :
  end proc:
```

Например, обратимся к ***m_k*** со списком ***pt***

```
> m_k(pt);
```

$$[0, 0], \left[\frac{1}{9}, 0\right], \left[\frac{2}{9}, 0\right], \left[\frac{1}{3}, 0\right], \left[\frac{2}{3}, 0\right], \left[\frac{7}{9}, 0\right], \left[\frac{8}{9}, 0\right], [1, 0] \qquad \textbf{(1.3)}$$

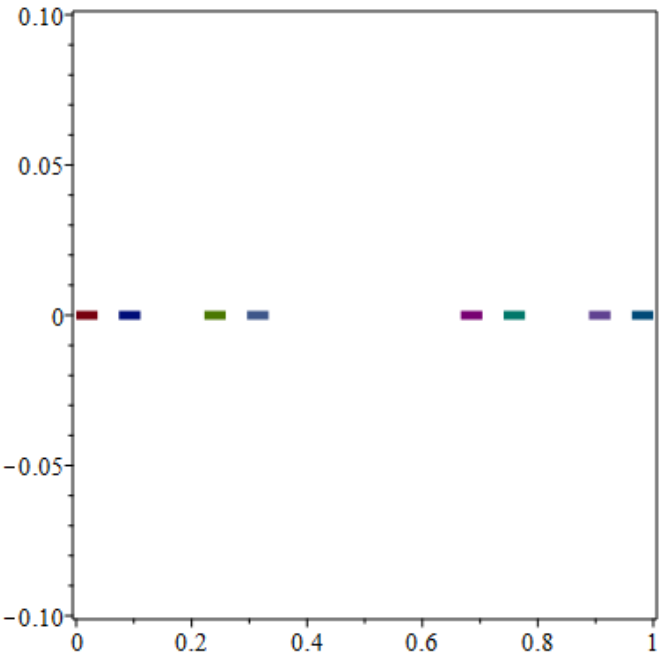
Теперь можно определить ***n-й*** шаг построения множества Кантора

```
> kantor := proc(n::integer)
    local i,j,lin,kan,kant:
    lin := [0, 0], [1, 0]:
    for i to n do lin := m_k(lin) end do:
    j := 1:
    for i by 2 to nops([lin]) do
      kan[j] := [lin[i], lin[i + 1]]:
      j := j + 1:
  end do:
  kant := convert(kan, list);
  plot(kant, axes = box, thickness = 5, view = [0..1, -0.1..0.1]):
  end proc:
```

Обратившись к процедуре ***kantor***, сделаем три шага построения множества

```
>
  K := 3;
  kantor(K);
```

$$K := 3$$



Кривая Коха

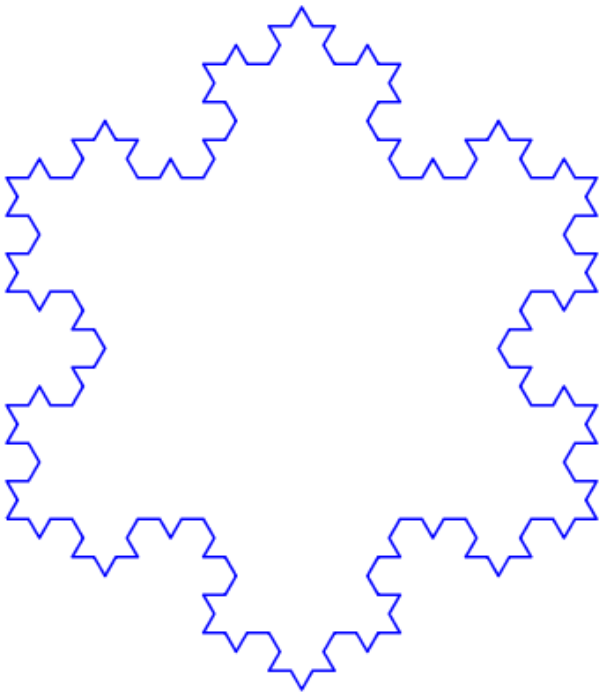
Норвежский математик Гельге фон Кох в 1904 году описал одну кривую линию, необычную тем, что она не имеет касательной ни в одной своей точке. Эта кривая определяется как предел последовательности кривых, порождаемых процедурами, аналогичными описанными выше (для множества Кантора). Точно так же рассматривается отрезок единичной длины, из которого удаляется средняя треть. Отличие в том, что разорванный отрезок дополняется двумя отрезками длиной ⅓ , превращаясь в ломаную из четырех звеньев. Далее та же самая операция применяется к каждому из отрезков ломаной. Следующий набор процедур позволяет отображать замкнутую кривую (снежинку) Коха.

```
> restart; with(plots) :
```

Снежинка Коха

```
> basic := proc(p1, p2)
    local dx, dy, p3, p4, p5 :
    dx := (p2[1] - p1[1])/3 : dy := (p2[2] - p1[2])/3 :
    p3 := [p1[1] + dx, p1[2] + dy] :
    p4 := [p1[1] + 2·dx, p1[2] + 2·dy] :
    p5 := [p1[1] + 1.5·dx - dy·sqrt(3)/2, p1[2] + 1.5·dy + dx·sqrt(3)/2] :
    p3, p5, p4, p2;
  end proc:
> flake := proc(fl)
    local i, curve;
    curve := fl[1] :
    for i to nops(fl) - 1 do
      curve := curve, basic(fl[i], fl[i + 1]) :
    end do:
  end proc:
> snowflake := proc(n)
    local i, curve :
    curve := [[-0.5, 0], [0, sqrt(3)/2], [0.5, 0], [-0.5, 0]] :
    for i from 2 to n do curve := [flake(curve)] od;
  plot(curve, color = blue, scaling = constrained, axes = none) :
  end proc:
```

```
> snowflake(4);
```



Кривая Пеано

Особенность следующей непрерывной кривой, которую описал в 1891 году Давид Гильберт, состоит в том, что она заполняет собою некоторое заданное пространство, таким образом, в пределе можно сказать, что ее длина составляет определенное количество квадратных метров.

```
> restart; with(plots) :
```

Кривая Пеано

```
> basic := proc(p1, p2, p3)
    local p4, p5, p6, p7, p8, p9;
  p4 := 0.5·(p1 + p2);
  p9 := 0.5·(p2 + p3);

  p5 := p4 + (p2-p9);
  p6 := p2 + (p2 − p9);
  p7 := p2 + (p4 − p1);
  p8 := p9 + (p4 − p1);
  p4, p5, p6, p2, p7, p8, p9, p3;
  end proc:
> peano := proc(fl)
    local i, cur;
  cur := [fl[1]];
  for i by 2 to nops(fl) − 2 do
    cur := [op(cur), basic(fl[i],fl[i + 1],fl[i + 2])]
  end do;
  end proc:
```

Начальная ломаная:

```
> fl := [[0, 0], [1, 1], [2, 0]] :
```

Выполним четыре шага построения кривой Пеано

```
> for i to 4 do fl := peano(fl) end do:
```

```
> plot(fl, style = line, scaling = constrained);
```