FractPy

Release 0.0.4

Amarjit Singh Gaba

Feb 02, 2022

CONTENTS:

This is a Python library to generate fractals of various kinds.

CHAPTER

ONE

TUTORIAL: GENERATING NEWTON FRACTAL FOR A SIMPLE FUNCTION

1.1 Introduction to Fractal

A fractal is never ending pattern. Fractals are a type of mathematical shape that are infinitely complex. They are created by repeating a simple process over and over in an ongoing feedback loop.

In essence, a Fractal is a pattern that repeats forever, and every part of the Fractal, regardless of how zoomed in, or zoomed out you are, it looks very similar to the whole image. A shape does not have to be exactly identical to be classified as a Fractal. Instead shapes that display inherent and repeating similarities are the main requirement for being classified as a Fractal. [Falconer1990]

Fractals are found all over nature, spanning a huge range of scales. We find the same patterns again and again, from the tiny branching of our blood vessels and neurons to the branching of trees, lightning bolts, snowflakes, river networks and even the clustering of galaxies. Regardless of scale, these patterns are all formed by repeating a simple branching process.

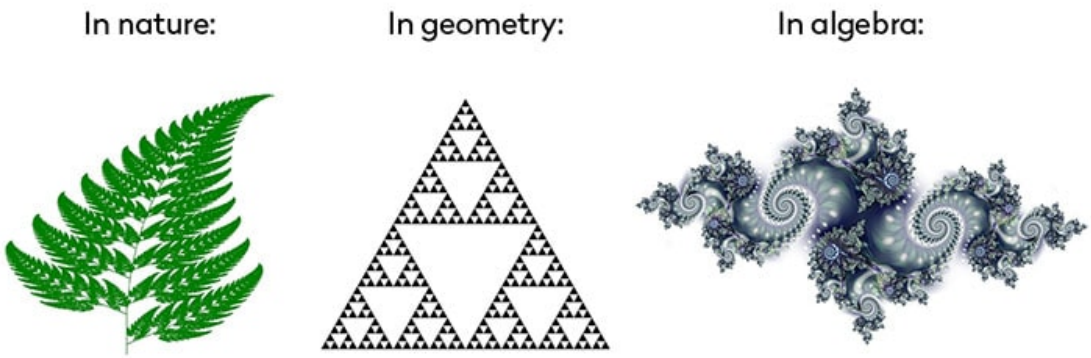


Fig: Fractal in nature, geometry, and algebra [Jasser]

For more background information on fractals: [FractalFoundation2009].

1.1.1 Newton Fractal

One way of generating fractals is using Newton-Raphson Method, also known as Newton Fractal. Newton fractals are fractals created in the plane of complex numbers. An iteration process with Newton's method is started at each point on a grid in the complex plane, and a color is assigned to each point according to which of the roots of a given function the iteration converges to. [Sahari2006]

A generalisation of Newton's iteration is:

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}$$

where $z \in \mathbb{C}$ represents any point in the plane, $n \in \mathbb{N}$ represents the number of step, and $f(z)$ is a polynomial or transcendental function.

For $f(z) = z^3 - 1$, the iteration is:

$$z_{n+1} = z_n - \frac{z^3 - 1}{3z^2}$$

1.2 Installing FractPy

Let us now see how to plot fractal for this function using FractPy, but first we need to install it. FractPy requires Python 3.6 or greater, so assuming you have it, to install FractPy:

- On Mac OSX or linux open a terminal;
- On Windows open the Command prompt or similar

and type:

```
$ python -m pip install fractpy
```

1.3 Using FractPy

Generating fractals in `fractpy` can be divided into 2 steps:

1. Creating a model
2. Generating the fractal

1.3.1 Creating a Model

A model represents the technique being used to generate the fractal, so generating fractal from Newtons' method would involve making a model `NewtonFractal` which is a class of module `fractpy.models`, and then passing the required function in the form `str` as an argument during initialisation. The following code shows how to make a `NewtonFractal` model for the above function:

```
>>> from fractpy.models import NewtonFractal
>>> model = NewtonFractal("x**3 - 1")
>>> model
### FractPy Model ###
Type: Newton Fractal
Function: x**3 - 1
```

Note: For complex values use **I** (upper case i) instead of commonly used convention of **i**. For example: function $f(x) = (x-1)(x+i)(x-i)$ would be passed as `"(x - 1)(x + I)(x - I)"`.

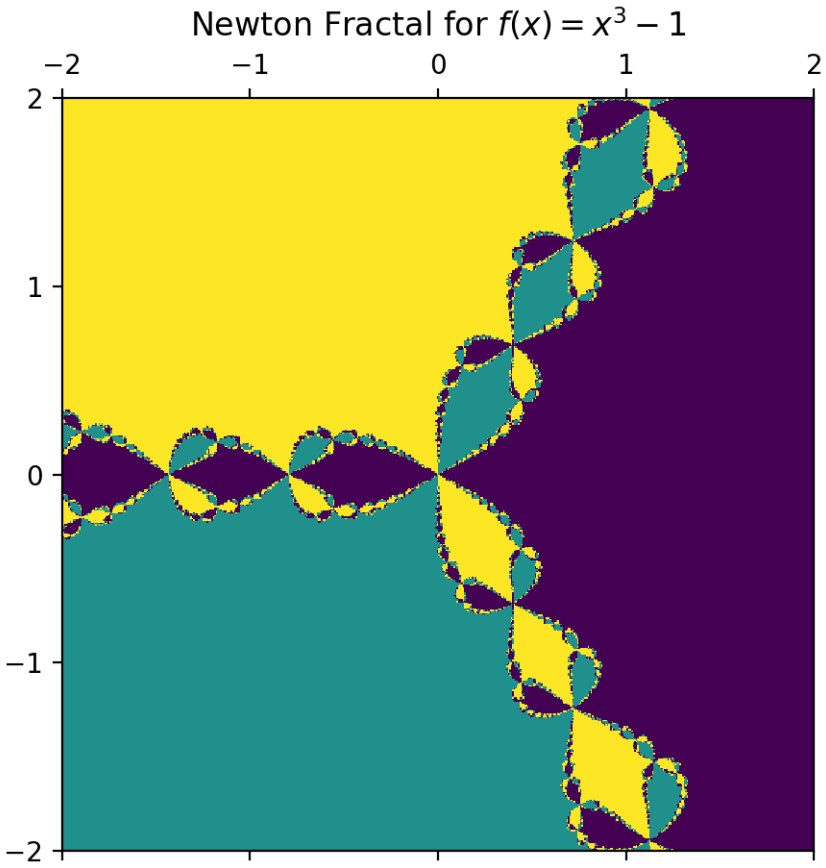
1.3.2 Generating Fractal

To generate the fractal all we have to do now is call the method `plot`, and pass in the axes limits along with the desired resolution of the image which returns a `matplotlib.figure.Figure`:

```
>>> xmin, xmax, ymin, ymax = -2, 2, -2, 2
>>> p = model.plot(xmin, xmax, ymin, ymax, (500, 500))
>>> p.show()
```

The above code will generate the Newton Fractal for $x^3 - 1$, in the range -2 to 2 for both x-axis and y-axis, and the resolution of the image would be 500X500.

This creates the following plot:



Note: Generating fractal requires some heavy computation so it may take seconds, or minutes depending on the computing power of the system.

CHAPTER

TWO

HOW TO

How to:

2.1 Create a Model

A model in `fractpy` is an object which represents the method it is using to generate fractals.

Note: FractPy currently supports only Newton Fractal method of fractal generation for polynomial functions with real powers. More methods are currently in development.

To make a Newton Fractal Model for the function $f(x) = (x^2 - 1)(x^2 + 1)$ all we have to do is pass in the function as `str` to the `NewtonFractal` class from module `fractpy.models`:

```
>>> from fractpy.models import NewtonFractal
>>> model = NewtonFractal("(x**2 - 1)(x**2 + 1)")
>>> model
### FractPy Model ###
Type: Newton Fractal
Function: (x**2 - 1)*(x**2 + 1)
```

Note: For complex values use **I** (upper case i) instead of commonly used convention of **i**. For example: function $f(x) = (x - 1)(x + i)(x - i)$ would be passed as `"(x - 1)(x + I)(x - I)"`.

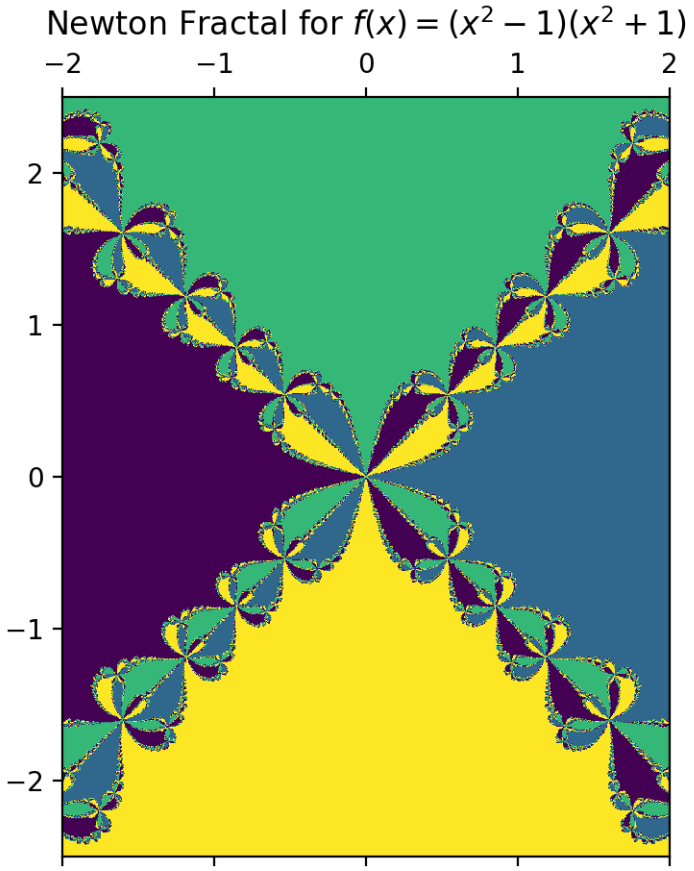
We can use this model to generate fractals!

2.2 Create a Plot

To create plot from the model, we call the method `plot`, and pass in the axes limits, along with the resolution of the plot we want. This returns a `matplolib.figure.Figure`. To plot a fractal for the range -2, 2 and -2.5 , 2.5 for x and y-axis respectively and of resolution (600,900):

```
>>> p = model.plot(-2, 2, -2.5, 2.5, (600,900))
>>> p.show()
```

This creates the following plot:



2.3 Create a Zoom Plot

Creating a simple plot for fractals is pretty boring, especially if you cannot see the beauty of how it repeats itself.

So `fractpy` offers a funtionality in which you can dynamically zoom in any region of the plot. To create such plot we will use the method `zoom_plot` which creates two identical panels. Zooming in on the right panel will show a rectangle in the first panel, denoting the zoomed region. And as done in `plot` we will gave to pass in the initial axes range, along with the resolution of the plot to be generated, and get the `matplotlib.figure.Figure`:

```
>>> p = model.zoom_plot(-2, 2, -2, 2, (200,200))
>>> p.show()
```

This creates a plot like this, which can be zoomed in:

Note: This currently does not work in Jupyter Notebook, and has to be run using a python script.

2.4 To Find the Derivative and the Roots

Fractpy has a class `Function` which can be used to perform basic calculus operations on a single-variable function. To initialise an object of this class we pass in the function as type `str`.

For example to initialise with function $f(x) = x^4 - 3x^3 + 2x^2 - 9$:

```
>>> from fractpy import Function
>>> f = Function("x**4 - 4*x**3 + 4*x**2 - 4*x + 3")
>>> f
x**4 - 4*x**3 + 4*x**2 - 4*x + 3
```

Note: For complex values use **I** (upper case i) instead of commonly used convention of **i**. For example: function $f(x) = (x-1)(x+i)(x-i)$ would be passed as `"(x - 1)(x + I)(x - I)"`.

2.4.1 To Calculate Roots

Use the method `roots`, which returns list of the roots:

```
>>> f.roots()
[1, 3, I, -I]
```

2.4.2 To Find the Derivative:

Use the method `differentiate`, which returns the derivative of the function in the form of a `sympy` expression:

```
>>> f.differentiate()
4*x**3 - 12*x**2 + 8*x - 4
```

CHAPTER

THREE

EXPLANANTION

3.1 Newton Fractal

Newton Fractal is generated by iterating Newton-Raphson method of finding roots in the complex plane. [Sahari2006]

3.1.1 Newton-Raphson Method of Finding Roots

The Newton-Raphson method (also known as Newton's method) is a way to quickly find a good approximation for the root of a real-valued function $f(x) = 0$ It uses the idea that a continuous and differentiable function can be approximated by a straight line tangent to it.

To perform a Newton-Raphson approximation, suppose you have a function $f(x)$, with derivative $f(x)$, and you have an approximation x_0 to a root of the function. The Newton-Raphson procedure is to calculate:

$$x_1 = x_0 \frac{f(x_0)}{f(x_0)}$$

which is a closer approximation to the root. Typically you would then iterate this again, and again, until the successive values were extremely close together, at which point you would conclude that you had a very good approximation to the actual value r for which $f(r) = 0$.

The iteration performed is:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

3.1.2 Using Newton-Raphson Method for Generating Fractal

If we used this method to start iteration at each point on the real-line, run the iteration until it converged to within tolerance level of a root, and then colour the starting point according to which root it ended up at, what we get is a fractal.

But Fractal in 1 Dimension is not that intuitive, so taking this process to complex plane i.e. performing Newton-Raphson Method at each point in complex plane and colour that point according to the root to which it converged, would give us colourful fractals.

3.1.3 Algorithm used in the Code

1. Choose a function $f(z)$, remember how interesting the fractal looks depends on this choice.
2. Find the roots of $f(z)$ and find the function required for the iteration i.e. $\frac{f(z)}{f'(z)}$.
3. Choose the range of the complex plane, and divide the x-axis and y-axis into m and n points respectively (assuming the dimensions of the image to be generated is (m X n)).
4. Run the iteration for each point on the plance with given tolerance and max number of iterations.
5. Assign the colour to each point according to the root to which it converged.
6. Plot the resulting colour for each grid point.

3.1.4 Precautions

1. As generating fractals requires some heavy computation (can take minutes depending on your configuration), it is recommended to start with generating a low-resolution image and once you are sure everything works as expected, generate the final high-resolution image.
2. The tolerance you choose should be small enough so that if roots lie really close to one another, the program can correctly assign the colour.
3. Choose the initial range of the graph such that it covers almost majority of the roots, so as to see the interesting things happening, and then focus on the part you want.

CHAPTER

FOUR

REFERENCE

4.1 Bibliography

This is a collection of various bibliographic items referenced in the documentation.

4.2 Source Code

4.2.1 Subpackages

fractpy.models package

Submodules

fractpy.models.newton module

class fractpy.models.newton.**NewtonFractal**(*func*, *prec_goal=1e-11*, *nmax=200*)

A class for plotting Newton Fractal for a given function.

Newton fractals are fractals created in the plane of complex numbers. An iteration process with Newton's method (or Newton-Raphson Method) is started at each point on a grid in the complex plane, and a color is assigned to each point according to which of the roots of a given function the iteration converges to. [Sahari, M. and Djellit, I., 2006. Fractal Newton basins. Discrete Dynamics in Nature and Society, 2006, pp.1-16.]

Parameters

- **func** (sympy expression) – The function for which we want to plot fractal (single- variable).
- **prec_goal** (*float, optional*) – Tolerance for how small the iteration step be relative to the point to break the loop for that point i.e. if relative difference of the iteration step and the point is smaller than this value, it will stop the loop for this point.
- **nmax** (*int, optional*) – Number of iterations to be run (default is 200). Minimum recommended value is 50, but for some functions may required over 500.

function

The function for which the Newton Fractal is being generated.

Type fractpy.Function

roots_list

Roots of the function.

Type list

Example

To plot Newton Fractal for f(x) = x**4 - 2x**3 + 10 we first make a model, and then plot it in required range and resolution:

```
>>> model = NewtonFractal("x**4 - 2x**3 + 10")
>>> p = model.plot(-2, 2, -2, 2, (200, 200))
```

Where p is an object of `matplotlib.figure.Figure`.

To make a plot which can be zoomed use `model.zoom_plot()`.

See also:

fractpy.Function A class for performing basic calculus operations on a function (like finding roots).

plot(*xstart*, *xend*, *ystart*, *yend*, *dim=(100, 100)*)

Plots the fractal for given range and dimensions.

Parameters

- **xstart** (*float*) – Lower limit of x-axis
- **xend** (*float*) – Upper limit of x-axis
- **ystart** (*float*) – Lower limit of y-axis
- **yend** (*float*) – Upper limit of y-axis
- **dim** (*list of int, optional*) – The dimensions of the plot to be generated (resolution of the plot, width X height)(default is (100, 100)).

Return type `matplotlib.figure.Figure`

zoom_plot(*xstart*, *xend*, *ystart*, *yend*, *dim=(100, 100)*)

Plots the fractal in two identical panels. Zooming in on the right panel will show a rectangle in the first panel, denoting the zoomed region.

Parameters

- **xstart** (*float*) – Lower limit of x-axis
- **xend** (*float*) – Upper limit of x-axis
- **ystart** (*float*) – Lower limit of y-axis
- **yend** (*float*) – Upper limit of y-axis
- **dim** (*list of int, optional*) – The dimensions of the plot to be generated (resolution of the plot, width X height)(default is (100, 100)).

Return type `matplotlib.figure.Figure`

4.2.2 Submodules

fractpy.function module

A class for performing basic operations on functions.

class fractpy.function.**Function**(*function*)

A class for performing basic operations on given single-variable function. The operations include finding roots, calculating derivative.

Parameters **function** (*str*) – The function of interest (has to be a single variable function).

function

The function of interest.

Type sympy expression

variable

The variable in terms which the function is defined.

Type sympy.Symbol

Notes

This class was mainly developed to be used for fractpy.models.NewtonFractal class.

differentiate()

Differentiates the function.

Returns Derivative of the function.

Return type sympy expression

roots()

Calculate roots of the function.

Returns Roots of the function.

Return type list

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Jasser] Jasser, n.d. Fractal in nature, geometry and algebra. [image] Available at: <http://jasser.nl/about/fractal-geometry/> [Accessed 12 April 2021].

[FractalFoundation2009] Fractal Foundation (2009) Fractal Pack Educators' Guide. [pdf] Available at: <https://fractalfoundation.org/fractivities/FractalPacks-EducatorsGuide.pdf> [Accessed: 27 March 2021].

[Falconer1990] Falconer, K., 1990. Fractal geometry: mathematical foundations and its applications. Chichester: John Wiley & Sons.

[Sahari2006] Sahari, M. and Djellit, I., 2006. Fractal Newton basins. Discrete Dynamics in Nature and Society, 2006, pp.1-16.

INDEX