Узоры и фракталы на Python

В. Ф. Очков, А. И. Тихонов

Национальный исследовательский университете «МЭИ»
111250, Москва, Красноказарменная 14

e-mail: ochkov@twt.mpei.ac.ru, TikhonovAI@mpei.ru

Аннотация. В статье рассмотрено применение STEM- и STEAM-технологий при освоении студентами минимального подмножества Python для решения научно-технических и инженерных задач. Конструкции языка вводятся параллельно с генерацией и визуализацией узоров и фракталов. Рассматриваются возможности геймификации для освоения библиотек экосистемы Python, при проведении многовариантных расчетов, визуализации и интерпретации полученных результатов.
Ключевые слова: STEM, STEAM, Python, узор, фрактал.

1. Введение

В настоящее время в качестве инструментальной платформы STEM-технологий [1] используются такие математические программные системы как проприетарные Mathematica, Mathcad, Maple, Matlab, либо свободно распространяемые Octave, SMath, Scilab, Sage. Здесь рассматривается возможность использовать для этих целей экосистему Python.

Экосистема Python широко применяется в машинном обучении, науке о данных (Data Science), а сам язык программирования выходит, по разным оценкам, на второе-четвертое места по популярности языков программирования [2], а по некоторым оценкам лидирует в образовательной сфере [3].

Основной причиной распространенности экосистемы являются: 1) наличие огромного числа библиотек для решения практических задач, включая вычислительную математику, научную визуализацию, системное администрирование, веб-приложения; 2) бесплатность; 3) простота и низкий порог вхождения, ортогональность языка, заключающаяся в том, что для решения конкретных задач необходимо ограниченное подмножество языковых средств, которое можно расширять по мере необходимости.

Популярности платформы способствует использование удобных сред для проведения расчетов и интерпретации результатов, включая Jupyter Notebook (JN) и JupyterLab (JL), реализующих метафору вычислительного документа и позволяющих в любом современном браузере записывать условия задачи с помощью форматированного текста (HTML, Markdown), работать с изображениями, видео, формулами (Latex), проводить многовариантные расчеты,

визуализировать их результаты, даже создавая анимации, подготавливать отчеты в форматах html и pdf, создавать приложения с графическим пользовательским интерфейсом. Кроме того, встроенный модуль позволяет без дополнительных усилий превратить документ JN в презентацию, используемую, например, при защите типового расчета или квалификационной работы.

К этому следует добавить, что при использовании дистрибутивов WinPython [4] для Windows и Anaconda [5] для Linux, Mac OS X, Windows рассмотренная выше функциональность доступна «из коробки» после установки дистрибутива. Следует отметить, что простота и доступность работы с экосистемой практически не вызывает вопросов и в условиях пандемии короновируса…

В настоящее время существует достаточно много учебников различного уровня, посвященных решению научно-технических и инженерных задач с использованием экосистемы Python [6–9]. Среди них следует отметить классический учебник [6], не переведенный, к сожалению, на русский язык.

В то же время мы вынуждены отметить, что подготовка студентов последние три года снижается особенно в области освоения навыков программирования и решения даже простых задач.

Именно поэтому мы попытались сделать курс как можно более наглядным, отказались от лекций в пользу практических занятий, увеличили число решаемых студентами задач. Практически обязательной стала визуализация результатов решения задач. В то же время оказалось, что достаточно много времени стало уходить на обсуждение физических постановок задач и фактически повторение пройденного на первых трех семестрах. Это привело к решению увеличить число геометрических задач и иллюстрировать излагаемый материал графическими образами. Например, работа с двумерными массивами NumPy строится на преобразовании изображений в градациях серого, для работы с трехмерными массивами добавляется цвет. Даже достаточно сложный материал, связанный с использованием сингулярного разложения, удалось рассмотреть, анализируя влияние числа и величин удерживаемых сингулярных чисел на качество изображения.

Одним из навыков, приобретаемых студентами, является умение искать и использовать библиотеки экосистемы. Поэтому на протяжении всего курса мы стараемся демонстрировать дилемму велосипеда, сводящуюся к тому, что если не найдены, установлены, опробованы и применены созданные сообществом средства для решения задач, то приходится изобретать велосипед. Практически для всех библиотек экосистемы доступны исходные тексты, в связи с этим нужным и полезным навыком является умение читать, а иногда и дорабатывать исходные тексты. Мы работаем с инженерами-электриками и инженерами электронной

техники, которым необходимо уметь рисовать электрические схемы. В течение пары минут на практическом занятии отыскивается библиотека SchemDraw, она устанавливается на компьютеры студентов, еще 20 минут уходит на чтение документации и выполнение примеров, после чего оказывается, что применяемые в библиотеке обозначения некоторых элементов электрических и электронных схем отличаются от отечественных. После этого совместными усилиями анализируется структура библиотеки и принципы рисования элементов схем. Важным побочным результатом является приобретение навыков оформления исходных текстов программ. Результатом занятия является минимальная библиотека отечественных элементов схем, создаваемая студентами по индивидуальным заданиям. В свою очередь это дает возможность рисования электрических и электронных схем при выполнении контрольных работ и типовых расчетов.

Примерно половина студентов испытывает трудности по декомпозиции исходной задачи на последовательность подзадач, решаемых с помощью библиотечных процедур, и сборки решения с помощью написания простого исходного текста для преобразования результатов решения одной подзадачи в исходные данные для решения следующей подзадачи. Эта трудность носит в основном психологический характер и для ее преодоления осуществляется разбор типовых задач, выполнение примеров, наработка стандартных приемов преобразования данных.

Наконец, даже у магистров первого курса отсутствуют навыки локализации и исправления ошибок, элементарные приемы тестирования. Эти пробелы ликвидируются с помощью простого приема, когда исходный текст с ошибками демонстрируется группе с помощью проектора, а поиск и исправление осуществляется путем совместных усилий всей группы.

Еще одним эффективным, но достаточно трудоемким для преподавателя приемом является «разбор полетов» после контрольных, когда нерешенные задачи разбираются, но вместо нерешенной задачи предлагаются решить не одну, а две аналогичные задачи.

2. Визуализация при знакомстве минимальным подмножеством Python

Уже на первом занятии студенты должны получить простой, но видимый результат решения задачи, поэтому начинаем с импорта библиотечных модулей, функций, данных и визуализации. В качестве средства визуализации для начального обучения используется собственная реализация так называемой черепашьей графики. Дело в том, что стандартная библиотека turtle использует Tk в качестве графического интерфейса, не встраивается в блокноты JN, поэтому было решено

реализовать возможности этой библиотеки на основе matplotlib, основного средства визуализации экосистемы. Это, кроме всего прочего, обеспечивает возможности масштабирования рисунков и облегчает переход к интенсивному использованию этой библиотеки.

Любой язык программирования осуществляет последовательность действий над объектами и преобразований данных. В случае черепашьей графики объектом служит черепашка, которая перемещается по документу, оставляя след заданного цвета и толщины, если она опущена, и перемещается, не оставляя следа, если черепашка поднята. Командами служат поднять, опустить, переместить вперед или назад на заданное число пикселов, повернуть черепашку вправо или влево, перейти в точку плоскости с заданными координатами, задать толщину и цвет следа, сохранить нарисованный рисунок. В терминах такого микроязыка программирования студентам предлагается нарисовать различные фигуры. При рисовании многоугольников выясняется, что крайне необходимы языковые конструкции для выполнения последовательности одинаковых действий, после чего рассматриваются циклы.

Для обеспечения возможности многократного рисования фигур, отличающих друг от друга положением, вводятся функции, рисуются правильные многоугольники и звезды. При этом обыгрываются возможности использования позиционных, именованных параметров, значений параметров по умолчанию. На рис. 1 представлены пятиконечные звезды, повернутые и наложенные друг на друга [10].

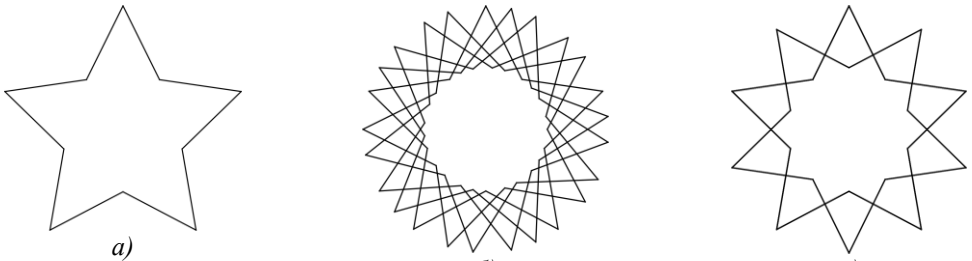


Рисунок 1. Пятиконечные звезды
а) одна звезда; б) пять звезд, повернутых друг относительно друга на 45 градусов;
в) пять звезд, повернутых друг относительно друга на 36 градусов

Смещение и повороты друг относительно друга многоугольников и звезд дает практически неисчерпаемый источник узоров, которых можно нарисовать с помощью функции, укладывающейся в десять строк исходного текста (рис. 2).

Обсуждение полученных результатов показывает, что мы создали функцию — генератор узоров, который позволяет генерировать практически бесконечное количество узоров из многоугольников, достаточно менять число их сторон, количество, взаимное расположение. При этом одни узоры кажутся привлекательными, а другие нет. Если написать программу — генератор узоров достаточно просто, то отбирать красивые узоры приходится вручную. Попытка написать программу распознавания красивых узоров на основе методов машинного обучения не привела к удовлетворительным результатам.

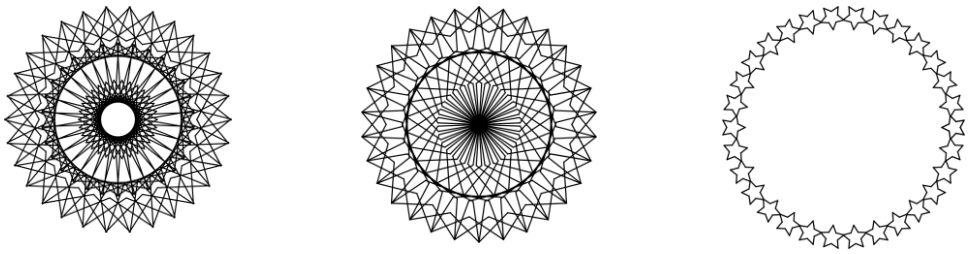


Рисунок 2. Узоры из 30 пятиконечных звезд с различным взаимным расположением друг относительно друга

Здесь мы от технологии STEM переходим к STEAM (Science, Technology, Engineering, Art, Mathematics — синтез науки, технологий, инженерного дела, искусства и математики [11]).

Далее проводился эксперимент по определению того, сколько сторон должно быть у многоугольника, чтобы считать его кругом. Оказалось, что для работы за монитором пятидесяти сторон вполне достаточно.

В качестве домашнего задания моделировалось простое устройство, называемое спирографом [12]. Спирограф представляет собой пластину с вырезанной окружностью радиуса R, по которой без проскальзывания катится круг радиуса r. В этом круге на расстоянии ρ от центра имеется отверстие, в которое вставлено перо, оставляющий след. Этот след образует узор. На рис. 3 представлена схема спирографа, нарисованная при выполнении домашнего задания.

Уравнение, описывающее траекторию следа, имеет вид:

$$\begin{aligned} x &= (R-r)\cos\alpha + \rho\cos\left(\frac{\alpha R}{r}\right), \\ y &= (R-r)\sin\alpha + \rho\sin\left(\frac{\alpha R}{r}\right). \end{aligned} \tag{1}$$

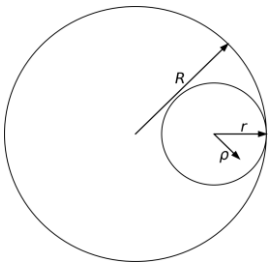


Рисунок 3. Схема спирографа

Сразу отметим, что при $r < 0$ круг с этим радиусом перемещается по внешней стороне окружности с радиусом R. Спирограф позволяет использовать несколько перьев, расположенных на различном расстоянии от центра круга радиуса r. Естественно, что перья рисуют траектории различными цветами, на рис. 4 представлены четыре узора, нарисованные программной моделью спирографа.

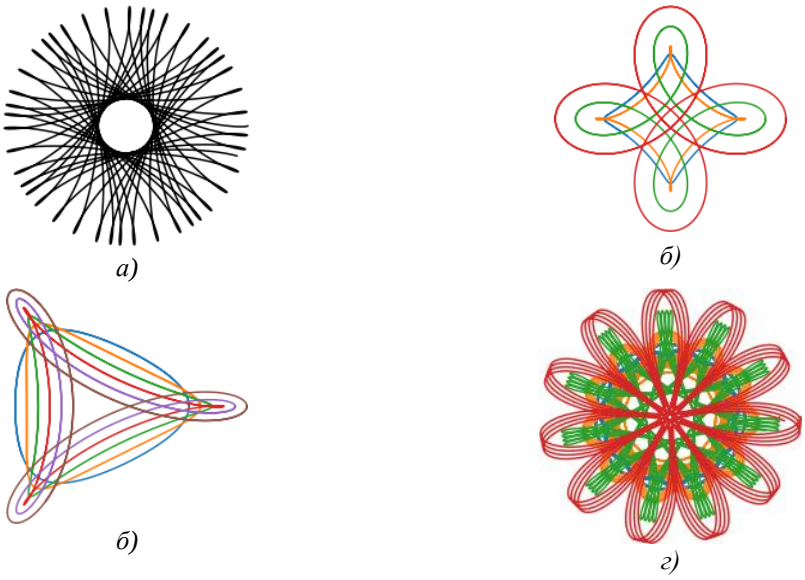


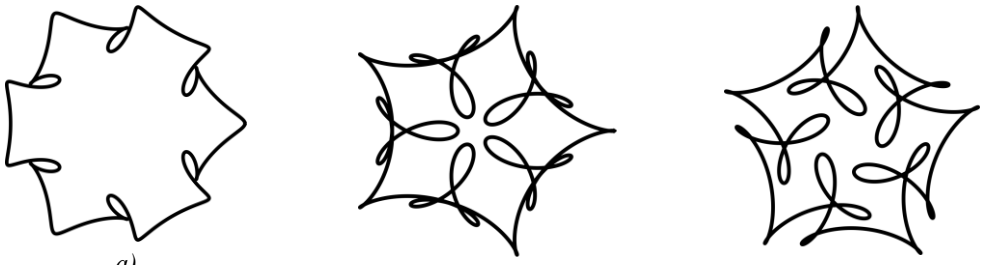
Рисунок 4. Спирограф с тремя инструментами R=1; *а*) r = –0.59, ρ =(1.,); б) r = –1/3, ρ =(0.3, 0.5, 1., 1.5);*в*) r = –.5, ρ =(0.2, 0.4,0.6, 0.8, 1., 1.2); *г*) r = 0.57, ρ =(0.2, 0.5, 1., 1.5)

3. «Волшебные» кривые

Имеется обобщение спирографа, называемое «волшебными» кривыми [13], основывающееся на формуле Эйлера и позволяющее работать с произвольным числом окружностей и других выпуклых фигур:

$$w(t) = x(t) + iy(t) = \sum_{k=0}^{n} a_k e^{i2\pi b_k t}, \qquad (2)$$

где в (2) i — мнимая единица, a_k — комплексные числа, b_k — действительные или целые числа, $n \geq 2$. Можно показать, что при $n = 2$ выражение (2) описывает спирограф. Без ограничения общности можно считать $a_0 = 1$.



а) *б)* *в)*

Рисунок 5. «Волшебные» кривые с целыми коэффициентами b_k: а) a = (1, 0.2, 0.1), b = (1, 7, –14); б) a = (1, 0.5, 0.3), b = (1, 6, –14); в) a = (1, 0.5, 0.3i), b = (1, 6, –14)

Библиотека NumPy позволяет работать с комплексными массивами, что в свою очередь позволяет реализовать вычисление координат узора в четыре строки исходного текста. При этом устраивается соревнование, кто реализует самую компактную функцию, а кто обеспечит максимальную функциональность, включая сохранение узора в полиграфическом качестве. Три «волшебные» кривые с целыми коэффициентами b_k приведены на рис. 5.

Для периодичности кривых, описываемых выражением (2), достаточно, чтобы все b_k были целыми. На рис. 6 приведены три кривые с действительными коэффициентами b_k.

Рисунок 6. Кривые с действительными коэффициентами b_k

Кривые на рис. 5 обладают симметрией, можно показать, что для того, что кривая совмещалась сама с собой при повороте на угол $2\pi/m$ радиан достаточно выполнения условия:

$$b_k \,\%\, m \equiv 1, \text{ для } \forall k = 0, 1, \ldots, \tag{3}$$

т. е. остаток от деления коэффициентов b_k на m был равен единице. В свою очередь это позволяет получить простую формулу для генератора симметричных узоров для заданных m и числа коэффициентов b_k, равного n:

$$b_k = g_k \cdot m + 1, g_0 = 1, \text{ для } \forall k = 1 \ldots n. \tag{4}$$

На рис. 7, 8 представлены наборы кривых для $m = 3$ и 8, $n = 3$ и предэкспоненциальных коэффициентов $a = (1, 0.5, 0.3i)$. Под *ipic* на рисунках понимается номер сгенерированной кривой для данного m.

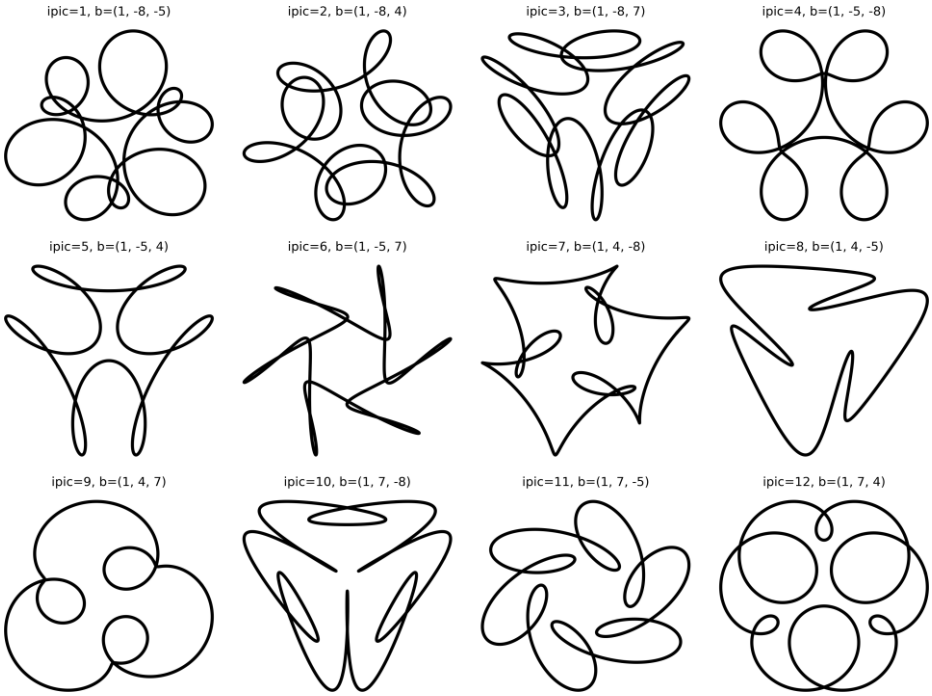


Рисунок 7. Набор кривых для m = 3

Генератор в течение нескольких секунд позволяет получить сотни и тысячи кривых, анализ которых позволяет разбить их на условные классы, например, на рис. 9 изображены кривые класса «шестеренки».

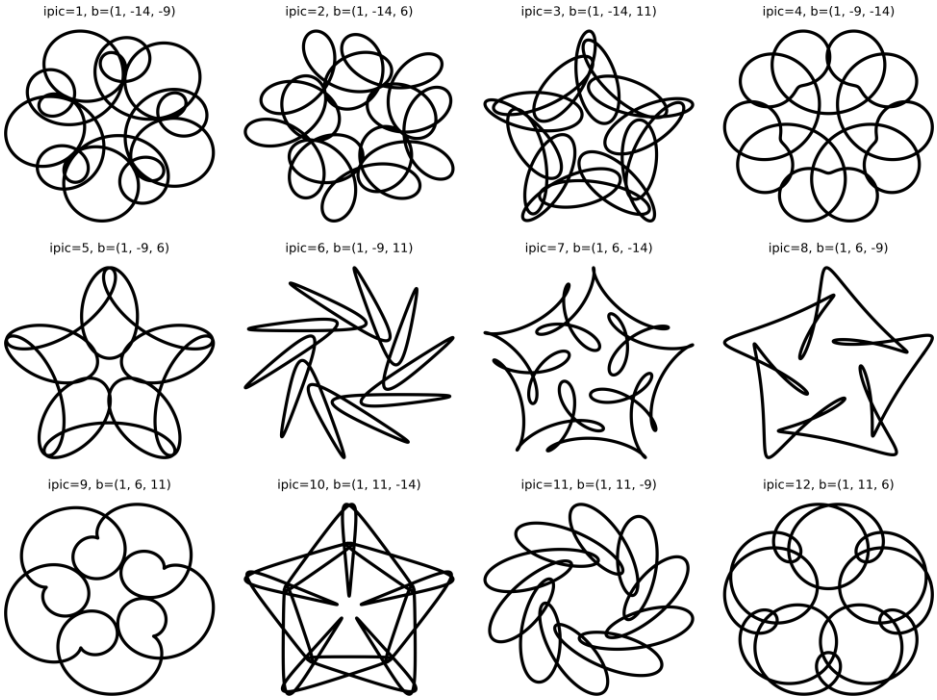


Рисунок 8. Набор кривых для m = 8

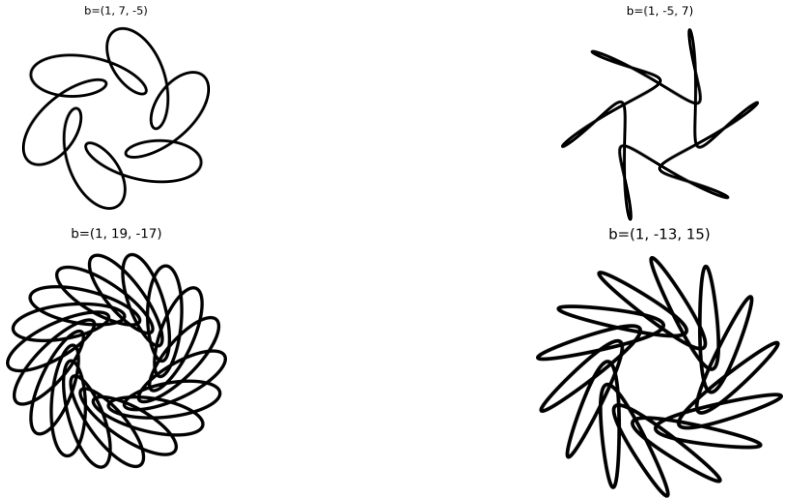


Рисунок 9. Кривые класса «шестеренки»

Другой класс представляет собой «розетки» (рис. 10).

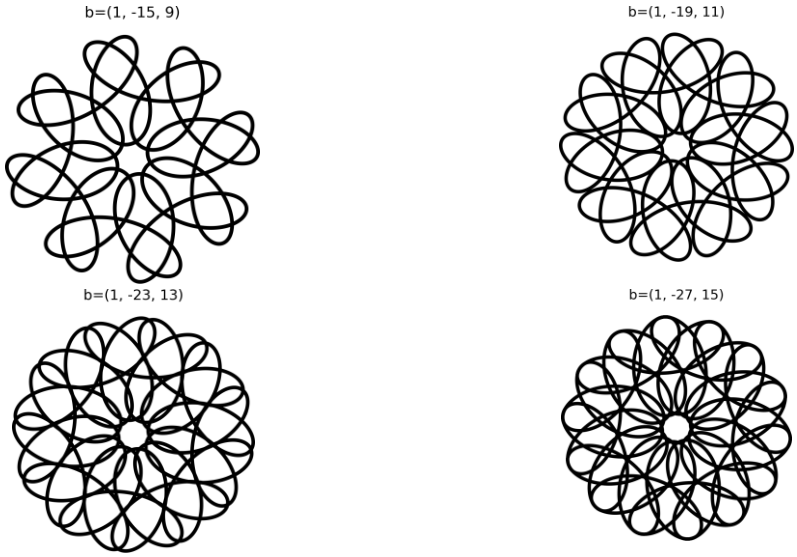


Рисунок 10. Кривые класса «розетки»

На рис. 10 можно наблюдать эволюцию розеток от $m=4,\ b=(1,-15,9)$ до $m=7,\ b=(1,-27,15).$

«Звезды» походят на «розетки», более того для небольших m «розетки» превращаются в «звезды» (рис. 11).

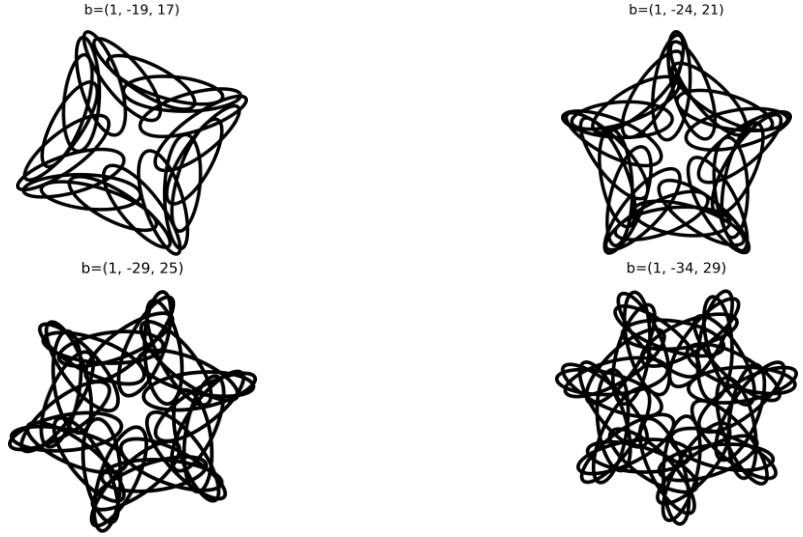


Рисунок 11. «Звезды»

Здесь приводятся четырех-семиконечные звезды $m = (4-7)$. Приведем еще две звезды с меньшими абсолютными значениями коэффициентов (рис. 12).

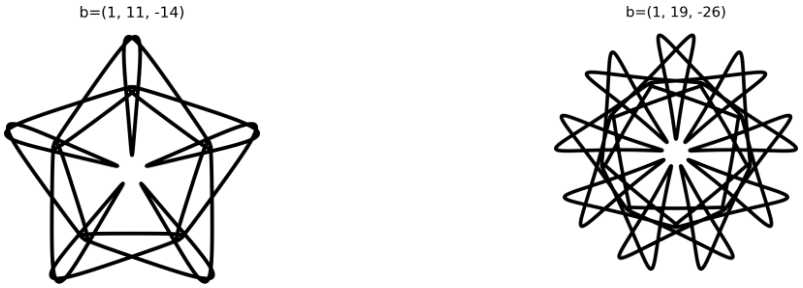


Рисунок 12. Две звезды для m = 5,8

Далее мы вступаем на зыбкий путь, отнеся к «волшебным» понравившиеся кривые, отличающиеся своим необычным поведением. На рис. 13 приведены четыре такие «волшебные» кривые, соответствующие $m = 3-6$. Наиболее «волшебными» являются кривые для $m = 4$ и 5. При больших значениях m «волшебные» кривые превращаются в гибриды «звезд» и «шестеренок», как показано на рис. 14. Левая кривая на рис. 14 соответствует $m = 7$, а правая $m = 8$.

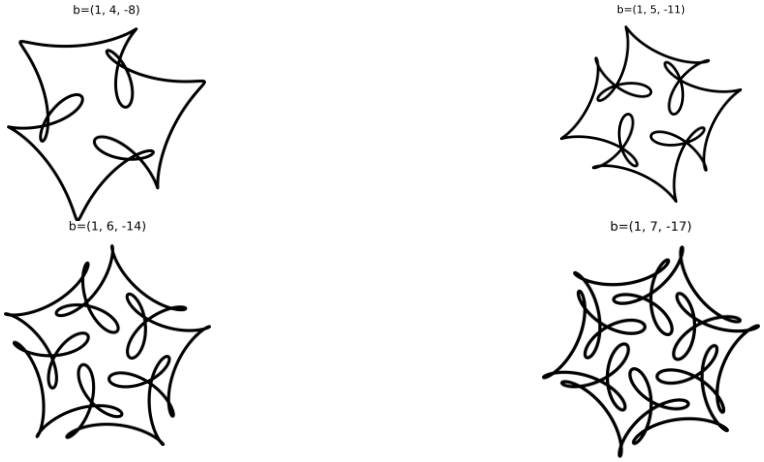


Рисунок 13. «Волшебные» кривые

Рисунок 14. Гибриды «звезд» и «шестеренок»

До сих пор мы анализировали эволюцию кривых при изменении коэффициентов b_k, при фиксированном кортеже $a=(1.,0.5,0.3i)$. Теперь же зафиксируем $b=(1,5,-11)$ и будем изменять кортеж a. Анализ показывает, что «волшебная» кривая (рис. 15*а*, *в*, *г*, *д*) при изменении всего одного коэффициента может превращаться в звезду (рис. 15 ж, з) и розетку (рис. 15 б, и).

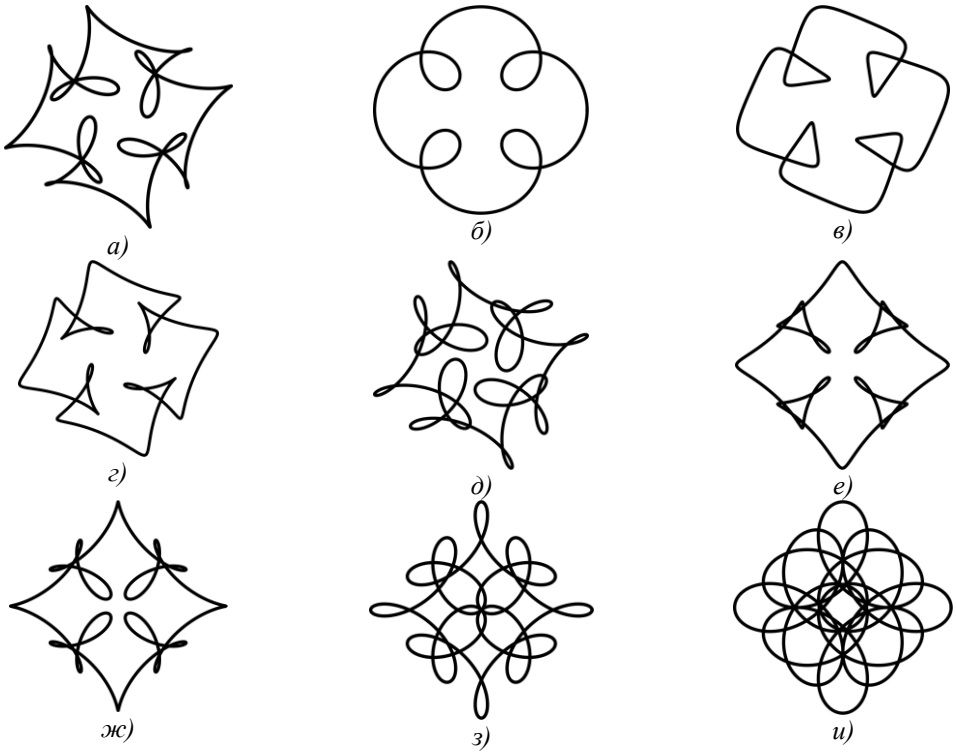


Рисунок 15. Эволюция кривых при изменении кортежа a и b = (1,5,–11), *что соответствует m* = 3*: а) a* = (1., 0.5, 0.3*i*) – *исходная кривая; б) a* = (1., 0.5, 0*i*)*; в) a* = (1., 0.5, 0.1*i*); *г*) *a* = (1., 0.5, 0.2*i*); *д*) *a* = (1., 0.5, 0.4*j*); *е*) *a* = (1., 0.5, 0.2); *ж*) *a* = (1., 0.5, 0.3); *з*) *a* = (1., 0.5, 0.5); *и*) a = (1., 0.5, 1.0)

Казалось бы, увеличение числа членов в выражении (2) позволяет генерировать все более сложные и красивые кривые, однако сделать это не удается, увеличение сложности приводит ко все более запутанным и хаотическим кривым даже для небольших m.

4. Другие задачи, приводящие к узорам

Часть задач, реализация которых приводит к красивым кривым и поверхностям, сформулированы в виде домашних заданий. В частности, для направления Энергетика и электротехника предлагались задачи визуализации картины электрического поля методом конформных отображений [14]. На рис. 16 приведено несколько конформных отображений единичного квадрата с помощью элементарных функций, построенных студентами.

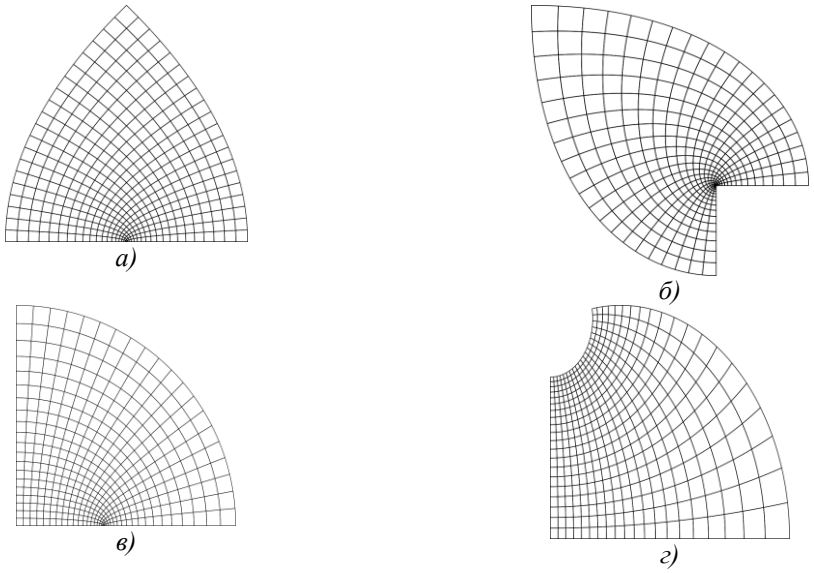
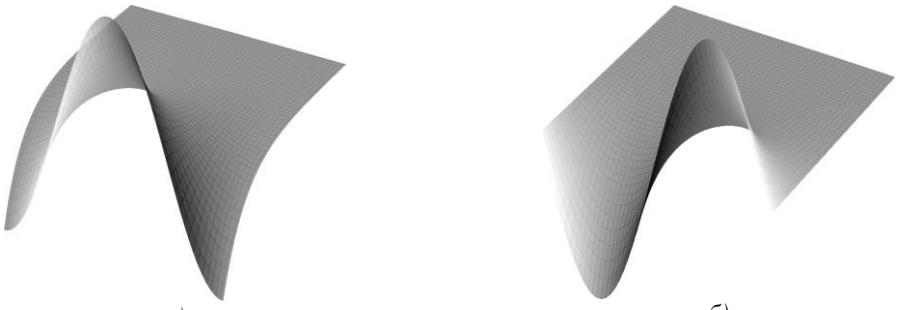


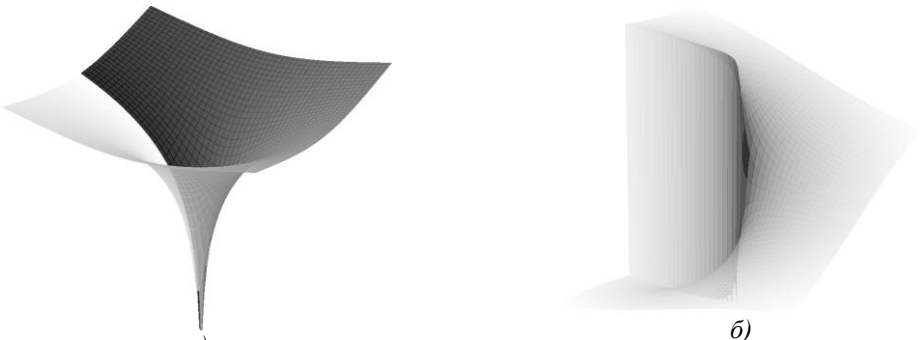
Рисунок 16. Конформные отображения единичного квадрата с помощью функций:
а) z^2; *б)* z^3; *в)* $\sin(\pi z/2)$; *г)* $tg(z)$

Двум студентам удалось решить более сложную задачу отображения поверхностей, когда единичный квадрат $z = x + iy$ отображается на комплексную плоскость $w = u + iv$. Функция представляет собой поверхность в четырехмерном пространстве, одну из переменных приходится отображать цветом из заданной палитры, обычно это u или v. Стандартного типа графика для этого в библиотеке matplotlib нет, но удалось написать собственную функцию для этого (рис. 17, 18).



а) *б)*

Рисунок 17. Риманова поверхность для функции $e^{i2\pi z}$*: а) действительная составляющая отображается по вертикали, мнимая составляющая отображается градациями серого; б) мнимая составляющая отображается по вертикали; действительная составляющая отображается градациями серого*



а) *б)*

Рисунок 18. Риманова поверхность для функции $\ln(z)$*:*
а) действительная составляющая отображается по вертикали, мнимая составляющая отображается градациями серого; б) мнимая составляющая отображается по вертикали; действительная составляющая отображается градациями серого

Возвращаясь от поверхностей в четырехмерном пространстве к кривым на плоскости, рассмотрим простой способ рисования узоров в полярных координатах. Давайте создадим массив *t* чисел с плавающей точкой от нуля до *m*, разбитый на *n* частей, *m*, *n* — целые числа. Казалось бы, чего ждать от такого массива при визуализации, однако если перейти к полярным координатам, то с помощью matplotlib получим узоры (рис. 19).

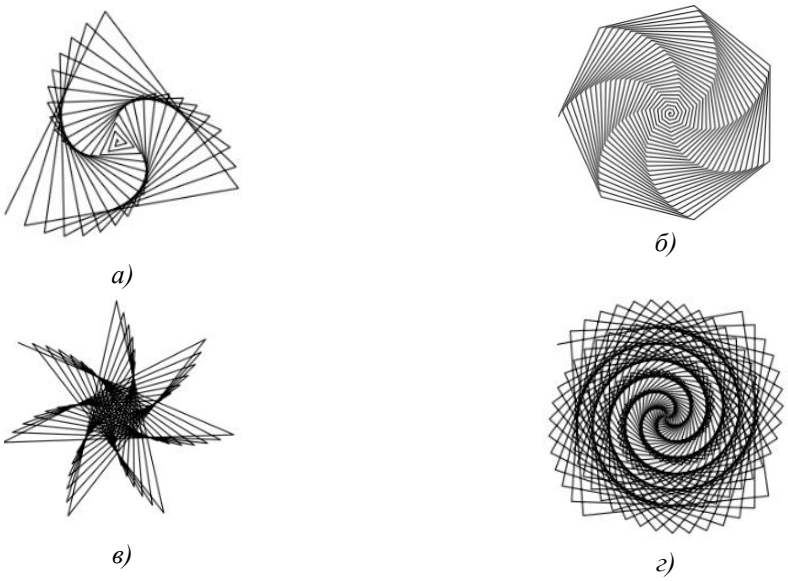


Рисунок 19. Узоры в полярных координатах.
а) $m = 100$, $n = 50$*; б)* $m = 180$, $n = 200$*; в)* $m = 200$, $n = 80$*; г)* $m = 400$, $n = 250$

Построить узор можно двумя способами: либо используя функцию polar() matplotlib, либо явно преобразуя координаты точек графика из полярных в декартовы координаты

5. Рекурсия и фракталы

Фракталы [15–17] — самоподобные фигуры, которые выглядят почти одинаково в разных масштабах. Примером фрактала, подпадающего под данное неформальное определение, являются карта береговой линии.

Фракталы вводились в курсе в утилитарных целях, чтобы объяснить и иллюстрировать использование рекурсии, но как оказалось, на их основе можно решить большое число интересных задач, а также проиллюстрировать использование, например, анимации.

Начнем с кривой Коха [18], описанной в 1904 году шведским математиком Хельге фон Кохом. Кривая состоит из четырех равных частей длиной l, для ее рисования достаточно провести отрезок прямой длины $l/3$, повернуть на 60 градусов влево, провести отрезок прямой, повернуть вправо на 120 градусов, провести отрезок прямой, для завершения рисования поворачиваем на 60 градусов влево и рисуем последний отрезок длины $l/3$.

Построение кривой Коха (рис. 20) продолжается удалением средней трети каждого отрезка и заменой изъятого на ломаную, описанную в начале

предложении. Процесс заканчивается тогда, когда текущая длина l не станет меньше наперед заданной величины $lmin$. Рисуются фракталы, рассматриваемые в данном разделе, с помощью черепашьей графики. Оказалось, что этапы построения фракталов удобно иллюстрировать с помощью анимации, сохраняя промежуточные этапы в файловой системе с последующим «монтажом» анимации с помощью штатных средств matplotlib.

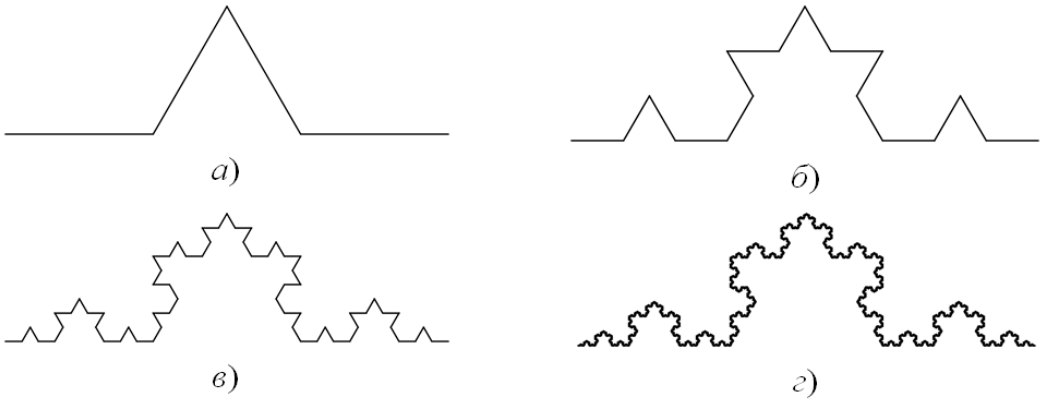


Рисунок 20. Кривая Коха при различных значениях $lmin$ и $l = 100$
а) $lmin$ =100; *б)* $lmin$ =60; *в)* $lmin$ =30; *г)* $lmin$ =2

Кривая Коха используется с еще одной целью: для демонстрации работы с функциями с переменным числом параметров. Данный механизм достаточно важен, так как широко используется в библиотеках экосистемы Python. В начале курса на занятиях подробно разбирается функция рисования многоугольника. Здесь же мы разбираем, как нужно доработать эту функцию с тем, чтобы для заменить сторону многоугольника произвольной кривой, в том числе и фракталом. На рис. 21 а нарисован треугольник и снежинка на его основе, то же самое на рис. 21 б сделано для шестиугольной снежинки.

Рисунок 21. Снежинки на основе кривой Коха:
а) треугольники и снежинка на его основе; б) шестиугольник и снежинка на его основе.

Рассмотрим еще один фрактал, — треугольник Серпинского [19]. Представим себе равносторонний треугольник, из которого изымается середина, также представляющая собой равносторонний треугольник. Та же операция применятся к оставшимся трем треугольникам и так далее до тех пор, пока длина стороны изымаемого треугольника не станет меньше заданной. Для рисования этого фрактала также удобно воспользоваться рекурсией (рис. 22).

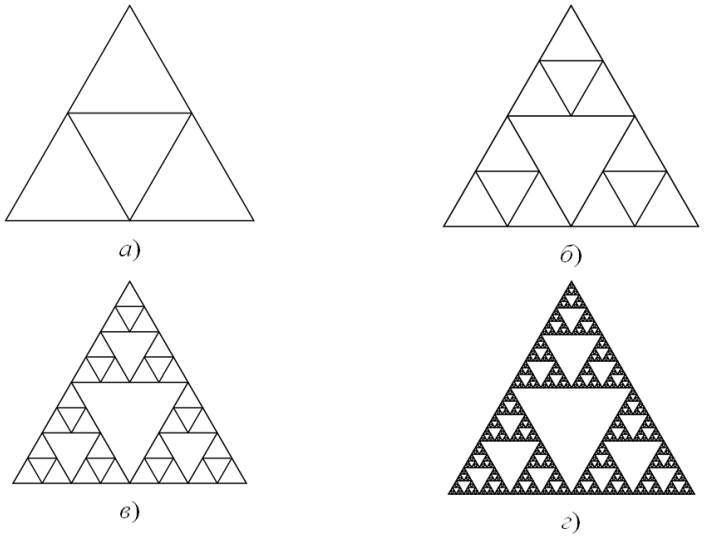


Рисунок 22. Треугольники Серпинского для l = 100 и различных lmin:
а) lmin = 100; б) lmin = 50; в) lmin = 25; г) lmin = 2

6. Предметно-ориентированный язык для рисования узоров и фракталов

Общим для узоров и фракталов, рассмотренных ранее, является то, что для рисования каждого из них пришлось писать отдельную функцию.

Возникает вопрос, можно ли сделать так, чтобы можно было задавать правила построения узора или фрактала, а рутинная работа по генерации последовательности команд рисования выполнялась автоматически. Положительный ответ на данный вопрос дал биолог Аристид Линденмайер [20], которому понадобился инструмент для выяснения правил, по которым, формируются кроны деревьев. В честь него эти системы получили название L-систем. В книге [21] приводится большое число реалистических цветных изображений растений и других биологических систем, синтезированных с помощью L-систем.

L-система представляет собой специализированный предметно-ориентированный язык, основой которого является алфавит — набор символов.

Например, мы можем в качестве набора символов рассматривать укороченную систему команд черепашьей графики:

F перемещение черепашки вперед на l пикселов с рисованием, которое мы связали с командой forward(l);

f перемещение черепашки вперед на l пикселов без рисования, которое мы связали с командой move(l);

\+ поворот налево на угол θ градусов (команда left(θ));

– поворот направо на угол θ градусов (команда right(θ));

[сохранить состояние черепашки (положение, угол поворота, поднята черепашка или опущена);

] восстановить состояние черепашки (положение, угол поворота, поднята она или опущена).

Набор символов алфавита необязательно ограничивается указанным выше набором команд. Можно включить в алфавит дополнительные символы, например, *X*, *Y*, 1, 9.

Работа L-системы начинается с задания начальной строки символов, которую называют аксиомой, например, для рисования начального состояния кривой Коха необходимо задать аксиому F.

Начальная строка символов преобразуется с помощью одного или нескольких правил. Правило состоит из левой части — преобразуемого символа, и правой части — строки символов, в которую преобразуется символ, заданный в левой части. Продолжая пример с кривой Коха, приведем единственное правило для преобразования символов:

$$F \to F + F - -F + F.$$

После применения правила к аксиоме мы получим строку символов $F + F - -F + F$, а после применения правила к результату первого преобразования будет получена строка

$$F - F + +F - F - F - F + +F - F + +F - F + +F - F - F - F + +F - F.$$

Правило может применяться произвольное число раз, результатом всегда будет строка символа алфавита.

Рассмотрим другой пример преобразования, так называемой «драконьей» кривой:

аксиома: *FX*,

правило 1: $X \to X + YF+$,

правило 2: $Y \to -FX - Y$,

угол поворота θ = 90°.

Начинаем преобразовывать аксиому. Первый символ F не содержится в левых частях правил, копируем F в результат преобразования. Второй символ X — левая часть первого правила, поэтому передаем в результирующую строку правую часть первого правила. Результатом первого преобразования является строка:

$FX+YF+$

Применим правила второй раз. Первый символ F копируем в результирующую строку, второй символ X преобразовываем с помощью первого правила, как это делалось на первом шаге. Третий символ также копируется в результирующую строку, четвертый символ Y преобразуется с помощью второго правила, пятый и шестой символы копируются в результирующую строку. После второго применения правил мы получим:

$FX+YF++-FX-YF+.$

После проведения преобразований необходимо визуализировать получившийся результат, т. е. преобразовать символы результирующей строки в команды визуализации, например, команды черепашьей графики.

Трудность состоит в том, что часть символов (в данном примере это *X*, *Y*) не связана с командами визуализации и необходимо принять решение, что с ними делать. Вариантов решений несколько: либо в явном виде связывать все символы с командами, либо пропускать несвязанные символы при визуализации, либо отображать несвязанные символы с помощью заданной команды визуализации.

Возвращаясь к рассматриваемому примеру, заменим *X*, *Y* на *F*:

$$FF+FF++-FF-FF+$$

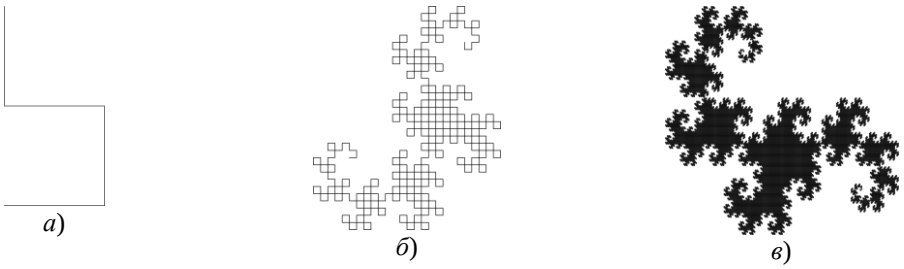


Рисунок 23. «Драконья» кривая:а) после двух применений правил (число команд 14); б) после 9 применений (число команд 2046); в) после 15 применений правил (число команд 131070)

Этому соответствует изображение, полученное с помощью черепашьей графики (рис. 23). После двух применений кривая на дракона не слишком похожа, но, если применить правила многократно, что-то вроде дракона мы увидим (рис. 23 б, в). В подрисуночную подпись вынесено число команд, необходимых для визуализации рисунка, из которых очевидно, что выполнить преобразования без применения компьютера весьма затруднительно.

Экосистема Python отлично подходит для реализации предметно-ориентированных языков. В данном случае функциональность L-системы реализована классом Lsystem. Алфавит в классе Lsystem представлен в виде словаря Python, ключами которого являются символы алфавита, а значениями —

команды. В свою очередь команда — это кортеж, содержащий строку имени команды, например, 'forward' и передаваемые этой команде параметры. Стандартный набор символов, включает 'F', 'f', '+', '–' '[', ']' . Указанные символы включаются в экземпляр класса по умолчанию. При инициализации экземпляра класса алфавит дополняется символами и командами пользователя. Если символ не связан с командой, то его можно не кодировать в словаре алфавита.

Команды параметризированы, поэтому для поворота черепашки на угол, отличающийся от θ и перемещения черепашки с рисованием вперед на h пикселов, необходимо дополнить алфавит:

```
alphabet = {
  '*': ('left', 'γ'),
  'H': ('forward', 'h')
}
```

При этом необходимо задать значения параметров γ и h. Параметры задаются словарем, ключом является строка — имя параметра, значением — числовое значение параметра, например:

```
params = {'γ':60, 'θ':90, 'h':20}
```

Аксиома, т. е. начальная последовательность символов алфавита задается строкой, для рассмотренного выше примера:

```
axiom = 'FX'
```

Правила задаются в именованном параметре rules, значением которого должен быть словарь. Ключи словаря — левые части правил, значения — правые части правил. Например, правила для «драконьей» кривой задаются так:

```
rules = {
  'X': 'X+YF+',
  'Y': '–FX–Y'
}
```

По умолчанию при визуализации значения несвязанных параметров заменятся символом 'F'. Это умалчиваемое поведение можно изменить, задав значение именованного параметра replace равным False. При этом несвязанные символы при визуализации не будут отображаться.

Можно выборочно задать неотображаемые при визуализации символы, передав именованному параметру exclude строку с неотображаемыми символами алфавита. Естественно, что при этом значением параметра replace должно быть True.

Для работы с L-системами необязательно явно создавать экземпляр класса Lsystem и вызывать его методы, например, метод apply(n) для применения правил n раз и visualize() для собственно визуализации, достаточно вызвать статический метод draw(), передав этому методу число применений правил, строку аксиомы,

словари правил и параметров и, если необходимо, изменить умалчиваемые значения параметров. Например, для рисования снежинки на основе кривой Коха с измененными углами и квадрата достаточно:

```
axiom = 'F*F*F*F'
rules = {'F':'F+F--F+F',}
p = {'theta':80, 'gamma':90}
alphabet = {'*':('right', 'gamma'),}
L.draw('snow_flake_90', 4, axiom, rules,
       params=p, lw=0.5, alphabet=alphabet)
```

Результат представлен на рис. 24.

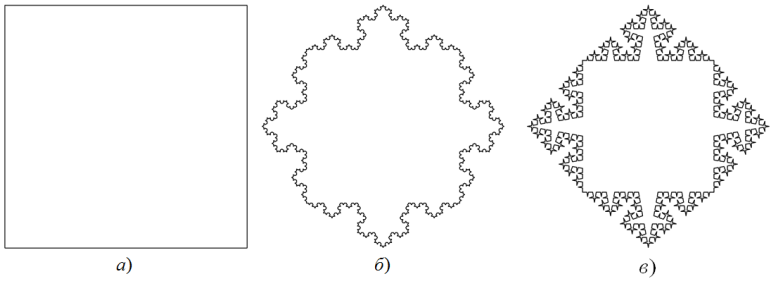


Рисунок 24. Снежинка на основе квадрата:
а) правила не применялись; б) четырехкратное применение правил при θ = 60°;
в) четырехкратное применение правил при θ = 90°

Тема снежинок неисчерпаема. Ранее в кривой Коха лучики снежинки были треугольные, попробуем нарисовать их на основе пятиугольников [22] (рис. 25).

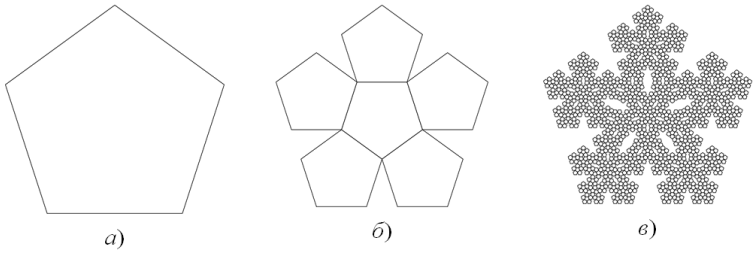


Рисунок 25. Снежинка с пятиугольными лучами:а) отображение аксиомы;
б) однократное применение правила; в) четырехкратное применение правила

Для рисования пятиугольников нам нужно будет поворачивать влево на угол 72°, для этого нам придется ввести дополнительный элемент алфавита и, кроме того, переопределить значение угла θ:

```
axiom = 'F*F*F*F*F'
rules = {'F':'F*F*F**+F-F*F'}
p = {'theta': 36, 'gamma':72}
```

```
alphabet = {'*':('left','gamma')}
L.draw('snowflake_5', 7, axiom, rules, params=p,
    alphabet=alphabet)
```

Эта снежинка отличается от снежинки Коха только тем, что треугольные выступы заменяются пятиугольниками.

Простые правила, использующие только перемещения черепашки и повороты, позволяют построить не только затейливые кривые, но и дендриты — сталактиты и сталагмиты, растущие в пещерах (рис. 26).

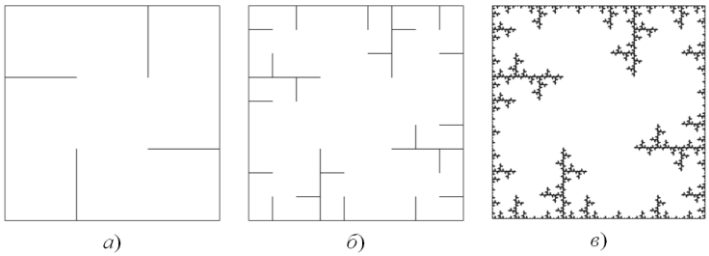


Рисунок 26. Сталактиты и сталагмиты: а) однократное применение правила; б) двукратное применение правила; в) пятикратное применение правила.

В нашем случае сталактиты и сталагмиты будут расти не только на полу и потолке пещеры, но и на стенах:

```
axiom = 'F-F-F-F'
rules = {'F':'FF-F--F-F'}
p = {'theta':90}
L.draw('stalactite1', 1, axiom, rules, params=p)
```

Кривая Гилберта, покрывающая квадрат [23], формируется с помощью аксиомы и правил:

```
axiom = 'L'
rules = {'L':'+RF-LFL-FR+', 'R':'-LF+RFR+FL-'}
p = {'theta': 90}
L.draw('hilbert_curve', 1, axiom, rules, params=p)
```

Результат их применения представлен на рис. 27.

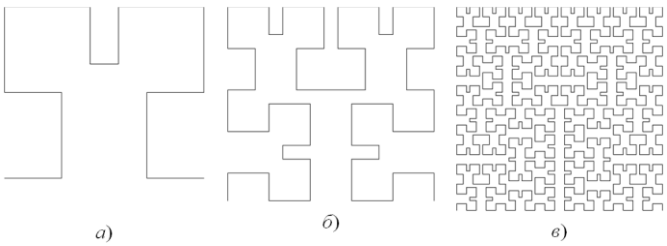


Рисунок 27. Кривая Гилберта: а) двукратное применение правил; б) трехкратное применение правил; в) пятикратное применение правил.

Рисование узоров и фракталов сводится к следующим действиям: информационному поиску правил рисования интересных объектов, созданию видео на основе отдельных кадров, для чего была написана функция lsystem_animation(), позволяющая на основе алфавитов, аксиом, правил и максимального числа их применения формировать файлы видео, описывающие эволюцию узоров при применении правил. На рис. 28 представлены некоторые узоры.

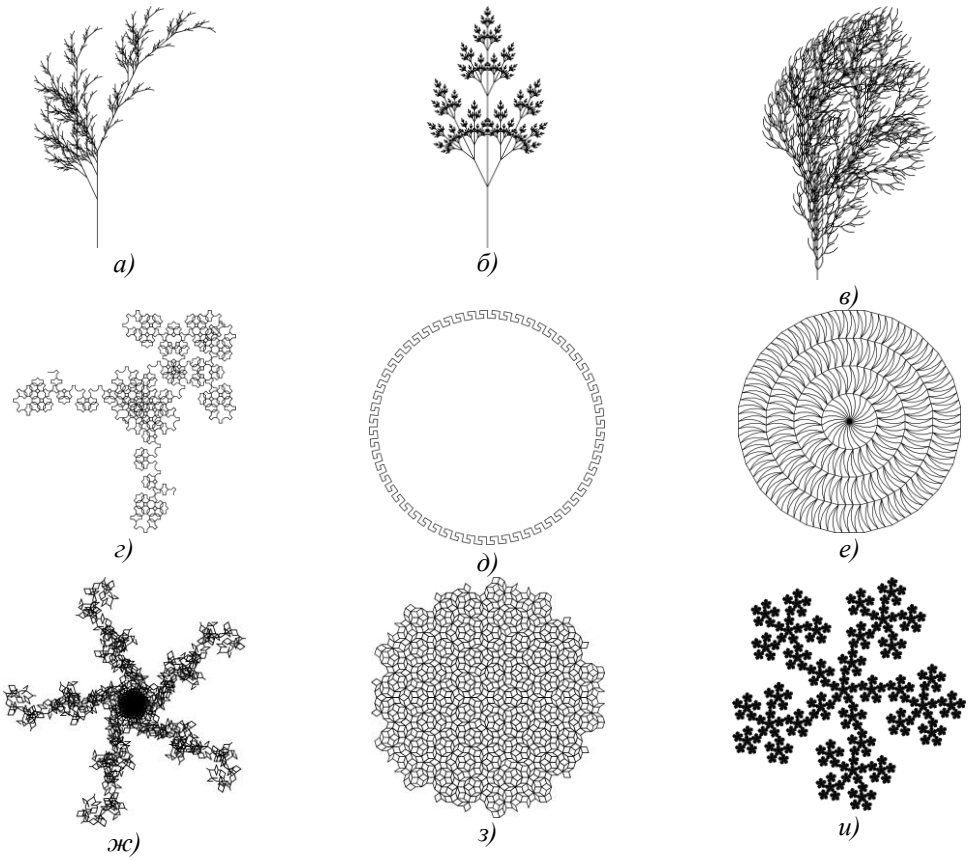


Рисунок 28. Узоры, построенные с помощью L-систем: а), б), в) деревья; г) робот-футболист; д) меандр; е) колам — индийское украшение; ж) морская звезда; з) мозаика Пенроуза; и) пентадендрит

7. Заключение

Рассмотренные здесь простые средства позволяют сделать процесс освоения подмножества Python, необходимый для решения расчетных научно-технических задач, наглядным, внести в него элементы геймификации, решая кроме этого задачи освоения библиотек экосистемы, научной визуализация, стиля оформления исходных текстов Python и технологии проведения многовариантных расчетов с

использованием Jupyter Notebook и JupyterLab. Аналогичные подходы с примерами из квантовой механики, теории динамических систем используются при рассмотрении явлений, описываемых обыкновенными дифференциальными уравнениями и уравнениями в частных производных.

Литература

[1] *Очков В. Ф., Калова Я., Никульчев Е. В.* Оптимизированный фрактал или ФМИ // *Cloud of Science*. 2015. Т. 2. № 4. С. 544–561.

[2] TIOBE Index for January 2020 [Электронный ресурс] URL: https://www.tiobe.com/tiobe-index/ (дата обращения 10.01.20)

[3] *Schlothauer S.* Top programming languages in 2019: Python sweeps the board [Электронный ресурс] URL: https://jaxenter.com/python-ranking-2019-161880.html (дата обращения 10.01.20).

[4] WinPython. The easiest way to run Python, Spyder with SciPy and friends out of the box on any Windows PC, without installing anything! [Электронный ресурс] URL: https://winpython.github.io/ (дата обращения 22.04.20).

[5] Anaconda Individual Edition. The World's Most Popular Python/R Data Science Platform [Электронный ресурс] URL: https://www.anaconda.com/distribution/ (дата обращения 22.04.20).

[6] *Langtangen H. P.* A Primer on Scientific Programming with Python. — 5th Ed. — Berlin : Springer-Verlag, 2016.

[7] *Доля П. Г.* Введение в научный Python. — Харьков : Изд-во Харьковского национального университета, 2016.

[8] *Johansson R.* Numerical Python. Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib. Second Edition. — NY : APRESS MEDIA, 2019. P. 719.

[9] *Pine D. J.* Introduction in Python for Science and Engineering. — London : CRC Press, 2019.

[10] *Штейнгауз Г.* Математический калейдоскоп. — М.–Л. : Гос. изд. тех.-теор. лит., 1949.

[11] *Асадова Н.* 3 вопроса об образовательной технологии STEAM, которая изменит российскую школу. [Электронный ресурс] URL: https://letidor.ru/obrazovanie/3-voprosa-o-sisteme-steam-kotoraya-izmenit-rossiiskuyu-shkolu.htm (дата обращения 22.04.2020).

[12] Spirograph [Электронный ресурс] URL: https://en.wikipedia.org/wiki/Spirograph (дата обращения 12.01.2020)

[13] *Farris F. A.* Creating Symmetry. The Artful Mathematics of Wallpaper Patterns. — Princeton : Prinston University Press, 2015.

[14] *Иванов В. И., Попов В. Ю.* Конформные отображения и их приложения. — М. : Едиториал УРСС, 2002.

[15] *Терехов С.В.* Фракталы и физика подобия. — Донецк : Цифровая типография, 2011.

[16] *Gulic D.*, *Scott J.* The Beauty of Fractals: Six Different Views. — Washington DC : MAA Service Center, 2010.

[17] Фракталы на Python. Пошаговое руководство [Электронный ресурс] URL: https://habr.com/ru/company/piter/blog/496538/ (дата обращения 22.04.20).

[18] Кривая Коха [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/Кривая_Коха (дата обращения 10.03.2020).

[19] Треугольник Серпинского [Электронный ресурс] URL: https://elementy.ru/posters/fractals/Sierpinski (дата обращения 10.03.2020).

[20] *Lindenmayer A.* Mathematical models for cellular interaction in development // *J. Theoret. Biology*, 1968. Vol. 18. No. 3. P. 280–315.

[21] *Prusinkiewicz P.*, *Lindenmayer A.* The Algorithmic Beauty of Plants. — Berlin : Springer Verlag, 2004. 240 p.

[22] L-system generator World's simplest math tool. [Электронный ресурс] URL: https://onlinemathtools.com/l-system-generator (дата обращения 21.01.2020)

[23] Кривая Гилберта [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/Кривая_Гилберта (дата обращения: 25.01.2020)

Авторы:

Валерий Федорович Очков — доктор технических наук, профессор, профессор кафедры теоретических основ теплотехники, Национальный исследовательский университет «МЭИ»

Антон Иванович Тихонов — кандидат технических наук, старший научный сотрудник, профессор кафедры физики и технологии электротехнических материалов и компонентов, Национальный исследовательский университет «МЭИ»

Ornaments and fractals in Python

V. F. Ochkov, A. I. Tikhonov

National Research University Moscow Power Engineering Institute, Moscow, 111250 Reussia
e-mail: ochkov@twt.mpei.ac.ru; TikhonovAI@mpei.ru

Abstract: The article deals with the application of STEM- and STEAM-technologies when students master a minimum subset of Python to solve scientific, technical and engineering problems. Language constructions are introduced in parallel with generation and visualization of ornaments and fractals. The possibilities of gamification for studing libraries of the Python ecosystem are considered. Multivariant calculations, visualization and interpretation of the obtained results are carried out.
Keywords: STEM, STEAM, Python, ornament, fractal.

References

[1] *Ochkov V. F.*, *Kalova J.*, *Nikulchev E. V.* (2015) *Cloud of Science*, **2**(4):544–561. [Rus]

[2] https://www.tiobe.com/tiobe-index/

[3] https://jaxenter.com/python-ranking-2019-161880.html

[4] https://winpython.github.io/

[5] https://www.anaconda.com/distribution/

[6] *Langtangen H. P.* (2016) *A Primer on Scientific Programming with Python*. (5th Ed., Berlin, Springer-Verlag).

[7] *Dolya P. G.* (2016) *Introduction Scientific Python*. (Kharkov,) 265 p. [Rus]

[8] *Johansson R.* (2019) *Numerical Python. Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib*. Second Edition. (N.Y., APRESS MEDIA). P. 719.

[9] *Pine D. J.* (2019) *Introduction in Python for Science and Engineering*. (London, CRC Press).

[10] *Steingaus G.* (1949) *Mathematical kaleidoscope*. (Moscow-Leningrad, State publishing house of technical-theoretical literature). [Rus]

[11] https://letidor.ru/obrazovanie/3-voprosa-o-sisteme-steam-kotoraya-izmenit-rossiiskuyu-shkolu.htm

[12] https://en.wikipedia.org/wiki/Spirograph

[13] *Farris F. A.* (2015) *Creating Symmetry. The Artful Mathematics of Wallpaper Patterns*. (Princeton, Prinston University Press).

[14] *Ivanov V. I., Popov V. Yu.* (2002*) Conformal mappings and their applications*. (Moscow). [Rus]

[15] *Terekhov S.V.* (2011) *Fractals and Similarity Physics.* (Donetsk, Digital Typography). [Rus]

[16] *Gulic D., Scott J.* (2010) *The Beauty of Fractals: Six Different Views.* (Washington DC, MAA Service Center).

[17] https://habr.com/ru/company/piter/blog/496538/

[18] https://ru.wikipedia.org/wiki/Кривая_Коха

[19] https://elementy.ru/posters/fractals/Sierpinski

[20] *Lindenmayer A.* (1968) *J. Theoret. Biology*, **18**(3):280–315.

[21] *Prusinkiewicz P., Lindenmayer A.* (2004) *The Algorithmic Beauty of Plants.* (Berlin, Springer Verlag).

[22] https://onlinemathtools.com/l-system-generator

[23] https://ru.wikipedia.org/wiki/Кривая_Гилберта