mathematics

MDPI

Article

A Concretization of an Approximation Method for Non-Affine Fractal Interpolation Functions

Alexandra Băicoianu [1], Cristina Maria Păcurar [1] and Marius Păun [2,*]

[1] Department of Mathematics and Computer Science, Faculty of Mathematics and Computer Science, Transilvania University of Braşov 50, Iuliu Maniu Str., 500090 Braşov, Romania; a.baicoianu@unitbv.ro (A.B.); cristina.pacurar@unitbv.ro (C.M.P.)

[2] Department of Forest Engineering, Forest Management Planing and Terrestrial Measurements, Faculty of Silviculture and Forest Engineering, Transilvania University of Braşov, 1 Şirul Beethoven, 500123 Braşov, Romania

* Correspondence: m.paun@unitbv.ro

Abstract: The present paper concretizes the models proposed by S. Ri and N. Secelean. S. Ri proposed the construction of the fractal interpolation function (FIF) considering finite systems consisting of Rakotch contractions, but produced no concretization of the model. N. Secelean considered countable systems of Banach contractions to produce the fractal interpolation function. Based on the abovementioned results, in this paper, we propose two different algorithms to produce the fractal interpolation functions both in the affine and non-affine cases. The theoretical context we were working in suppose a countable set of starting points and a countable system of Rakotch contractions. Due to the computational restrictions, the algorithms constructed in the applications have the weakness that they use a finite set of starting points and a finite system of Rakotch contractions. In this respect, the attractor obtained is a two-step approximation. The large number of points used in the computations and the graphical results lead us to the conclusion that the attractor obtained is a good approximation of the fractal interpolation function in both cases, affine and non-affine FIFs. In this way, we also provide a concretization of the scheme presented by C.M. Păcurar.

Keywords: fractal interpolation function; non-affine FIFs; countable affine probabilistic scheme; countable affine deterministic scheme; countable non-linear probabilistic scheme; countable deterministic non-linear scheme

MSC: Primary 28A80; 41A05; Secondary 26A18; 37C25; 37C70

Citation: Băicoianu, A.; Păcurar, C.M.; Păun, M. A Concretization of an Approximation Method for Non-Affine Fractal Interpolation Functions. *Mathematics* **2021**, *9*, 767. https://doi.org/10.3390/math9070767

Received: 10 March 2021
Accepted: 28 March 2021
Published: 1 April 2021

1. Introduction

The notion of fractal interpolation has been introduced by Barnesley in [1] (see also [2]) and it represents a different interpolation method, which results in functions that are continuous, but not necessarily differentiable at every point. The fractal interpolation function (FIF) is a continuous function with real values, which interpolates a given set of data

$$\{(x_i, y_i) \in [x_0, x_N] \times \mathbb{R}, i = \{0, \ldots, N-1\}\}$$

(i.e., $f(x_i) = y_i$ for all $i \in \{0, \ldots, N-1\}$), where x_i are sorted in an ascending order such that its graph is the attractor of an iterated function system.

The significance of FIFs is emphasized by the numerous research directions that have been broadly studied ever since they were introduced. Among these directions, we mention the hidden variable fractal interpolation, which was introduced by Barnsley et al. (see [3]) and generates functions which are not self-referential; thus, being much more less restrictive (see [4–6]), the extension to a countable iterated function systems (a notion introduced in [7–9]) to obtain the corresponding FIFs (see [10,11]) and the replacement of the fixed

point result (Banach fixed point theorem), which guarantees the existence of the FIF with different fixed point results (see [12–15]).

Among the different types of FIFs existing in the literature, there were studied affine FIFs (see [1]), but also non-affine FIFs (see [16]). However, if for the affine case, there have been studies undertaken towards the computational part (see [17–21]), as far as we know, there have not yet been any studies related to non-affine FIFs in this respect.

The aim of the present paper is to offer a concretization of an approximation method for non-affine fractal interpolation functions. Starting from the results in [8,12], in this paper, we propose two different algorithms to produce the fractal interpolation functions in both cases affine and non-affine FIFs. The theoretical context we were working in, suppose a countable set of starting points and a countable system of Rakotch contractions. Due to the computational restrictions the built algorithms, in applications, have the weakness that they uses a finite set of starting points and a finite system of Rakotch contractions. With this respect, the attractor obtained is a two-step approximation. The big amount of points used in the computations and the graphical results lead us to the conclusion that the attractor obtained is a good approximation of the fractal interpolation function in both affine and non-affine cases. In this way, we also provide a concretization of the scheme presented by C.M. Păcurar (see [15]).

In this study, we want to solve also the problem of viewing a big set of data, generated by the iterations schemes mentioned above, in order to better understand the theoretical knowledge in the function plotting field. We study the nature of data plotting in C++ regarding its pros and cons (limitations). The scope of the application is to generate graphs for various functions and to observe the steps taken by the algorithm in order to obtain the correct plotting. Using C++ (one of the fastest and most memory efficient languages) and Qt (a C++ cross-platform framework for GUI - Graphical User Interface), we developed an application that puts into use most of the modern features offered by C++ (especially, C++11 issues).

2. Mathematical Preliminaries

Let (X, d) be a metric space.

Definition 1. *The map $f : X \to X$ is called a Picard operator if f has a unique fixed point $x_* \in X$ (i.e., $f(x_*) = x_*$) and*

$$\lim_{n\to\infty} f^{[n]}(x) = x_*,$$

for every $x \in X$, where $f^{[n]}$ denotes the n-times composition of f with itself.

Definition 2. *1. A map $f : X \to X$ is called Lipschitz if there exists a real non-negative C such that*

$$d(f(x), f(y)) \leq Cd(x, y),$$

for every $x, y \in X$. The smallest C in the above definition is called Lipschitz constant and it is defined as

$$lip(f) = \sup_{x\neq y} \frac{d(f(x), f(y))}{d(x, y)}$$

2. A map $f : X \to X$ is called Banach contraction if there exists $C \in (0, 1)$ such that

$$d(f(x), f(y)) \leq Cd(x, y),$$

for every $x, y \in X$.

3. A map $f : X \to X$ is called φ-contraction if there exists a function $\varphi : [0, \infty) \to [0, \infty)$ such that

$$d(f(x), f(y)) \leq \varphi(d(x, y)),$$

for every $x, y \in X$.

4. *A map $f : X \to X$ is called Matkowski contraction if it is a φ-contraction where $\varphi : [0, \infty) \to [0, \infty)$ is non-decreasing and $\lim_{n\to\infty} \varphi^{[n]}(t) = 0$ for all $t > 0$.*
5. *A map $f : X \to X$ is called Rakotch contraction if it is a φ-contraction where $\varphi : [0, \infty) \to [0, \infty)$ is such that the function $t \to \frac{\varphi(t)}{t}$ is non-increasing for every $t > 0$ and $\frac{\varphi(t)}{t} < 1$ for every $t \in (0, \infty)$.*

Remark 1. 1. *Every Banach contraction is Lipschitz where the Lipschitz constant is smaller than* 1.
2. *Every Banach contraction is a φ-contraction, for*

$$\varphi(t) = C \cdot t,$$

for every $t > 0$.
3. *Every Rakotch contraction is a Matkowski contraction.*

In [22], the following fixed point result was proved.

Theorem 1. *Every Matkowski contraction on a complete metric space is a Picard operator.*

2.1. Iterated Function Systems

Hutchinson introduced the notion of iterated function systems in [23]. Secelean extended the notion to countable iterated function systems, composed of a countable number of constitutive functions (see [8]).

Definition 3. *Let (X, d) be a compact metric space and the continuous functions $f_n : X \to X$. The system of all functions f_n is called a countable iterated function system (CIFS), which will be denoted by $\mathcal{S} = \{(f_n)_{n\geq 0}\}$*

Let $\mathcal{P}_{\text{cp}}(X)$ be the class of all non-empty compact subsets of X.
The fractal operator associated to $\mathcal{S}$ is the map $F_{\mathcal{S}} : \mathcal{P}_{\text{cp}}(X) \to \mathcal{P}_{\text{cp}}(X)$ defined as

$$F_{\mathcal{S}}(K) = \overline{\bigcup_{n\geq 1} f_n(K)}$$

for every $K \in \mathcal{P}_{cp}(X)$.

If the functions f_n are Matkowski contractions (or Rakotch contractions, or φ-contractions, or Banach contractions), the fractal operator associated to the CIFS $\mathcal{S}$ is a Picard operator and its unique fixed point is called the attractor of $\mathcal{S}$, which will be denoted by $A_{\mathcal{S}}$.

2.2. Countable FIFs

Let (Y, d) be a compact metric space and the countable system of data

$$\{(x_n, y_n) \in [x_0, m] \times Y, n \geq 0\}, \tag{1}$$

where the sequence $(x_n)_{n\geq 0}$ is strictly increasing and bounded and $m = \lim_{n\to\infty} x_n$, and the sequence $(y_n)_{n\geq 0}$ is convergent. We make the notation $M = \lim_{n\to\infty} y_n$.

Definition 4. *An interpolation function for the system of data (1) is a continuous function $f : [x_0, m] \to Y$ such that $f(m) = M$ and $f(x_n) = y_n$ for all $n \geq 0$.*

Let us recall from [15], the way we can construct a family of functions associated to the system of data (1):

Let $l_n : [x_0, m] \to [x_n, x_{n+1}]$ be a family of contractive homeomorphisms such that

(i) there exists $C_n \in [0,1)$ such that

$$|l_n(x) - l_n(x')| \leq C_n|x - x'|$$

for every $x,\ x' \in [x_0, m]$;

(ii)

$$l_n(x_0) = x_{n-1} \quad \text{and} \quad l_n(m) = x_n;$$

(iii)

$$\sup_{n\geq 1} C_n < 1.$$

By $diam(A)$ we denote the diameter of A.

Let $F_n : [x_0, m] \times Y \to Y$ be continuous functions such that

(j)

$$F_n(x_0, y_0) = y_{n-1} \quad \text{and} \quad F_n(m, M) = y_n;$$

(jj) $\lim_{n\to\infty} diam(Im\, F_n) = 0$.

We can now define the family of functions $(f_n)_{n\geq 0}$

$$f_n : [x_0, m] \times Y \to [x_0, m] \times Y$$

associated to the system of data (1) as

$$f_n(x, y) = (l_n(x), F_n(x, y)),$$

for every $x \in [x_0, m]$ and $y \in Y$.

Let

$$\mathcal{F}([x_0, m]) = \{f : [x_0, m] \to Y |\, f(x_0) = y_0,\ f(m) = M,\ f \text{ - continuous}\}$$

endowed with the uniform metric $d_{\mathcal{F}([x_0,m])}$.

Remark 2. *(see Theorems 3.2 and 3.3 from [15])*

1. *If the functions F_n are Lipschitz with respect to the first variable and Rakotch contractions with respect to the second variable, then the functions f_n are Rakotch contractions with respect to d_θ, where*
$$d_\theta((x, y), (x', y')) := |x - x'| + \theta d(y, y')$$
for all $(x, y),\ (x', y') \in [x_0, m] \times Y$, where $\theta = \frac{1-\sup_{n\geq 1} C_n}{2(C+1)} \in (0, 1)$.
2. *Given the same aforementioned framework, there exists an interpolation function f_* corresponding to the system of data (1) such that its graph is the attractor of the CIFS $\mathcal{S} = (([x_0, m] \times Y, d_\theta), (f_n)_{n\geq 1})$.*

In the particular case that Y is a compact real interval, $Y \subset (0, \infty)$, we can choose the non-affine functions f_n as follows (see [15]):

$$\begin{aligned} f_n(x, y) = \Bigg(& \frac{x_n - x_{n-1}}{m - x_0} x + \frac{m x_{n-1} - x_0 x_n}{m - x_0}, \\ & \left(\frac{y_n - y_{n-1}}{m - x_0} - \frac{1}{m - x_0} \left(\frac{M}{1 + nM} - \frac{m}{1 + nm} \right) \right) x + \frac{y}{1 + ny} \\ & + y_{n-1} - x_0 \frac{y_n - y_{n-1}}{m - x_0} + \frac{x_0}{m - x_0} \frac{M}{1 + nM} - \frac{m}{m - x_0} \frac{m}{1 + nm} \Bigg). \end{aligned}$$

For the affine case, when Y is a compact real interval, one can choose the functions f_n as follows (see [10]):

$$f_n(x,y) = \left(\frac{x_n - x_{n-1}}{b-a}x + \frac{bx_{n-1} - ax_n}{b-a}, \left(\frac{y_n - y_{n-1}}{b-a} - d_n\frac{M-m}{b-a}\right)x + d_n y + \frac{by_{n-1} - ay_n}{b-a} - d_n\frac{bm - aM}{b-a}\right).$$

3. Computational Background

3.1. Applied Technologies. Motivation (Pros)

Qt is a widget toolkit for creating graphical user interfaces as well as cross-platform applications that run on various software and hardware platforms such as Linux, Windows, macOS, Android or embedded systems with little or no change in the underlying codebase, while still being a native application with native capabilities and speed. Qt Creator is a cross-platform C++, JavaScript and QML integrated development environment which simplifies Graphical User Interface (GUI) application development. It includes a visual debugger and an integrated WYSIWYG (What You See Is What You Get) GUI layout and forms the designer. The editor has features such as syntax highlighting and autocompletion.

One of the main problems encountered during development was using user-input flexible functions, that is why we prioritised adding a fairly robust mathematics parsing engine written in C++. We chose CmathParser (https://github.com/NTDLS/CMathParser, accessed on 15 February 2021) that provides a robust collection of functions and structures that give users the ability to parse and evaluate various types of expressions. Although it is fairly lightweight, CMathParser can interpret a various list of mathematical functions and operations and its performance is convenient in relation to the advantages that this engine brings. The mathematical functions used need to follow a specific syntax (for example, $\sqrt{x}$ is SQRT(x)), this is why we found it useful to read our functions from a file. We opted for reading from an XML (Extended Markup Language) file because we can use the tags in our advantage and clearly define every function and every parameter for that function.

Below, Listing 1, is an example for an XML file accepted by the application:

Listing 1: Example for an XML file.

```
<?xml version="1.0"?>

<function>
<Xn>SQRT(n)/(1+SQRT(n))</Xn>
<Yn>SIN(n)/SQRT(1+n)</Yn>
<FnX>(((XN)-(XN-1))*X+b*(XN-1)-a*(XN))/(b-a)</FnX>
<FnY>((YN)-(YN-1)-(DN)*((M)-m))*X/(b-a)+(DN)*Y+(b*(YN-1)-a*(YN)-(DN)*(b*m-a*(
    M)))/(b-a)</FnY>
<a>0</a>
<b>1</b>
<m>0</m>
<M>0</M>
<dn>0.4</dn>
</function>
```

QcustomPlot (https://www.qcustomplot.com/, accessed on 15 February 2021) is a Qt C++ widget for plotting and data visualization. It has no further dependencies and is well documented. This plotting library focuses on making good looking, publication-quality 2D plots, graphs and charts, as well as offering high performance for real time data visualization applications.

3.2. Technical Notes on Performance

Our target regarding the application's performance focused around optimising the algorithm in a way that it brings up the graphs as soon as possible. To increase the performance, we used multithreading: we use all the available threads on the CPU and developed the algorithm in a way that favors concurrency. The algorithm finds out how

many threads are available and uses them (for a CPU with 4 threads, the algorithm will use a maximum of 4 threads on max load) although it is possible to start any number of threads (the OS scheduler will put them in a priority queue). A program that starts 100 threads for 100 tasks on a 4 threaded CPU will be less performant than the same program that splits those 100 tasks in a way that the CPU takes 4 tasks at a time.

This, Listing 2, is a threading code snippet that spawns as many threads as available to generate points:

Listing 2: Multithreading.

```
int numberThread;

if (std::thread::hardware_concurrency() > 0)
numberThread = std::thread::hardware_concurrency();
else
numberThread = 4;

auto spawnThreads = [&]()
{
std::vector<std::thread> threads;
for (int index = 1; index < numberThread; index++)
{
threads.push_back(std::thread(generate, 0, numberPoint / numberThread,
"..//.//QtExample\\Step1\\file" + std::to_string(index) + ".txt"));
}
threads.push_back(std::thread(generate, 0, numberPoint - numberPoint /
    ↪ numberThread *
(numberThread - 1), "..//.//QtExample\\Step1\\file
    " + std::to_string(numberThread) + ".txt"));

for (auto& th : threads)
{
th.join();
}
};
```

The number of used threads impacts performance, and overall, the quality and time; see Figure 1. It can also be seen that as the number of points increases, the difference between the time associated with running with 8 threads versus 16 threads increases considerably.

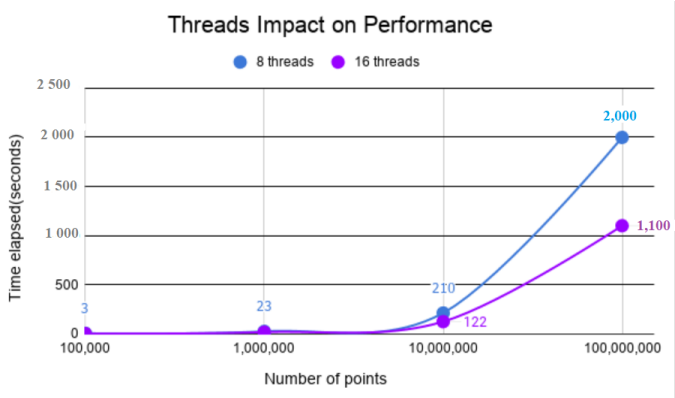

Figure 1. Threads impact on performance.

3.3. Limitations (Constraints)

At the present time, the only known limit of the application is the time required to generate and plot the points when they are billions or even more in number. The next examples were run on a i7-7700HQ with 8 threads. RAM memory is not really relevant because the algorithm is very CPU heavy.

For the probabilistic scheme, generating and plotting 100,000 points are made in approximately 2–3 s, and for every $p * 100{,}000$ (p is a signed integer), the time will be approximately $p * x$, where $x \in [2, 3]$.

For the deterministic scheme, things become challenging. At a low level, the time for generating and plotting points will be the same as for Step 1. The difference occurs in the number of points the scheme generates for parameters k, n, p. leading to run time fluctuations and may occur due to the insertion and processing of points in the files.

4. Main Results

4.1. Countable Fractal Non-Affine Interpolation Schemes

We start the study producing an approximation for the fractal countable non-affine interpolation scheme in two different ways. Let us consider the positive, increasing, convergent sequence $(x_n)_{n\in\mathbb{N}}$, the convergent sequence $(y_n)_{n\in\mathbb{N}}$ defined by $(x_n)_{n\in\mathbb{N}}$, $(y_n)_{n\in\mathbb{N}}$ given by the following:

$$x_n = \frac{3\sqrt{n}+1}{\sqrt{n}+1}, \quad y_n = \frac{\left|\sin\left(\frac{180\cdot n}{\pi}\right)\right|+1}{\sqrt{n}+1}$$

and the sequence of non-affine functions given by the following:

$$\begin{aligned} f_n(x,y) = \Bigg(& \frac{x_n - x_{n-1}}{m-x_0}x + \frac{mx_{n-1} - x_0x_n}{m-x_0}, \\ & \left(\frac{y_n - y_{n-1}}{m-x_0} - \frac{1}{m-x_0}\left(\frac{M}{1+nM} - \frac{m}{1+nm}\right)\right)x + \frac{y}{1+ny} \\ & + y_{n-1} - x_0\frac{y_n - y_{n-1}}{m-x_0} + \frac{x_0}{m-x_0}\frac{M}{1+nM} - \frac{m}{m-x_0}\frac{m}{1+nm}\Bigg). \end{aligned}$$

for all $(x, y) \in [x_0, m] \times Y$. The argument of the function sinus we used when defining y_n was imposed by the fact that the Mathematical Function Library demands us to work in radians.

The absolute value of sinus in the definition of y_n was imposeed by the fact that every second coordinate of the points obtained in the process must be positive in order to have a Rakotch contraction. The result obtained is an approximation of the countable non-affine interpolation schemes. For the desired approximation, we take the following subsets

$$XP = \{x_n \mid n \leq 100\}, YP = \{y_n \mid n \leq 100\}, FP = \{f_n \mid n \leq 100\}$$

The two schemes we used are the probabilistic interpolation scheme and the deterministic interpolation scheme. The probabilistic scheme is described in Algorithm 1.

The deterministic scheme we are triyng to apply is given by the Algorithm 2.

The probabilistic scheme (Algorithm 1), after 100,000 steps, leads us to the result shown in Figure 2.

Algorithm 1: The Probabilistic scheme.

1. Consider an empty set of points $\mathcal{P} \in \mathbb{R}^2$ and p a signifiant big signed positive integer.
2. Generate an arbitrary point $(xa, ya) \in [0,1] \times [0,1]$.
3. Determine $\mathcal{P} \bigcup \{(xa, ya)\}$.
4. Generate a random signed integer $0 < k \leq 100$.
5. Compute $(xa, ya) = f_k(xa, ya)$.
6. Repeat steps 3, 4, 5 p times
7. Sort the elements of the set $\mathcal{P}$ in ascending order regarding the first component of the elements.
8. Plot the function passing through all the points of the set $\mathcal{P}$.

Algorithm 2: The Deterministic Scheme.

1. Consider k the number of the initial points, n the number of functions involved, p the number of steps and define an empty set of points $\mathcal{P} \in \mathbb{R}^2$.
2. Generate randomly a set K_0 of k points in $[0,1] \times [0,1]$.
3. Determine $\mathcal{P} \bigcup K_0$.
4. Compute $\mathcal{P} = f_1(\mathcal{P}) \bigcup f_2(\mathcal{P}) \bigcup .. \bigcup f_n(\mathcal{P})$
5. Repeat step 4 for p times
6. Sort the elements of the set $\mathcal{P}$ in ascending order regarding the first component of the elements.
7. Plot the function passing through all the points of the set $\mathcal{P}$.

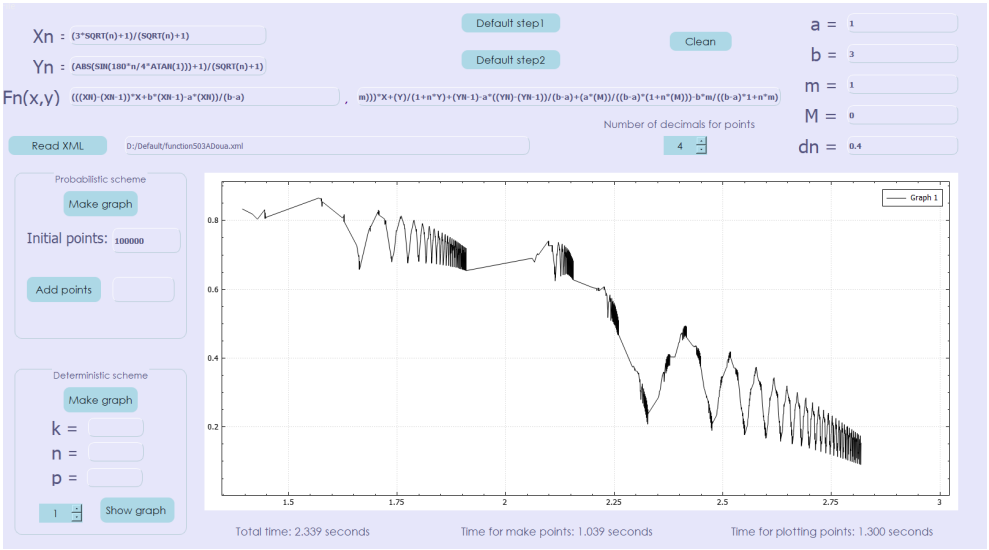

Figure 2. Non-affine probabilistic interpolation scheme (approximation with 100,000 points).

The time spent to generate the points of the scheme was 1.039 s and the plotting time 1.3 s.

Using Algorithm 2, in the initial conditions of the probabilistic scheme, for the deterministic scheme taking $k = 100$, $n = 100$ and $p = 3$, we obtain the graph given in Figure 3. The scheme generated 100,000,000 points and the total duration of computing and plotting was 1474.477 s.

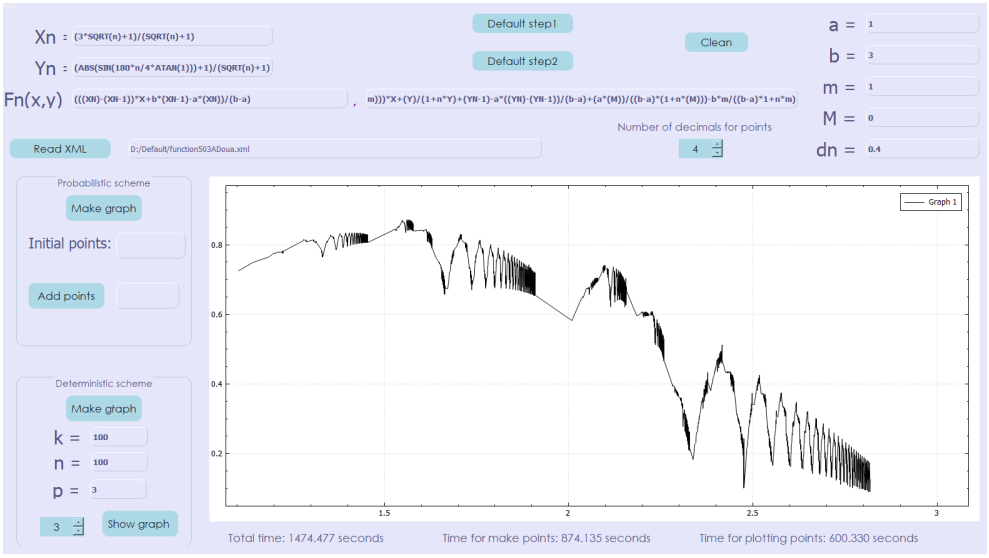

Figure 3. Non-affine deterministic interpolation scheme (approximation with 100,000,000 points).

In Figure 4, a graph with the plotting of the interpolation function in every step of the algorithm is shown.

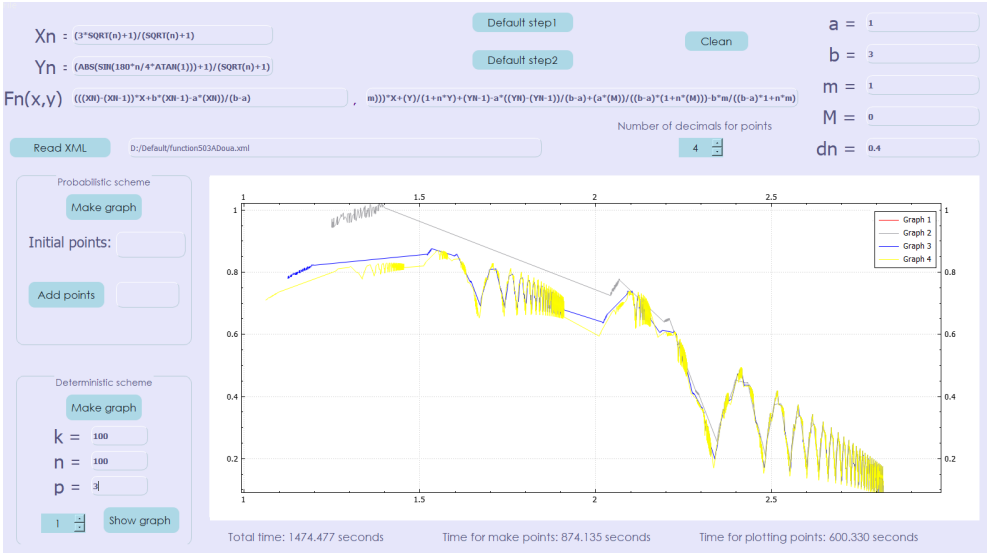

Figure 4. Non-affine deterministic interpolation scheme (approximation with 100,000,000 points), plotting the graph of every step in the algorithm.

4.2. Countable Fractal Affine Interpolation Schemes

We made the study producing an approximation for the fractal countable affine interpolation scheme in the same two ways as for the non-affine case. Let us consider the positive, increasing, convergent sequence $(x_n)_{n\in\mathbb{N}}$, the convergent sequence $(y_n)_{n\in\mathbb{N}}$ defined by

$$x_n = \frac{3\sqrt{n}+1}{\sqrt{n}+1}, \quad y_n = \frac{\cos\left(\frac{180\cdot n}{\pi}\right)+1}{\sqrt{n}+1}$$

and the affine sequence of functions $(f_n(x,y))_{n\in\mathbb{N}}$

$$f_n(x,y) = \left(\frac{x_n - x_{n-1}}{b-a}x + \frac{bx_{n-1} - ax_n}{b-a}, \right.$$
$$\left. \left(\frac{y_n - y_{n-1}}{b-a} - d_n \frac{M-m}{b-a} \right) x + d_n y + \frac{by_{n-1} - ay_n}{b-a} - d_n \frac{bm - aM}{b-a} \right).$$

for all $(x,y) \in [x_0, m] \times Y$. The argument of the function cosinus we used when defining y_n was imposed by the fact that the Mathematical Function Library demands us to work in radians.

For the desired approximation, we take the following subsets

$$XP = \{x_n \mid n \leq 100\}, YP = \{y_n \mid n \leq 100\}, FP = \{f_n \mid n \leq 100\}$$

The probabilistic scheme (Algorithm 1), after 100,000 steps, leads us to the result shown in Figure 5.

The deterministic scheme produces, for $k = 100$, $n = 100$ and $p = 3$, the graph in Figure 6.

The time for obtaining the points in these conditions was 595.631 s and the time for plotting the function was 1594.470 s. The graph with the plotting of the interpolation function in every step of the algorithm is given in Figure 7.

The red graph is for the function given by the random generated 100 points. Graph 2 is the function for 10,000 thousand points (step for p = 1), graph 3 is for the function after step p = 2 (1,000,000 points) and the yellow graph is the final one—the same as in Figure 7.

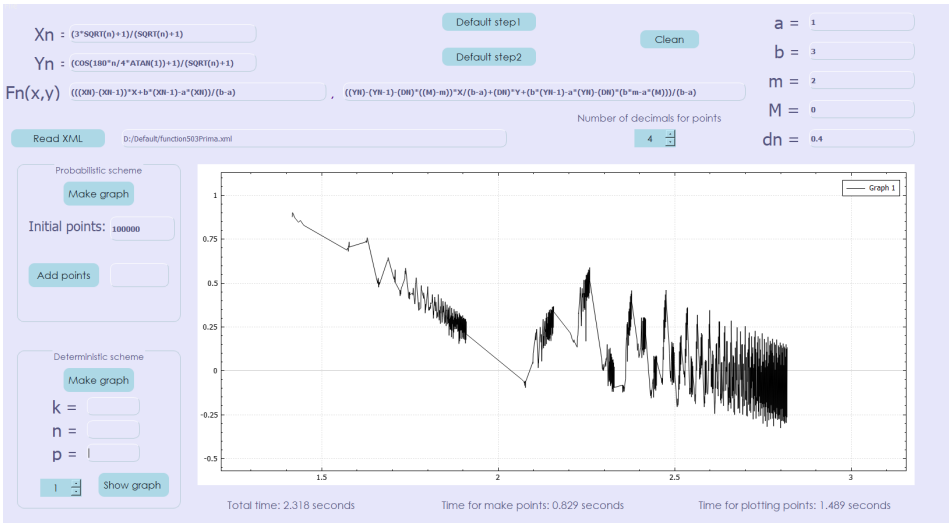

Figure 5. Affine probabilistic interpolation scheme (approximation with 100,000 points).

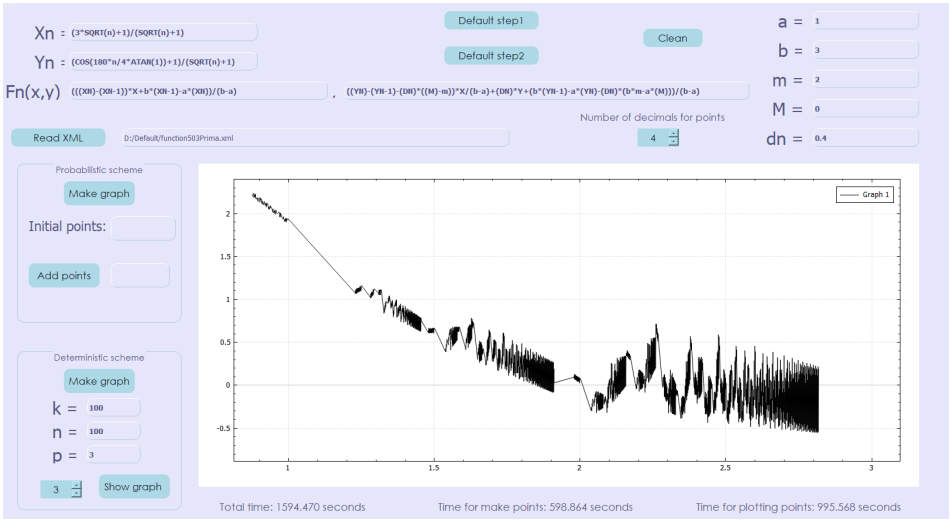

Figure 6. Affine deterministic interpolation scheme (approximation with 100,000,000 points).

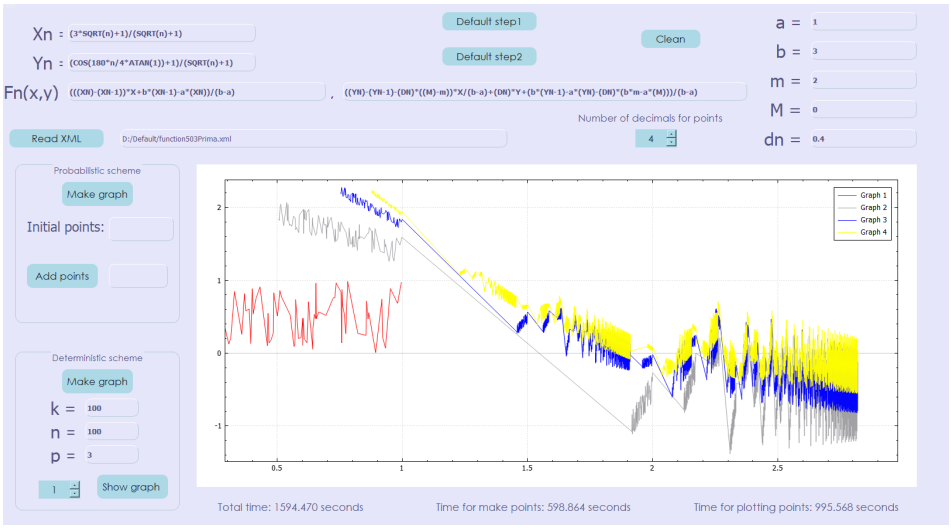

Figure 7. Affine deterministic interpolation scheme (approximation with 100,000,000 points, step by step).

5. Conclusions

The main conclusion of this study is that the algorithms presented give similar approximations of the FIFs for both schemes, affine and non-affine. For the probabilistic scheme (Algorithm 1), significant results are obtained for more than 10,000 steps and the time elapsed to plot the graph is less than two seconds. The deterministic scheme (Algorithm 2) permits the study of the variation of FIFs, step by step, but, in order to obtain significant results one must perform more than three steps. The time elapsed in this case is more than 1000 s. In the applications presented, both algorithms have an imposed number of steps.

Further studies are to be made in order to obtain a condition for stopping the algorithms if some conditions are fulfilled .

Besides the two algorithms, the computer application is a useful tool for plotting big sets of data generated by the iteration schemes described above through functions made in C++. It truly shows the capabilities of this programming language and it pushes it to the maximum using threading and modern programming techniques.

Author Contributions: Conceptualization: M.P.; introduction and preliminaries: C.M.P. and M.P.; methodology: M.P. and A.B.; visualization: A.B.; original draft preparation: M.P.; C.M.P. and A.B.; review, editing, validation, and formal analysis: M.P., C.M.P. and A.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors want to express their gratitude for the reviewers. Their observations were very useful to improve the scientific value of the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Barnsley, M.F. Fractal functions and interpolation. *Constr. Approx.* **1986**, *2*, 303–329. [CrossRef]
2. Barnsley, M. *Fractals Everywhere*; Academic Press: New York, NY, USA, 1988.
3. Barnsley, M.F.; Elton, J.; Hardin, D.; Massopust, P. Hidden variable fractal interpolation functions. *SIAM J. Math. Anal.* **1989**, *20*, 1218–1242. [CrossRef]
4. Mazel, D.S.; Hayes, M.H. Hidden-variable fractal interpolation of discrete sequences. In *ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing*; IEEE Computer Society: Toronto, ON, Canada, 1991.
5. Chand, A.K.B.; Kapoor, G.P. Hidden variable bivariate fractal interpolation surfaces. *Fractals* **2003**, *11*, 277–288. [CrossRef]
6. Bouboulis, P.; Dalla, L. Hidden variable vector valued fractal interpolation functions. *Fractals* **2005**, *13*, 227–232. [CrossRef]
7. Fernau, H. Infinite iterated function systems. *Math. Nachrichten* **1994**, *170*, 79–91. [CrossRef]
8. Secelean, N. Countable Iterated Fuction Systems. *Far East J. Dym. Syst.* **2001**, *3*, 149–167.
9. Secelean, N. *Countable Iterated Function Systems*; LAP Lambert Academic Publishing: Saarbrueken, Germany, 2013.
10. Secelean, N. The fractal interpolation for countable systems of data. *Univ. Beograd. Publ. Elektrotehn. Fak. Ser. Mat.* **2003**, *14*, 11–19. [CrossRef]
11. Secelean, N. Fractal countable interpolation scheme: Existence and affine invariance. *Math. Rep. (Bucur.)* **2011**, *13*, 75–87.
12. Ri, S. A new idea to construct the fractal interpolation function. *Indag. Math.* **2018**, *29*, 962–971. [CrossRef]
13. Kim, J.; Kim, H.; Mun, H. Nonlinear fractal interpolation curves with function vertical scaling factors. *Indian J. Pure Appl. Math.* **2020**, *51*, 483–499. [CrossRef]
14. Ri, S.; Drakopoulos, V. How Are Fractal Interpolation Functions Related to Several Contractions? In *Mathematical Theorems—Boundary Value Problems and Approximations*; Alexeyeva, L., Ed.; IntechOpen: London, UK, 2020.
15. Pacurar, C.M. A countable fractal interpolation scheme involving Rakotch contractions. *arXiv* **2021**, arXiv:2102.09855.
16. Dalla, L.; Drakopoulos, V.; Prodromou, M. On the box dimension for a class of non-affine fractal interpolation functions. *Anal. Theory Appl.* **2003**, *19*, 220–233. [CrossRef]
17. de Amo, E.; Chiţescu, I.; Diaz Carrillo, M.; Secelean, N.A. A new approximation procedure for fractals. *J. Comput. Appl.* **2003**, *151*, 355–370. [CrossRef]
18. Dubuc, S.; Elqortobi, A. Approximations of fractal sets. *J. Comput. Appl. Math.* **1990**, *29*, 79–89. [CrossRef]
19. Chiţescu, I.; Miculescu, R. Approximation of fractals generated by Fredholm integral equations. *J. Comput. Anal. Appl.* **2009**, *11*, 286–293.
20. Chiţescu, I.; Georgescu, H.; Miculescu, R. Approximation of infinite dimensional fractals generated by integral equations. *J. Comput. Appl. Math.* **2010**, *234*, 1417–1425. [CrossRef]
21. Miculescu, R.; Mihail, A.; Urziceanu, S.-A. A new algorithm that generates the image of the attractor of a generalized iterated function system. *Numer. Algorithms* **2020**, *83*, 1399–1413. [CrossRef]
22. Matkowski, J. Integrable solutions of functional equations. *Dissertationes Math.* **1975**, *127*, 68.
23. Hutchinson, J. Fractals and self similarity. *Indiana Univ. Math. J.* **1981**, *30*, 713–747. [CrossRef]