Linear Fractal Interpolation Function (FIF) Method

Your Name

November 3, 2024

Introduction

- Barnsley established the foundation of fractal interpolation function (FIF) by adopting the principle of iterated function system (IFS).The graph of a FIF is the attractor of a suitable hyperbolic IFS, which is constructed by using several join-up conditions at internal nodes.
- Each interval between two consecutive control points is represented by an affine transformation.
- This transformation introduces self-similar fractal patterns while maintaining the overall shape defined by the control points.

Control Points and Vertical Scaling Factors

- Control points: $(x_0, f_0), (x_1, f_1), \ldots, (x_N, f_N)$
- Vertical scaling factors: $d_1, d_2, \ldots, d_N$
- Total width: $b = x_N - x_0$

Parameter Definitions

For each interval (x_{n-1}, f_{n-1}) to (x_n, f_n), we define the following parameters:

- **Horizontal scaling**: $a_n = \frac{x_n - x_{n-1}}{b}$
- **Horizontal translation**: $e_n = \frac{x_N \cdot x_{n-1} - x_0 \cdot x_n}{b}$
- **Vertical scaling**: $c_n = \frac{f_n - f_{n-1} - d_{n-1} \cdot (f_N - f_0)}{b}$
- **Vertical translation**: $f\_f_n = \frac{x_N \cdot f_{n-1} - x_0 \cdot f_n - d_{n-1} \cdot (x_N \cdot f_0 - x_0 \cdot f_N)}{b}$

Affine Transformation

For any point (x, y) within an interval, the affine transformation is defined as:

$$x_{\text{new}} = a_n \cdot x + e_n$$

$$y_{\text{new}} = c_n \cdot x + d_{n-1} \cdot y + f\_f_n$$

Random Iteration Algorithm (RIA)

The Random Iteration Algorithm applies these transformations iteratively:

1. Randomly select one of the N transformations.
2. Compute x_{new} and y_{new} using the chosen transformation.
3. Plot or record $(x_{\text{new}}, y_{\text{new}})$.

Repeating this process across many iterations generates the fractal pattern.

Python Code Example

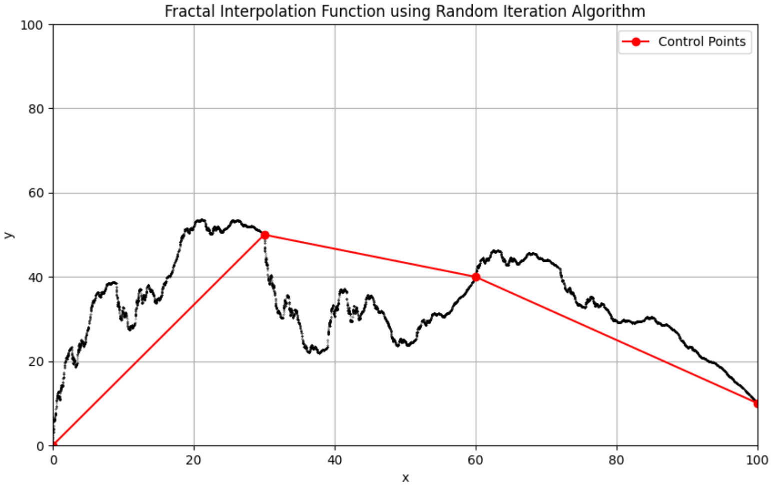
Here's an example of Python code to implement the Random Iteration Algorithm:

```
import random
import matplotlib.pyplot as plt

# Define control points and vertical scaling factors
control_points = [(0, 0), (1, 1), (2, 0.5), (3, 0.75)]
d = [0.9, 0.7, 0.5, 0.3]
N = len(control_points) - 1
b = control_points[-1][0] - control_points[0][0]

# Compute affine transformation coefficients
affine_params = []
for n in range(1, N+1):
    xn, fn = control_points[n]
    xn_1, fn_1 = control_points[n-1]
    dn_1 = d[n-1]
    an = (xn - xn_1) / b
    en = (control_points[-1][0] * xn_1 - control_points[0][0] * xn) / b
    cn = (fn - fn_1 - dn_1 * (control_points[-1][1] - control_points[0][1])) / b
    f_fn = (control_points[-1][0] * fn_1 - control_points[0][0] * fn - dn_1 * (control_points[-1][0] * control_
    affine_params.append((an, en, cn, f_fn))
```

Python Code Example

```
# Function to apply affine transformation
def affine_transform(x, y, params):
    an, en, cn, f_fn = params
    return an * x + en, cn * x + d[n-1] * y + f_fn

# Generate fractal points
points = []
x, y = control_points[0]
for _ in range(10000):
    n = random.randint(0, N-1)
    params = affine_params[n]
    x, y = affine_transform(x, y, params)
    points.append((x, y))

# Plot the result
x_vals, y_vals = zip(*points)
plt.scatter(x_vals, y_vals, s=0.5)
plt.show()
```

Figure



Fractal Interpolation Function using Random Iteration Algorithm
Control Points
100
80
60
40
20
0
y
0
20
40
60
80
100
x

Recursive Function for Fractal Generation

Instead of using the Random Iteration Algorithm (RIA), we can generate the fractal using recursion.
The recursive function splits the interval between control points and applies the affine transformation to generate new points.

- Start at the first control point.
- Recursively apply transformations to refine the fractal curve.
- Stop when a specified depth is reached (e.g., maximum recursion depth).

Python Code Example (Recursive Method)

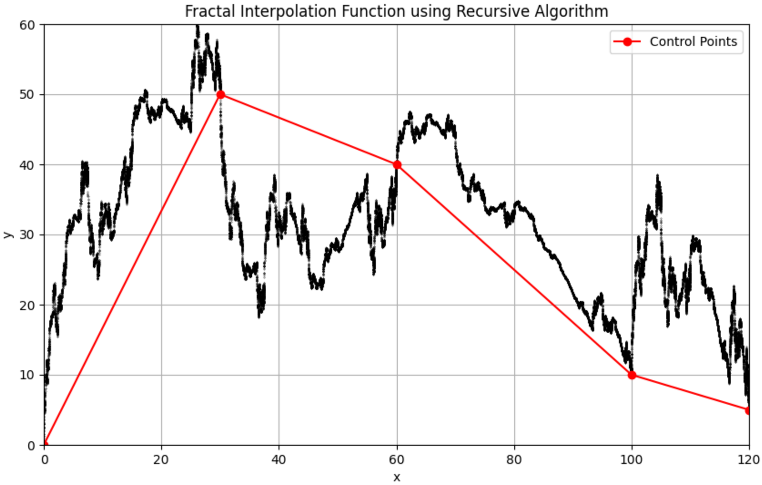
Here's a Python implementation of the fractal generation using recursion:

```
import matplotlib.pyplot as plt

# Define control points and vertical scaling factors
control_points = [(0, 0), (1, 1), (2, 0.5), (3, 0.75)]
d = [0.9, 0.7, 0.5, 0.3]
N = len(control_points) - 1
b = control_points[-1][0] - control_points[0][0]

# Compute affine transformation coefficients
affine_params = []
for n in range(1, N+1):
    xn, fn = control_points[n]
    xn_1, fn_1 = control_points[n-1]
    dn_1 = d[n-1]
    an = (xn - xn_1) / b
    en = (control_points[-1][0] * xn_1 - control_points[0][0] * xn) / b
    cn = (fn - fn_1 - dn_1 * (control_points[-1][1] - control_points[0][1])) / b
    f_fn = (control_points[-1][0] * fn_1 - control_points[0][0] * fn - dn_1 * (control_points[-1][0] * control_
    affine_params.append((an, en, cn, f_fn))
```

Python Code Example (Recursive Method)

```
# Recursive function to generate fractal points
def generate_fif_recursive(x, y, depth, max_depth):
    if depth > max_depth:
        return [(x, y)]

    points = [(x, y)]

    for n in range(N):
        params = affine_params[n]
        an, en, cn, f_fn = params

        # Apply affine transformation
        x_new = an * x + en
        y_new = cn * x + d[n-1] * y + f_fn

        # Recurse for the new point
        points.extend(generate_fif_recursive(x_new, y_new, depth + 1, max_depth))

    return points

# Generate fractal points with recursion
points = generate_fif_recursive(control_points[0][0], control_points[0][1], 0, 5)

# Plot the fractal
x_vals, y_vals = zip(*points)
plt.scatter(x_vals, y_vals, s=0.5)
plt.title('Fractal Interpolation Function (Recursive)')
plt.show()
```

Figure


Fractal Interpolation Function using Recursive Algorithm
Control Points
60
50
40
30
20
10
0
y
0
20
40
60
80
100
120
x

Benefits of Recursive Approach

- Recursion provides a structured approach to refine the fractal curve with depth-based control.
- Recursion can be easier to follow for deterministic fractal generation.
- Can be used to limit the depth of fractal details and control precision.

Conclusion

- The FIF method introduces self-similar fractal detail while maintaining interpolation between control points.
- By using affine transformations, the method allows for random iterative generation of a fractal pattern.
- Applications of FIF include image compression, curve fitting, and more.