informatics

MDPI

Article

Exact Analysis of the Finite Precision Error Generated in Important Chaotic Maps and Complete Numerical Remedy of These Schemes

Constantinos Chalatsis [1], Constantin Papaodysseus [1,*], Dimitris Arabadjis [2], Athanasios Rafail Mamatsis [1] and Nikolaos V. Karadimas [3]

[1] School of Electrical & Computer Engineering, National Technical University of Athens, Iroon Polytechniou 9, Zografou, 15780 Athens, Greece; khalatsi@mail.ntua.gr (C.C.); trmamatsis@mail.ntua.gr (A.R.M.)
[2] School of Engineering, University of West Attica, Petrou Ralli & Thivon 250 Egaleo, 12241 Athens, Greece; darampatzis@uniwa.gr
[3] Division of Mathematics and Engineering Science, Department of Military Science, Hellenic Army Academy, Evelpidon Avenue, 16672 Vari, Greece; nkaradimas@sse.gr
* Correspondence: cpapaod@cs.ntua.gr; Tel.: +30-21-0772-1476

Abstract: A first aim of the present work is the determination of the actual sources of the "finite precision error" generation and accumulation in two important algorithms: Bernoulli's map and the folded Baker's map. These two computational schemes attract the attention of a growing number of researchers, in connection with a wide range of applications. However, both Bernoulli's and Baker's maps, when implemented in a contemporary computing machine, suffer from a very serious numerical error due to the finite word length. This error, causally, causes a failure of these two algorithms after a relatively very small number of iterations. In the present manuscript, novel methods for eliminating this numerical error are presented. In fact, the introduced approach succeeds in executing the Bernoulli's map and the folded Baker's map in a computing machine for many hundreds of thousands of iterations, offering results practically free of finite precision error. These successful techniques are based on the determination and understanding of the substantial sources of finite precision (round-off) error, which is generated and accumulated in these two important chaotic maps.

Keywords: finite precision error generation; finite precision error accumulation; stabilization of algorithms; stable Bernoulli's map; stable Baker's map

Citation: Chalatsis, C.; Papaodysseus, C.; Arabadjis, D.; Mamatsis, A.R.; Karadimas, N.V. Exact Analysis of the Finite Precision Error Generated in Important Chaotic Maps and Complete Numerical Remedy of These Schemes. *Informatics* **2021**, *8*, 54. https://doi.org/10.3390/informatics8030054

Academic Editor: Olga Kurasova

Received: 7 June 2021
Accepted: 10 August 2021
Published: 15 August 2021

1. Introduction

In recent years, a quite extensive research in connection with Bernoulli's and Baker's maps applications takes place. However, as we demonstrate in the present work, and as it is also referred to in the bibliography (e.g., [1]), a serious amount of finite precision error is accreted during the execution of these two algorithms.

As far as the applicability of Bernoulli's map is concerned, we would like to emphasize that the use of Bernoulli chaotic maps grows constantly. Thus, for example, this chaotic map is employed for image watermarking [2]; in this publication, the authors perform a statistical analysis of a watermarking system based on Bernoulli chaotic sequences. In [3], a comparison of the performance of various chaotic maps, used for image watermarking, is presented.

Another application of the Bernoulli map is associated with the construction of reliable random number generators. In this framework, the authors of [4] employ chaotic true orbits of the Bernoulli map on quadratic algebraic integers, in order to implement a pseudo-random number generator. The authors report that the generated numbers manifest good randomness. In [5], a hardware implementation of pseudo-random number generators based on chaotic maps is introduced.

Bernoulli map is also employed in cryptography. In particular, a number of image encryption algorithms based on this chaotic map have been proposed, given that the associated encryption schemes are fast and easily implemented in hardware and software. For example, the authors of [6] use the Bernoulli chaotic map in order to embed a secret message in an image.

In connection with Baker's map, we would like to cite the following: In [7] an alternative chaotic image encryption based on Baker's map is presented. The authors mention that this enhanced symmetric-key algorithm can support a variable-size image, in contrast to other encryption algorithms, which are mainly based on Baker's map that require only a square image. In [8], a method for obtaining cryptographically strong 8×8 substitution boxes (S-boxes) is presented. The method is based on chaotic baker's map and a "mini version" of a new block cipher. In [9], a family of pure analog coding schemes, with good properties, constructed from dynamic systems, which are governed by chaotic functions—Baker's map, is proposed. The authors of [10] show that pattern classification systems can be designed based upon training algorithms, such as Baker's map, designed to control the qualitative behavior of a nonlinear system. In [11], a comparison of the efficiency of three chaotic maps—Baker's, Cat and Line maps—is performed, as far as cryptography is concerned.

However, both Bernoulli's and Baker's maps manifest serious finite precision error, when implemented in a contemporary computing machine. The accumulation of this finite precision error makes both algorithms generate completely unreliable results, after a relatively very small number of iterations (e.g., [1,12]).

Brief Summary and Organization of the Present Work

Here, the instability of these two chaotic maps, the Bernoulli map and the Baker's one, is confirmed and the actual reason for the causation and accumulation of the related finite word length numerical error is spotted and demonstrated. Moreover, novel methods for the complete stabilization of these algorithmic schemes are introduced. More specifically:

In Section 2 of the manuscript in hand, the authors give the employed notation and symbolism. Moreover, they state a set of fundamental definitions upon which the comparison of two arbitrary floating-point numbers is achieved in finite precision. Eventually, a crucial definition is derived, which allows for the exact evaluation of a number of erroneous digits with which an arbitrary quantity is computed in a machine that employs a finite word length.

In Section 3, the authors demonstrate that, in any subtraction executed in a computing machine using a finite word length, there might be generated an even serious amount of finite precision error, which is due to two types of numerical inaccuracy: a deterministic or causal one and a random or an erratic error. The authors establish that the causal error is due to the difference between the exponent of the obtained subtraction result and the maximum exponent of the subtraction operands. On the other hand, the erratic error is intimately connected to the method that the computing machine employs in order to fill in the missing digits.

In Section 4, the exact sources of finite precision error in the Bernoulli chaotic map are shown for the first time. It is demonstrated that these sources make the results this algorithm offers totally unreliable, after a relatively very small number of iterations; this renders the classical execution of the Bernoulli map totally inapplicable in practice. We strongly emphasize that this rapid failure of the Bernoulli algorithm occurs even when a particularly large finite word length is employed, say including 40,000 decimal digits or more, e.g., by using tools offering unlimited precision arithmetic. On the basis of these results, a novel method for the complete stabilization of the Bernoulli map is introduced.

In Section 5, the authors proceed along similar directions, in connection with Backer's map. In fact, the exact sources of finite precision error of this chaotic map are introduced for the first time. It is, again, demonstrated that these sources make the results offered by Baker's algorithm completely unreliable, after a relatively very limited number of recursions; this renders the classical execution of the Baker's map, too, entirely inapplicable

in practice. We, once more, would like to emphasize that these serious numerical problems appear causally, too, independently of the employed finite word length, no matter how big this number is. Using the previous results, a novel method for the complete stabilization of the Baker's map is accomplished for the first time.

In Section 6, a conclusion incorporating a summary of the obtained results is presented.

2. Employed Symbolism, Notation and a Number of Fundamental Definitions

For any quantity a expressed in scientific format or canonical form, like for example the IEEE 754 floating-point format, we shall employ the notation (i) $man(a)$ for the mantissa of a and (ii) $E(a)$ for the exponent of a.

The analysis that will be presented below is made in connection with the decimal arithmetic system, simply because this is far more familiar to the user. However, we stress that the related analysis is very well applicable to the binary system, too. In fact, both the theoretical and the experimental results introduced in the present work show that the associated approach, which is based on the decimal radix, is a very reliable and robust model for the actual procedures that take place in a contemporary computing machines that use the binary radix.

Abbreviations and Employed Notation

- Acronym e. d. d. stands for "erroneous decimal digits", while acronym c. d. d. stands for "correct decimal digits" and acronym d. d. stands for "decimal digits".
- The abbreviation f. p. e. stands for "finite precision error". We note, in passing, that a number of researchers prefer the name "round-off error" to the "finite precision" one. However, we will establish in the following that the generated and accumulated error in an algorithm, after a number of iterations, when it is executed, is not a "round off error" anymore; for this reason, we have definitely preferred to use the term f. p. e.
- The term "the algorithm is destroyed due to f. p. e." or "the algorithm fails" expresses the fact that the algorithm in hand offers completely unreliable/erroneous results, at a specific recursion.

Next, suppose that one uses a computing machine in which all quantities are written in canonical form with n decimal digits in the mantissa. Suppose in addition that all operations in this machine are performed with the same finite word length and let a be an arbitrary quantity in this computing environment. In the ideal case, where all number representations and operations are made with infinite precision, then let this arbitrary quantity a take the value a^c (superscript c for correct value). In the following, we will give a rigorous relation among a and a^c by means of the subsequent definitions.

Definition 1. *Consider two numbers, p_n, q_n of the same sign, written in canonical exponential form, with the same number n of decimal digits in the mantissa, say:*

$$\begin{array}{l} p_n = \xi_1 \cdot \xi_2 \xi_3 \ldots \xi_n \cdot 10^{\tau} \\ q_n = \delta_1 \cdot \delta_2 \delta_3 \ldots \delta_n \cdot 10^{\rho} \end{array} \quad , \tag{1}$$

and without any loss of generality, let us assume that $\tau \geq \rho$.

Therefore, these two numbers differ by k decimal digits, $k \geq 0$, if and only if

$$|p_n - q_n| = w \cdot 10^{\tau-(n-k)} \quad where \;\; 1 \leq w < 10 \tag{2}$$

We emphasize that if the aforementioned formula offers a negative k then, by definition, it holds that $k = 0$, namely p_n and q_n are identical as far as the n first decimal digits are concerned.

On the contrary, if $k \geq n$, then the two numbers p_n and q_n differ completely.

Thus, according to this definition, the two numbers, written below:

$$n_1 = 5.12345\mathbf{678912} \times 10^9,$$
$$n_2 = 5.12347\mathbf{654321} \times 10^9$$

differ by seven (7) decimal digits, shown in bold, since:

- $\tau = \rho = 9$
- $n = 12$
- Their absolute difference is $|n_1 - n_2| = 1.975409 \times 10^4$

  Hence:

$$\tau - (n - k) = 4 \Leftrightarrow k = 7$$

Another representative example is demonstrated below:

$$n_1 = 6.957600112134568 \times 10^8$$
$$n_2 = 6.957599577134568 \times 10^8$$

A simple inspection might lead someone to deduce that these two numbers differ by twelve (12) decimal digits. Actually, and according to Definition 1 the following hold:

- $\tau = \rho = 8$
- $n = 16$
- Their absolute difference is $|n_1 - n_2| = 5.350000000000000 \times 10^1$

  Hence:

$$\tau - (n - k) = 1 \Leftrightarrow k = 9$$

Therefore, the two aforementioned numbers differ in nine (9) decimal digits, contrary to a probable initial expectation.

Similarly, the two numbers:

$$n_1 = 1.000000000 \times 10^{\tau}$$
$$n_2 = 9.999999999 \times 10^{\tau - 1}$$

have $n-$ correct decimal digits, since:

- $n = 10$
- $\tau \in Z$
- $|n_1 - n_2| = 1 \times 10^{-10+\tau}$

  Hence:

$$\tau - (n - k) = -10 + \tau \implies k = 0$$

On the basis of the previous analysis, the sought-for relation between a and a^c is given via the subsequent definition:

Definition 2. *The floating-point number a has been evaluated with exactly the last λ decimal digits erroneous, if and only if the restriction-projection a^c into n decimal digits and a, differ by λ digits on the basis of Definition 1.*

Subsequently, consider any computing machine, which uses a precision practically equivalent to n decimal digits in the mantissa; then, for an arbitrary floating-point number represented in this machine, it holds that:

$$(\text{number of correct decimal digits of } a) + (\text{erroneous d.d. of } a) = n \tag{3}$$

Or equivalently, $\lambda^c + \lambda = n$, where λ^c is the number of the correct decimal digits of a. For this reason, we shall use terms $c.d.d.$, $e.d.d.$, λ^c and λ interchangeably.

We also note that in IEEE 754 floating-point format, the mantissa is represented in a computer machine by a specific number of bits, say m. Then, the same number is represented in the decimal

system by n decimal digits (d. d.), where n is approximated in practice by the nearest integer of quantity m·lg2.

3. Establishing That in an Arbitrary Subtraction the Finite Precision Error Consists of Two Types: A Deterministic and an Erratic One

In the present section, we will show that during the execution of an arbitrary subtraction in a computing machine, two types of finite precision error may emerge: the first kind of f. p. e. necessarily arises in many circumstances, when specific conditions hold and then, it may be computed explicitly via a closed formula; for this reason, we shall employ the term "causal" or "deterministic" f. p. e. for it. The other kind of f. p. e., which may be generated in an arbitrary subtraction is strongly associated with the random way with which the computer fills in the mantissa digits, when it executes a left shift so as to restore the number in its canonical form; consequently, we shall employ the name "erratic f. p. error" for this form of f. p. e. We will try to analyze these two types of "round-off error", immediately below.

The associated analysis will be made in the decimal arithmetic system without any loss of generality; as we have already pointed out, this approach will offer a very reliable, robust and comprehensive model of the actual procedures that take place in a computing machine, based on the standard IEEE 754 form.

3.1. The Deterministic f. p. Error during Subtraction

In the present section, we will demonstrate that during any subtraction of numbers having the same sign, it is possible to have an amount of finite precision error generated causally. We would like to stress that this generated amount of f. p. e. may be arbitrarily large, in the sense that the number of erroneously computed decimal digits in the operation of subtraction may increase up to the finite word length itself. In the following, we will establish that this deterministic f. p. e. is due to the difference between the exponent of the result of the subtraction and the maximum exponent of the subtracted terms. In fact, if τ is the maximum exponent of the two subtraction operands, both written in scientific format, and if ε is the exponent of the difference, when written in canonical form, then, causally, $(\tau - \varepsilon)$ additional e. d. d. are accreted in the mantissa. For this reason, exactly, we shall call this type of f. p. error "causal" or "deterministic", while we shall also use the term "exponent plunge" for the difference:

$$d = (\tau - \varepsilon) \tag{4}$$

We will attempt to clarify the previous statements by means of the examples that follow.

Example 1. *Consider the two positive, undermentioned numbers, p_{32}, q_{32} with thirty two (32) decimal digits in the mantissa. Moreover, consider the representation of the same numbers with sixteen (16) d. d. in the mantissa, say p_{16}, q_{16}, having all their decimal digits correct, as shown below:*

$$p_{16} = 6.321456789456123 \times 10^5$$
$$q_{16} = 6.321456712896587 \times 10^5$$
$$p_{32} = 6.32145678945612328965874123654 78 \times 10^5$$
$$q_{32} = 6.32145671289658712547896325874 12 \times 10^5$$

then,

$$s_{16} = p_{16} - q_{16} = 7.655953640000000 \times 10^{-3}$$
$$s_{32} = p_{32} - q_{32} = 7.655953616417977 \times 10^{-3}$$

The maximum exponent of p_{16}, q_{16} is five (5) while the exponent of the result s_{16} is minus three (-3); therefore, the exponent's plunge is $d = \tau - \varepsilon = 8$. Consequently, one expects that a deterministic f. p. e. will be generated in the execution of this subtraction, corresponding to eight (8) additional e. d. d. This may become evident by comparing s_{16} and s_{32}.

Example 2. *Consider the following two numbers:*

$$p_{16} = 4.985214589614258 \times 10^1$$
$$q_{16} = 1.958741286325416 \times 10^0$$
$$p_{32} = 4.9852145896142581187456321456789 \times 10^1$$
$$q_{32} = 1.9587412863254169854123458965412 \times 10^0$$

Then:

$$s_{16} = p_{16} - q_{16} = 4.789340460981717 \times 10^1$$
$$s_{32} = p_{32} - q_{32} = 4.789340460981716 \times 10^1$$

Consequently, from Definitions 1 and 2, it follows immediately that s_{16} differs from s_{32} *in exactly one decimal digit.*

In fact, the following rigorous result holds:

Proposition 1. *Suppose that all associated quantities are computed with n decimal digits in the mantissa. Consider an arbitrary number evaluated via a subtraction, i.e., by means of a formula of the type:*

$$s_n = p_n - q_n \,, \quad p_n \cdot q_n > 0 \tag{5}$$

where, $s_n = \gamma_1 \cdot \gamma_2 \ldots \gamma_n \cdot 10^{\varepsilon}$, $p_n = \xi_1 \cdot \xi_2 \xi_3 \ldots \xi_n \cdot 10^{\zeta}$, $q_n = \delta_1 \cdot \delta_2 \delta_3 \ldots \delta_n \cdot 10^{\theta}$ *and* $\zeta \geq \theta$.

Suppose that, due to the previous finite word length computations, both quantities p_n , q_n *have been evaluated with exactly* λ^c *correct decimal digits or equivalently with* $\lambda = n - \lambda^c$ *erroneous d. d. In addition, let us write* $p_n = p^c + x \cdot 10^{\zeta - \lambda^c}$ *and* $q_n = q^c + y \cdot 10^{\theta - \lambda^c}$, *where* p^c, q^c *are the correct values of quantities* q_n, p_n, *should all operations were made with infinite precision, and* x, y *are the mantissae of f. p. error. Suppose moreover that inequality* $1 \leq \left| x - y \cdot 10^{\theta - \zeta} \right| < 10$ *holds. In this case* s_n *is computed with the first* $(\lambda^c - d)$ *correct decimal digits.*

In other words, the initial number of erroneous decimal digits has been increased by d in a causal manner. We should stress that this number of e. d. d. is generated deterministically; however, it may be modified somehow in an erratic manner, as it will be briefly indicated in Section 3.2.

Proof of Proposition 1. In the beginning, we must clarify that, since λ_c is the number of correct d. digits in the mantissa of p_n and q_n, then it indeed holds that $p_n = p^c + x \cdot 10^{\zeta - \lambda^c}$ and $q_n = q^c + y \cdot 10^{\theta - \lambda^c}$. □

Now, we consider that $(p^c - q^c)$ is of order ε, strictly smaller than ζ and θ, and let $d = \zeta - \varepsilon$. Consequently, $s_n = p_n - q_n = (p^c - q^c) + (x - y \cdot 10^{\theta - \zeta}) 10^{\zeta - \lambda^c} = \gamma^c \cdot 10^{\zeta - d} + (x - y \cdot 10^{\theta - \zeta}) 10^{\zeta - \lambda^c} = \left(\gamma^c + (x - y \cdot 10^{\theta - \zeta}) 10^{d - \lambda^c} \right) \cdot 10^{\zeta - d}$. In addition, we have adopted the assumption that $1 \leq \left| x - y \cdot 10^{\theta - \zeta} \right| < 10$ holds. Therefore, quantity $(p_n - q_n)$ is computed with d additional erroneous decimal digits or equivalently with $(\lambda^c - d)$ correct decimal digits, for the following reasons: After subtracting the mantissa of p_n and q_n and since the exponent of the subtraction result plunges by d, the obtained result will have the first d decimal digits equal to zero; i.e.,

$$s_n = p_n - q_n = \xi_1 \cdot \xi_2 \xi_3 \ldots \xi_n \cdot 10^{\zeta} - \delta_1 \cdot \delta_2 \delta_3 \ldots \delta_n \cdot 10^{\theta} = 0.00 \ldots 0 \gamma_1 \ldots \gamma_{n-d} \cdot 10^{\zeta} \tag{6}$$

However, at this stage, the machine performs an equivalent of d decimal left shifts in order to restore s_n into its scientific format. These left shifts will generate d extra incorrect decimal digits in the "tail" of the mantissa of the difference. Since we have assumed that inequality $1 \leq \left| x - y \cdot 10^{\theta - \zeta} \right| < 10$ holds, error mantissae x and y do not contribute additionally to the overall f. p. error during subtraction.

We would like to point out that the operation of subtraction has the very important peculiarity "to be able" to generate at once, an arbitrarily large number of e. d. d. up to the employed word length. The other fundamental operations, addition, multiplication and division, do not have this property, as the authors will support in other research works.

However, both multiplication and division may also generate a large number of e. d. d., when they are repeatedly applied.

3.2. The Erratic Finite Precision Error Appearing in the Operation of Subtraction

According to the analysis introduced in Section 2, the eventual causal error that is generated in the operation of subtraction may be modified in an erratic-stochastic manner, strongly associated with the way the computing machine fills the digits that are deterministically lost during the subtraction. Evidently, the restoration of the lost digits takes place via successive left shifts, made as a rule in the binary radix environments of a contemporary computing machine. However, due to the aforementioned relation between the numbers of a set of binary digits in one hand and the equivalent decimal digits on the other, one may safely consider that the left shifts that restore the canonical form, are made in the decimal arithmetic system; this statement will be made definitely clear in the subsequent analysis and it will be established in a forthcoming manuscript of the authors.

In any case, the underlying reasons for the appearance of an erratic finite precision error in the result of an arbitrary subtraction are the following:

1. Suppose that the first digit to be replaced is the one located at the $\psi - th$ position, where $\psi = \lambda^c - d + 1$; according to the convention in symbolism adopted in the present work, λ^c is the number of the correct most significant (MS) decimal digits and d the exponent's eventual plunge occurring after the subtraction (Section 3.1). Then, the event of obtaining an additional correct decimal digit is equivalent to the event that the computing machine fills the $\psi - th$ position of the mantissa of difference s_n with the correct digit.
2. The event of obtaining two correct decimal digits in the mantissa of s_n is equivalent to the case where the machine fills both the $\psi - th$ and $(\psi + 1) - th$ digits with the correct ones and so forth.
3. The eventuality that the number of correct d. d. after the evaluation of the deterministic error is increased by one, is equivalent to the case where the round-off procedure that occurs in the $\psi - th$ position of the mantissa of s_n, generates an additional erroneous decimal digit.
4. Clearly, if the round-off approximation that the machine performs in the mantissa of s_n does not generate an additional erroneous d. d., then the erratic error does not change the effects of the deterministic one.

Suppose that one knows the algorithm with which the computing machine fills in the least significant (LS) lost digits after subtraction and the associated statistical properties. Then, it is rather straightforward, but lengthy and tedious to evaluate the various probabilities of modifying the deterministic error by a specific number of decimal digits. Of course, such an evaluation requires knowledge of the statistical distribution of the accreted finite precision error, during the execution of an algorithm; the authors will exhaustively tackle this problem for various pdfs in future work. For the time being, we quote a restricted number of quite typical probability values concerning the appearance of an erratic finite precision error in an arbitrary subtraction result. A class of associated results are presented in Table 1. This table refers to the case where the finite precision error generated by the computing machine follows a binomial Gaussian distribution. In fact, it refers to the corresponding probabilities of changing the deterministic error, for two different values of the standard deviation σ of the binomial Gaussian distribution.

Table 1. The probabilities of the deterministic modification of normally distributed finite precision error.

-	**Examples of Theoretical Erratic Probabilities, Symbolized as P^{eq} , during Subtraction, for Two f. p. e. Standard Deviations σ**	
Number of erroneous d. d. randomly generated during subtraction	First std of the Gaussian pdf $\sigma_1 = 1.65$	Second std of the Gaussian pdf $\sigma_2 = 4.9$
Increased by one	6.20×10^{-3}%	14.31%
Retained the same	68.27%	68.66%
Reduced by one	28.06%	10.86%
Reduced by two	3.53%	1.05%
Reduced by three	0.23%	6.63×10^{-2}%

The results appearing in this table refer to the case where the finite precision error generated during an arbitrary subtraction follows a joint Gaussian distribution with standard deviation σ_1 or σ_2. Each row of the Table corresponds to the modification of the deterministic error by a specific number of decimal digits. Finally, we have employed the symbolism P^{eq}, since the probabilities presented here, correspond to the "worst case scenario" by far, where, in the subtraction $s_n = p_n - q_n$, operands p_n and q_n have already been evaluated with exactly the same number λ of erroneous LS d. d.

4. Finite Precision Error Analysis and Stabilization of Bernoulli Map

4.1. A Brief Description of Bernoulli's Map and Its Finite Precision Error Properties

Let x_0 be a floating-point quantity such that $0 \leq x_0 < 1$. Then, the Bernoulli map, starting from x_0 generates a sequence of floating-point numbers x_{k+1} as follows:

$$x_{k+1} = \begin{cases} 2x_k & if \quad 0 \leq x_k < \frac{1}{2} \\ 2x_k - 1 & if \quad \frac{1}{2} \leq x_k < 1 \end{cases} \tag{7}$$

It has been observed by many researchers (e.g., [1]) that the number of incorrect decimal digits generated during the computation of the Bernoulli map grows with the number of performed iterations. This continuous increase of the accumulated amount of finite precision error in x_{k+1}, eventually makes the results of the computations totally unreliable. In [1], it has been shown that this round-off error practically doubles at each iteration.

4.2. The Actual Cause of Failure of the Bernoulli Map Due to Finite Precision

In the following, we shall give an explanation concerning how this finite precision error is generated and accumulated and, using these results, we will show a method for generating the correct values x_{k+1} of the Bernoulli map, for an arbitrarily large number of iterations, when x_0 has a finite representation.

In order to achieve the goal of determining exactly how the finite precision error is accreted during the execution of the Bernoulli map, the following Lemma will prove very useful:

Lemma 1. *Suppose that one executes a series of operations in a computing machine that uses a finite word length, say with n decimal digits in the mantissa. Then, suppose, in addition, that a specific but otherwise arbitrary quantity z has already been computed with a total amount z^e of finite precision error that gives rise to λ incorrect decimal digits in the mantissa of z. Without harming the general case, suppose that the exponent of z in canonical form is τ, i.e., $z = man(z)\ 10^{\tau}$.*

Suppose, now, that quantity z is multiplied by an arbitrary error-free quantity, say ω, $1 \leq |\omega| < 10$; according to the analysis of the previous sections, this restriction of the $\omega-$ value suffices for the finite precision error analysis in connection to an arbitrary error-free value of ω. Then, the following hold:

i. *If* $|\omega \cdot man(z)| \geq 10$ *and* $|\omega \cdot man(z^e)| < 10 \Rightarrow \omega \cdot z$ *is computed with an additional correct decimal digit in its mantissa in comparison with* z.

ii. *If* $|\omega \cdot man(z)| < 10$ *and* $|\omega \cdot man(z^e)| \geq 10 \Rightarrow$ *quantity* $\omega \cdot z$ *is computed with one less correct digit than* z.

iii. *If* $|\omega \cdot man(z)| < 10$ *and* $|\omega \cdot man(z^e)| < 10$ *hold* $\Rightarrow$ *quantities* $\omega \cdot z$ *and* z *are computed with* λ *erroneous decimal digits in their mantissa. The same occurs when* $|\omega \cdot man(z)| \geq 10$ *and* $|\omega \cdot man(z^e)| \geq 10$ *simultaneously hold.*

Proof of Lemma 1. Since z has the last λ decimal digits erroneous, then it can be written as follows:

$$z = z^c + man(z^e) \cdot 10^{\tau-(n-\lambda)} \tag{8}$$

where, as before, z^e is the total amount of finite precision error accumulated in z. Consequently, when z is multiplied by the error free number ω it follows that:

$$\omega \cdot z = \omega \cdot z^c + \omega \cdot man(z^e) \cdot 10^{\tau-(n-\lambda)} \tag{9}$$

□

Now, one may distinguish the following cases:

1. $|\omega \cdot man(z)| \geq 10$ and $|\omega \cdot man(z^e)| < 10$ simultaneously hold.

Then, according to Definitions 1 and 2, quantity $\omega \cdot z$ is computed with an additional correct decimal digit. Indeed, in this case Equation (9) is transformed into:

$$\begin{array}{c} man(\omega \cdot z) \cdot 10^{\tau+1} = man(\omega \cdot z^c) \cdot 10^{\tau+1} + man(\omega \cdot z^e) \cdot 10^{\tau-(n-\lambda)} \Leftrightarrow \\ man(\omega \cdot z) \cdot 10^{\tau+1} = \left(man(\omega \cdot z^c) + man(\omega \cdot z^e) \cdot 10^{-(n-\lambda+1)} \right) \cdot 10^{\tau+1} \end{array} \tag{10}$$

The last equation confirms the fact that quantity $\omega \cdot z$ has been computed with one less erroneous decimal digit in the mantissa.

2. $|\omega \cdot man(z)| < 10$ and $|\omega \cdot man(z^e)| \geq 10$ simultaneously hold. Now, one deduces:

$$\begin{array}{c} man(\omega \cdot z) \cdot 10^{\tau} = man(\omega \cdot z^c) \cdot 10^{\tau} + man(\omega \cdot z^e) \cdot 10^{1} \cdot 10^{\tau-(n-\lambda)} \Leftrightarrow \\ man(\omega \cdot z) \cdot 10^{\tau} = \left(man(\omega \cdot z^c) + man(\omega \cdot z^e) \cdot 10^{-(n-\lambda-1)} \right) \cdot 10^{\tau} \end{array} \tag{11}$$

Then, according to Definitions 1 and 2, quantity $\omega \cdot z$ is computed with an additional erroneous decimal digit in its mantissa, as compared to z.

3. $|\omega \cdot man(z)| < 10$ and $|\omega \cdot man(z^e)| < 10$ simultaneously hold or $|\omega \cdot z| \geq 10$ and $|\omega \cdot man(z^e)| \geq 10$ are verified.

It follows immediately from the analysis performed in 1. and 2. above, together with application of Definitions 1 and 2 that $\omega \cdot z$ is computed with exactly the same number of correct digits in the mantissa, as compared to z.

Next, suppose that we execute Bernoulli's map starting from a non-zero initial value x_0; we shall demonstrate the finite precision error causation and propagation, by means of the entire previous analysis, for a specific, but completely arbitrary and representative example. Indeed, let us assume that all operations are executed in a computing machine with sixteen (16) decimal digits in the mantissa and let:

$$x_0 = 5.055022710000000 \times 10^{-1}$$

Then, x_1 = 1.100454200000001 $\times$ 10^{-2} where the last digit '1', shown in bold, is erroneous. The actual cause for the generation of this erroneous decimal digit, according to the analysis of Section 3, is the following: Multiplication $2x_0$ generates a quantity of order

10^0. When one (9) is subtracted from it, the result is a quantity of order 10^{-2}; in other words, a plunge of order two takes place in the exponent of the obtained result. In fact,

$$x_1 = 2x_0 - 1 = 1.011004542000000 \times 10^0 - 1 \Rightarrow$$
$$x_1 = 1.100454200000001 \times 10^{-2}$$

This plunge generates a deterministic numerical error of two decimal digits; in addition, an erratic error is generated with probabilities of order shown in Table 1. From this table one deduces that, since the exponents $2x_0$ and 1 are equal to zero, it is quite probable that the finite precision error maybe relaxed by one decimal digit. Application of Lemma 1, Case (ii), demonstrates that this is indeed the case.

Then, the next step of the computations offers:

$$x_2 = 2.200908400000001 \times 10^{-2},$$
$$x_3 = 4.401816800000002 \times 10^{-2},$$
$$x_4 = 8.803633600000005 \times 10^{-2},$$

where, once more, the erroneous digits are shown in bold. We observe that, in the computation, there is no need for subtracting 1 from x_2, x_3, x_4, since x_1 and all these three numbers are smaller than $1/2$. On the contrary, each result has been obtained by doubling the previous value of the sequence. Therefore, Lemma 1 can be applied, with $\omega = 2$, ensuring that the number of erroneous decimal digits of quantities x_2, x_3, x_4 remains equal to one ($\lambda = 1$).

During the computation of x_5, the f.p. error has been doubled temporarily, thus generating an additional e. d. d., but only for a while. In fact, execution of the multiplication $2x_4$ has also increased the order of the obtained result by one; when right-shift is performed in order to restore the canonical form, an additional correct digit is produced, in full accordance with the results of Lemma 1. Therefore, x_5 is eventually computed with $2 - 1 = 1$ erroneous decimal digit in the mantissa, as shown below:

$$x_5 = 1.760726720000001 \times 10^{-1}$$

By similar arguments, one may explain the finite precision error generation and accumulation in x_6, x_7, x_8, shown below:

$$x_6 = 3.521453440000002 \times 10^{-1}$$
$$x_7 = 7.042906880000004 \times 10^{-1}$$
$$x_8 = 4.085813760000008 \times 10^{-1}.$$

During the computation of x_9 an additional erroneous decimal digit will appear, since the error is doubled, and its exponent increases by one, while the overall exponent remains the same and no right shift is performed this time; this corresponds to Case (ii) of Lemma 1. In fact,

$$x_9 = 8.171627520000015 \times 10^{-1}$$

With a very similar reasoning, one may predict that the number of incorrect decimal digits in x_{10}, x_{11} will remain the same, namely two; indeed, doubling of the number relaxes the error by one digit, while the subtraction increases the number of erroneous digits by one, according to the analysis results of Sections 2 and 3. Therefore,

$$x_{10} = 6.343255040000031 \times 10^{-1}$$
$$x_{11} = 2.686510080000062 \times 10^{-1}$$

In the next iteration, when x_{11} is doubled to compute x_{12}, we observe that the order of the result does not change, given that the integer part of x_{11} is smaller than five (5), while the exponent of the accumulated numerical error (6.2×10^{-15}) increases by one.

Consequently, Case (ii) of Lemma 1 holds and x_{12} is evaluated with the last three d. d. erroneous. In fact,

$$x_{12} = 5.373020160000\mathbf{124} \times 10^{-1}$$

Next, $x_{13} = 2x_{12} - 1$ and the conditions introduced in Sections 2 and 3 hold. Consequently, one expects that x_{13} will be computed with four (4) e. d. d. and indeed:

$$x_{13} = 7.460403200002\mathbf{475} \times 10^{-2}$$

Now, evaluation of x_{14}, x_{15} and x_{16} follows the results of Lemma 1, Case (iii) while evaluation of x_{17} satisfies the analysis made in Section 3, offering:

$$x_{17} = 3.873290240007\mathbf{918} \times 10^{-1}$$

Doubling x_{17} in order to evaluate x_{18}, gives rise to the "activation" of Case (ii) of Lemma 1 and thus

$$x_{18} = 7.746580480015\mathbf{837} \times 10^{-1}$$

In full accordance with all previous analysis

$$x_{19} = 5.493160960031\mathbf{673} \times 10^{-1}$$

However, $x_{20} = 2x_{19} - 1$ and now the plunge of the exponent of x_{20} is not compensated by the generation of a correct decimal digit and hence,

$$x_{20} = 9.863219200\mathbf{633466} \times 10^{-2}$$

The computation of x_{k+1} continued in a very analogous manner and so in the 53rd iteration the x_{53} element of the Bernoulli map has been computed with all digits of the mantissa incorrect.

The analysis associated with the aforementioned example dictates that the actual reason of generation and accumulation of finite precision error during the execution of the Bernoulli's map in a computing machine is independent of the finite word length that the machine employs. A large number of performed experiments using finite word length from eight (8) decimal digits to forty thousand (40,000) d. d. fully confirmed this statement.

4.3. Obtaining a Bernoulli Map Free of Finite Precision Error

The main observation which may lead to a robust computation of any desired number of x_{k+1}, is the following:

Suppose that the initial value of the Bernoulli's map x_0 is correctly given in canonical form, with n decimal digits in the mantissa and exponent -1. Then, all subsequent terms x_{k+1} of the Bernoulli map, where $(k+1)$ is an arbitrary natural number, will also be computed with precisely n decimal digits in the mantissa. In addition, all intermediate calculations will offer results with at most $(n+1)$ decimal digits in the mantissa, including all the eventual finite precision procedures. The demonstration of this claim follows by an inductive argument:

Suppose that all Bernoulli elements up to x_k are indeed calculated with n decimal digits in their mantissa and that they are of order 10^{-1}. Then, (i) if $x_k < \frac{1}{2}$ holds, doubling x_k will not change the exponent of the result. (ii) On the other hand, if $x_k \geq \frac{1}{2}$, then the exponent of the result $2x_k$ will be zero (0) while, at the same time, temporarily the mantissa of the product $2x_k$ will include $n+1$ decimal digits. At this point, when one is subtracted from $2x_k$ to generate x_{k+1}, then the order of the result becomes smaller than or equal to 10^{-1}. Then, x_{k+1} in canonical form has at most n digits in the mantissa.

Similarly, suppose that x_0 or x_k $(k \in N)$ are of order $10^{-\rho}$, with $\rho > 1$, having n decimal digits in its mantissa. Then, with similar arguments as above, one may deduce that all elements x_{k+1} of the Bernoulli map are computed with at most $(n+\rho-1)$ decimal digits in the mantissa, when all intermediate finite precision procedures are taken into

consideration. At the same time, all intermediate quantities involved in the computation of x_{k+1} are temporarily evaluated with at most $n + \rho$ decimal digits, again when all actions taking place in the finite precision environment are considered.

At this point, we choose to perform all computations with a finite word length, corresponding to $t = n + \rho + 2$ decimal digits in the mantissa. According to the results of the previous sections, if λ is the number of the digits in the mantissa of x_k which are incorrect, then the number of e. d. d. of x_{k+1} is at most $\lambda + 2 + E(x_k) - E(x_{k+1})$, where $E(x_{k+1})$ and $E(x_k)$ are the exponents of x_{k+1} and x_k, both expressed in canonical form. Hence, if $\psi = \lambda + 2 + E(x_k) - E(x_{k+1})$, then all incorrect decimal digits in the mantissa of x_{k+1} are located between the $\psi - th$ decimal place and the last one, namely the $t - th$.

Consequently, if we force the last, after the $\psi - th$ one, decimal digit in the t representation of x_{k+1} to be zero then, the obtained result will always be free of round-off error. We accomplish that, in a rather straightforward manner, by transforming the number into a string and obtaining both the mantissa of the current x_{k+1} as well as its exponent. Subsequently, we zero the last $t - \psi + 1$ entries of the string of the mantissa and we project the number back to floating point arithmetics. In this way, we will obtain totally error free values x_{k+1} of the Bernoulli map, in connection to all decimal digits located between the first and the $\psi - th$ digits' places in the mantissa.

A flowchart of the Bernoulli map version that offers f. p. error—free results is presented in Figures 1 and 2.

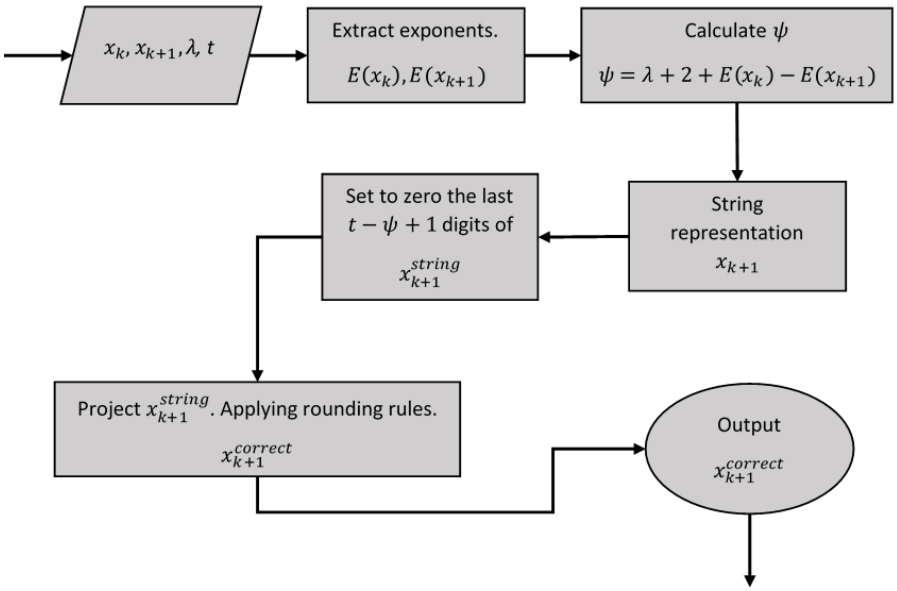

Figure 1. The flowchart of the introduced method for obtaining versions of both Bernoulli and Baker's maps free of finite precision error; we define "correct_x(x_k, x_{k+1}, λ, t)" to be the procedure. In this figure-flowchart: x_k, x_{k+1} are the values offered at the $k - th$ and $(k + 1) - th$ iteration of the classical Bernoulli map, λ is the number of incorrect digits appearing in the mantissa of x_k and t is the necessary extension of the employed finite word length for obtaining f. p. e. free member x_{k+1}. The exact number of t is $n + 2 - E(x_k)$.

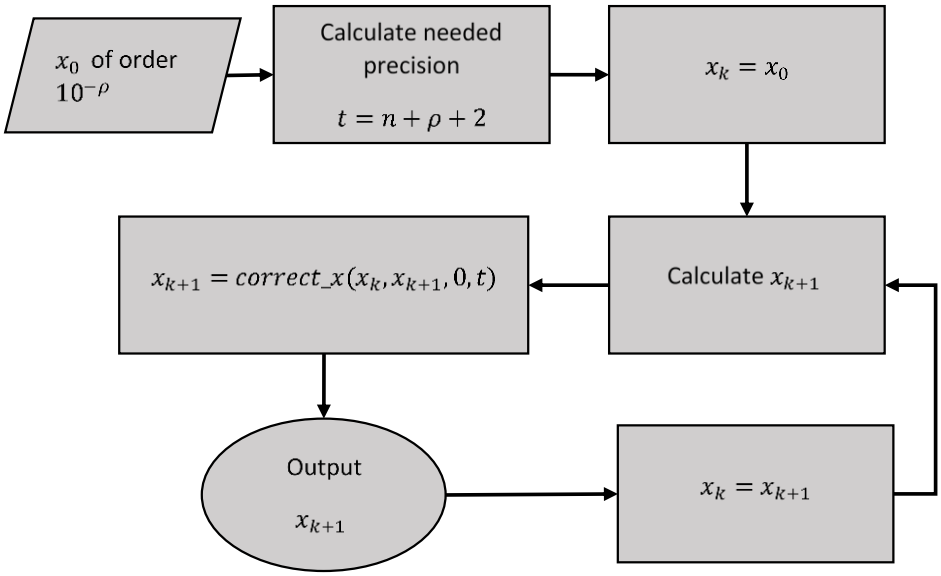

Figure 2. The chart of the entire flow of the Bernoulli map, which generates finite precision error free results, for tenths of millions of iterations.

We note that special care has been taken to circumvent eventual difficulties due to the equivalent representation of a floating-point number with a number of terminal nines. We will manifest the related approach with an example: in a computing machine, which uses a finite word length corresponding to $n = 8$ (eight) decimal digits in the mantissa, numbers $y = 1.2399999$ and $z = 1.24$ must be treated as equivalent. To achieve that, we have rounded the floating-point number from the 4-th to the 8-th decimal digit. Evidently, the position where the rounding process takes place is automatically spotted by applying simple logical rules, string conversions and rounding methods.

We stress that the entire aforementioned method and the corresponding implementation add negligibly to the overall execution time. Clearly, in the case where $t \leq 18 \Leftrightarrow n + \rho \leq 16$, then the aforementioned method for correcting Bernoulli's map results may be implemented using the hardware capabilities of the computing machine. On the contrary, when $t > 18$, one must switch to software implementation of operations with arbitrary precision, like the powerful multiple-precision floating-point computations with correct rounding (MPFR) library.

The related performed experiment fully supports the previous analysis. For example:

$$x_0 \quad = \quad 1.375675568 \times 10^{-1}$$

where $n = 16$ the numbers of digits in the mantissa of x_0, $\rho = -1$ the exponent of x_0. Then, the employed finite word length is the classical double precision offered by the vast majority of contemporary computing machines.

A. The classical method of Bernoulli map computation offers completely unreliable/ erroneous results after fifty-three (53) iterations, when it is implemented by the standard hardware double precision arithmetic (IEEE 754 double-precision floating-point format). Here are the results where the algorithm fails completely for the first time:

$$\begin{array}{rcl} x_{53} & = & 1.250 \times 10^{-1} \\ x_{53}^{correct} & = & 8.62483456 \times 10^{-2} \end{array}$$

The correct values $x_{53}^{correct}$ have been obtained by executing the Bernoulli's algorithm with approximately one thousand (1000) decimal digits in the mantissa in the MPFR

environment. Evidently, the associated results have been projected in a 16 decimal digits word length.

B. When mantissa of twenty thousand (20,000) decimal digits precision is used, the classical method of Bernoulli map gives incorrect results after the 16th d. d. in the 66,387-th recursion. We will manifest this failure of the classical algorithm, by projecting the obtained results in seventeen (17) decimal digits due to obvious space limitations. Then, one obtains.

$$x_{66386} = 7.8099604275200000 \times 10^{-1}$$
$$x_{66387} = 5.6199208550399997 \times 10^{-1}$$

The correct results have been obtained by computing the Bernoulli map with forty thousand (40,000) decimal digits in the mantissa in the MPFR environment and they are shown below:

$$x_{66386} = 7.8099604275200000 \times 10^{-1}$$
$$x_{66387} = 5.6199208550400000 \times 10^{-1}$$

Clearly, again, due to the space limitations, only the first seventeen (17) decimal digits are shown while in x_{66387} all subsequent decimal digits up to twenty thousand (20,000) were totally erroneous. We note that after a very small number of subsequent iterations the Bernoulli's map offer totally unreliable/erroneous results.

C. The method proposed in the present work continues to give totally error free results. Actually, the computation of the classical Bernoulli map performed in forty thousand (40,000) decimal digits failed completely, while the proposed method continued to give absolutely correct results. Evidently, the failure of the classical Bernoulli map and the error free results of the proposed method have been verified by comparing them with the corresponding results generated when eighty thousand (80,000) decimal digits in the mantissa have been use.

At this point, we must emphasize that it is impossible to stabilize any algorithm which generates such a type of f. p. e. whatsoever, when the initial value x_0 is a finite representation of a non-rational number. For example, if $x_0 = \sqrt{2}$ and one keeps n decimal digits for representing it, then a straight-forward analysis as the one given above indicates that the f. p. error will inevitably propagate very fast and that it will eventually destroy the algorithm. Namely, the initial truncation, taking place in $\sqrt{2}-$ representation, will inevitably propagate in any analogous algorithm whatsoever. This means that if one succeeds in obtaining an analytic solution of the problem, then the Bernoulli algorithm or any other similar one, when executed in a computing machine with n d. d. word length, will eventually offer results radically different than the analytic solution. This holds true for any other number, which needs an infinite non-periodic sequence of digits for its representation.

5. Finite Precision Error Analysis and Stabilization of Baker's Map

5.1. A Brief Description of Baker's Map and Its Finite Precision Error Properties

Let x_0, y_0 be floating point quantities such that $0 \leq x_0 < 1$ and $0 \leq y_0 < 1$. Then, the folded Baker's map is a sequence $(x_{k+1},\ y_{k+1}), k \in N_0$ of floating-point numbers, starting from (x_0, y_0) defined via:

$$(x_{k+1},\ y_{k+1}) = \begin{cases} (2x_k, \frac{y_k}{2}) & if \quad 0 \leq x_k < \frac{1}{2} \\ (2(1-x_k),\ 1-\frac{y_k}{2}) & if \quad \frac{1}{2} \leq x_k < 1 \end{cases} \tag{12}$$

Baker's map suffers from very serious finite precision error, quite similar to the one referred in Section 4.1, in connection with Bernoulli's map.

5.2. The Actual Cause of Failure of the Baker's Map Due to Finite Precision

The actual cause of generation and accumulation of f. p. e., as far as x_k is concerned, is practically the same with the previously analytically presented case of the Bernoulli shift computation. In other words, in connection to quantities x_{k+1}, the following sequence of actions in the computing machine generates a continually increasing error due to the employed finite word length:

A. Execution of the subtraction $(1 - x_k)$, whenever necessary, generates a plunge of order d in the exponent of the obtained result; equivalently the closer x_k is to one, the smaller the exponent 10^{-d} of the difference and hence, according to Proposition 1, the greater the number of decimal digits of x_{k+1} that they will be erroneous, as far as the deterministic f. p. error is concerned.

B. On the other hand, multiplication of quantity $(1 - x_k)$ by two may modify the deterministic error, according to the results of Lemma 1. In fact, if conditions of Case (ii) of Lemma 1 occur, then the overall f. p. e. increases by one decimal digit in comparison with the causal finite precision error.

C. For similar reasons, multiplication of x_k by two also may increase the overall number of the already generated and/or accumulated erroneous digits, again when Case (ii) of Lemma 1 holds. Thus, in the end, the accumulation of f. p. e. becomes dominant and the folded Baker's map fails, after an impressively small number of iterations.

On the contrary, y_{k+1} computation does not, in practice, generate f. p. e. for the following reasons:

i. We note that the operation $\left(1 - \frac{y_k}{2}\right)$ may indeed generate f. p. e. However, in contrast to what happens in the x_{k+1} computation, the plunge of the exponent of the obtained result y_{k+1} is one (9) when $y_k > 0$, since inequality $0 \leq \frac{y_k}{2} < \frac{1}{2}$ always holds. Consequently, y_{k+1} may be computed with one erroneous d. d. at most, due to the exponent's plunge of one, according to Proposition 1. We must point out that an additional erratic error is not generated this time, because the term 1 in $\left(1 - \frac{y_k}{2}\right)$ is error free.

ii. At the same time, each division of y_k by two also divides the amount of error with which y_{k+1} is produced. One may safely say that two or three successive divisions by two (2) in practice reduce the number of the accumulated e. d. d. by one decimal digit. Moreover, the essence of the folded Baker's map algorithm itself, forces the operation of division $\frac{y_k}{2}$ to occur after a rather quite limited number of subtractions of type $\left(1 - \frac{y_k}{2}\right)$.

Eventually, one expects that statistically, with very high probability, y_k will be computed with only one or two or non-erroneous decimal digits in the mantissa. Extensive related experiments performed by the authors fully support this claim.

Finally, we would like to, once more, emphasize that the classical Baker's map fails after a relatively particularly small number of recursions even when a large finite word length is employed, e.g., 40,000 decimal digits in the mantissa.

5.3. A Method for Generating the Baker's Map Elements Free of Finite Precision Error

Therefore, in order to stabilize folded Baker's map algorithm, a quite analogous method to the one presented in Section 4.3 has been applied here, too. In fact, one may prove with analogous arguments that

A. When the initial value x_0, is of order $10^{-\rho}$, with a finite number n of decimal digits in the mantissa, then quantity x_{k+1} of folded Baker's map is always correctly computed, should $n + \rho - 1 \leq 2n$ decimal digits in the mantissa were employed. The justification of this statement follows with arguments completely analogous to the ones stated in the Bernoulli's map case.

B. Therefore, one may apply a very similar method to the one presented in Section 4.3, in order to stabilize the component x_{k+1} of folded Baker's map, too. Extended experiments, performed by the authors, fully support this statement.

A flowchart of the Baker's map version that offers f. p. error—free results is given in Figure 3.

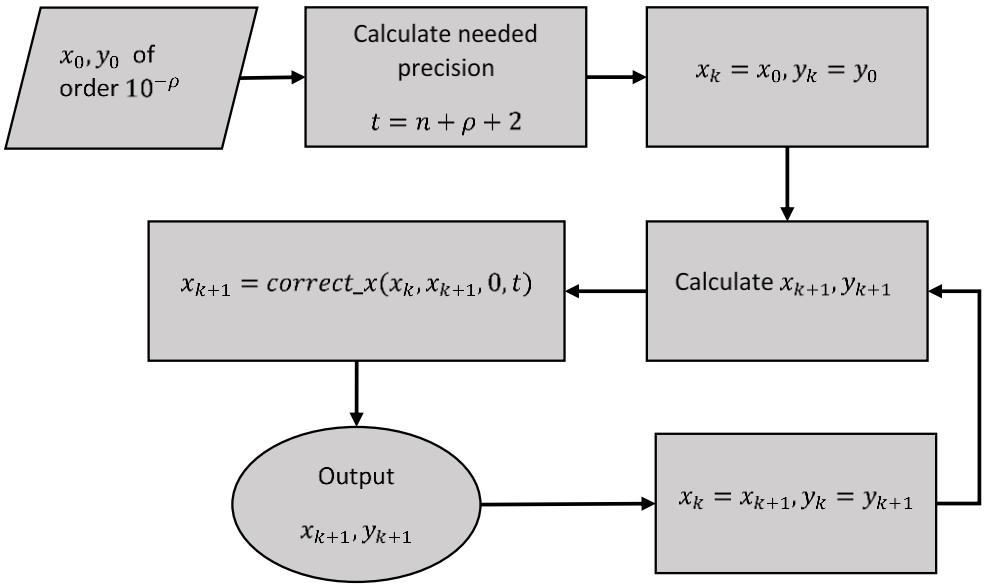

Figure 3. The chart of the entire flow of the Baker's map, which generates finite precision error free results, for tens of millions of iterations. The present flowchart also employs the procedure correct_x introduced in the flowchart of Figure 1.

An indicative example follows.

We have computed the folded Baker's map with the following initial condition for x_0, using sixteen (16) decimal digits in the mantissa and $\rho = -1$:

$$x_0 \quad = \quad 3.333356668880000 \times 10^{-1}$$

The choice of the initial value y_0 may be arbitrary, since a practically negligible f. p. e. is generated in the y_{k+1} computation, as described previously.

We have evaluated the x_{k+1} and y_{k+1} values of this map for a number of iterations, first with the classical manner and next with the previously introduced stabilization method. The related comparative results are shown in Table 2, from where it is evident that, after fifty-four (54) iterations, the classical manner of computation completely fails, i.e., it generates results with all the sixteen d. d. erroneous. On the contrary, the method introduced in this manuscript manifested no erroneous decimal digits at all. We note that in order to obtain these results, the folded Baker's map has been executed in parallel with four thousand (4000) decimal digits precision and by application of Definitions 1 and 2, where the first sixteen digits of the 4000 d. d. representation played the role of the correct quantity. We would like to stress that the evaluation of x_{k+1} by means of the introduced method continues to give correct results for many more thousands of iterations. Moreover, in the end, comparison of these results with those obtained with a mantissa even larger than 4000 d. d., indicates that the x_{k+1} evaluation with 4000 d. d. eventually fails, while the introduced method offered practically completely correct results.

Table 2. Comparative results of the introduced method for the folded Baker's map.

No. of Iteration	Values of x_{k+1} via Classical Method	Values of x_{k+1} via Introduced Method	Correct Values of x_{k+1}
1	$6.666713337760000 \times 10^{-1}$	$6.666713337760000 \times 10^{-1}$	$6.666713337760000 \times 10^{-1}$
2	$6.666573324480000 \times 10^{-1}$	$6.666573324480000 \times 10^{-1}$	$6.666573324480000 \times 10^{-1}$
3	$6.666853351040001 \times 10^{-1}$	$6.666853351040000 \times 10^{-1}$	$6.666853351040000 \times 10^{-1}$
53	$\mathbf{7.500000000000000} \times 10^{-1}$	$8.359693271040000 \times 10^{-1}$	$8.359693271040000 \times 10^{-1}$
54	$\mathbf{5.000000000000000} \times 10^{-1}$	$3.280613457920000 \times 10^{-1}$	$3.280613457920000 \times 10^{-1}$

The results of folded Baker's map obtained via the following three different methods: (a) via the classical manner indicated as "Classical Method" in the Table, (b) via the introduced method, labeled "Introduced Method" and (c) with four thousand digits in the mantissa, where the sixteen of these digits are presented in the Table and they are labeled "Correct Values". We stress that the values of the Classical Method have been obtained by evaluating the iterative Equation (12) in "triple" precision, From the table, it is evident that when using the classical method, x_{k+1} completely fails after 54 iterations; on the contrary during all these iterations, evaluation of x_{k+1} by by means of the introduced method, offered totally correct results.

We would like to emphasize that the folded Baker's map manifested the same behavior, as far as f. p. error is concerned, in connection with many hundreds of tested inputs x_0 having a great variability in the order $10^{-\rho}$, the exact value and the number of decimal digits of x_0.

We would simply like to once more emphasize that if the input values x_0, y_0 in the folded Baker's map are irrational numbers, then stabilization of the algorithm is impossible due to the nature of the computing machines that use a finite word length for all computations.

6. Conclusions

In the present manuscript, it has been established that the Bernoulli's and Baker's maps indeed offer totally erroneous results, after an impressively small number of recursions; for example, when IEEE standard double precision is employed, both chaotic maps fail after few tens of iterations, frequently less than sixty. The actual reason for the causation and accretion of this, exponential growing, finite precision error, follows immediately from the analysis introduced in Section 3 and Lemma 1; the corresponding approach is novel.

Using the aforementioned results, methods for properly executing the Bernoulli's and Folded Baker's maps are introduced in Sections 4 and 5. These methods, for the first time, allow the two chaotic maps to run for many hundreds of thousands of iterations, offering correct results practically free of finite precision error. A considerable number of associated experiments fully confirm this claim, as it is discussed in Sections 4 and 5 of the present manuscript.

Author Contributions: Conceptualization, C.P., D.A., C.C. and A.R.M.; funding acquisition; investigation, A.R.M. and C.C.; project administration, C.P.; resources, N.V.K. and C.C.; software, C.P., C.C., A.R.M., N.V.K. and D.A.; supervision, C.P. and D.A.; validation, C.P., D.A., N.V.K., A.R.M. and C.C.; writing—original draft, C.P. and C.C.; writing—review and editing, A.R.M., C.P., N.V.K. and C.C. 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: The study did not employ any data sets.

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

References

1. Selvam, A.M. *Chaotic Climate Dynamics*; Luniver Press: Bristol, UK, 2007; pp. 67–68.
2. Tsekeridou, S.; Solachidis, V.; Nikolaidis, N.; Nikolaidis, A.; Tefas, A.; Pitas, I. Statistical Analysis of a Watermarking System. *Signal Process.* **2001**, *81*, 1273–1293. [CrossRef]
3. Nikolaidis, A.; Pitas, I. Comparison of different chaotic maps with application to image watermarking. *Proc. IEEE Int. Symp. Circuits Syst.* **2000**, *5*, 509–512.
4. Saito, A.; Yamaguchi, A. Pseudorandom number generation using chaotic true orbits of the Bernoulli map. *Chaos Interdiscip. J. Nonlinear Sci.* **2016**, *26*, 063122. [CrossRef] [PubMed]
5. de la Fraga, L.G.; Torres-Pérez, E.; Tlelo-Cuautle, E.; Mancillas-López, C. Hardware implementation of pseudo-random number generators based on chaotic maps. *Nonlinear Dyn.* **2017**, *90*, 1661–1670. [CrossRef]
6. Martínez-González, R.F.; Díaz-Méndez, J.A.; Palacios-Luengas, L.; López-Hernández, J.; Vázquez-Medina, R. A steganographic method using Bernoulli's chaotic maps. *Comput. Electr. Eng.* **2016**, *54*, 435–449. [CrossRef]
7. Salleh, M.; Ibrahim, S.; Isnin, I.F. Enhanced chaotic image encryption algorithm based on Baker's map. *Proc. IEEE Int. Symp. Circuits Syst.* **2003**, *2*, 11.
8. Gondal, M.A.; Abdul, R.; Hussain, I. A Scheme for Obtaining Secure S-Boxes Based on Chaotic Baker's Map. *3D Res.* **2014**, *5*, 17. [CrossRef]
9. Liu, Y.; Li, J.; Lu, X.; Yuen, C.; Wu, J. A family of chaotic pure analog coding schemes based on baker's map function. *EURASIP J. Adv. Signal Process.* **2015**, *2015*, 58. [CrossRef]
10. Rogers, A.; Keating, J.; Shorten, R. A novel pattern classification scheme using the Baker's map. *Neurocomputing* **2003**, *55*, 779–786. [CrossRef]
11. Prasadh, S.K.; Chakole, S.; Mohammed, V.N. Performance Analysis of Chaotic Map Algorithms Comparison of efficiency of Line Map, Bakers and Cat Algorithmfor image cryptography. In Proceedings of the 2018 Fourteenth International Conference on Information Processing (ICINPRO), Bangalore, India, 21–23 December 2018.
12. Schmitz, R. Use of chaotic dynamical systems in cryptography. *J. Frankl. Inst.* **2001**, *338*, 429–441. [CrossRef]