# Estimation of Input Variable as Initial Condition of a Chaos Based Analogue to Digital Converter

**Rajlaxmi Basu**

**A thesis submitted to the University of Huddersfield in partial fulfilment of the requirements for the degree of Doctor of Philosophy**

**February 2018**

# ABSTRACT

A realization of an analogue-to-digital converter (ADC) with improved conversion accuracy, using the chaotic behaviour of the tent map, is presented. In this approach, the analogue input signal to be measured, termed as the initial condition is applied to a chaotic map, and the symbolic dynamics resulting from the map evolution, is used to determine the initial condition in digital form. The unimodal piecewise linear tent map (TM) has been used for this purpose, because of its property of generating uniform distribution of points and robust chaos.

Through electronic implementation of the TM it is practically impossible to produce an 'ideal' TM behaviour with parameter values in the full range [0,1]. Due to component imprecision and various other factors, a non-ideal map with reduced height is observed. For such a map, converting the equivalent symbolic trajectory generated by TM iterations return erroneous results as the partitioning of the phase space embodied in the finite symbolic dynamics no longer has unique correspondence with the initial condition.

Two algorithmic solutions have been proposed to minimise the errors associated with a practical system. For one, it has been established that for a reduced-height map the partitioning will not remain of equal size. Considering that the height of the tent map used for this purpose is known from an independent but related research, a technique of partitioning the state space unevenly, depending on the map height has been proposed and has been shown that if the correct partitioning is used, the resulting symbolic dynamics again map uniquely to the initial condition.

Alternatively, it has been shown that the degree of deviation of the iterate values can be determined based on the parameter value, which in turn can be adjusted for depending on the symbolic sequence generated by the initial condition to determine the correct decimal equivalent values.

The both the approaches proved to be highly effective in obtaining a digital outcome corresponding to the initial condition using 8 symbolic iterations of the map in hardware domain, with the second approach outperforming the first in terms of accuracy, while the first method can easily be pipelined alongside generating the iterates and thus improve the speed. This development is promising because, in contrast to the commercially available ADCs, it places lower demand on the hardware resource and can be effectively implemented to give a real-time operation.

# ACKNOWLEDGEMENT

I would like to take this opportunity to acknowledge the individuals who have been integral parts of my journey with their patience, advice and encouragement on both personal and professional levels.

First of all, I would like to express my sincere gratitude to my academic supervisor Dr Peter Mather for his relentless support that was not limited to my research and thesis alone. His patience, motivation and advice through some of the most difficult times have been invaluable, and I could not have asked for a better supervisor. I would also like to thank Dr Violeta Holmes for her support and advice every time I turned to her for help. Special thanks to Dr Soumitro Banerjee, without whose guidance, I couldn’t have come this far.

Apart from my supervisors, I had the chance to work with some of the greatest people I have ever met. In particular, I would like to thank Dhrubajyoti Dutta, who has often gone out of his way to help and support whenever I needed. I would also like to thank Dennis Town for being who he is, because nothing I say can be enough; Jan Berkenbusch for being one of the most endearing person I have ever known, and Michael Agolom and David Upton for being amazing friends and colleagues.

I would like to thank my parents Bani and Swapan, and my brother Debkumar, for keeping up with my whims and still managing to motivate and support me, albeit 5000 miles away and despite a tedious time difference.

Finally, my sincere thanks to Chris Sentance and Cathie Raw for supporting me whenever I reached out to them, which was often.

# CONTENTS

Abstract ........................................................................................ 1

Acknowledgement ................................................................................ 3

List of figures ..................................................................................... 7

List of tables ..................................................................................... 10

List of Publications ............................................................................. 11

List of abbreviations and acronyms .................................................... 12

1 Introduction ..................................................................................... 14

1.1 Analogue to Digital Converters ........................................................ 14

1.2 Nonlinear Chaos .............................................................................. 15

1.2.1 Chaos based ADC ......................................................................... 18

1.3 Aims and objectives ......................................................................... 27

1.3.1 Initial conditions as intervals ......................................................... 27

1.3.2 Analysis of the intervals generated ................................................ 28

1.3.3 Recovering initial conditions ......................................................... 29

1.4 Original contribution ....................................................................... 30

1.5 Document overview .......................................................................... 31

2 Background review ............................................................................ 33

2.1 A brief account of ADC .................................................................... 33

2.1.1 Flash type ADC ............................................................................ 37

2.1.2 ΔΣ type ADC ................................................................................ 38

2.1.3 SAR type ADC .............................................................................. 39

2.1.4 Pipeline ADC ................................................................................. 40

2.1.5 Hybrid flash ADCs ........................................................................ 41

2.2 Chaotic maps: formal definition and properties ..................... 43

2.2.1 Bitshift Map ..................................................... 44
2.2.2 Logistic Map ..................................................... 46
2.2.3 Tent Map ......................................................... 48

2.3 Symbolic dynamics ..................................................... 52

2.4 Previous chaos-based ADC attempts .................................... 55

2.4.1 ADC based on BM ................................................. 55
2.4.2 ADC based on TM ................................................. 56

3 Analysis of non-ideal behaviour .......................................... 59

3.1 Parametric deviation .................................................. 60

3.2 Narrowed dynamical attractor .......................................... 62

3.3 Skewness of the intervals ............................................. 65

4 Initial condition estimation .............................................. 69

4.1 Self-Similarity ....................................................... 71

4.2 Interval arithmetic ................................................... 74

4.3 Addressing non-ideal patterns ......................................... 75

4.4 Orientation of the interval arithmetic ................................ 78

4.4.1 The interval arithmetic method ................................... 80

4.5 Deviation adjustment for the GON ...................................... 84

4.5.1 The deviation adjustment method .................................. 85
4.5.2 Details of the calculation ....................................... 87

5 Results and Discussion ................................................... 95

5.1 Simulation results .................................................... 95

5.1.1 Results for interval arithmetic algorithm ........................ 96

5.1.2 Results for deviation adjustment algorithm .................... 107
5.2 Implementation .................................................................. 116
6 Conclusion and further work ..................................................... 123
6.1 Future directions ............................................................... 126
6.1.1 ADC implementation ........................................................ 126
6.1.2 Application in Cryptography ......................................... 127
6.1.3 Application in communication ....................................... 128
References ........................................................................... 129
Appendix 1 ........................................................................... 136
Appendix 1.1 ........................................................................ 137
Appendix 1.2 ........................................................................ 138
Appendix 2 ........................................................................... 139
Appendix 2.1 ........................................................................ 140
Appendix 2.2 ........................................................................ 143
Appendix 2.2.1 ..................................................................... 143
Appendix 2.2.2 ..................................................................... 145
Appendix 2.2.3 ..................................................................... 146
Appendix 2.3 ........................................................................ 148
Appendix 2.2.1 ..................................................................... 148
Appendix 2.2.2 ..................................................................... 152
Appendix 3 ........................................................................... 161
Appendix 4 ........................................................................... 172
Appendix 5 ........................................................................... 174

# LIST OF FIGURES

Fig. 1.1 Behaviour of LM across a range of parameter ....................... 17

Fig. 1.2 Using a chaotic map as a quantisation block.......................... 19

Fig. 1.3 Using a chaotic map as a quantisation block.......................... 21

Fig. 1.4 Uniform intervals in the state space....................................... 28

Fig. 1.5 Skewed intervals remaps the same input................................ 29

Fig. 2.1 ADC step - size and desired level of detection....................... 34

Fig. 2.2 Block diagram of a 3-bit flash ADC ...................................... 37

Fig. 2.3 Simplified block diagram of a Delta-Sigma ADC.................. 38

Fig. 2.4 Simplified block diagram of a SAR type ADC ...................... 39

Fig. 2.5 Block diagram of a pipelined ADC........................................ 40

Fig. 2.6 Block diagram of a folding-interpolating ADC...................... 41

Fig. 2.7 The Bitshift (Bernoulli) Map (BM) behaviour ....................... 45

Fig. 2.8 Bifurcation diagram of BM: points escape to infinity ............ 45

Fig. 2.9 The Logistic Map (LM) behaviour ........................................ 47

Fig. 2.10 BD of LM: periodic window over the entire range............... 47

Fig. 2.11 The Tent Map (TM) behaviour ............................................ 48

Fig. 2.12 BD of TM: points above ~0.7 exhibit robust chaos [17] ...... 49

Fig. 2.13 A 3D view of the iterates over the entire state space............ 51

Fig. 2.14 Fractal dynamics of the TM over the state space ................. 52

Fig. 2.15 Intervals with binary signature generated by the BM ........... 56

Fig. 2.16 Intervals with Gray signature generated by the TM ............. 57

Fig. 3.1 TM with reduced heights due to various parametric values.... 61

Fig. 3.2 Cobweb diagram for points originating before $x_c$................... 63

Fig. 3.3 Cobweb diagram for points originating before $x_c$ ................... 64

Fig. 3.4 Shift in interval partition due to reduced height ..................... 66

Fig. 4.1 Fractal behaviour of ideal Gray codes for $\mu = 1$ ..................... 69

Fig. 4.2 Distortion of the fractal code for $\mu = 0.75$ ............................. 70

Fig. 4.3 GON values calculated and compared with the ideal case ...... 71

Fig. 4.4 A closer view of the GON values ......................................... 72

Fig. 4.5 GON calculated with base of $2\mu$ and rescaled by $1/2\mu$ .......... 74

Fig. 4.6 Fractal growth of sub-intervals over iterations ....................... 75

Fig. 4.7 Shift of the partitions towards $x_c$ .......................................... 77

Fig. 4.8 Generating Gray code over iterations .................................... 79

Fig. 4.9 Reconstructing sub-intervals ................................................ 81

Fig. 4.10 Flow chart for the interval arithmetic algorithm [28] ........... 83

Fig. 4.11 Tabular representation of deviation adjustment algorithm ... 87

Fig. 5.1 Percent error using 8 symbols .............................................. 96

Fig. 5.2 Percent error using 12 symbols ............................................ 97

Fig. 5.3 Percent error using 16 symbols ............................................ 97

Fig. 5.4 Exponential reduction with increasing symbols ..................... 98

Fig. 5.5 Logarithmic view of the exponential reduction ...................... 99

Fig. 5.6 The maximum log-of-error for multiple parameters ............. 100

Fig. 5.7 The maximum log-of-error using 8, 16 and 32 symbols ....... 101

Fig. 5.8 Histogram of the data set and the calculated GON ............... 102

Fig. 5.9 Histograms for interval arithmetic method .......................... 104

Fig. 5.10 Histogram for dataset shifted by half-step ......................... 106

Fig. 5.11 Percent error using 8 symbols ........................................... 107

Fig. 5.12 Percent error using 12 symbols ......................................... 108

Fig. 5.13 Percent error using 16 symbols .......................................... 108

Fig. 5.14 Histograms for deviation adjustment method ..................... 111

Fig. 5.15 Histograms for dataset shifted by half-step ........................ 113

Fig. 5.16 Comparison of various estimation techniques .................... 114

Fig. 5.17 Effect of noise on symbolic outcomes ............................... 116

Fig. 5.18 Schematic diagram adapted from the TM circuit [42]. ....... 117

Fig. 5.19 Physical hardware of the TM function [28]. ...................... 117

Fig. 5.20 Image capture of the map on oscilloscope ......................... 118

Fig. 5.21 Time series for 200mV (left), and 518mV (right) .............. 119

Fig. 5.22 GON of a 3-cycle ramp .................................................... 119

Fig. 5.23 The 8-bit and 16-bit % error using both the algorithms ...... 120

Fig. 5.24 The 8-bit and 16-bit histograms using both the algorithms 121

Fig. A. 1 3D view of the real iterates across the state space ........................ 172

Fig. A. 2 Unequal interval partitioning of the real state space ..................... 173

Fig. A. 3 A 3D view combining Fig. 6.6 and 6.7 ........................................ 174

# LIST OF TABLES

Table 2.1 Comparison chart of various types of ADCs ........................ 42

Table 2.1 Correspondence between Sequences and Input Intervals ..... 55

Table 3.1 Change in trajectory with change in parameter ................... 61

Table 3.2 Imperfect mapping of symbolic trajectory .......................... 62

Table 4.1 Calculation of weightings ................................................... 88

Table A. 1 Input signal with the symbolic sequences........................ 161

# LIST OF PUBLICATIONS

1. **An Algorithmic Approach for Signal Measurement Using Symbolic Dynamics of Tent Map**

   **Reference**
   Basu, R., Dutta, D., Banerjee, S., Holmes, V., & Mather, P. (2018). An Algorithmic Approach for Signal Measurement Using Symbolic Dynamics of Tent Map. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *65*(7), 2221-2231.

   **Individual contribution**
   - Effect on the symbolic dynamics of the tent map due to non-ideal parameter
   - Shift of the tent map partitions under non-ideal parametric conditions
   - Association of the non-ideal symbolic sequences with the correct interval through repositioning of the partitions
   - Determination of the initial condition from the symbolic sequence through repeated partitions proportional to the parameter value

2. **Parameter estimation for 1D PWL chaotic maps using noisy dynamics**

   **Reference**
   Dutta, D., Basu, R., Banerjee, S., Holmes, V., & Mather, P. (2018). Parameter estimation for 1D PWL chaotic maps using noisy dynamics. *Nonlinear Dynamics*, 1-15.

   **Individual contribution**
   - Contribution to validation of the proposed estimation method to other 1D PWL maps apart from tent map

# LIST OF ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| µ | Parameter |
| 1D | One dimensional |
| ADC | Analogue to digital converter |
| BD | Bifurcation diagram |
| BM | Bitshift or Bernoulli Map |
| DAC | Digital to analogue converter |
| div | division |
| FPGA | Field programmable gate array |
| GON | Gray Ordering Number |
| LM | Logistic Map |
| LSB | least significant bit |
| LUT | Look up table |
| MSB | mot significant bit |
| mV | milivolt |
| mV/div | milivolt per division |
| S/H (alt. S&H) | Sample and hold |

| | |
|---|---|
| SAR | Successive approximation register |
| SNR | Signal to noise ratio |
| TM | Tent Map |
| V | Volt |
| Vdac | DAC voltage |
| Vin | input voltage |
| Vref | reference voltage |
| ΔΣ | Delta Sigma |

# 1 INTRODUCTION

One of the most important requirements in engineering systems is the accuracy of the data collected from the environment, leading to the necessity for precision measurement. This is particularly true for sensory devices which record changes in various physical quantities from the environment and generate signals proportional to those changes. With sensors typically being analogue instruments, parameter signals with theoretically infinite resolution are produced, which thus can potentially detect the finest possible change and generate signals accordingly. However, with the signals being analogous in nature, complicated computations and manipulations of the collected signals become complex without digitisation. This requires the analogue signals from the sensors to be converted into the corresponding digital equivalents performed by an Analogue to Digital Converter (ADC). The resolution of the converted signal is ascertained by the number of digital bits representing each of the voltage or current levels. Increased number of bits improves the resolution of the measurement, leading to improved accuracy in the resultant computations.

## 1.1 Analogue to Digital Converters

Converting analogue signals to digital numbers with an increased number of bits comes at the expense of increased complexity leading to increased chip surface area, power consumption and cost, typically along with exponentially reduced speed. For example, the fastest ADC architecture – the flash type – doubles in complexity for a single bit resolution increase. Also, in terms of improved tolerance and precision, in order to retain the accuracy at an optimum level, the quality of the components adds to the cost. Evidently, the higher resolution and precision, as well as accuracy, necessary for increased digitisation of the data lead to major trade-offs. In an effort to minimise the trade-off costs for improving resolution, previous attempts had been made to consider *chaotic maps* as an alternative quantisation block in an ADC. In the following section, the principles of chaotic dynamics are briefly introduced, prior to discussing the previous techniques for chaotic map based ADCs.

## 1.2 Nonlinear Chaos

Chaotic maps are nonlinear functions that exhibit complex evolution over time and the development of the state space is dependent on the initial input values [1]. Such functions are broadly studied as nonlinear dynamical systems, a branch of science that had been developed from the three-body-problem famously studied by Henri Poincaré [2] in the late 19th century, where he observed deterministic systems demonstrating aperiodic behaviour. This led to a geometric approach of treating nonlinear problems where notably Aleksandr Lyapunov [3] addressed the nonlinearity through approximations. However, the studies of chaotic behaviour of the nonlinear systems were limited until the later half of the 20th century, i.e. until the introduction of high-speed computers in the 1950's, which allowed experimentations with long term dynamics and observations of chaotic trajectories.

In 1961 meteorologist Edward Lorenz accidentally discovered the sensitivity of the dynamics to initial conditions while working on weather prediction models, when he attempted to re-examine some of his previous results. He had used data accurate up to three decimal places while the computer could produce results accurate up to six decimal places [4]. This led to inaccuracies by up to 0.1% leading to completely divergent solutions. By 1963 Lorenz discovered the chaotic motion on strange attractors involving three variables, which led him to conclude that long term weather prediction was impossible, however, there is a structure in the apparent chaos and these observations eventually led to what is today known as *Chaos Theory* [5]. Chaotic phenomena have since been observed and applied in many other fields, e.g., economics, biology, cosmology etc. The evolutionary dynamics of several systems in these fields might appear to be random, however, after a certain period, such systems result in behaviour that may appear to be deterministic in nature, hence are defined as chaotic systems [3]. Despite the apparent randomness in the chaotic evolution the underlying deterministic information can be defined i.e. with infinite precision, the exact behaviour can be repeated, and also tracked in reverse. The dynamical behaviour of a system involves time domain evolution of the system states. The continuously changing states of the system, in most cases, is governed by more than one variable whose relationship with the evolutionary process can be defined mathematically as a function. One such variable is the initial condition, which is the origin point or the initial state of the system from which the future evolution is observed.

Additional influential factors affecting the behaviour are control parameters that can be perceived as scaling factors governing the amount of change introduced in the system states causing the system to visit several other states resulting into the dynamic behaviour. Since the dynamical evolution is mainly iterative in nature, as the present state of the dynamics is responsible for the future states, the mathematical function that defines the dynamics can also be referred to as maps.

A state of a system can be considered as the initial condition or system input for further dynamic process where the resultant dynamics can be analysed as the trajectory of the initial condition. Chaotic maps, which can be either discrete where the trajectory is not differentiable with respect to the chaotic function can be continuous time domain as it can be differentiable with respect to the operating function i.e. the intermediate states between the two iterates can also be determined numerically. Chaotic functions with one or more spacial dimensions exhibit diverging points over changing parameters (Fig. 1.1). This phenomenon is defined as *bifurcation* in behaviour over a range of parameters and specific conditional ranges leading to aperiodic characteristics over time, generating chaotic behaviour [6,7].

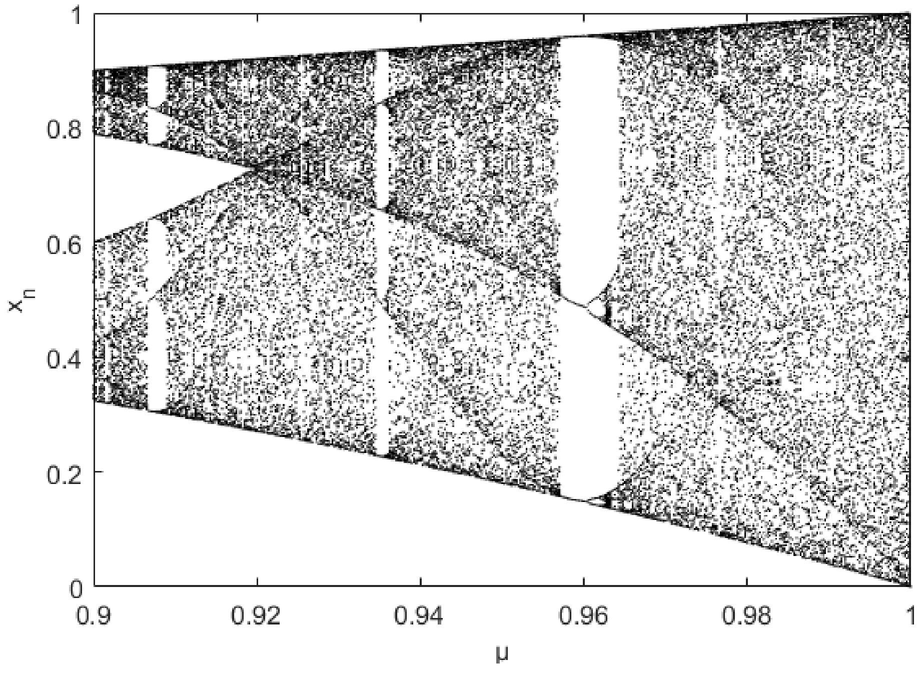


Fig. 1.1 Behaviour of LM across a range of parameter

The deterministic property of chaotic maps has found use in a myriad of applications, from cryptanalysis [8] to secure digital communication [9,10] to pattern recognition [11] to name a few. In this work, in order to determine the absolute values of signals, the unimodal unidimensional chaotic maps are used for generating unique trajectories, effectively resulting in an ADC where the maps have been envisioned as alternative to the existing quantisation blocks.

### 1.2.1 Chaos based ADC

The fact that the dynamics produced by the chaotic maps are sensitive to the initial condition, the application of chaotic maps, as a measurement system, can be proved to be feasible [12,13], chaotic maps can thus be chosen as the quantisation block for an ADC (Fig. 1.2) where the terms *initial condition* is used interchangeably to imply the *input signal* to the ADC, wherever relevant. As a quantisation block, the chaotic map can be utilised to generate a 0 when the input voltage is less than 0.5V and a 1 otherwise. However, using chaotic maps iteratively practically eliminates the increase in cost for increasing bit-resolution because the same block can be reused when incorporating feedback, virtually infinitely in order to generate further iterations. The fundamental idea relies on the fact that the sensitivity of the maps on the initial condition can be utilised to identify the input conditions to the map from the uniqueness of the iterated trajectories generated by them. Thus, if a chaotic map is implemented electronically through an analogue circuit, the magnitude of the input signal entering the circuit could effectively be recovered by reverse calculating from the specific resultant trajectory, given the knowledge of the iterative behaviour of the map.

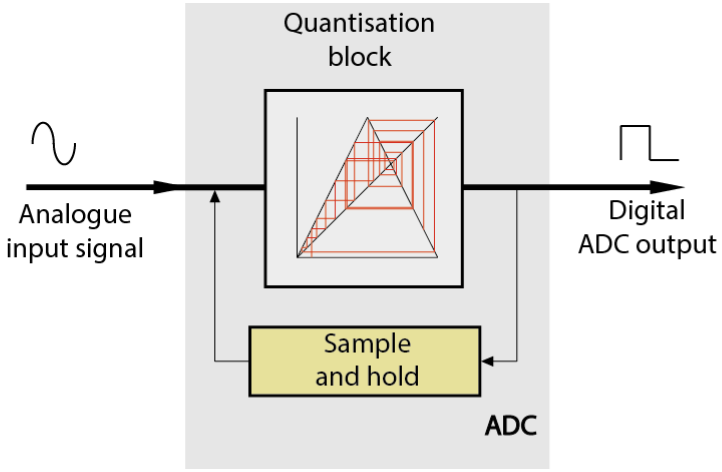


Fig. 1.2 Using a chaotic map as a quantisation block

The outcome of the resulting trajectories in the implemented circuit is greatly affected by the inevitable presence of noise and the accuracy of the analogue implementation, due to component tolerances. Hence, intensive computations are involved in the reverse-calculation procedure and must be performed in the digital domain. However, the sensitivity of the dependence of the map dynamics on the iterate values is crucial. A slight deviation from the actual iterate value will lead to a completely different initial condition over sufficient number of iterates. As a result, in order to recover initial conditions from the respective trajectories, accurate measurement of the iterate values is critical. However, as the map is implemented through a physical circuit, a high-resolution ADC must be associated with every iteration stage of the implemented map. This, however, is dictated by the degree of accuracy and resolution required for the initial condition. Also, the resolution of the ADCs required may exceed the resolution of the ADC thus generated. As a result, the target device or the end-product becomes a requirement in the hardware setup and the cost is escalated exponentially.

A solution to avoid high resolution measurement of each iterate is to employ another useful tool called symbolic dynamics, which is another way of treating and analysing the map dynamics without resorting to actual values of iterates of the map. Observing and analysing chaotic trajectories through symbolic representation was first introduced by Metropolis et al [14] where the iterates were categorised through symbols depending on the position of the iterate values around the Markovian partition [6,15]. Therefore, if each iterate were assigned a letter or a symbol and the pattern generated by the trajectories are studied, any iterate "will then be said to be of "type L" or of "type R"" [13] depending on which side (left or right) of the Markovian partition the iterate lies. As a result, the trajectories produced a sequence consisting of The 'patterns' thus exhibited by the symbolic treatment were eventually converted to binary digits and were soon utilised efficiently in digital analysis of the map. In 1D chaotic maps, the symbolic dynamics are generated by assigning '1' to iterate values that exceed the midpoint of the possible range of inputs while '0' to the rest of them. Under the ideal parametric conditions, the symbolic trajectory correspond to the initial condition through binary (generated by BM) or Gray code (generated by LM or TM) [6,12,16]. Fig. 1.3 shows the Gray code generation over the iterations. Since the state space is partitioned by each operation of the chaotic map there are many intervals generated due to the partitioning as the dynamics continues. In ideal conditions, the intervals are divided into two equal halves and a symbol or a series of symbols represent any such interval. Hence, if such sequences are either binary or Gray codes then conventionally converting them to equivalent real numbers that involves division by two in each of the subsequent stages resulting into determining the initial condition or an iterate in the trajectory, therefore, making such maps and its symbolic dynamics the ideal choice for measurement applications.

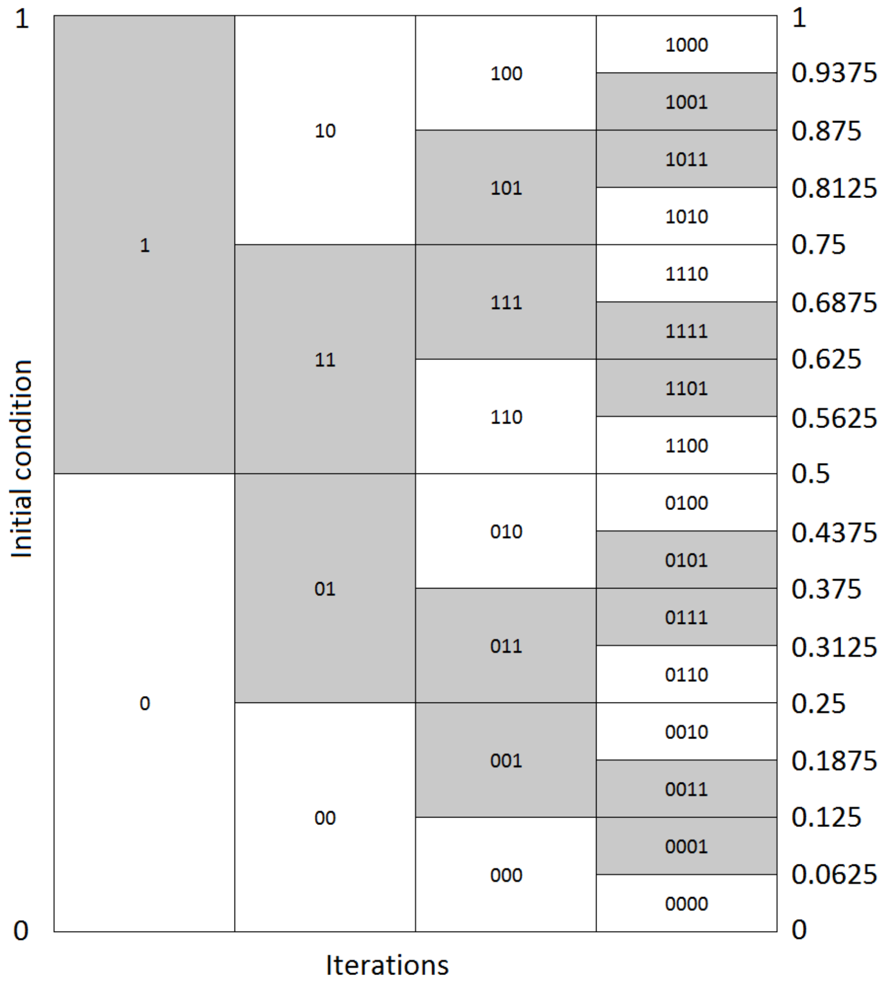


Fig. 1.3 Using a chaotic map as a quantisation block

The most recent work for the chaos based ADC development was also attempted by Berberkic et al. [17] aimed at determining relative changes in incoming signals. Various 1D maps were investigated for performance, where the LM and the TM proved to be feasible options. The performances were weighed by comparing the shift in the long term trajectories of the signals through the maps. Owing to the sensitive dependence of the initial conditions, trajectories gradually diverged for the slightest changes and were noticeable over sufficient time steps of the map operation. It has been shown that the resulting trajectory of the difference between individual signals showed unique characteristics and as a result could be utilised to identify the relative change in the signals. The TM emerged to be the optimum option in terms of implementation and performance [18] and could successfully measure signal differences (as opposed to absolute signal values) up to 20μV. However, the lack of determination of absolute values of the signals and the shift in parametric values in the implemented circuits were unaddressed. This required further investigations, which addressed both issues. Also, the work in [18] approached the problem using real valued trajectories of the map, which entailed measuring the individual iterates with high precision. This involved sufficiently developed measuring techniques and converting the readings into digital values for computation. This proved to be challenging for long term aims of implementing an ADC. However, the study remains important for the assurance of uniqueness of trajectories for physically implemented versions of the maps as well as the choice of map.

The next stage of the research involved correctly identifying the absolute value of the signal value from the symbolic dynamics of the map function. However, the implemented map would lead to reduced height which needs to be identified in order to perform necessary modifications in determining the shift in the symbolic behaviour and thus accommodating the shifted dynamics within the state space of the map maintaining the correct correspondence with the initial condition. The direction of investigation, therefore, became bifurcated, one being identification of the implemented map through its parameter, which has been led by Dhrubajyoti Dutta as an independent yet related research. The other direction, as described in this work, involves remapping the affected symbolic sequence generated by the map to its initial condition, keeping in mind the non-ideal behaviour produced by the reduced parameter TM. The breakdown of the research areas has been shown in Fig 1.7.

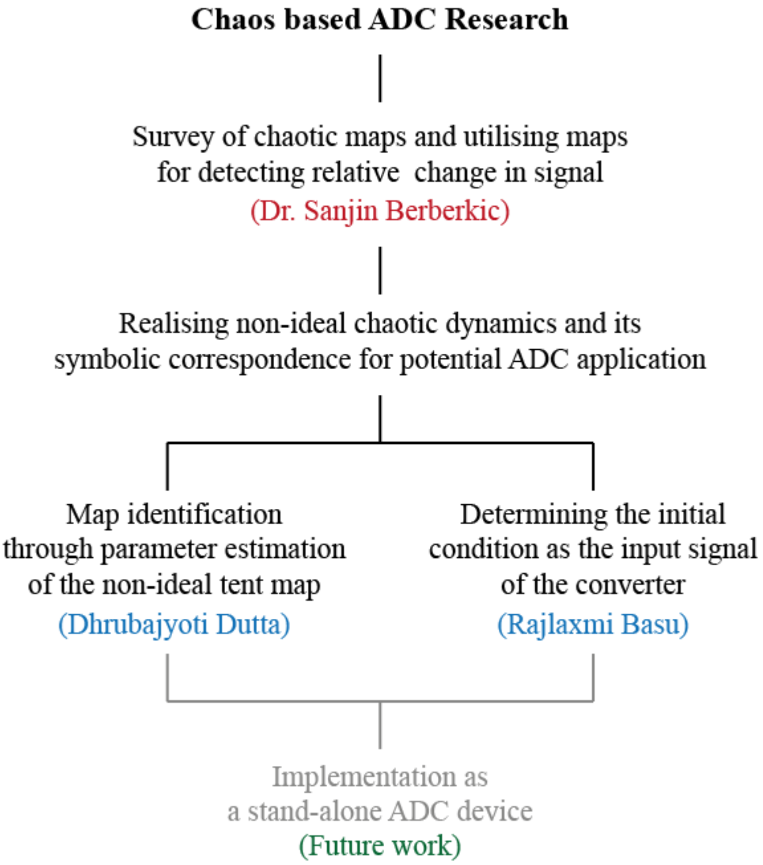

Chaos based ADC Research
Survey of chaotic maps and utilising maps for detecting relative change in signal
(Dr. Sanjin Berberkic)
Realising non-ideal chaotic dynamics and its symbolic correspondence for potential ADC application
Map identification through parameter estimation of the non-ideal tent map
(Dhrubajyoti Dutta)
Determining the initial condition as the input signal of the converter
(Rajlaxmi Basu)
Implementation as a stand-alone ADC device
(Future work)

Fig. 1.7 Breakdown of the areas of the research

Discrete 1D maps are unidimensional systems whose dynamics are expressed by a single state variable and is non-differentiable in nature that makes such systems discrete, e.g. Logistic Map (LM), Tent Map (TM) and Bernoulli Map (BM) of which LM and TM are the two most fundamental forms of 1D discrete chaotic systems that are also unimodal in nature. The LM consists several regions within certain parameter ranges where the map dynamics become periodic, i.e., iterative points visit the same set of points in a cyclic pattern. Thus, the distribution of the dynamics contains windows of periodicity which is discussed further in section 2.2.2. In case of the TM, the windows of periodicity cease to exist beyond a certain low parameter value (about ~0.7), and therefore the distribution is much less periodic (discussed in section 2.2.3). Thus, the TM showed better linearity and sensitivity and is known to produce *robust chaos* [19] for a wide range of parametric value that controls the behaviour of the map, and can therefore accommodate the dynamics of the reduced height map caused by the electronic implementation.

The TM produces unique, mutually exclusive *Gray code sequences* as output under ideal parametric values, which can be directly utilised to generate symbolic signatures for the initial conditions. However, ideal implementation of the map as a physical circuit is impossible, owing to the imprecision in the components aside from the noise introduced by the hardware [13]. This results in a reduction in the parametric value and therefore the height of the map function is reduced, leading to substantial deviation in the map trajectories for the same initial conditions. This is referred to as the non-ideal condition of the map, henceforth being addressed as the non-ideal TM, and the symbolic sequence thus generated as non-ideal sequence. As a result, if this behaviour, the dynamics continue infinitely, i.e. does not converge to zero, and must be truncated after a finite number of iterations as it is not feasible to continue the physical process indefinitely, as well as in order to avoid complete corruption of the trajectories owing to system noise [20,21]. However, regardless of the variations in the sources of the non-idealities – offset, linearity issues, along with noise – observed in the implementations, only the parametric value sufficiently reflects the shift in a measurable way that can be addressed to recover the initial condition.

While generating the symbolic sequences from such non-ideal trajectories, it results in different codes than what are expected from an initial condition. In fact, without a significant number of iterations, insufficient definition of the sequences results in overlapping codes, and therefore mapping to the same real valued point, despite being produced by different initial conditions [16]. With a reduced height TM, if the generated Gray code sequences are converted directly into the corresponding decimal values, the resultant mapping is incorrect and therefore measurement accuracy is affected. This has led to incorrect digitisation of the input signal as observed in [20]. If, however, a lengthier symbolic time series is considered, it is demanding in terms of resource required for generating the additional symbols.

An analysis has been conducted, but in a theoretical setting in [23], where it is shown that the use of a map with ideal parameter is preferable. Thus, a limitation in the applicability of the TM is encountered because in a physical implementation, deviation of the parameter is inevitable, as can be seen from the work of Kapitaniak et al. [13]. In [13], there was an earlier attempt to propose a theoretical model to measure electrical signals using 1D PWL maps where it was observed that the traditionally measured outcomes were greatly affected by slightest error. These errors have been introduced due to the offsets and tolerances of the components used in the physically implemented map, which significantly reduced the parametric domain of the map. It can also be seen from the works of Cong et al. [24] where the problem of recovering initial conditions was approached using the inverse map with suitable use of symbolic sequences as a footmark for the back-track algorithm. Their results have shown a good agreement to the actual input values. However, since the approach is performed backwards through the sequence, the final symbol for a desired length of sequence will have to be known. Therefore, the accumulation process of the entire sequence i.e. all the iterations must be completed before any conversion process can begin. This might add some time overhead in the conversion. However, as each iterative action of the TM on the input signal generates partitions and doubles the number of intervals that the entire range of the initial condition is divided into, finite length symbolic sequences generated by those iterations define each of these specific intervals uniquely [6]. Therefore, the number of bits in the finite length non-ideal symbolic trajectories directly relates to the number of intervals generated by the iterations [15,25,26]. Because the overlapping caused by the non-ideality of the map causes the intervals to be unequal in size. In order to successfully retrieve the correct initial condition from a finite length Gray code generated by a non-ideal TM, it is prudent to identify the initial conditions by the interval in which it belongs. The size of the intervals can be guided by the bit-resolution of the magnitude of the initial condition.

The proposed solution is a forward operating conversion algorithm which can be applied from the starting symbol through each symbol of the sequence to the end as the trajectory generation continues to progress with the map iterations. This implies that such a conversion technique can be applied as a pipelined stage along with the iterations, thus saving the time overhead to collect the entire sequence before conversion. Keeping this aim in mind, the targets have been set objectively which are explained in detail in the following section.

## 1.3 Aims and objectives

The aim of this work is to determine the digital equivalent of signals by using it as an initial condition of a non-ideal TM from the resultant symbolic sequence generated by the iterations to ultimately develop an accurate ADC structure. As the non-ideality of the electronically implemented map most severely affects the parametric value of the TM, the mapping of the symbolic trajectory generated by the iterative process to the initial condition through any direct means, is complex. In particular, the intervals generated by the repeated partitioning of the state space are responsible for the mutation of the symbolic sequence. Acquiring the parametric value would lead to, tracking the shift in the resulting partitions and thus determining the deviation in the trajectory must be accounted for while determining the initial condition that generated the dynamics. In order to execute the said task, the overarching aim can be achieved by fulfilling a set of objectives as are described in the following sub-sections.

### 1.3.1 Initial conditions as intervals

The foremost task is to establish the validity of identifying the initial conditions by the intervals they belong to. Any initial condition, belonging to the possible range of the inputs, can be identified as a point within a short interval. The objective is to define this interval in a way which can later be narrowed down to, among the entire range of possibilities. Depending on how finely defined this interval is, the resolution of the input signal can be defined.

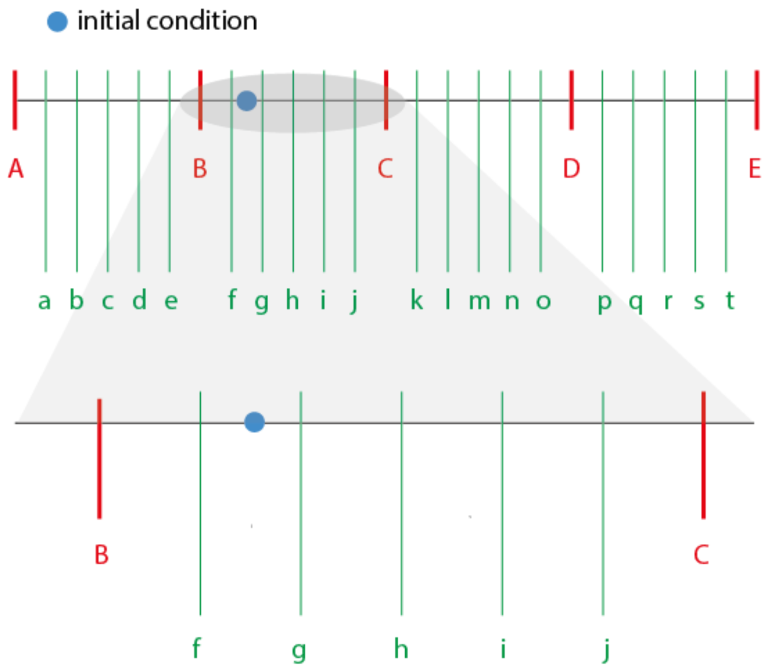


Fig. 1.4 Uniform intervals in the state space

For example, as can be seen from Fig. 1.4, a random initial condition can be either defined as belonging to the interval "*BC*", or "*fg*". The resolution of the input signal, therefore, depends on how sharply the intervals are defined. This is analogous to identifying input signals through the step size of an ADC and thus establishing the efficacy of utilising the tent map as a quantisation block of an intended ADC. Thus, the intervals must be appropriately defined before proceeding to identify the correct interval.

### 1.3.2 Analysis of the intervals generated

The next step is to identify the nature of the intervals generated. While it is ideal to have equally spaced intervals as the origin of the initial condition, the non-ideality of the map parameter affects the dynamics of the map [15,27]. This behaviour is introduced by the component tolerances as well as the inherent noise of the circuit implementation of the map. As a result, the intervals generated are skewed and unequal in nature.

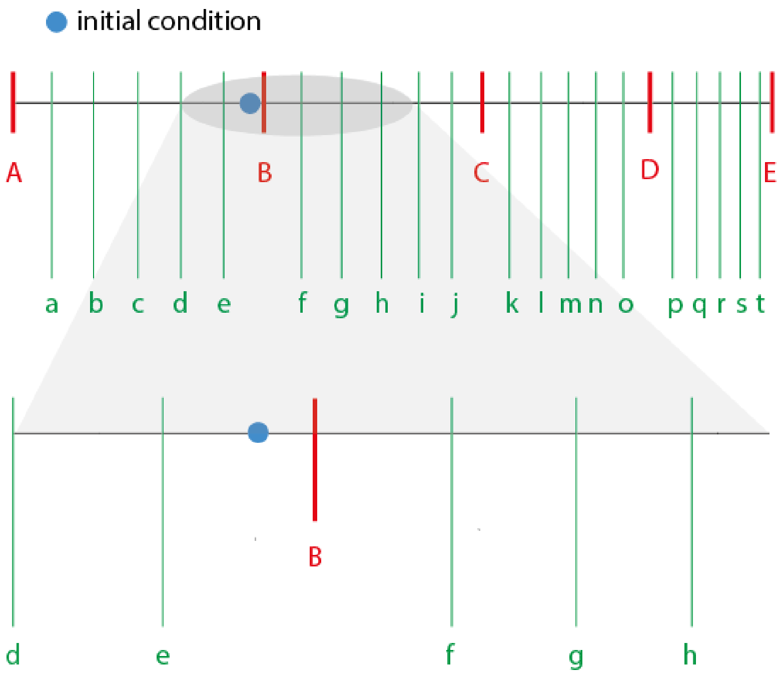


Fig. 1.5 Skewed intervals remaps the same input

As can be seen from Fig. 1.5 the unequal intervals might result in redistribution of the initial conditions and is especially true when the precision of the input signal is important. The same initial condition as in Fig. 1.8 can be now defined to be in the interval "*AB*", or in case of increased precision, in "*eB*". Therefore, in order to determine the underlying dynamics, the next step is to identify and analyse the nature of the non-ideality of the map and thereby the unequal intervals generated by it. This then forms the basis of the initial condition estimation technique.

### 1.3.3 Recovering initial conditions

The precise values of the generated trajectories cannot be retrieved without a precision ADC, therefore tracing back to the initial condition by reverse calculating the identity of the originating interval must solely depend on the symbolic signature associated with the trajectory. Thus, the final step is to utilise the acquired knowledge about the nature of non-ideality to identify how the dynamics of the map has been altered and where the initial condition has been remapped among the skewed intervals. This must be done to determine the modification required for the interval arithmetic to accommodate the deformed symbolic sequences with the skewness of the intervals, tallying the correct sequence with the correct interval. The modified arithmetic is to be utilised to develop an algorithm to determine the initial condition producing the symbolic trajectory in question. The algorithm must be tested both in simulated as well as a physically implemented test scenario to validate the applicability as a successful ADC.

## 1.4 Original contribution

The fundamentals of the work presented here has led to a journal publication titled “An Algorithmic Approach for Signal Measurement Using Symbolic Dynamics of Tent Map”, published in Transactions in Circuits and Systems—I, IEEE [28]. Additionally, further contributions were made to another publication, the complete details can be viewed under the List of Publications. The primary areas of contributions associated to the work presented in this thesis are listed here.

- Analysis of the intervals generated by the iterations of the non-ideal variation of the map, identifying the shift in the partitions generating unequal, asymmetrical intervals.
- Identification of the non-ideal symbolic sequences with respect to the skewed intervals.
- Defining the shifted partitions with the help of the symbolic sequence with respect to the non-ideal parameter value.

- Remapping the initial conditions to the new intervals due to the shifted partitions as opposed to where the initial condition would be mapped by direct binary conversion of the sequences.
- Development of an algorithm to determine the initial condition solely from the symbolic sequence of a known non-ideal TM, keeping in mind the constraints of an implemented map, and the resulting sequences, contributing to the successful implementation of a chaos-based ADC.

## 1.5 Document overview

The background literature of the proposed work is reviewed in detail in the next chapter. A brief account of the types of analogue to digital converters is presented, followed by a summary of the map of choice: tent map. This chapter also sets up the basic premise of the analogue to digital conversion technique and previous works in this direction is discussed.

The third chapter explains the challenges faced by the basic proposition and establishes the need for further investigation. The details of the specific reasons that affect the outcome of the proposed method are mentioned. This chapter concludes by pointing out the specific directions that require the attention in order to improve the accuracy of the results.

The approach developed in this work to determine the initial condition is outlined in the fourth chapter and the necessary analyses of the problems are presented. The basis of the solution is established by summarising applications of the analytical outcomes of the previous chapter.

The fifth chapter explains the mechanism of how the solution works and is summarised in the form of an algorithm. The results and validity of the algorithm is verified in chapter six. Finally, the seventh chapter concludes the overall achievements along with the future steps in order to achieve a fully implementable stand-alone ADC. The other possible directions where the present work can be utilised, including other fields such as communication or cryptography are also discussed.

# 2 BACKGROUND REVIEW

The research for the implementation of a low-cost ADC devices with significant accuracy is very popular. Various architectures have been developed and often combined together in order to achieve a balance of cost, accuracy, resolution, conversion speed, etc. This chapter explores a brief history of the types of ADC architectures, followed by a summary of the properties of 1D maps, and how these properties can theoretically be utilised for an ADC. Finally, the previous attempts at chaos-based ADC that have been proposed are discussed.

## 2.1 A brief account of ADC

Analogue to digital converters (ADCs) convert analogue electrical signals from the sensors etc, to digital representation 1's and 0's of computational electronics. The basic principles of ADCs involve comparing the input voltage to a number of voltage levels and determining the maximum numeric digital value equivalent to the voltage detected. The step size of the increment of voltage levels to which the input signal is compared, is the minimum change that an ADC can detect and therefore determines the resolution offered by the ADC. Thus, for the step size shown in Fig. 2.1, the change in signal desired to be detected does not produce any change in the outcome.

The step size ($Z$) of an ADC with maximum input of $V_{ref}$ whose output is defined by $n$ bits is given by (2.1).

$$Z = {}^{V_{ref}}\!/_{2^n - 1}. \tag{2.1}$$

The step size therefore, depends on the resolution on which the input signal is to be defined by the digital output and is referred to as the *resolution* of the instrument. The higher the number of bits used to define the signal, the higher is the resolution. However, this does not guarantee the *precision* of the ADC, which is dependent on the repeatability and reliability. The components used in manufacturing the ADC are responsible for the degree of precision offered. Therefore, the components that vary greatly with temperature and other physical conditions offer poorer precision than the components whose behaviour is maintained over wider ranges of physical parameters.

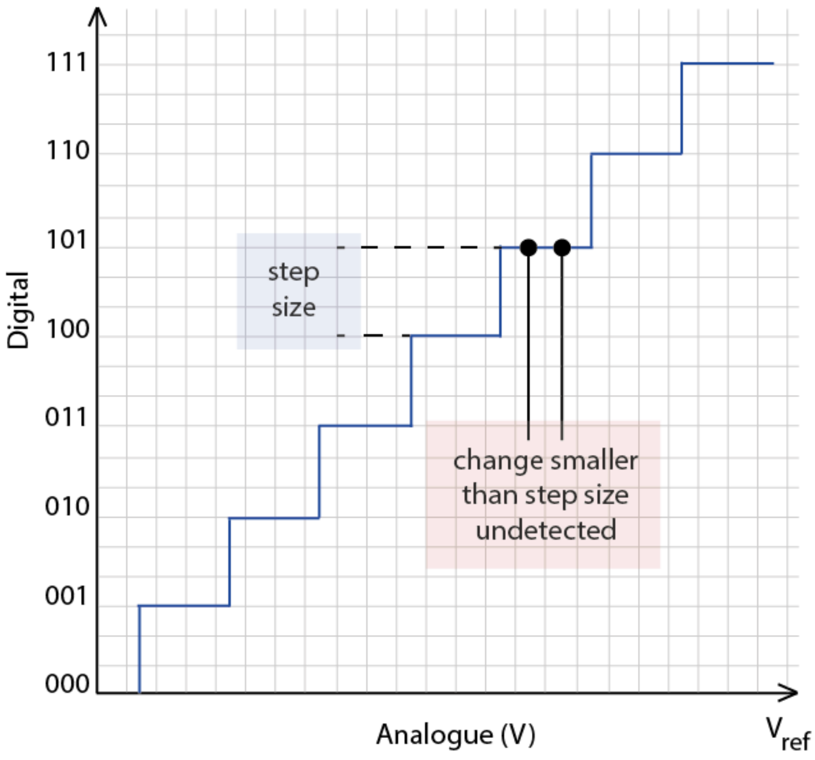


Fig. 2.1 ADC step - size and desired level of detection

The *accuracy* of the device however is the degree of error caused by the gain or off-set parameters, which result in the outcome being scaled by a certain factor, or the entire outcome being off-set by a certain amount. Although both the accuracy and the precision are dependent on the components used, the conditions are independent of each other. Therefore, it is possible to have a good degree of repeatability with the outcome effectively being incorrect due to scaling or off-set error (or both). The accuracy of the step size, in particular, is of great importance as unequal step size results in non-linear error in the outcome, greatly affecting the resolution of the digital output, leading to incorrect mapping of the input signals to the corresponding digital value.

Finally, the *sensitivity* of an ADC is the minimum absolute change in the measurement that can be detected. Unlike resolution, which is the smallest amount of change that an instrument is *theoretically* capable of indicating (as an output value) depending on the number of bits it utilises to do so, sensitivity is the smallest change in measurement (of the input signal) that is capable of triggering a change in the reading or the output. Attaining a balance between these four parameters – resolution, accuracy, precision and sensitivity – influenced the development of various ADC architectures. The *conversion speed* of the ADC is determined by the amount of time required to realise the input signal in terms of digital values. The conversion speed of an ADC is governed by Nyquist's criterion [29] which states that the sampling (measurement attempt) frequency $f_s$ of the ADC should be at least be double the input signal bandwidth given by $f_B < 0.5f_s$ to avoid aliasing. Aliasing happens when signals become indistinguishable between transition of the inputs or a void between the points whenever fast changing input signals are sampled. Therefore, to avoid unnecessary gaps or coarseness in the converted signals, it is always recommended to follow Nyquist's Criterion for the sampling of input signals. Depending on the several architectures of ADCs and the amount of resources involved in each design, conversion times may vary and there might be some conversion overheads as well depending on the measurement technique followed. In order to comply with the Nyquist criterion there is a limit on maximum bandwidth of the input signal that can be chosen for a specific ADC architecture with a certain conversion time.

Currently, there is a wide range of ADC architectures available, with the choice of a particular type of architecture over the others dependent on the application specifications. Each type can be analysed through several performance metrics such as cost, precision, speed, chip area and power dissipation which have been proposed by [30,31]. While for some applications, the speed is of prime importance, for others, lowered power consumption might be of utmost necessity. For critical applications, the precision of the ADC might be of prime importance even at the cost of other parameters while a non-critical device may focus on reducing the cost even if that means compromising on other aspects. Of course, there could be a combination of priorities as well, and all these have led to extensive research in the direction of ADCs, of which delta-sigma (ΔΣ) type, successive approximation register (SAR) type, pipelined and modified flash ADC types are some of the most commonly used architectures [32].

Each of these architectures can be weighted in terms of benefits and shortcomings, resulting in one type to be more profitable over the other depending on the judging criteria. Generally, performance and design complexity of different ADC types are judged based on the quantisation factors (gain, offset, transfer function, noise, etc.), speed, structural organisation and resource consumption [33].

### 2.1.1 Flash type ADC

Flash type converters have the simplest quantisation block of all types of ADCs, it consists of quantised segments of resistive dividers, each of which are referenced to a set of parallel implementations of comparators [32]. The resolution of this type of ADCs depends on the number of voltage divider segments that the input range is divided into. The resulting comparator codes therefore result in a series of ones until the level which is just above the input voltage. This is usually referred to as a thermometer code and is dealt in a priority encoder to determine the actual binary output.

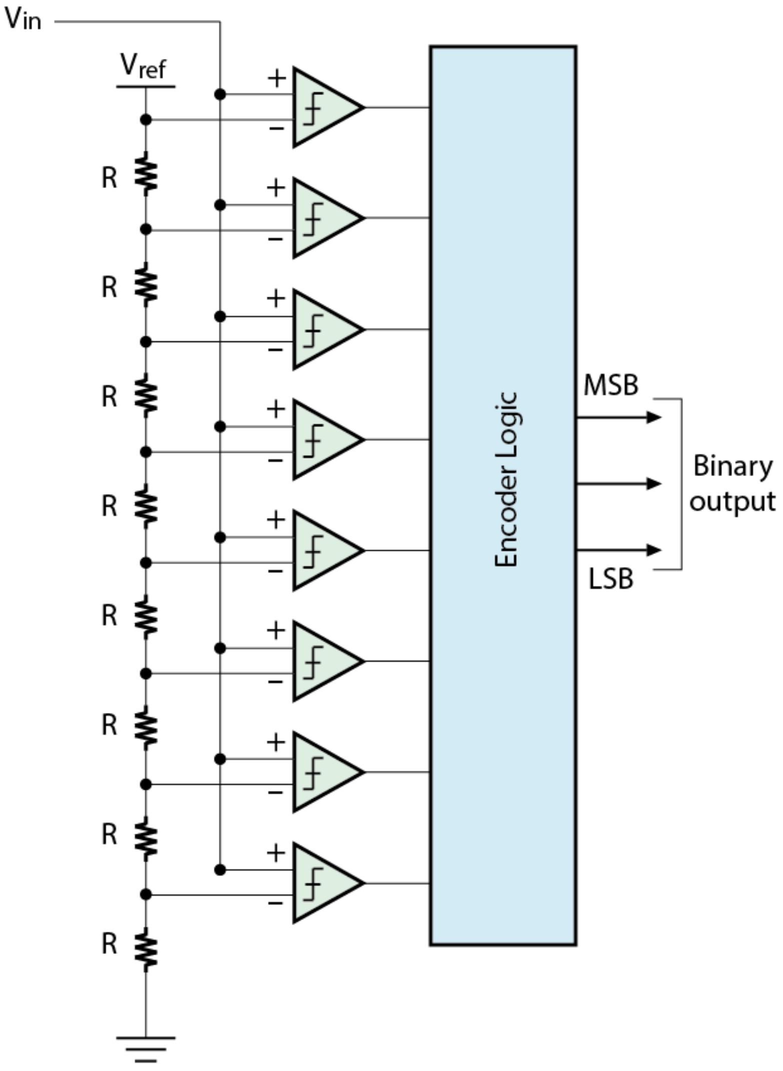


Fig. 2.2 Block diagram of a 3-bit flash ADC

The block diagram of a 3-bit flash-based ADC is shown in Fig. 2.2. Though flash ADCs are well known for high speed operations due to its parallel architecture, it is challenging to achieve higher bit resolution as the number of comparators given is doubled for each increment in bit resolution, thus $2^n$ comparators would be required for an *n*-bit ADC. This increases the chip area requirement significantly for the designs over 6 bits. Currently time-interleaved Flash ADCs are being considered [34,35].

### 2.1.2 ΔΣ type ADC

As an alternative to flash ADCs other kinds of ADC architectures have been introduced, which save sufficient resource but at the cost of reduced speed [32,36]. ΔΣ ADCs measure the input signal in terms of the frequency of a pulse modulated signal generated by the integrator with a thresholding as shown in Fig. 2.3.

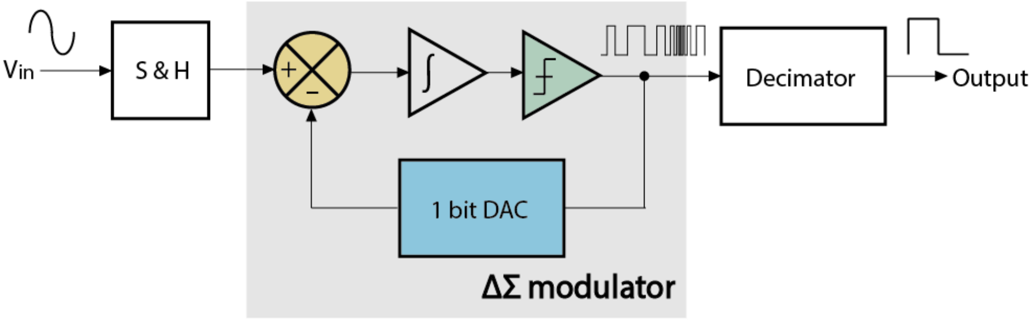


Fig. 2.3 Simplified block diagram of a Delta-Sigma ADC

The difference between the input signal and the 1 bit digital to analogue converter (DAC or the Δ sub-circuit) is integrated until the threshold value is reached and the pulse count is buffered and added together to produce the digital outcome. These ADCs are mostly preferred for better precision and power consumption; however, it offers moderate speed which can be attributed to oversampling. Also, due to the higher order system implementation, a large amount of chip area is required, and the stability factors are affected by the order of modulation.

### 2.1.3 SAR type ADC

The SAR type ADCs are comprised of a register controlled by a successive approximation sub-circuit where the input signal ($V_{in}$) is compared to a reference voltage ($V_{dac}$) controlled by a residual feedback [37]. The ADC initialises the most significant bit (MSB) as 1 and the rest of the bits as 0. The resulting code is converted to the analogue equivalent through a DAC. The outcome is compared with the input signal. If the $V_{in} > V_{dac}$, then the next bit is set as 1, otherwise the previous bit is set as 0 and the following bit is set as 1. The process continues until all the digits in the code, coming from MSB to the least significant bit (LSB) have been set, which denotes end of conversion [38].

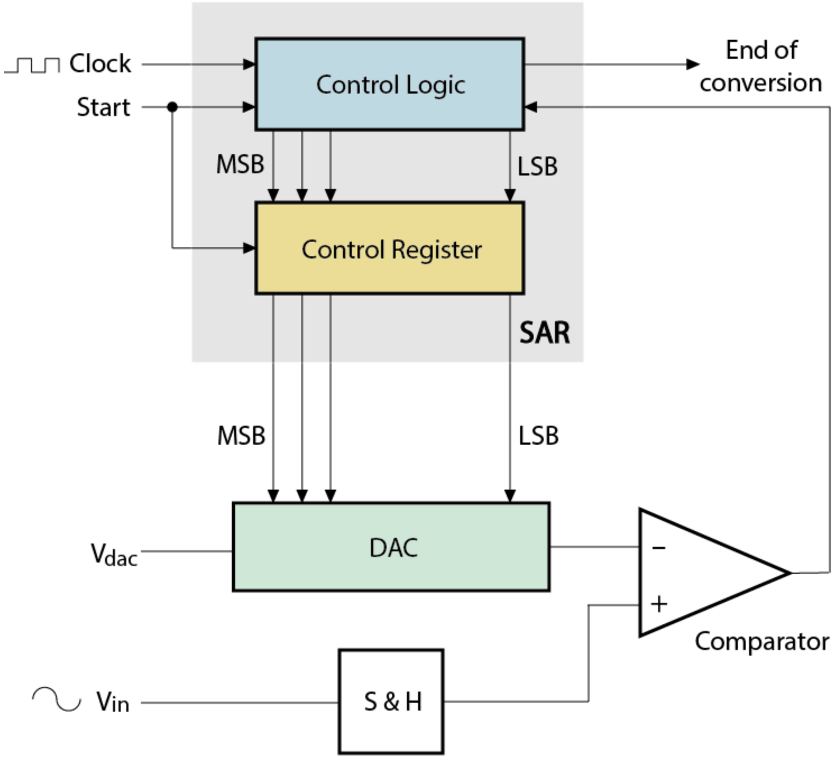


Fig. 2.4 Simplified block diagram of a SAR type ADC

Since the number of bits is fixed, the time taken for values is fixed too. The simplified block diagram of an SAR type ADC is shown in Fig. 2.4. For successive approximation type ADCs, improved resolution is achieved through higher level of design complexity and resource consumption, but at the cost of reduced speed.

### 2.1.4 Pipeline ADC

A flash-based architecture is pipelined ADC, which involves series implementation of quantisation blocks that are operated in parallel. Each quantisation stage generally includes a 3-bit flash ADC which contributes 2 bits to the final outcome, a 3-bit DAC and a multiplier of 4 [32]. Four of such blocks are implemented in series with an additional 4-bit flash ADC at the end stage to complete 12 bits of conversion. A simplified block diagram shows the operation is Fig. 2.5.

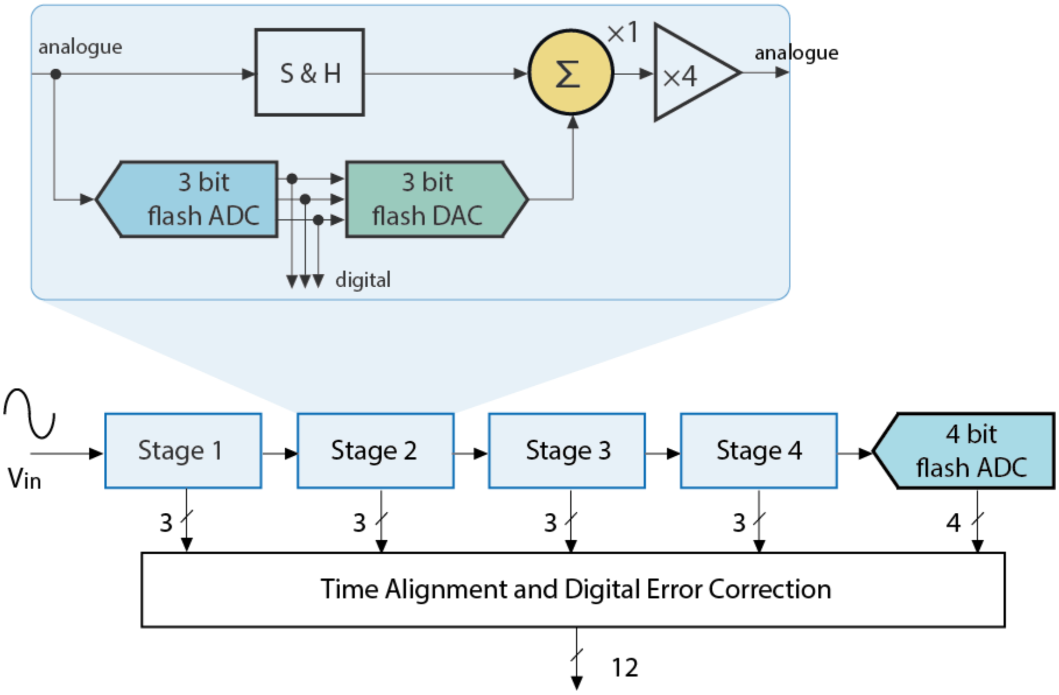


Fig. 2.5 Block diagram of a pipelined ADC

### 2.1.5 Hybrid flash ADCs

Increased resolution is achieved through several other hybrids of flash architecture such as *interpolation type* which reduces the number of pre-amplification units by using additional voltage dividers between two consecutive pre-amplifier outputs [39]. Though chip area is drastically reduced through interpolation, the number of latches required is still the same as the classical flash architecture. This can further be reduced by incorporating additional *folding* stages [40]. The folding stage includes a coarse grain ADC and a fine grain ADC with a folding circuit.

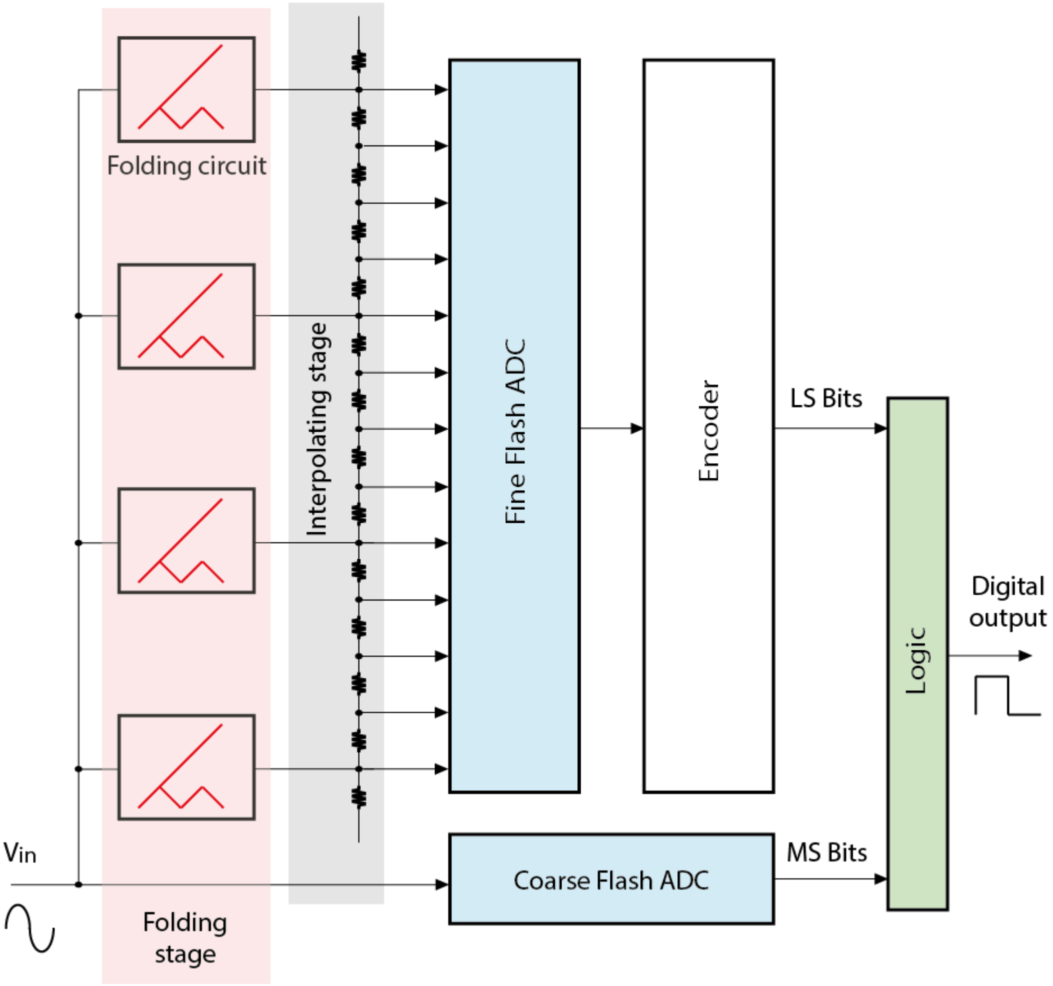


Fig. 2.6 Block diagram of a folding-interpolating ADC

A combined block diagram incorporating folding circuits followed by interpolating stages can be seen in Fig. 2.6. The success of such ADCs depends largely upon the accurate implementation of the folding circuit. As summarised in Table 2.1 developed with the help of the data from Saima et al [33], there is a trade-off between resolution, power dissipation, and speed for flexible design architecture.

Table 2.1 Comparison chart of various types of ADCs

| ADC | Sampling speed | Conversion cycles | Power consumption | Chip area | Accuracy | Cost |
|---|---|---|---|---|---|---|
| Flash | High | 1 | High | High | Low | High |
| ΔΣ | Low | Variable | Low | Medium | High | Low |
| SAR | Medium | Variable | Ultralow | Low | Medium | High |
| Pipelined | Medium | 2N/2-1 | High | High | Medium | High |
| Hybrid flash (folding and interpolation) | Medium | Variable | High | High | Medium | High |

From the aforementioned discussions on the various types of ADC architectures, it is evident that, for improved performance, most of the ADC architectures rely heavily on additional quantization blocks such as increased number of comparators or coarse ADC/DAC as well as folding circuits leading to increased resource consumption, which result in increased chip area with greater design complexity and high power consumption. Therefore, given chaotic maps are simple mathematical functions and can be easily implemented with simpler structures, and that a single block of chaotic map can be reused iteratively to generate the dynamics and symbolic representations corresponding to an input signal, the use of chaotic maps as a quantisation unit is investigated. The physical implementation of a chaotic map can suffer errors that are caused due to the gain and offset of the components used in the design. However, using the technique that is proposed here, such errors can be corrected algorithmically through the principles of dynamics while analysing the symbolic sequence of an input. All such analysis can be carried out in the digital domain, thus making the potential system architecture less complicated at the hardware level.

## 2.2 Chaotic maps: formal definition and properties

Chaotic maps are classified according to the dimensions and topologies defined by the function. Depending on the univariate or multivariate state mapping of the chaotic systems the map definition may be categorised as unidimensional or multidimensional maps. The behaviour of chaotic systems is widely understood through one dimensional, which produce the most fundamental type of chaos that may offer a wide spectrum of chaos under different parametric conditions. One dimensional systems can further be classified according to the system topology e.g. unimodal, multimodal etc. A certain class of chaotic maps called *unimodal maps*, which shall henceforth be denoted as $\mathcal{F}$, is considered. For any mapping given by $f \in \mathcal{F}$ if the function $f$ maps the elements of the set $I$ back to itself, i.e., $f : I \rightarrow I$, where $I = [a,b] \subset \mathbb{R}$, a < b and simultaneously satisfies the following conditions:

– $f$ has a unique maximum $f_{max}$, in the interval $I$,

– $f_{max} = f(x_c)$ where $x_c \in$ I is called the critical point of the map, and

– *f* is monotonically increasing in the interval [$a$,$x_c$] and monotonically decreasing in the interval [$x_c$,$b$], then *f* is unimodal.

When such stretching-and-folding-like behaviours are involved in the evolutionary process such maps show chaotic dynamics as the monotonic progression of the evolution is prevented by the function. The most fundamental type of chaotic maps are one dimensional (1D) chaotic maps e.g. Logistic Map (LM), Bernoulli or Bitshift Map (BM), Tent Map (TM) with an ideal response to the initial condition, shown in Fig. 1.1 – Fig. 2.11 respectively [6].

The class $\mathcal{F}$ consists of certain maps that can be defined using a *control parameter*, $\mu$, such that $f_\mu(x) \in \mathcal{F}$ is valid for $x \in I$ and $\mu \in J \subset \mathbb{R}$, and $f_\mu(x)$ is a map on $I \times J$. The BM, LM and TM, all belong to this family of parametric self-maps $f_\mu : I \rightarrow I$ such that $I = [0,1]$ and also $J = [0,1]$.

### 2.2.1 Bitshift Map

Of the three types of maps, BM (Fig. 2.7), which is also referred to as the Bernoulli Map is the most restrictive in the sense that it is defined for only the ideal parametric value. The BM, $B$ can be defined as $B(x) \in \mathcal{F}$ where

$$B(x) = f_\mu(x) = \begin{cases} 2\mu x & 0 \le x \le x_c \\ 2\mu x - 1 & x_c < x \le 1 \end{cases}. \tag{2.2}$$

where the map is defined only when $\mu$ is 1 (i.e. ideal). The map fails to remain chaotic and generated trajectories that do not remain trapped within the state space and approaches infinity (or negative infinity) for even for the slightest deviation in the parametric value.

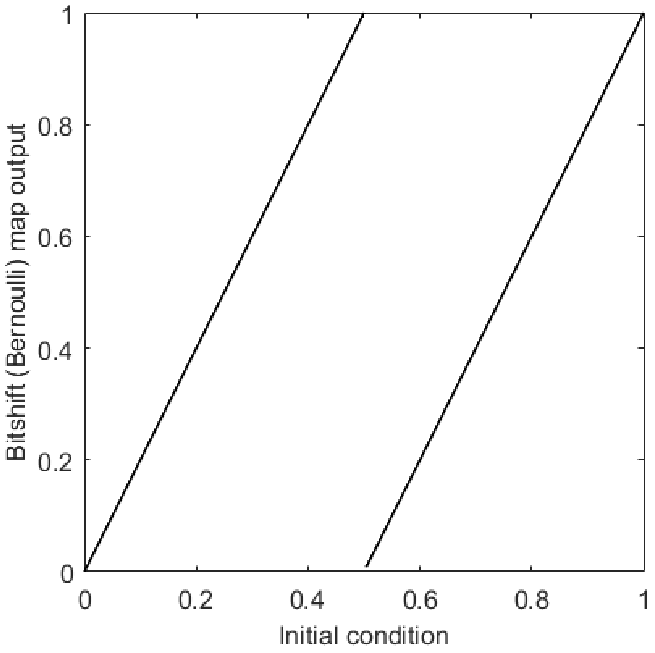


Fig. 2.7 The Bitshift (Bernoulli) Map (BM) behaviour

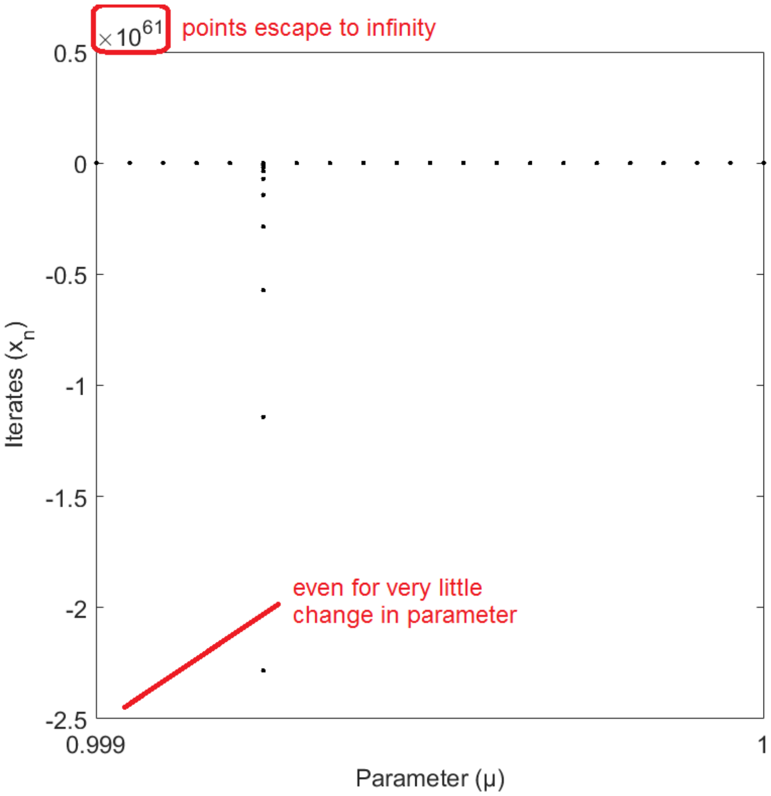


Fig. 2.8 Bifurcation diagram of BM: points escape to infinity

The behaviour can be observed from its bifurcation diagram (BD) shown in Fig. 2.8 where the points within the parametric range [0.999,1] approach astronomical upper limits  and up to -2.5 for the lower limit. As a result, a physical implementation of the map and using it to realise an ADC is rendered futile because the dynamic generated fails to remain chaotic. Therefore, although the map could produce binary sequences theoretically, practicality of achieving an ADC through the BM is is not possible.

### 2.2.2 Logistic Map

As an alternative to the BM, the LM and the TM, both present a large range of workable parameters to achieve chaotic trajectories and thus the possibility of practical implementation. In particular, the LM has been successfully implemented by [41] for practical purposes through an electronic circuit. The map, not being defined as a pair of piecewise linear equations, is more readily adaptable for electronic implementation. The LM function, $L$ is defined as $L(x) \in \mathcal{F}$ where

$$L(x) = 4\mu x(1 - x). \tag{2.3}$$

The map is defined for a wide range of parameters, where the behaviour begins with a single period orbit gradually bifurcating into 2, 4, 8 periods before briefly lapsing into chaotic behaviour and therefore a dense chaos where all the points are defined within the state space of the map. However, as can be seen from the bifurcation diagram of the LM in Fig. 2.10, the behaviour often lapses into a number of wide periodic windows for certain parameters (the periodic windows are marked with numbers referring to the periodicity of the window).

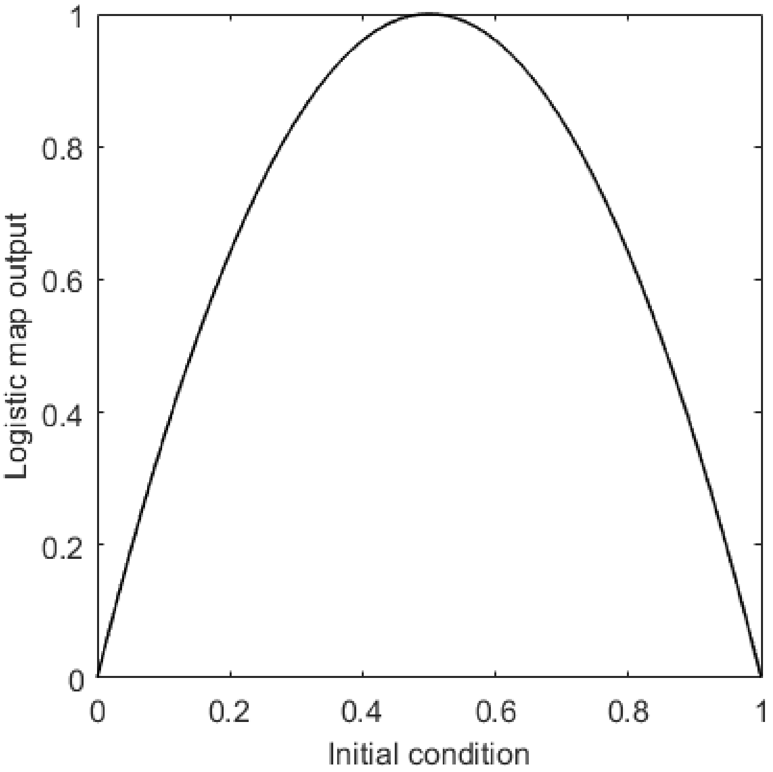


Fig. 2.9 The Logistic Map (LM) behaviour

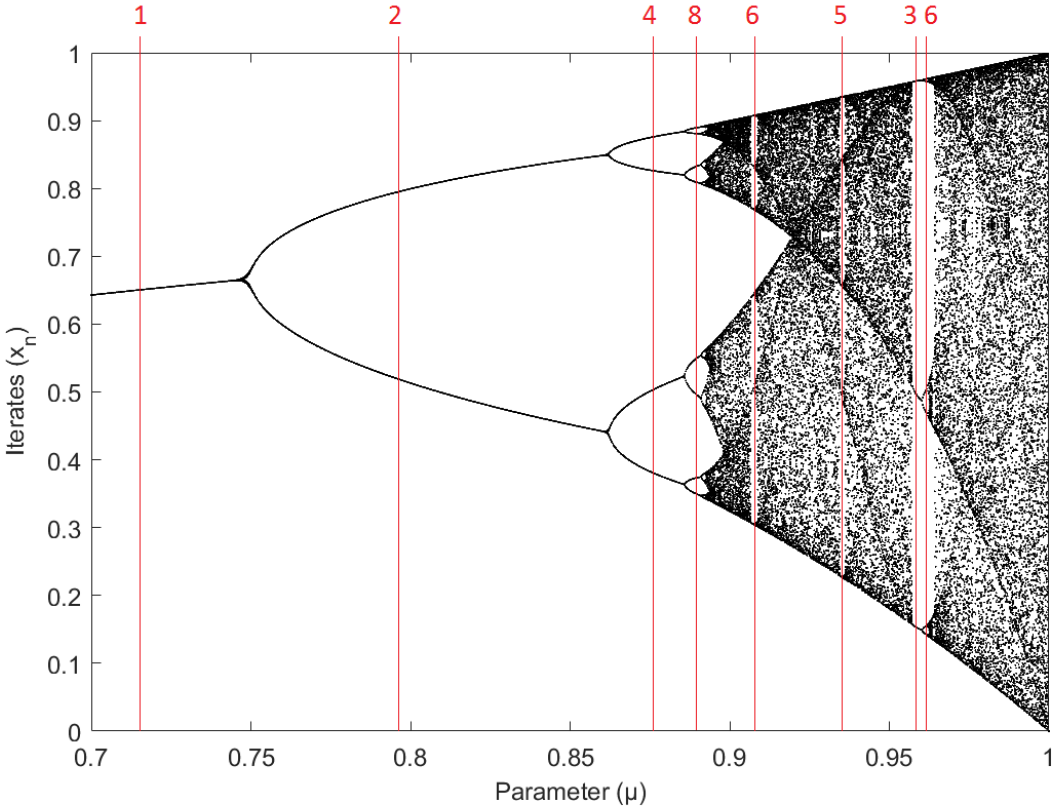


Fig. 2.10 BD of LM: periodic window over the entire range

When the map is implemented electronically, there is every possibility that the component tolerances might drift the parameter into one of the periodic windows, if not already in it. This poses a possible hindrance for the ADC to be reliably utilised in all conditions.

### 2.2.3 Tent Map

The TM shows dense and robust chaos over a significant range of parameters (above ~0.7) where there are no windows of periodicity. Any periodicity is limited only to a small region towards the lower range of the parametric value, leaving sufficient play for the parameter value to drift due to tolerances. This can be seen from the bifurcation diagram in Fig. 2.12. The uniformly dense chaos [19] exhibited by the TM enables a successful utilisation of the chaos where unique sequences can be generated if sufficiently long trajectories are considered.

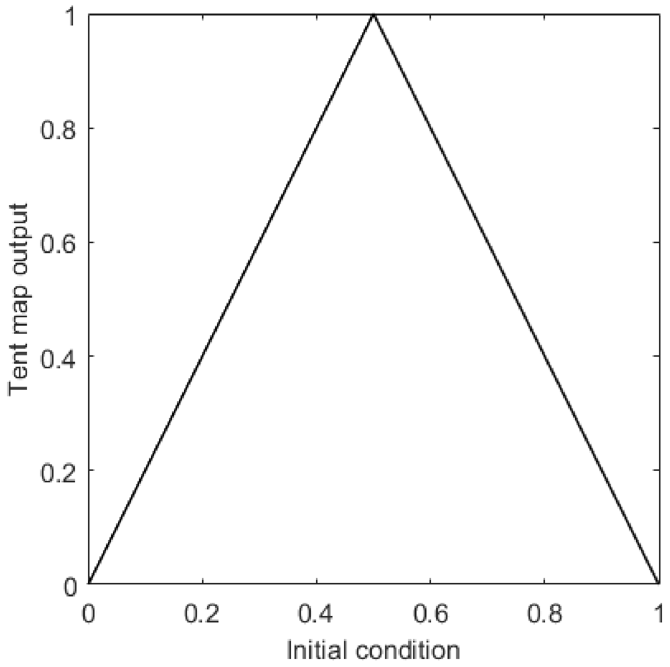


Fig. 2.11 The Tent Map (TM) behaviour

A successful implementation of the map has been done by Campos et al. [42] which can be readily utilised for practical purposes. The TM, which can be defined as a function  belonging to the function described in section 2.2 as $T \in \mathcal{F}$, which is and $T(x)$ is defined as

$$T(x) = f_\mu(x) = \begin{cases} 2\mu x & 0 \le x \le x_c \\ 2\mu(1-x) & x_c < x \le 1 \end{cases}. \qquad (2.4)$$

where $x_c = 0.5 \in I$ is the critical point of the map. For the map to be chaotic, it is crucial that the range $J$ of the control parameter $\mu$ is given by $J = (0.5,1]$.

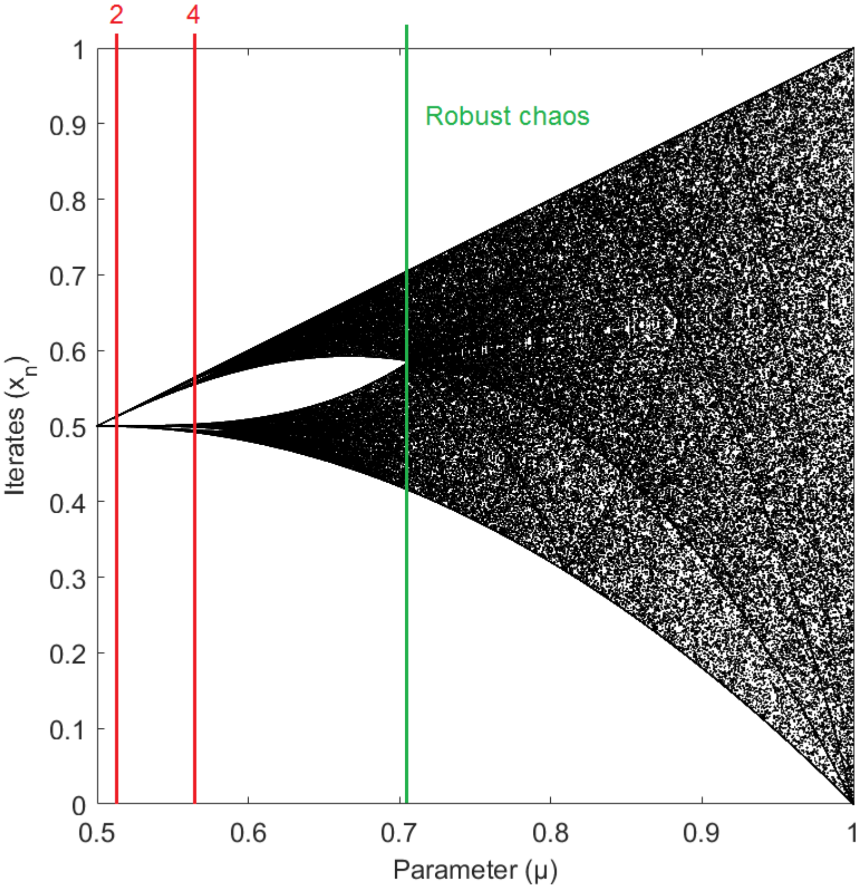


Fig. 2.12 BD of TM: points above ~0.7 exhibit robust chaos [17]

In the closed interval $I \subset \mathbb{R}$—also known as the state space of the map—the $i^{th}$ iterate of $T(x)$ is defined as $x_{i+1} = T(x_i)$, $i \in \mathbb{N}_0$ (where $\mathbb{N}_0 = \{0\} \cup \mathbb{N}$) such that,

1. $x_0 = T^0(x) = x$

2. $x_{i+1} = T^{i+1}(x_0) = T(T^i(x_0)) = T(x_i)$

3. $T(0) = T(1) = 0$ is the absolute minimum of the map

4. $T_{max} = T(x_c) \leq 1$, $T_{max}$ is the maximum height of the map, for $0 \leq \mu \leq 1$

5. $T(T_{max}) = T^2(x_c) \geq 0$, $T(T_{max})$ is the dynamic minimum of the long-term trajectory.

When the map is iterated for $n$ times, a set of $n$ values is generated. This is known as the trajectory of the map. The set of $n$+1 points (including the initial condition $x_0$) visited by the trajectory of a TM can be referred to as the orbit of that particular initial condition and is defined as $\mathcal{O}_T(x_0) = \{T^0(x_0), T^1(x_0), T^2(x_0), \ldots, T^n(x_0)\}$. The behaviour of $\mathcal{O}_T(x_0)$ is periodic at $J = 0.5$, with a period of one. As the value of $J$ increases to $J > 0.5$, the periodicity doubles into a two-period orbit. It then doubles again and eventually results into aperiodic orbits for higher values of $J$ producing the chaotic characteristics of the map. Eventually, the chaotic behaviour exhibited by the map at this stage is known as *robust chaos* [19].

Given that chaotic maps are sensitive to initial conditions, an infinitesimally small change in the initial condition results in substantially diverging trajectories and due to the folding nature of the map, points in the closed interval $I \subset \mathbb{R}$ will eventually map on to every other point in $I \subset \mathbb{R}$, or arbitrarily close to it [25].

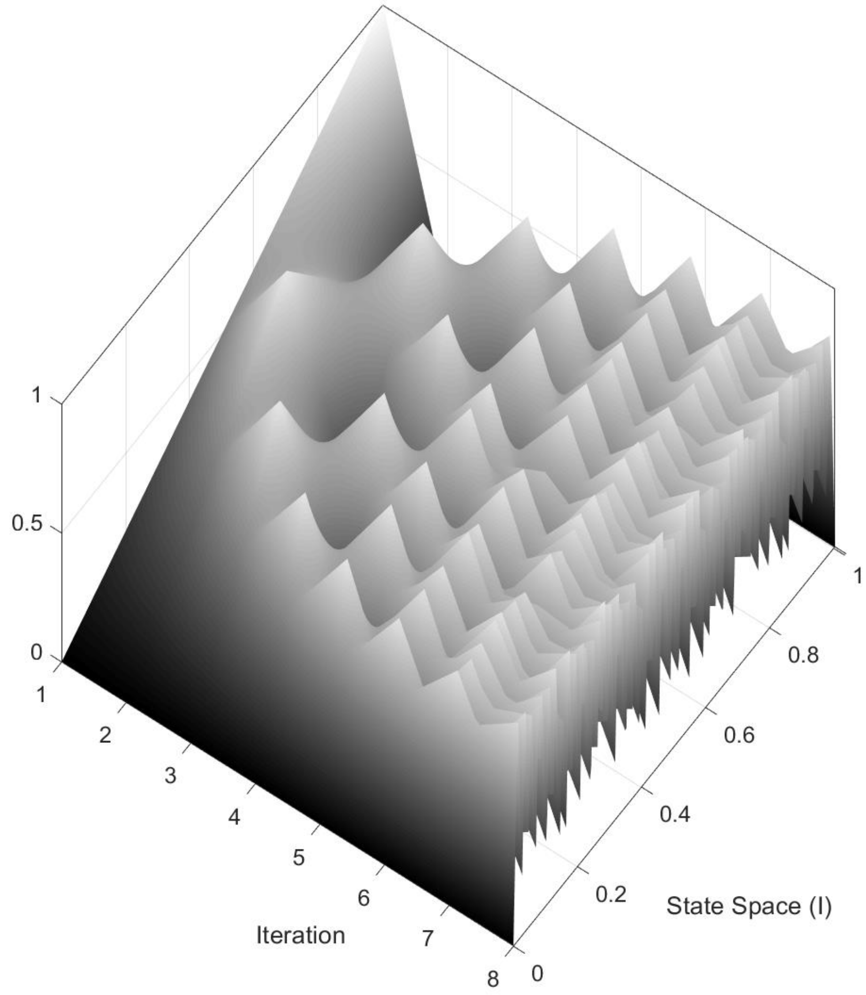


Fig. 2.13 A 3D view of the iterates over the entire state space

The stretching and folding nature of the map can be isolated as the orientation preserving and orientation reversing side of the map [6]. The monotonically increasing side of the map function, i.e., the restriction of the function that acts upon the points up to the $x_c$ are only stretched, but their orientation of increment is maintained. The other half, where the folding action takes place is called the orientation reversing half, where the map is monotonically decreasing, despite having a stretching action by the $2\mu$. This action is repeated over the iterations (Fig. 2.13) and result in mirroring effect across the state space over the iterations which can be seen from the top view of Fig. 2.13, as can be seen in Fig. 2.14.

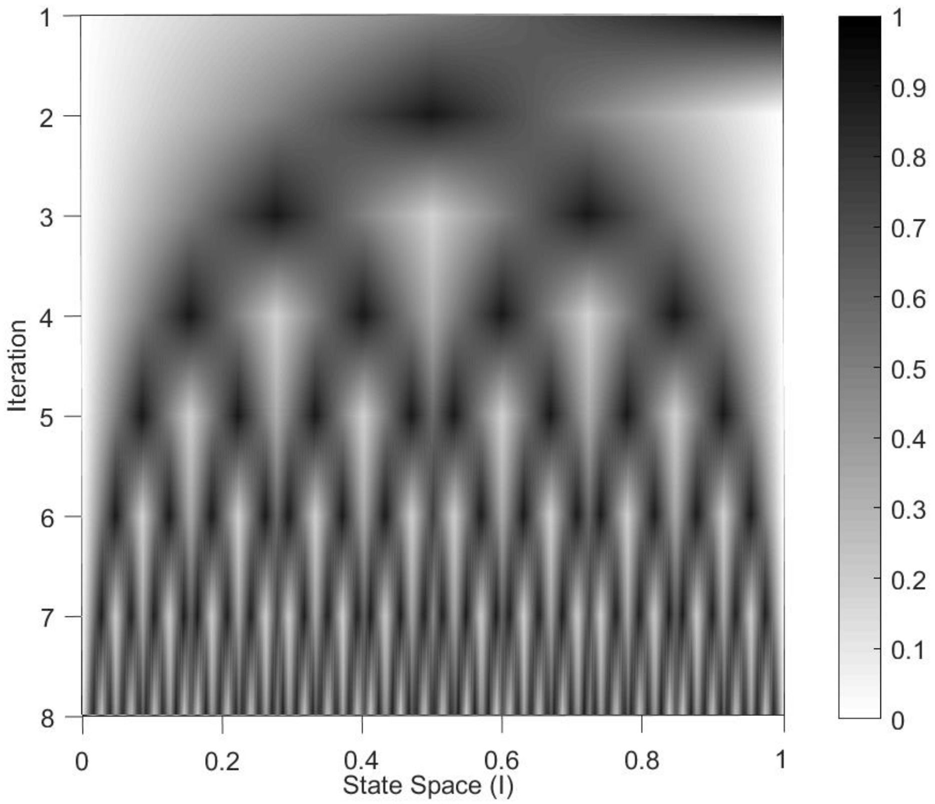


Fig. 2.14 Fractal dynamics of the TM over the state space

Observing Fig. 2.14, it can be seen that a self-similar (fractal) behaviour emerges from the long-term dynamics of the map over the entire state space. Through the repeated preservation and reversal of the orientation of the map, unique trajectories can be generated for any arbitrary initial condition in $I \subset \mathbb{R}$, that result in the aforementioned self-similarity, that was later utilised to determine its possible role in the interval partitioning. In the following section, the symbolic sequence generated by a TM has been described with its general features and functionalities that are relevant to this application.

## 2.3 Symbolic dynamics

The orbit of a TM, given by $\mathcal{O}_T(x)$, can be transformed into a symbolic sequence $\mathcal{S}_{n+1}$ of length $n$+1 where $\mathcal{S}_{n+1}(T,x) = s(x_0)s(x_1)s(x_2)\ldots s(x_n)$. The first attempt in this direction was initiated by Metropolis et al [14] who defined the symbolic sequences using three letters, $L$, $C$ and $R$ standing for *left*, *centre* and *right*. The symbol assignment was intuitive, the points to the left of the critical point are assigned $L$, the ones to the right are assigned $R$ and the critical point itself is assigned $C$. The realisation that the symbols exhibited pattern over the parametric space resulted in development of a strong tool for analysing chaotic dynamics. Eventually, the pattern in the symbolic dynamics over the state space was observed [14,43] and it was shown that for unimodal maps, if the symbols were replaced such that $L$ represented 0, $R$ represented 1 and $C$ could represent either one of the two, the patterns were either binary or Gray codes. The codes generated by the stretching and folding nature of the TM is always Gray code. Therefore, bypassing the $L$, $R$, $C$ convention, the symbolic sequence $\mathcal{S}_{n+1}(T,x)$ could be conveniently defined as $s : [0,1] \rightarrow \{0,1\}$ is defined as

$$s(x_i) = \begin{cases} 0 & x_i \leq x_c \\ 1 & x_i > x_c \end{cases}. \tag{2.5}$$

Furthermore, it has been shown that, the symbolic sequences generated are Gray codes [44]. On every $i^{th}$ iteration, the state space $I$ is partitioned into $2^{i+1}$ mutually exclusive sub-intervals $I_j^i$ where $0 \leq j \leq (2^{i+1}-1)$ is the count of the sub-interval increasing from the left endpoint 0 to the right endpoint 1 within $I$ and $i$ is the iteration count [16]. The input signal to the function must therefore belong to any *one* of the sub-intervals. The following properties relate the symbolic sequence $\mathcal{S}_{n+1}(T,x)$ to the sub-intervals generated by the map.

1. Every $x \in I_j^i$ results in same symbolic sequence $\mathcal{S}_{i+1}(T,x)$

2. If initial conditions $x \in I_j^i$ and $\hat{x} \in I_{j+1}^i$, then $\mathcal{S}_{i+1}(T,x)$ and $\mathcal{S}_{i+1}(T,\hat{x})$ differ by only one bit

3. $I_0^i \cup I_1^i \cup I_2^i \cup \ldots \cup I_{2^{i+1}-1}^i = I$

4. $I_j^i \cap I_k^i = \emptyset$ for $j \neq k$

Therefore from the properties 1, 2 and 4, the symbolic sequence $\mathcal{S}_{n+1}(T,x)$ can be interpreted as an $n$+1 bit long unique symbolic identity that corresponds to a sub-interval of the size $I_j^n$ and so, the longer the symbolic sequence, the narrower will be the size of the intervals. Each such $j^{th}$ interval can be identified by the corresponding symbolic sequence $\mathcal{S}_{n+1}$. The order of the symbolic sequences, as shown in [16], corresponds to the order $j$ = 1, 2, 3, …, $2^{n+1}$ according to which the intervals $I_j^n$ are ordered in $I$. For example, for all $\mathcal{S}_3(T,x)$, the order of the possible sequences corresponding to $j$ can be seen from Table 2.2. Therefore, for $\mathcal{S}_{n+1}(T,x)$, $I_j^n$can be written as $I_{\mathcal{S}_{n+1}}^n$and can be used as a basis to identify the originating interval of an initial condition.

In the case of ideal parameter, the initial conditions directly correspond to their originating intervals when their symbolic signatures $\mathcal{S}_{n+1}(T,x)$ are converted to the corresponding binary codes $\mathcal{B}$ : $b_0b_1b_2...b_n$

$$b_i = \begin{cases} s(x_i) & i = 0 \\ b_{i-1} \oplus s(x_i) & i > 0 \end{cases}. \tag{2.6}$$

$\mathcal{B}$ is further converted to the real values. This conversion from $\mathcal{S}_{n+1}(T,x)$ to real number is referred to as Gray Ordering Number (GON), given by the transformation

$$\text{GON}(\mathcal{S}_{n+1}) = \sum_{i=0}^{n} b_i^{-(i+1)}. \tag{2.7}$$

and can be ordered by its magnitude as described in [16].

Table 2.2 shows GONs for a 3-bit sequence generated using $T^2(x_0)$ for inputs ($x_0$) with a step-size of 0.125. Considering a longer sequence will result in identification of input signals with a finer step size.

Table 2.2 Correspondence between Sequences and Input Intervals

| $j$ | $\mathcal{S}_3(T,x)$ | Binary | GON |
|---|---|---|---|
| 0 | 000 | 000 | 0 |
| 1 | 001 | 001 | 0.125 |
| 2 | 011 | 010 | 0.25 |
| 3 | 010 | 011 | 0.375 |
| 4 | 110 | 100 | 0.5 |
| 5 | 111 | 101 | 0.625 |
| 6 | 101 | 110 | 0.75 |
| 7 | 100 | 111 | 0.875 |

## 2.4 Previous chaos-based ADC attempts

This mapping property of the TM has been considered previously and attempts have been made to utilise it in developing an ADC. The first attempt was that of Kennedy [12] in 1995, where both the TM and the BM were considered for conversion. Later, Kapitaniak et al. [13] also attempted independently, with similar results with further results exploring the non-ideal conditions. It was shown that the TM generated Gray code results while the BM directly produced binary outcomes. In the following sub-sections, the details of the inception of an ADC based on the BM and the TM respectively are shown.

### 2.4.1 ADC based on BM

The symbolic structure generated by the BM over the iterations for the entire state space is binary in nature. As can be seen from Fig. 2.15 the state space has been partitioned over repeated iterations. This, therefore, theoretically appears to be the most appropriate choice for the ADC. Depending on the number of symbols sampled from the map, the initial condition could be recovered up to that many bits of precision [13]. However, in a practical setting, the said map must be implemented through a physical circuit. The inherent noise in the implemented system and the precision of the components involved inevitably incurs a non-ideality, where the parametric value of the map is most severely affected. In such non-ideal parametric conditions, the BM fails to function as can be seen from the bifurcation diagram of the map (Fig. 2.8) and cannot be intended as an ADC for practical purposes.

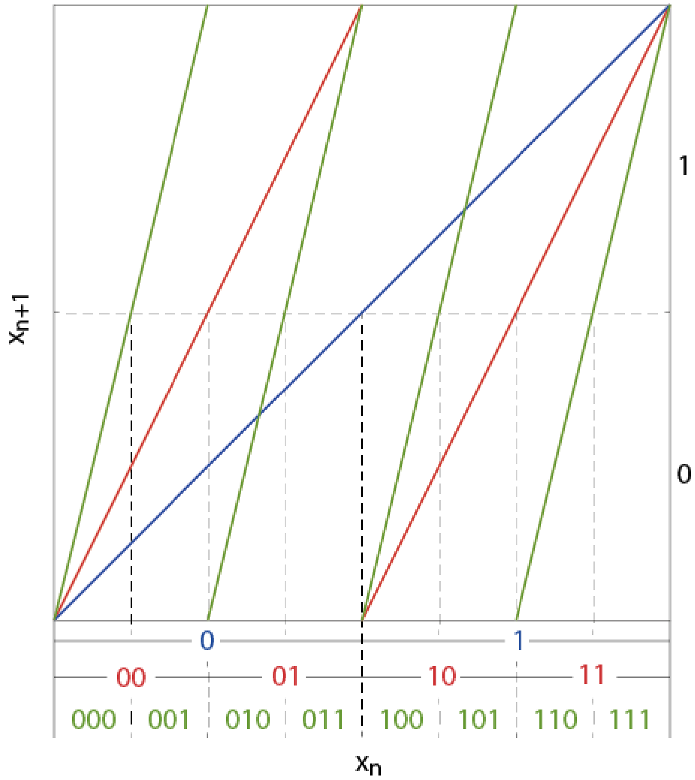


Fig. 2.15 Intervals with binary signature generated by the BM

### 2.4.2 ADC based on TM

An equivalent principle can be utilised for TM to develop an ADC. The symbolic structure for the state space as generated by the TM is in Gray codes (Fig. 2.16). In addition, the map functions chaotically even when the parametric value is non-ideal and can therefore be implemented in practical domain, and exhibits aperiodic dense chaos for an extended range of parameters (Fig. 2.14). In [12], the conversions were successful when the parameter for either maps must be maintained at the ideal value. The slightest deviation showed significant error in the outcomes. Similar experiments were carried out by Kapitaniak et al. [13] using both the maps, where the various sources of deviations were explored through simulation by varying separately both the elements that are susceptible to variation in an implemented map: the parameter as well as the critical point. The outcomes confirm that directly treating the non-ideal symbolic sequences as digitised output values will produce erroneous results. In particular, the effect of the parameter shift is more readily reflected on the symbolic sequences.

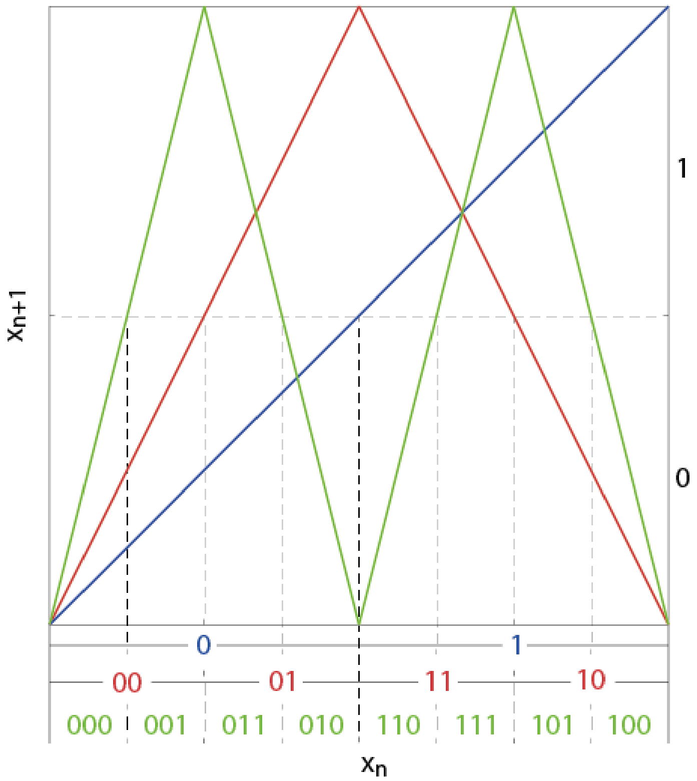


Fig. 2.16 Intervals with Gray signature generated by the TM

In the work of Alvarez et al. [45] it is seen that considering a large number of iterations and converting them using directly to the decimal values fairly approximate the initial condition even for non-ideal parametric values as the symbols contribute value in exponentially diminishing quantity thus asymptotically approaching the initial condition over the iterates. However, in a practical scenario, obtaining that many iterates, even without noise is an implausible idea.

Dinu et al. [46] developed a means of reverse-calculating the initial condition from a randomly assumed final iterate chosen based on the final symbol of the sequence. The generated outcomes were produced with sufficient accuracy; however, consistency cannot always be guaranteed. Cong et al. [24] have also employed similar techniques of reverse-calculations, which has shown good agreement of the outcomes with the input signals. However, as the reversal requires the entire sequence to be collected prior to conversion, it cannot be executed parallel to the iterations of the map. Also, there is a lack of real results testing the feasibility of the techniques in data from implemented circuit results of the map which inevitably involves the effect of noise distorting the symbolic sequences. Therefore, a conversion method is developed considering the challenges offered by a real circuit implementation which are described in the following chapter.

# 3 ANALYSIS OF NON-IDEAL BEHAVIOUR

The ideal condition for developing an ADC using the TM is to utilise a map with full parametric value. This would ensure that the symbolic sequence generated by the map uniquely identifies the initial condition that generated it. In fact, the symbolic sequence generated would produce the Gray code equivalent bits for the binary value of the initial condition. However, such a system is only feasible in a software simulated environment which is immune from component tolerances as well as the inherent noise in the system. Such a system would be of no practical use in engineering measurement and control where the signals collected by sensors are real electrical voltage or current. Therefore, producing a TM in the electronic domain is inevitable.

Since the primary intention for utilising chaotic maps to develop an ADC is to reduce the cost of the system while maintaining satisfactory accuracy, the components used for the implementation purposes must not be very highly rated in terms of tolerance. Also, such components and the nature of electronic circuits in general, are susceptible to noisy interference. Thus, when the TM is implemented, the ideal values of various parameters of the map function suffer from deviations from the ideal values. This has a significant effect on the map dynamics due to the sensitivity of the map to slightest changes. This in turn, affects the outcomes if the dynamical trajectories responsible for the set of initial conditions which are used as a signature for the initial condition measurement. As a result the generated Gray symbolic outcomes are not directly convertible to their binary or decimal equivalents. As the deviation is inevitable, a solution must be aimed at either recovering the ideal symbolic trajectories from the deviated ones, or devising a method to remap the initial conditions correctly through the non-ideal sequences. In order to determine the feasible and relevant solution, the effect of the non-ideality of the implemented TM needs to be understood thoroughly.

## 3.1 Parametric deviation

The most common form of non-ideality – reduction in the height of the map function – alters the intervals that correspond to each symbolic sequence [15]. In general, the symbolic representation of the TM dynamics of one half of the map is the mirror image of the other half. Therefore, when the intervals within the state space are charted symbolically, the structure of the symbolic codes are the same as that of the Gray codes. Gray code can be processed using a straightforward numerical exercise that involves the Gray to binary conversion and further into decimal equivalent values to estimate the initial condition. However, this is possible if and only if the tent map dynamics are generated by the circuit is 'ideal' i.e., its domain or the dynamical state space maps to the entirety of [0,1].

As stated before, a parallel independent research in the same direction has been conducted where the parameter of the non-ideal TM has been estimated by Dutta, once through the maximum sequence method as done in [28] and also in [47] where the inherent system noise has been utilised to determine the non-zero fixed point in a probabilistic approach. The resulting fixed point has then been utilised to determine the parameter that led to the estimated fixed point.

When the map is implemented physically, there is a reduction in the height of the tent map and therefore due to the non-ideal parameter as seen in

Fig. 3.1, the symbolic trajectories are no longer the same as it is with the ideal parameter. Hence converting the generated Gray code sequences directly into the corresponding real values, leads to a deviation in the estimated outcome to the actual initial condition, thus leading to incorrect mapping [48].

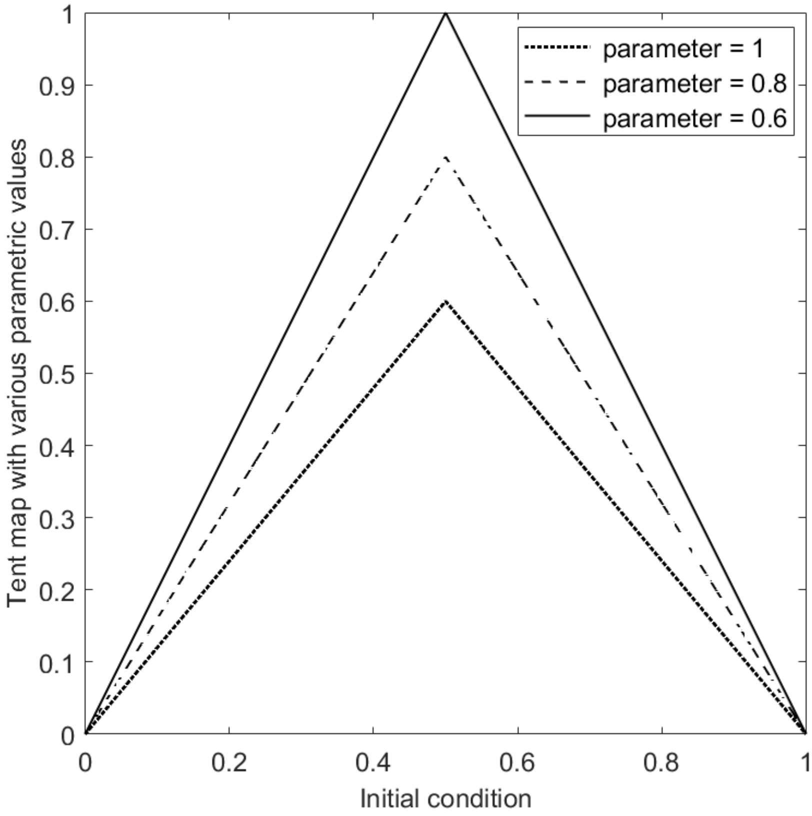


Fig. 3.1 TM with reduced heights due to various parametric values

For example, when the initial condition 0.4375 is iterated eight times through an ideal and a non-ideal TM, the resultant trajectory after 7 iterations, and the corresponding symbolic sequences are shown in Table 3.1. In the ideal situation, as can be seen from the table, the result is analogous to a conversion using an 8-bit ADC (any type). However, the results vary in case of the non-ideal map, where the outcome is greatly shifted from the expected result.

Table 3.1 Change in trajectory with change in parameter

| Ideal trajectory<br>$\mu = 1$ | Ideal<br>sequence | Non-ideal trajectory<br>$\mu = 0.95$ | Non-ideal<br>sequence |
|---|---|---|---|
| 0.4375 | 0 | 0.4375 | 0 |
| 0.875 | 1 | 0.83125 | 1 |
| 0.25 | 0 | 0.320625 | 0 |
| 0.5 | 0 | 0.6091875 | 1 |
| 1 | 1 | 0.74254375 | 1 |
| 0 | 0 | 0.489166875 | 0 |
| 0 | 0 | 0.9294170625 | 1 |
| 0 | 0 | 0.13410758125 | 0 |

As can be seen, even for a slight deviation of parameter ($\mu = 0.95$), the symbolic sequence generated is quite different from the ideal sequences. As a result, converting the symbols directly into the corresponding decimal values does not yield the initial condition as can be seen from Table 3.2.

Table 3.2 Imperfect mapping of symbolic trajectory

| Power factor | Ideal Gray | Ideal Binary | Decimal | Non-ideal Gray | Non-ideal Binary | Decimal |
|---|---|---|---|---|---|---|
| $2^{-1}$ | 0 | 0 | 0 | 0 | 0 | 0 |
| $2^{-2}$ | 1 | 1 | 0.25 | 1 | 1 | 0.25 |
| $2^{-3}$ | 0 | 1 | 0.125 | 0 | 1 | 0.125 |
| $2^{-4}$ | 0 | 1 | 0.0625 | 1 | 0 | 0 |
| $2^{-5}$ | 1 | 0 | 0 | 1 | 1 | 0.03125 |
| $2^{-6}$ | 0 | 0 | 0 | 0 | 1 | 0.015625 |
| $2^{-7}$ | 0 | 0 | 0 | 1 | 0 | 0 |
| $2^{-8}$ | 0 | 0 | 0 | 0 | 0 | 0 |
|  |  | Outcome = | 0.4375 |  | Outcome = | 0.421875 |

## 3.2 Narrowed dynamical attractor

The dynamical trajectories of all the initial conditions of the state space become trapped at one point in time within a boundary called the dynamical attractor. In the ideal TM, the iterated trajectory reaches the maximum height at some point for all possible initial conditions of the map. Therefore, in the ideal case, the dynamical attractor is the entire state space *I*. However, in the hardware implementation of the map, it may not be possible to maintain the parameter $\mu$ = 1 constantly. Under such non-ideal condition, when the map height (parameter) is reduced i.e. $\mu$ < 1, changes to certain degrees in the dynamical characteristics of the map that can be noticed. This can be viewed from the bifurcation diagram of the TM (Fig. 2.12) where, with changing parameter, there is a gradual narrowing of the attractor beyond which, the iterated point can no longer visit during further iterations. This can also be experienced for random individual initial conditions which also exhibit the trapped region where its dynamics is limited to. This can be viewed very well in a cobweb diagram showing a trajectory originating from $x_0$ = 0.000124 after 300 iterations is shown in Fig. 3.2.

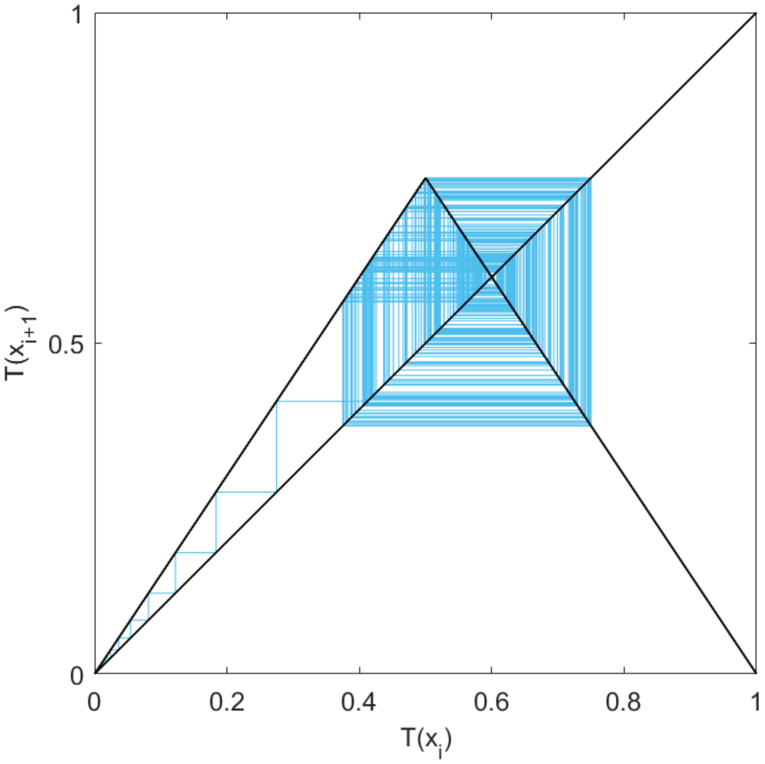


Fig. 3.2 Cobweb diagram for points originating before $x_c$

The cobweb diagram is generated by alternating between the points visited by the trajectory and the $x = y$ line, thus highlighting the growth and folding of the iterate values at each step. In a non-ideal TM, the cobweb diagram over the long-time dynamics exhibit the region where the folding nature of the map limits the boundary beyond which none of the iterates migrate. This property of the dynamical attractor of the tent map is characterised by the parameter $\mu$ and therefore, the maximum height $T_{max} = T(x_c)$ where it determines the upper limit of the attractor. As can be seen in Fig. 3.2, the points originate further away from the $T_{min}$ for $\mu = 0.75$, however, once the point enters the attractor, the trajectory stays limited within this region.

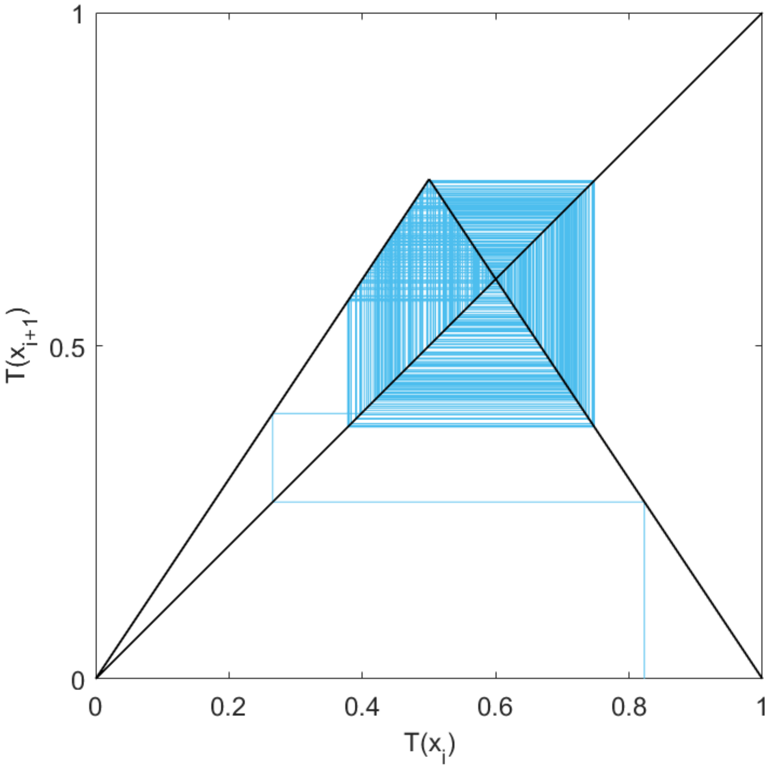


Fig. 3.3 Cobweb diagram for points originating before $x_c$

A similar plot can also be generated for a trajectory originating beyond the $T_{max}$. With an initial condition of $x_0 = 0.823$ (Fig. 3.3) the trajectory first maps to a point less than $T_{min}$ and then gradually enters the attractor and stays confined. This effect is also seen in the bifurcation diagram (Fig. 2.12), where the boundaries of the bifurcated points define the upper and lower limit.

Due to the folding nature of the tent map, the minimum value of the attractor can be determined as $T_{min} = T(T_{max}) = T(T(x_c))$. Thus, the upper limit and lower limit of the attractor are given by $T_{max} = T(x_c) = 2\mu x_c = \mu$, and $T_{min} = T^2(x_c) = T(T_{max}) = T(\mu) = 2\mu(1–\mu)$. The attractor can therefore be identified as the portion in the state space demarcated by these limits, which shall henceforth be referred to as $I' = [T_{min},T_{max}] = [2\mu(1–\mu),\mu]$. As a result, when $\mu < 1$, over time, it can be observed that points originating from arbitrary locations of the state space $I$ will eventually be attracted and be trapped within $I' = [T_{min},T_{max}]$, where $I' < I$. The dynamics continue infinitely as the iterated points never achieve the full height of 1 and therefore cannot reach 0 either on the next iterate. As both fixed points 0 and $2\mu/(1+2\mu)$ are unstable for $\mu > 0.5$, the dynamics stay trapped within $I'$ for a non-ideal $\mu$. The lower the value of the parameter, the narrower is the region where the dynamics is trapped and therefore the periodicities increase as the iterates repeatedly visit the few intervals that fit within that region.

## 3.3 Skewness of the intervals

The non-ideality of the map also affects the intervals generated by the map [15]. In an ideal map the first pair of partitions generated by the first iteration occur around 0.25 and 0.75 respectively. In a non-ideal map, where $\mu < 1$, after an iteration, it is observed that 0.25 falls short of producing 0.5 in the next iteration, as does 0.75 in the mirroring half. This can be noticed in Fig. 3.4, where the non-ideal iterate of 0.25 (and 0.75) shown in green, fails to meet the 0.5 line through the $Y$-axis. Therefore, the input value of $x$ in the map function $2\mu x$ (correspondingly $2\mu(1–x)$ for $x > 0.5$) is required to be higher (or lower) than the ideal to make up for the reduction in the parameter value. As a result, the partitions get shifted towards the critical point $x_c$, creating the intervals skewed and unequal.

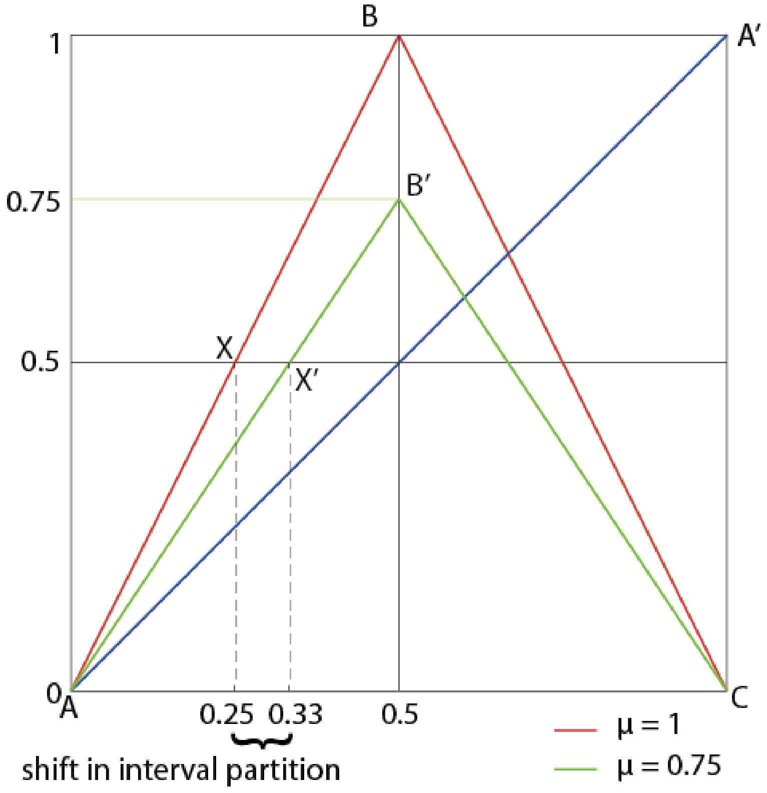


Fig. 3.4 Shift in interval partition due to reduced height

It can be seen from Fig. 3.4 that the ramp input represented by AA’ gets stretched and folded by the operation of the TM function – once using the ideal map ($\mu$ = 1) and once using the non-ideal map ($\mu$ = 0.75) – producing the ABC and AB’C respectively. To illustrate the shift of the partitions, the points where the stretching side of the two maps cross the critical value ($x_c$ = 0.5) are marked with X and X’ respectively. The point X’ can be calculated by reverse calculating the initial condition from the resulting iterate which must equate to the critical point. In this case, this is given by $1 - \frac{x_c}{2\mu x} = 0.33$. As can be seen, from the figure, in case of the non-ideal map, none of the inputs until 0.33 has crossed the 0.5 threshold after a single iteration.

In the process, the initial conditions are redistributed in the sub-intervals, i.e., in case of an ideal map, if a certain initial condition previously belonged to a certain sub-interval identified by a specific symbolic signature, in the non-ideal scenario, the same initial condition might not belong to the same sub-interval and might belong to an interval with a different signature. Thus the non-ideal symbolic sequences do not necessarily represent the same interval visits as the ideal sequences do.

This condition causes the non-ideal symbolic trajectories to be no longer the same as it is with the ideal parameter. Hence the trajectories do not correspond to the actual initial condition when the non-ideal symbolic sequences are converted to their decimal equivalents directly. Since the only relatively reliable outcomes of the system are the symbolic trajectories, in order to work out the initial condition, the information of the parameter of the map is also needed to be taken into account. It is assumed the information of the parameter is available, which is being taken care of through a parallel investigation. However, even the knowledge of the non-ideal parameter is insufficient for determining the initial condition from the symbolic sequence, if the conventional approaches are applied.

In order to determine the initial condition from the non-ideal symbolic trajectories, the dynamics should be traced with respect to the intervals visited by the iterates on every iteration. The information of the system parameter needs to be utilised to determine the measure of each interval visited by the trajectory. As the intervals of the non-ideal map are not equal in size and the positions are skewed, the measure of the non-ideal parameter as well as the symbol of the iterate should be utilised to govern and determine the interval size and thus trace the iterates to determine the initial condition that resulted in the sequence. The position of the partitions as well as the size of the interval for each symbol will aid in repeatedly narrowing down into the interval that should contain the initial condition which resulted in the non-ideal symbolic trajectory. To determine the shift of the partition with every iteration as well as keeping track of the shift with iterations including the folding behaviour of the map, the behaviour of the TM must be understood thoroughly, which is described in detail in the following chapter.

# 4 INITIAL CONDITION ESTIMATION

The ideal TM generates uniquely defined Gray code sequences which can be converted to binary codes first and then into the corresponding decimal values which will correspond to the initial condition that generated the trajectory. As shown for the LM in [27], if the symbolic sequences converted directly to decimal values without converting to the binary codes first, the resulting points when plotted against the initial conditions exhibit the fractal nature of the TM dynamics and is a very useful tool to observe the behaviour of the points in the state space when subjected to the TM iterations. This can be seen from the behaviour of the point 0.4 in Fig. 4.1. It must be noted that, since the point plotted is the decimal equivalent of the Gray code and therefore a direct doubling cannot be observed.

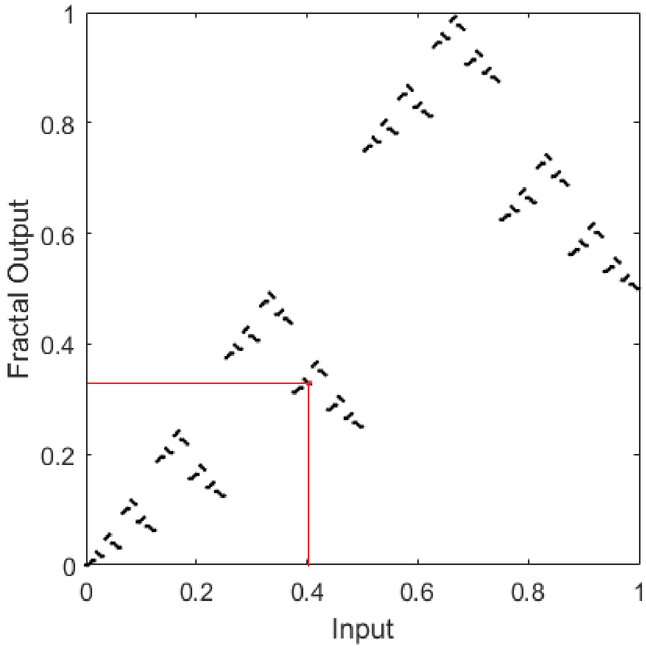


Fig. 4.1 Fractal behaviour of ideal Gray codes for $\mu = 1$

When the same observation is made for non-ideal ($\mu = 0.75$) map outcomes, the plot appears as shown in Fig. 4.2 and the equivalence of the point

0.4 is no longer maintained. The shift in the points introduced by the non-ideal parametric condition results in overlapping of points and drastic remapping that cannot be traced back in a straightforward manner. This is due to the shift of the interval partitions as described in [15] causing unequal and skewed sub-intervals. Therefore, the initial conditions from the state space *I* are redistributed in these sub-intervals. As a result of this phenomenon, the symbolic signatures associated with the specific initial conditions gets changed from the ideal symbolic trajectories. Therefore, converting the symbolic sequences directly into decimal values produces incorrect mappings.

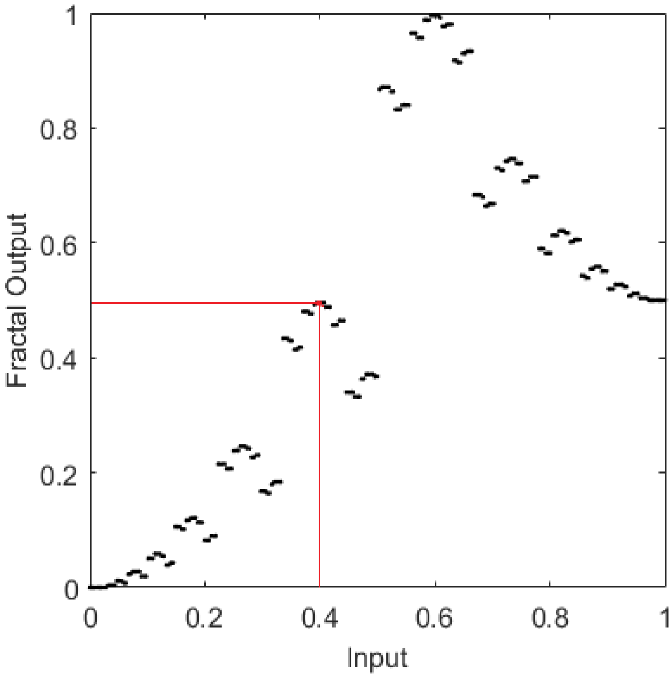


Fig. 4.2 Distortion of the fractal code for $\mu$ = 0.75

It can be seen from Fig. 4.2, however, the symbolic sequences generated by the iterations still maintain the fractal nature of the TM dynamics, even when the parameter value deviates from the ideal. Therefore, it might be possible to utilise the inherent self-similarity to recreate the skewed sub-intervals and identify the correct interval from where the initial condition originated.

Therefore, in order to determine the initial condition only from the symbolic sequence generated by a non-ideal TM, it is important to have a deeper understanding of how the dynamics of the trajectory gets affected with the change of parameter and how the change is reflected by the change in the symbolic sequence thus generated.

## 4.1 Self-Similarity

Initial experiments were conducted by directly converting the non-ideal symbolic sequences into decimal values (GON) and comparing with the initial conditions over a range of parameters (Fig. 4.3).

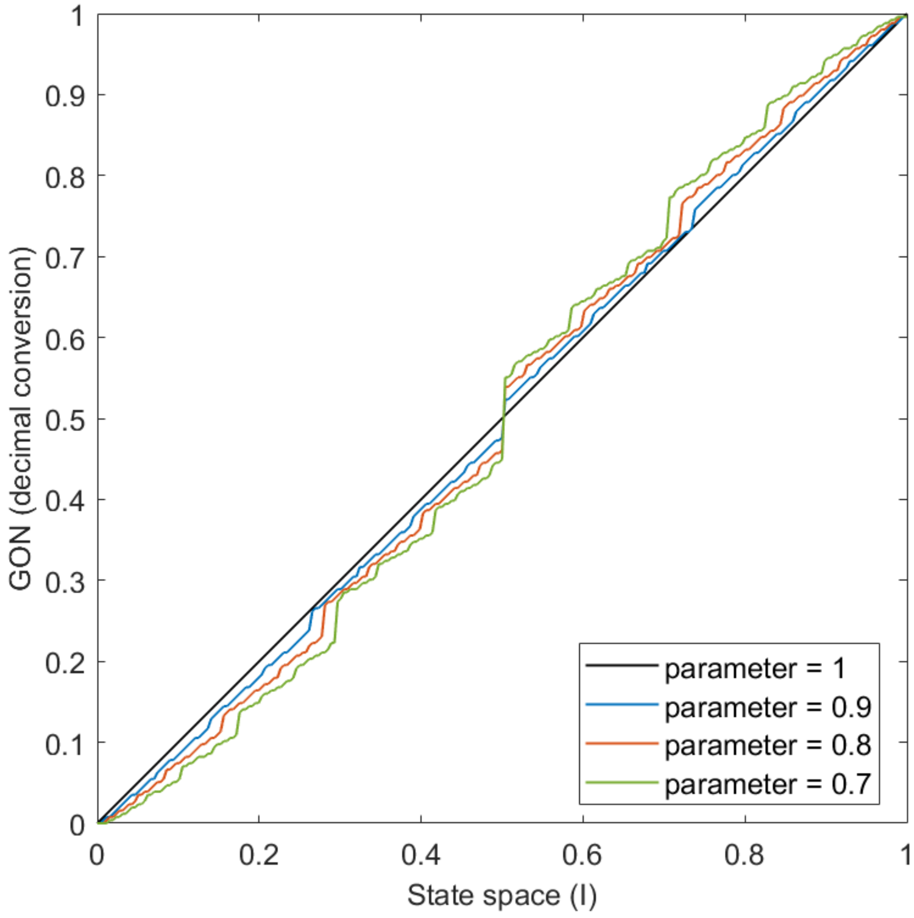


Fig. 4.3 GON values calculated and compared with the ideal case

For most of the initial conditions, therefore, the same initial condition maps to a variety of points depending on the parameter value of the operating function (Fig. 4.4).

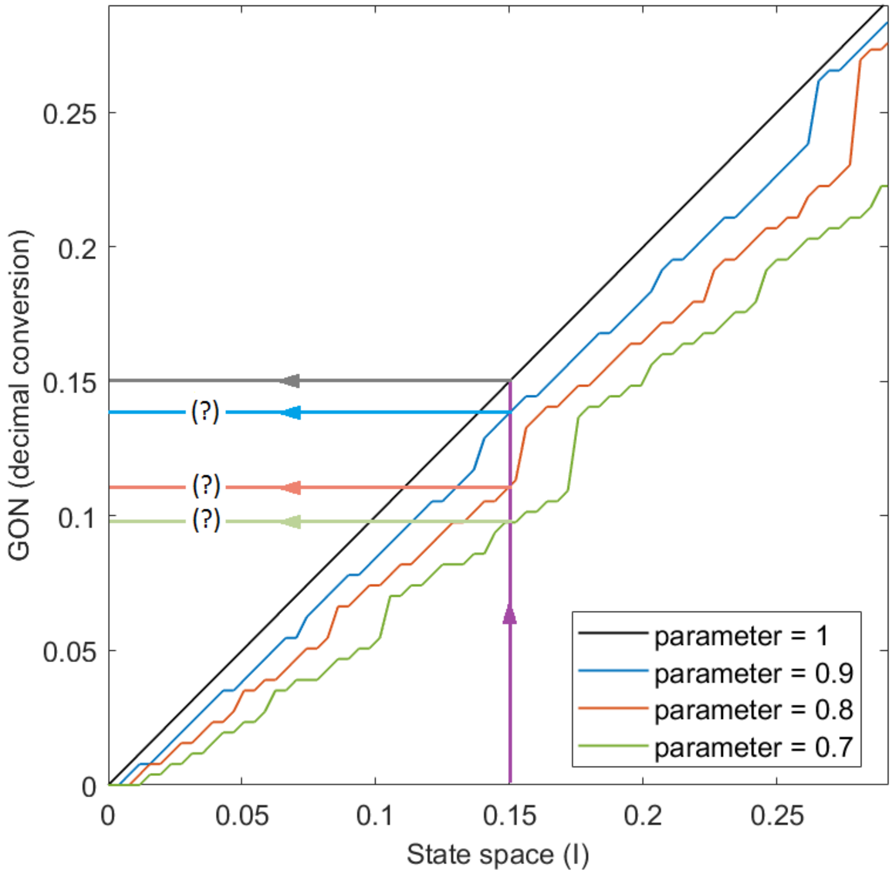


Fig. 4.4 A closer view of the GON values

It was observed that there exists a self-similarity in the GON values (as described in section 2.3) as well as the difference of the outcomes with the initial conditions. It was therefore logical to infer that there must be an overarching rule governing the behaviour of the deviation from the ideal values. As a result, it is evident that the initial condition can be recovered if the difference between the corresponding values for the initial condition in Fig. 4.3 can be made up for. As can be seen from closer observations, the magnitude of deviation from the ideal values is dependent on the parameter values. Once the parameter value is known – which is the objective of a parallel and related research – the initial conditions can be recovered from the GON values. Since the difference shows a fractal behaviour, the underlying governing rule must be recovered. Since the reduced parameter value results in a reduction in the subsequent iterate values as compared to the ideal condition, the difference between the parameter values can be utilised at every step of the iteration to account for the overall deviation of the initial condition. Now, if the initial condition requires to be reverse-calculated from the symbolic sequence, dividing the iterates and occasionally folding the outcomes depending on the symbolic footprint cannot be performed with the non-ideal parameter value since the accurate final iterate is not accessible.

On the other hand, replacing the base of 2 with the reduced parameter value in the GON calculation will not account for the folding behaviour since the symbolic sequence generated is not a true Gray code and will not generate true binary codes. This can be verified from Fig. 4.5, where the outcomes show scaling error from the ideal scenario and cannot be scaled back by $1/2\mu$. However, in the reverse calculation method, the deviation of the reverse-scaling factor can be independently calculated, even without the knowledge of the final iterate value. This difference, when accounted for along with the GON of the initial condition, depending on the symbolic sequence, should recover the initial condition successfully. The problem can be approached by determining how much deviation must be restored with every iteration. Since in a non-ideal map, the iterates gain by a factor of $2\mu$, the reversal should scale down the iterates by a factor of $1/2\mu$. However, as GON is performed with a base of 2, effectively, it results in a reverse-calculation with a scaling factor of 1/2. Therefore, there is a loss of scaling by $1/2\mu-1/2$. Over iterations, the power of the difference is compounded, and the power is increased.

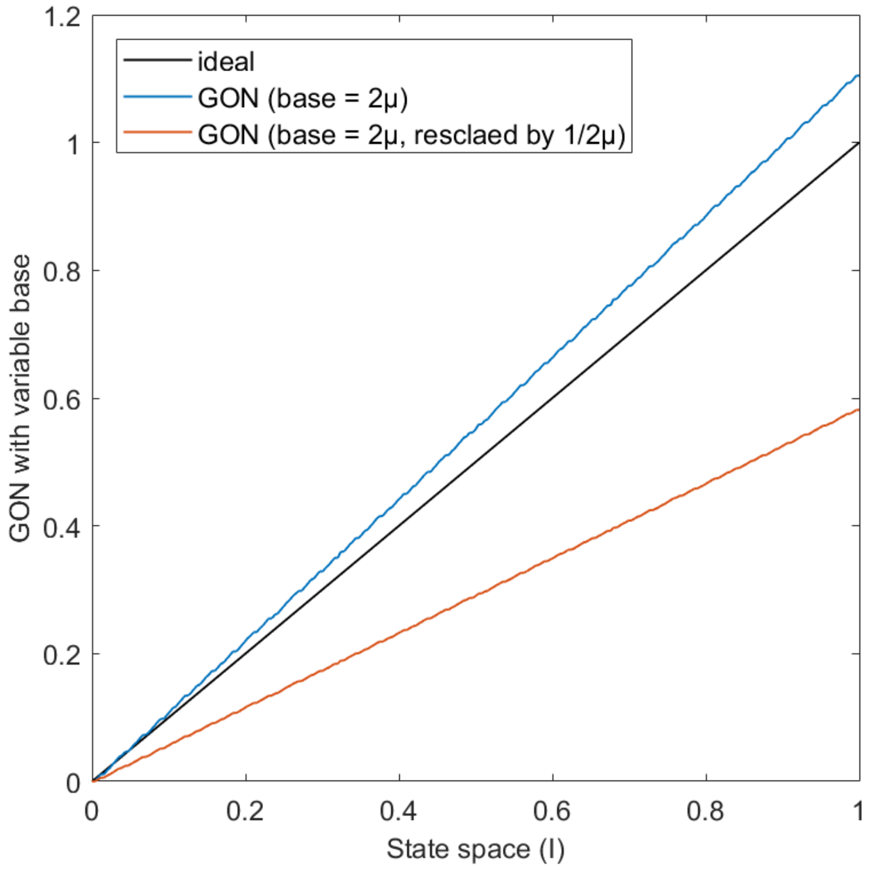


Fig. 4.5 GON calculated with base of $2\mu$ and rescaled by $1/2\mu$

## 4.2 Interval arithmetic

The self-similarity of the TM function was also reflected by the position and the size of the skewness and inequality of the intervals generated by a non-ideal TM. It was observed that the self-similarity had a mirroring property and it always occurred about the critical point of the map ($x_c$). It could therefore be linked to the reversed orientation of the two halves of the map as stated by Gilmore and LeFranc in [6]. Although the self-similar structure could be utilised along with the knowledge of the symbolic trajectory to triangulate an approximate area within the state space where the initial condition might belong, in a real situation the information of the entire state space will not be available and therefore, the information from the self-similar structure could not be accessed from a single symbolic sequence. Therefore, it was important to determine the property that lent the self-similarity to the observation in Fig. 4.6, and whether this property can aid in determining the initial condition through the symbolic sequences.

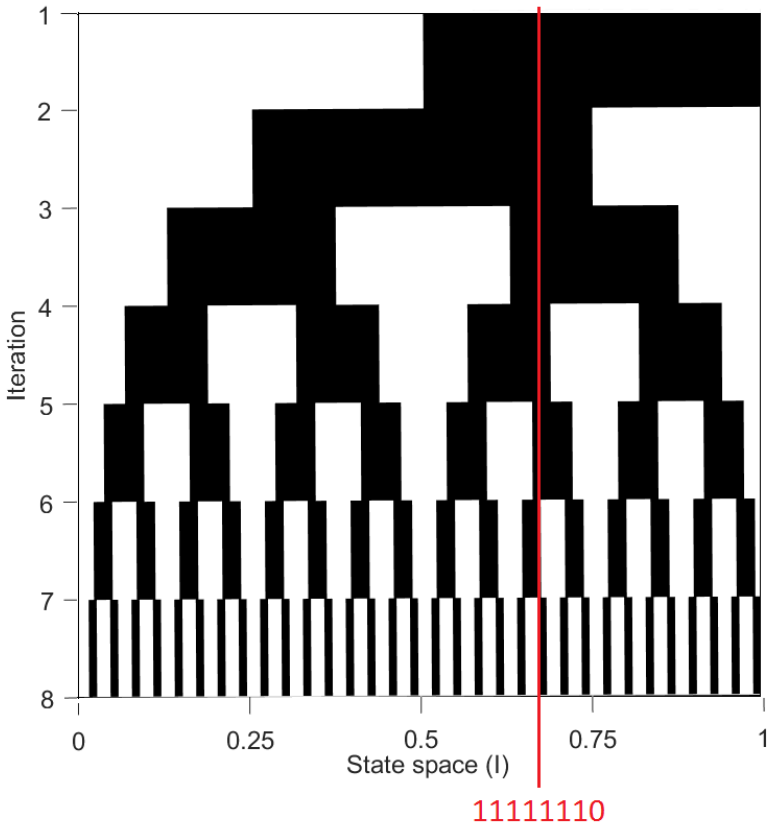


Fig. 4.6 Fractal growth of sub-intervals over iterations

## 4.3 Addressing non-ideal patterns

In order to correlate the underlying pattern or the self-similarity of the skewness of the intervals with the symbolic sequences which result in self-similar errors when converted to decimal values, the interval arithmetic of the map is considered. The interval arithmetic is the technique of observing and analysing the formation of the intervals within the state space, caused by repeated iterations of the map over the state space. In case of the TM the folding nature of the map result in a reflected fractal nature of the formation of the intervals. This is owed to the reversing and preserving nature of the two halves of the map. Thus, each partition created by the iterations result in sub-intervals on either side that also inherit the similar orientations as the original map.

As can be seen in Fig. 4.6, over iterations, each interval is again partitioned into further smaller sub-intervals with mirroring nature. As the newer intervals are produced, the generated intervals can also be uniquely identified as addressed by the symbols generated over iterations. Therefore, as the number of iterations is increased, the sub-intervals also double each time resulting in finer definition, producing higher resolution of the intervals where the initial condition might originate from. For example, for the initial condition through which the line is shown in Fig. 4.6, the originating interval can either be identified by the sequence 1111 if 4 iterations are considered but can be defined as 11111110 with a higher resolution if all the 8 iterations are considered.

In a non-ideal TM, the partitions get shifted due to reduced height (parameter) and thus the intervals are unequal. However, in order to correctly determine the shifted interval, it is important to note how the partitions have shifted and in which direction. It is observed that the rate of the partition shift is directly related to the change in parameter. Also, the direction of the shift is related to the orientation of the map as can be compared with Fig. 4.7.

As the partitions get repositioned, the interval sizes are either reduced or increased and accordingly, initial conditions or the subsequent iterates belonging to specific intervals in case of ideal maps get remapped into different intervals. Therefore, the symbolic sequences are altered. However, keeping a track of the partition shift along with the altered symbolic sequences might lead to partitioning the state space in the right way. If the partitioning is continued for all the symbols in a specific sequence, narrowing down and identifying the correct interval will be possible. This idea has been developed into an algorithm which is described in the next chapter.

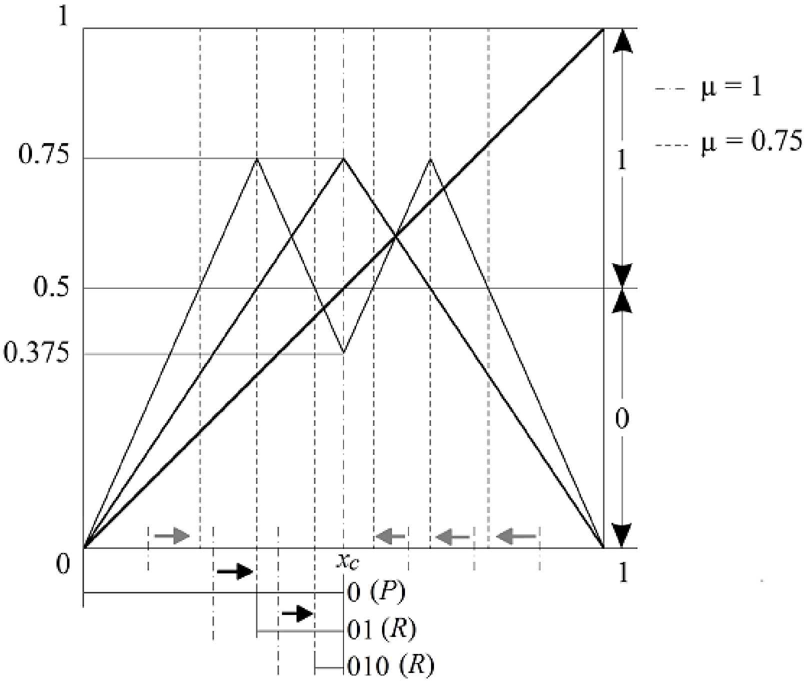


Fig. 4.7 Shift of the partitions towards $x_c$

In a TM, the new partitions are generated on the $n^{th}$ iteration through the points which, on the $n^{th}$ iterate, produce the critical value of 0.5. In case of non-ideal TM, the reduction in the parametric value must be made up for by the iterate values. As a result, for the orientation preserved side, the partitions are generated for a higher value compared to the ideal scenario. Similarly, in the orientation reversed side, due to the folding nature of the map, the partitions are generated through the points at values lower than the ideal case. Therefore, as can be seen from Fig. 4.7 the partitions are shifted towards the critical point of the map ($x_c$). As for the amount of the shift, since the partitions are generated on the $n^{\text{th}}$ iterate of the map, the reduction in parameter must be factored into the magnitude of the shift. In fact, for the $n^{\text{th}}$ iteration, the shift in partition is also produced by the $n$th power of the reduction in the parameter. Clearly, when the non-ideal parameter is known, if the orientations of the symbolic iterates are known the shifted partitions can be reconstructed. Therefore, the intervals can be reconstructed using the symbolic sequences for the initial conditions, which are redistributed in the skewed intervals of the non-ideal map.

## 4.4 Orientation of the interval arithmetic

For any initial condition $x_0$ that generated a symbolic sequence $\mathcal{S}_{n+1}$, the symbolic sequence identifies an interval $I^n_{\mathcal{S}_{n+1}}$ such that $x_0 \in I^n_{\mathcal{S}_{n+1}}$, which is described in section 2.3. Moreover, every $i^{\text{th}}$ symbol in $\mathcal{S}_{n+1}$ also indicates whether $x_i$ belongs to the left or right of $x_c$, i.e., to $I^0_0$ or $I^0_1$. As a result, $s(x_i) \in \{0,1\}$. This is true for every $x_i$. If $x_i = T^i(x_0) \in I^0_{s(x_i)}$, the inverse relation returns $x_0 \in \mathrm{T}^{-\mathrm{i}}(I^0_{s(x_i)})$. Since this operation can be performed for all bits in the sequence, for an $n$+1-bit sequence, this relationship can be combined for every $x_i$. Hence, the originating interval $I^n_{\mathcal{S}_{n+1}}$can be defined as

$$I^n_{\mathcal{S}_{n+1}} \equiv \bigcap_{i=0}^{n} T^{-i}(I^i_{s(x_i)}). \tag{4.1}$$

For instance, if $\mathcal{S}_{n+1} = 010...s(x_n)$, $s(x_0)$ is considered, the initial condition can be identified by $x_0 \in I^0_0$. After one iteration, the iterate $T(x_0) \in I^0_1$, which can also be expressed as $x_0 \in T^{-1}(I^0_1)$. Therefore, considering $s(x_0)s(x_1)$, $x_0 \in I^0_0 \cap T$

$^{-1}(I_1^0) \equiv I_{01}^1 \subset I_0^0$ [17,27]. In this manner, following all the symbols in the sequence, the originating sub-interval can be identified as

$$x_0 \in I_0^0 \cap T^{-1}\big(I_1^0 \cap T^{-1}(I_0^0 \ldots)\big) \equiv \ldots \subset I_{010}^2 \subset I_{01}^1 \subset I_0^0. \qquad (4.2)$$

As the TM is directly non-invertible (every point has two inverses), if the inverse operation $T^{-1}$ of the tent map function on an interval is performed, there is more than one choice for the restriction for $T^{-1}$. However, there is a way to determine this factor as it depends on the “orientation” of the map on the sub-interval [19]. *Orientation* of an interval is determined by the tendency of the function in that interval, which is dictated by the slope of the function. For a TM, a positive slope implies an orientation-preserving interval and a negative slope implies an orientation-reversing interval. Unlike a BM, this reversal of the orientation results in mirroring of the behaviour of the map and hence, generating Gray codes instead of binary codes. This is referred to as the reversal of the lexicographic order of the symbolic signature. Therefore, the orientation of the interval $I_{\mathcal{S}_{i+1}}^i$ can be determined from the sequence $\mathcal{S}_{i+1}$ associated with the *i*th iterate.

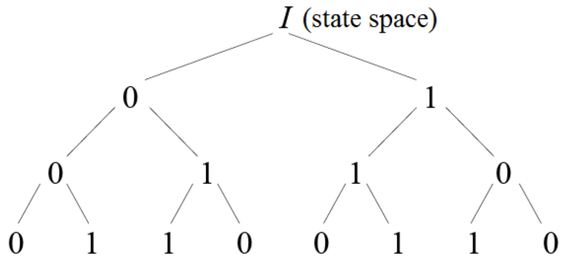


Fig. 4.8 Generating Gray code over iterations

From Fig. 4.8, it can be seen that the orientation of $I_{\mathcal{S}_{i+1}}^i$ gets reversed from how it was, every time a 1 is encountered in the trajectory. Hence, it can be deduced that two successive reversals result in restoring a reversed interval into a preserving one. Building up from here, it can be said that an even number of reversals can result in the same orientation. Hence, up to the $i^{th}$ iteration, occurrence of the orientation-reversing iteration for an even number of times restores the orientation of $I^{i}_{\mathcal{S}_{i+1}}$, while an odd count of the same behaviour results in a reversal. Since orientation-reversing iteration generates the symbol '1', the orientation of the interval $I^{i}_{\mathcal{S}_{i+1}}$ can be determined by checking whether $\alpha_i$ is even (preserved) or odd (reversed), where $\alpha_i$ is given by (5.3).

$$\alpha_i = \alpha_{i-1} + s(x_i) \tag{4.3}$$

This knowledge can be utilised to determine the restrictions of the inverse operation $T^{-1}$ of the tent map function can be chosen as

$$I^{i}_{\mathcal{S}_{i+1}} = T^{-1}\left(I^{i-1}_{\mathcal{S}_i}\right) = \begin{cases} \frac{I^{i-1}_{\mathcal{S}_i}}{2\mu} & \alpha\ is\ even \\ 1 - \frac{I^{i-1}_{\mathcal{S}_i}}{2\mu} & \alpha\ is\ odd \end{cases}. \tag{4.4}$$

In the following section, it is shown how the measure of shift in partitions is applied to the corresponding sub-intervals according to their orientation, given by each symbolic state in the sequence starting from $s(x_0)$ to $s(x_n)$, so that the originating interval of the initial condition $x_0$ can sharply be narrowed down from the state space $I$.

### 4.4.1 The interval arithmetic method

In a non-ideal TM, the magnitude of inequality of the resulting asymmetric sub-intervals is dependent upon the map parameter $\mu < 1$. Also, partitioning the state space as a nested sub-interval is guided by the orientation of the current sub-interval for determining whether the bigger or the smaller sub-interval needs to be chosen for the next step. Therefore, it must be decided regarding the direction, that, in which the partition of the current state needs to be shifted (from the midpoint of the current sub-interval) for each symbolic iterate in question.

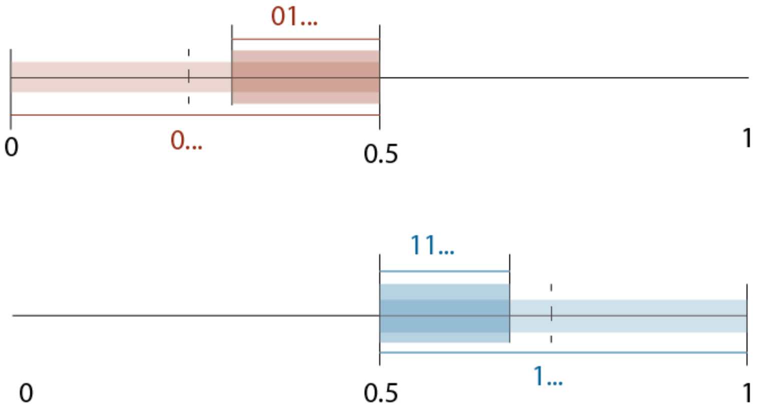


Fig. 4.9 Reconstructing sub-intervals

Here, it is shown how the orientation of $I^{i}_{\mathcal{S}_{i+1}}$ can be used to determine which direction the partition on $I^{i-1}_{\mathcal{S}_i}$ must be shifted to, and which of the two sub-intervals generated contains the originating interval of $x_0$. Using $\mathcal{S}_{n+1}$ = 01…$s(x_n)$ this can be illustrated in the following manner. For $x_0 \in I$, $s(x_0) = 0 \Rightarrow x_0 \in I^0_0$. $s(x_1) = 1 \Rightarrow x_1 \in I^0_1$ and therefore, $x_0 \in I^0_0 \cap T^{-1}(I^0_1) \equiv I^1_{01}$, which lies to the *right* of the newly generated partition as $\alpha_i$ is odd for $I^1_{01}$ (Fig. 4.9). Similarly, for $\mathcal{S}_{n+1}$ = 11…$s(x_n)$, despite $s(x_1) = 1$, $\alpha_i$ is even for $I^1_{11}$ and therefore $I^1_{11} \ni x_0$ lies to the *left* of the newly generated partition. Continuing for $n$+1 symbols, the originating interval $I^n_{\mathcal{S}_{n+1}} \ni x_0$ can be obtained [28].

A technique has been formulated (Fig. 4.10) to be able to measure signal value from the symbolic sequence generated by the TM, in the form of a computational algorithm. For easy adaptability in the digital processing domain, the task of partitioning the state space and the determining the next sub-interval of choice, has been adapted into a simple numerical exercise. The key essence is to shift one of the boundaries of the interval $I_{\mathcal{S}_i}^{i-1}$ towards the other, depending on the orientation of the resulting sub-interval $I_{\mathcal{S}_{i+1}}^{i} \ni x_0$, by a factor of $\mu$, in such a way that the sub-interval that does not contain $x_0$ is eliminated, leaving the correct $I_{\mathcal{S}_{i+1}}^{i}$ behind thereby leading to the originating interval of $x_0$ on the $n^{th}$ step.

For any given sequence $\mathcal{S}_{n+1}$, $s(x_0)$ is determined by $T^0(x_0)$ i.e. without having the initial condition iterated through the map function. This is because the critical point $x_c$ determines which half of the state space $I$ the point belongs to, and thus which symbol must be assigned to it. Hence the role of the first symbol $s(x_0)$ is simply to determine whether the algorithm must be performed on $I_0^0$ or $I_1^0$. $I_1^0$ being a mirror image of $I_0^0$ about $x_c = 0.5$, for any two symbolic sequences that are identical, except for the first symbol $s(x_0)$, their originating intervals are also mirror images of each other exactly about $x_c$. Therefore, for reducing computational complexity, the calculations for the symbolic sequence beginning with $s(x_0) = 1$ is performed on the preserving sub-interval and later amended for the reverse orientation.

From $s(x_1)$ onwards, the boundaries of the $i$–$1^{th}$ sub-interval $I_{\mathcal{S}_i}^{i-1}$ are denoted as A($i$–1) and B($i$–1). Therefore, the corresponding length of the sub-interval is given by $\ell(i-1) = \mathrm{B}(i-1) - \mathrm{A}(i-1)$ and $\delta(i-1) = \ell(i-1)/2\mu$ determines by how much one boundary needs to be shifted towards the other for creating the $i^{th}$ sub-interval. The algorithm [47] can be summarised into a flow chart as shown in Fig. 4.10.

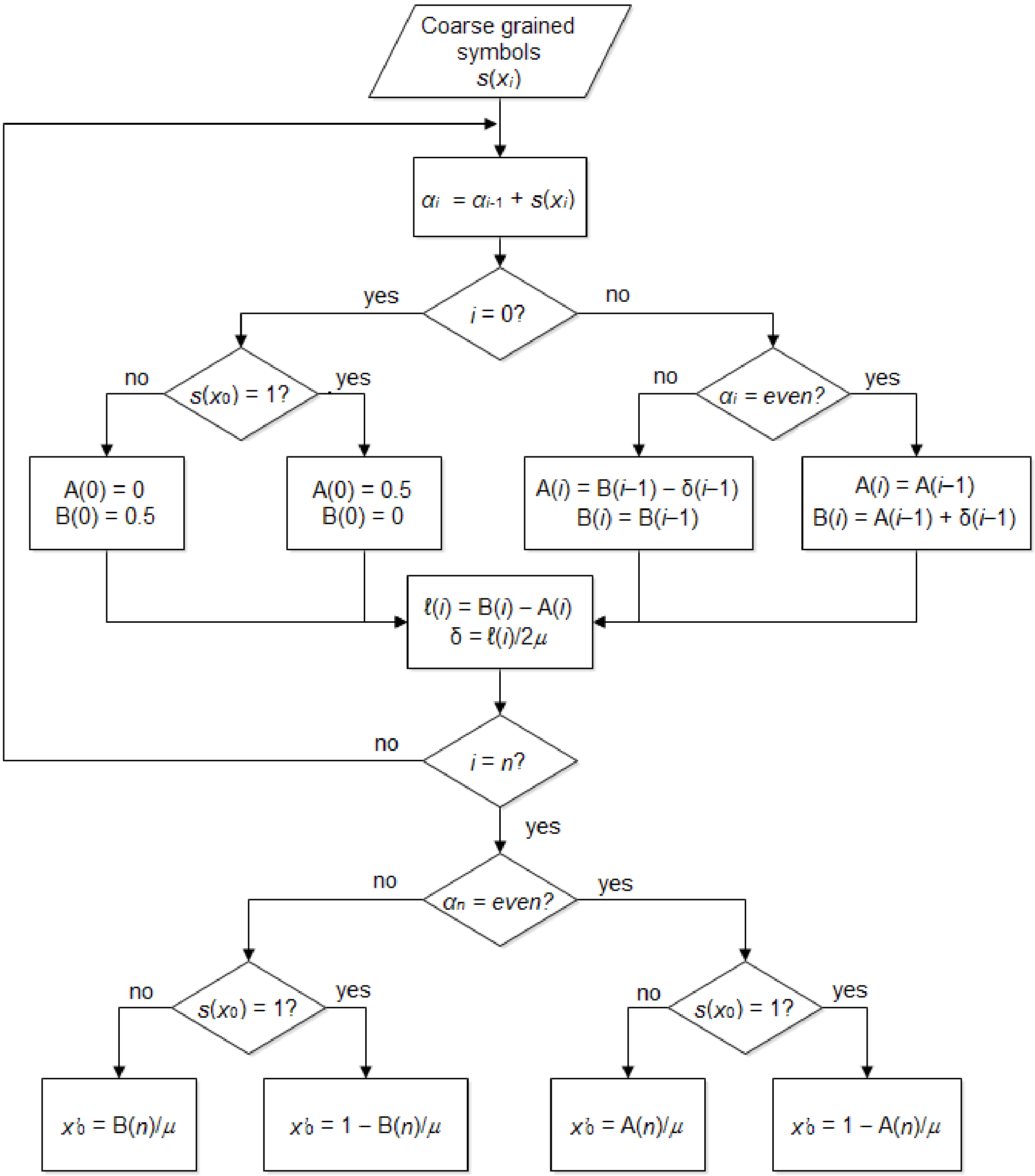


Fig. 4.10 Flow chart for the interval arithmetic algorithm [28]

Executing the steps in the flow chart evaluate the initial condition. The procedure is as follows:

1. For the interval $I_0^0$, i.e. for $T^0(x_0)$, the boundaries are referred to as A(0) = 0 and B(0) = 0.5 and ℓ(0) = B(0) – A(0) = 0.5 – 0 = 0.5. Similarly, by the previous proposition, the boundaries for $I_1^0$ are A(0) = 0.5 and B(0) = 0 and ℓ(0) = B(0) – A(0) = 0 – 0.5 = – 0.5. The negative value of the length is taken care of by the orientation of the symbols in the sequence.

2. From $s(x_1)$ onwards, the following step is repeated until $s(x_n)$. For $i = 1, 2, \ldots, n–1$:

   – $\alpha_i$ is even, $A(i) = A(i–1)$ and $B(i) = A(i–1) + \delta(i–1)$

   – $\alpha_i$ is odd, $A(i) = B(i–1) – \delta(i–1)$ and $B(i) = B(i–1)$

3. When the operation is performed with $\mu < 1$, the estimated initial condition $x_0'$ is scaled by a factor of $\mu$, resulting in $x_0' \in [0, \mu]$ which needs to be scaled back into $x_0' \in I = [0,1]$. Also, if $\mathcal{S}_{n+1}$ had $s(x_0) = 1$, the final sub-interval needs to be mirrored back into $I_1^0 = [0.5,1]$. Depending on the orientation of the sub-interval $I_{\mathcal{S}_{n+1}}^n$ of the nth iteration, keeping the conditions in mind, there are four cases for determining the initial condition $x_0 \in I_{\mathcal{S}_{n+1}}^n$:

   – If $\alpha_n$ is even and $s(x_0) = 0$, $x_0' = A(n)/\mu$

   – If $\alpha_n$ is odd and $s(x_0) = 0$, $x_0' = B(n)/\mu$

   – If $\alpha_n$ is even and $s(x_0) = 1$, $x_0' = 1 – [A(n)/\mu]$

   – If $\alpha_n$ is odd and $s(x_0) = 1$, $x_0' = 1 – [B(n)/\mu]$

This algorithm is tested in the chapter 6 in both simulation as well as real circuit results.

## 4.5 Deviation adjustment for the GON

Another method has also been proposed to determine the initial condition, based on the cumulative deviation incurred by each iteration of the map. This method also involves utilising the symbolic sequence of the map, by keeping a track of the deviation of the iterates based on the deviation of the parameter from the ideal. As mentioned in section 2.3, the deviation of the initial condition calculated using GON can be accounted for, if a technique is devised to restore the difference. Since the deviation is estimated for individual iterates, the cumulative deviation is finally adjusted for the GON value for the initial condition concerned. Unlike the previously discussed method, however, this method does not require performing the calculations every time, since the difference is estimated for the iteration numbers, without any dependence on the actual iterate values and are stored in a look-up-table (LUT) of bit difference values. Once a symbolic sequence is retrieved for an initial condition, only the GON value must be calculated, while the differences can be looked up from the LUT and applied with relevant signs (whether to be added or subtracted) depending on the symbolic footprint for the specific iterate. Finally, the total deviation can be adjusted at the end with the GON. This has been shown in detail through an example in section 4.5.2. This method may aid in saving sufficient computational power through occasional updates of the LUT which would be required only with a noticeable shift in parameter.

### 4.5.1 The deviation adjustment method

The problem is approached by trying to calculate how much deviation needs to be restored with every iteration while reverse-calculating the generated symbolic sequence. In order to perform inverse TM to go back to the initial condition, with each step, the reversal should be performed by scaling down by the amount of the multiplying factor of the TM function, $2\mu$. For non-ideal maps, this should have been by $1/(2\mu)$ but the direct conversion effectively results in 1/2. Thus, this difference of $(2\mu)^{-1}$-$2^{-1}$ must be accounted for, when $\mu$

is known. Accordingly, the next iteration results in a deviation of $(2\mu)^{-2}$-$2^{-2}$. Continuing in this manner, the deviation can be accounted for until the first symbol, and therefore can be added on to the decimal value calculated using basic conversion technique. Due to the reversing and preserving nature of the two halves of the map, there are two possibilities through which the map can be reversed back. This leads to four possibilities in how the deviations can be accounted for, when addressing them. This is easily addressed by considering the preceding and current symbol for each iteration.

The algorithm is executed on the Gray code symbolic sequence $\mathcal{S}_{n+1}(T,x)$ = $s(x_0)s(x_1)s(x_2)\ldots s(x_n)$ generated by the map. The steps are as follows:

1. Starting from the LSB of the sequence, i.e., $s(x_n)$, going towards the MSB $s(x_0)$, all the bits preceeding the first '1' are ignored, as those do not have any influence on the final difference value.

2. The following bit (i.e. after the first 1), say $s(x_m)$, the first weighted difference is calculated with the exponent equal to 1 (i.e. $(2\mu)^{-1}$-$2^{-1}$) which is denoted by the column A in Fig. 4.11. Depending on whether $s(x_m)$ is '0' or '1', the difference is positive or negative respectively.

3. For the following bits, going towards the MSB, $s(x_m)$ > $s(x_m)$ $s(x_{m-1})$ represents one of the four cases with the conditions as follows, leading to the calculations under the columns B, C, D, etc.(Fig. 4.11):

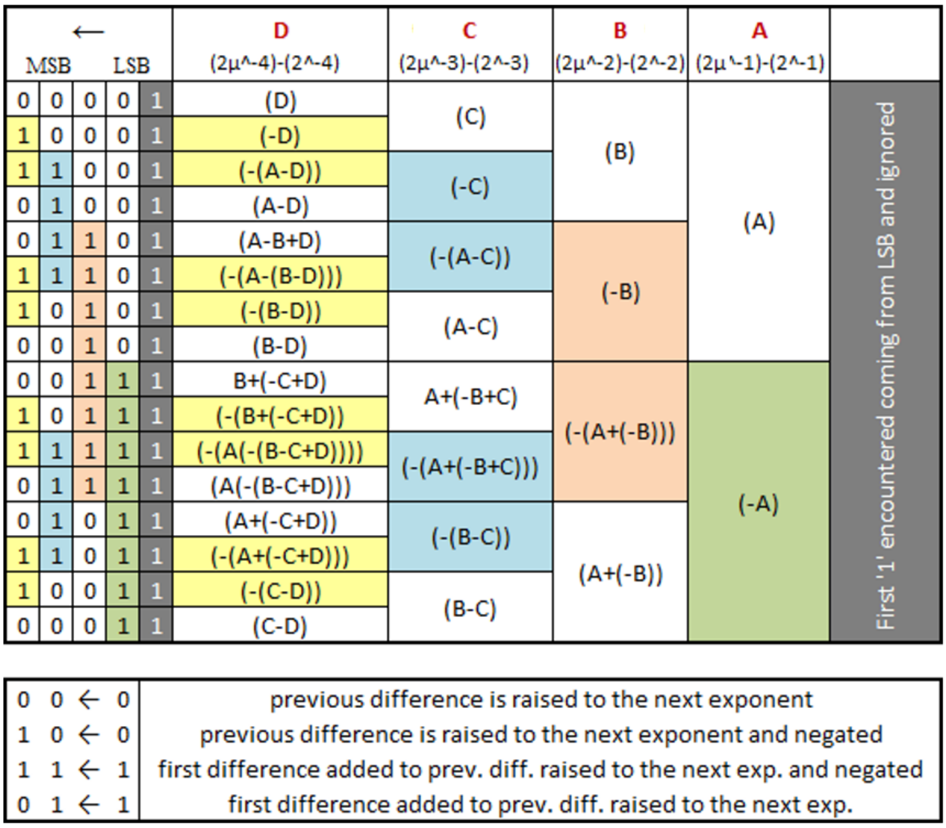
   - 0 > 00: every difference is raised to the next exponent

   - 0 > 01: every difference is raised to the next exponent and the overall sign is inverted

– 1 > 10: every difference is raised to the next exponent, the first difference is added to it and the overall sign is inverted

– 1 > 11: every difference is raised to the next exponent and the first difference is added to it

4. Finally, all the weighted differences are added to the GON value of the corresponding initial condition, in order to make up for the difference.



| ← | | | | | D | C | B | A | |
|---|---|---|---|---|---|---|---|---|---|
| MSB | | | LSB | | (2µ^-4)-(2^-4) | (2µ^-3)-(2^-3) | (2µ^-2)-(2^-2) | (2µ`-1)-(2^-1) | |
| 0 | 0 | 0 | 0 | 1 | (D) | (C) | (B) | (A) | First '1' encountered coming from LSB and ignored |
| 1 | 0 | 0 | 0 | 1 | (-D) | | | | |
| 1 | 1 | 0 | 0 | 1 | (-(A-D)) | (-C) | | | |
| 0 | 1 | 0 | 0 | 1 | (A-D) | | | | |
| 0 | 1 | 1 | 0 | 1 | (A-B+D) | (-(A-C)) | (-B) | | |
| 1 | 1 | 1 | 0 | 1 | (-(A-(B-D))) | | | | |
| 1 | 0 | 1 | 0 | 1 | (-(B-D)) | (A-C) | | | |
| 0 | 0 | 1 | 0 | 1 | (B-D) | | | | |
| 0 | 0 | 1 | 1 | 1 | B+(-C+D) | A+(-B+C) | (-(A+(-B))) | (-A) | |
| 1 | 0 | 1 | 1 | 1 | (-(B+(-C+D)) | | | | |
| 1 | 1 | 1 | 1 | 1 | (-(A(-(B-C+D)))) | (-(A+(-B+C))) | | | |
| 0 | 1 | 1 | 1 | 1 | (A(-(B-C+D))) | | | | |
| 0 | 1 | 0 | 1 | 1 | (A+(-C+D)) | (-(B-C)) | (A+(-B)) | | |
| 1 | 1 | 0 | 1 | 1 | (-(A+(-C+D))) | | | | |
| 1 | 0 | 0 | 1 | 1 | (-(C-D)) | (B-C) | | | |
| 0 | 0 | 0 | 1 | 1 | (C-D) | | | | |

| | |
|---|---|
| 0 0 ← 0 | previous difference is raised to the next exponent |
| 1 0 ← 0 | previous difference is raised to the next exponent and negated |
| 1 1 ← 1 | first difference added to prev. diff. raised to the next exp. and negated |
| 0 1 ← 1 | first difference added to prev. diff. raised to the next exp. |

Fig. 4.11 Tabular representation of deviation adjustment algorithm

### 4.5.2 Details of the calculation

In order to explain the procedure, a TM generated symbolic sequence $\mathcal{S}_{n+1}(T,x) = s(x_0)s(x_1)s(x_2)\dots s(x_n)$ for an initial condition $x_0$ is considered and the corresponding GON is assessed. Because the deviation is calculated in the reverse direction (i.e. from the LSB), the sequence $\mathcal{S}_{n+1}(T,x)$ is inverted and a sequence $\mathcal{S}'_{n+1}(T,x) = s(x_n)s(x_{n-1})s(x_{n-2})\ldots s(x_0)$ is produced. The rules detailed in Fig. 4.11 are followed, for $\mathcal{S}'_{n+1}(T,x)$. The differences (weightings) denoted by A, B, C, D, etc. are generated as shown in Table 4.1 and stored in a register ($W$).

Table 4.1 Calculation of weightings

| | **A** | **B** | **C** | **D** |
|---|---|---|---|---|
| $\boldsymbol{W}$ | $(2\mu)^{-1} - 2^{-1}$ | $(2\mu)^{-2} - 2^{-2}$ | $(2\mu)^{-3} - 2^{-3}$ | $(2\mu)^{-4} - 2^{-4}$ |
| $\boldsymbol{\Psi}$ | 0 or ±1 | 0 or ±1 | 0 or ±1 | 0 or ±1 |
| $\boldsymbol{\Xi}$ | 0 or ±A | 0 or ±B | 0 or ±C | 0 or ±D |

As the weightings produce very small float numbers, while implementing the algorithm, the values are scaled up by a *scaling_factor* of $2^{2n\text{-}1}$ (where $n$ = number of bits in the sequence) and rounded up to an integer value. Applying the above rule chart, an expression is formed. The expression signs are stored in a register ($\Psi$) whose cells correspond to the register W and the elements are either 0 or ±1 (Table 4.1), depending on the rules listed in step 3 of the algorithm. The equation register ($\Xi$) is formed by element-wise (*Hadamard product*: $\odot$) multiplying $W$ with $\Psi$, i.e. $\Xi = W \odot \Psi$.

Then sum of all the elements in the expression $\Xi$ is solved to determine the magnitude of deviation, determining the deviation of the GON for $\mathcal{S}_{n+1}(T,x)$ away from the initial condition that would have been produced by the TM if the parameter was ideal. As the differences have been scaled up for the ease of calculations, therefore the deviation is scaled down by the same scaling factor of $2^{2n\text{-}1}$ before adding to the GON.

To demonstrate with an example, let the entire 8-bit sequence for a particular initial condition x$_0$ = 0.0234375, generated with parameter $\mu$ = 0.95 be given as

| MSB | | | | | | | LSB |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |

Assuming that the parameter is known, the algorithm is executed as follows:

1. GON is calculated by converting the Gray code $\mathcal{S}_{n+1}(T,x) = s(x_0)s(x_1)s(x_2)\ldots s(x_n)$ first into binary: $\mathcal{B}$ : $b_0b_1b_2\ldots b_n$

$$b_i = \begin{cases} s(x_i) & i = 0 \\ b_{i-1} \oplus s(x_i) & i > 0 \end{cases}$$

$$\therefore \mathcal{B} = 0\ 1\ 0\ 0\ 1\ 1\ 0\ 0$$

and then the binary in to GON:

$$\text{GON}(\mathcal{S}_{n+1}) = \sum\nolimits_{i=0}^{n} b_i \times 2^{-(i+1)}$$

$$= \ 2^{-2} + 2^{-5} + 2^{-6} = 0.015625$$

2. The algorithm starts after the first 1 from the LSB towards MSB, so the code is reversed for convenience.

| LSB | | | | | | | MSB |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |

As the operations start after the first 1 closest to the LSB and continue towards the MSB, the two bits are discarded, and the remaining bits are numbered as shown:

| LSB | ⇀ | | | | | | MSB |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| | | (i) | (ii) | (iii) | (iv) | (v) | (vi) |

3. The differences are stored in $W$ acting as the set of weights, marked as A-H:

| **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** |
|---|---|---|---|---|---|---|---|
| $(2\mu)^{-1} - 2^{-1}$ | $(2\mu)^{-2} - 2^{-2}$ | $(2\mu)^{-3} - 2^{-3}$ | $(2\mu)^{-4} - 2^{-4}$ | $(2\mu)^{-5} - 2^{-5}$ | $(2\mu)^{-6} - 2^{-6}$ | $(2\mu)^{-7} - 2^{-7}$ | $(2\mu)^{-8} - 2^{-8}$ |

4. As the “differences” produce very small float numbers, the numbers are scaled up by a factor of $2^{2n-1}$ (where $n = 8$ is number of bits in the sequence) and rounded up to an integer value. This would produce:

| **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** |
|---|---|---|---|---|---|---|---|
| 862 | 885 | 681 | 466 | 299 | 185 | 111 | 65 |

5. The symbols dictate the sign of the differences in the equation, which decide whether the corresponding differences will be added or subtracted (or no operation).

   Therefore, the sign register ($\Psi$) having correspondence with $W$ is created, for storing the signs (±) and initiate it with 0:

| **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

   The array is filled up according to the rules in the diagram, therefore, starting from bit (i):

   (i) The first symbol is 0.

| LSB | | | | | | | MSB |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| | | (i) | (ii) | (iii) | (iv) | (v) | (vi) |

Accordingly, the "difference equation" starts with A = $G^{-1} - 2^{-1}$.

Therefore, the array will be filled up with +1:

| A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
| +1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

(ii) Next is 0 ⇢ 01.

| LSB | | | | | | | MSB |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| | | (i) | (ii) | (iii) | (iv) | (v) | (vi) |

Therefore, the previous "difference" is shifted to the higher order (exponent) by one place (B) which indicates that the sign-register should be shifted to the right, to align with the weight register position B. However, the first position received no new difference as the previous bit was 0.

Also, the overall sign is negated (i.e. the entire register undergoes sign reversal) since the current bit is 1:

| A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
| 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 |

(iii) Next, 1 ⇢ 10

| LSB | | | | | | | MSB |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| | | (i) | (ii) | (iii) | (iv) | (v) | (vi) |

The previous differences are shifted to the next order as before. The first difference is also introduced as the previous bit was 1, causing the first position in the sign register to be filled with +1. The signs are retained (i.e. no change) because the present bit is 0:

| A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
| +1 | 0 | -1 | 0 | 0 | 0 | 0 | 0 |

(iv) Next, 0 ⇢ 01

| LSB | | | | | | | MSB |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| | | (i) | (ii) | (iii) | (iv) | (v) | (vi) |

Therefore, the rule (ii) is followed:

| A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
| 0 | -1 | 0 | +1 | 0 | 0 | 0 | 0 |

(v) Next, 1 ⇢ 11

| LSB | | | | | | | MSB |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| | | (i) | (ii) | (iii) | (iv) | (v) | (vi) |

The previous “differences” are shifted to the next order as before. The first difference is also introduced as the previous bit was 1, causing the first position in the sign register to be filled with +1. The signs are inverted for the entire register because the present bit is 1 as well:

| A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
| -1 | 0 | +1 | 0 | -1 | 0 | 0 | 0 |

(vi) Finally, for 1 ⇢ 10

| LSB | | | | | | | MSB |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| | | (i) | (ii) | (iii) | (iv) | (v) | (vi) |

This is again rule (iii):

| A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
| +1 | -1 | 0 | +1 | 0 | -1 | 0 | 0 |

6. The equation for the entire sequence, up to the MSB can now be created. This is done by multiplying the two registers $W$ and $\Psi$ element wise (Hadamard product: $\odot$) and add the elements in an equation $\Xi$:

$$deviation = \sum \Xi$$

$$deviation = \sum (W \odot \Psi)$$

$$\therefore deviation = +A - B + D - F$$

$$= 862 - 885 + 466 - 185 = 258$$

7. Ideally, the magnitude of the deviation, scaled up (by $2^{15}$) of the GON of $x_0$ from the actual $x_0$ is produced. Therefore, the deviation must be scaled down to the normal range and added it to the GON calculated previously, to produce the estimated initial condition $\tilde{x}_0$, given by

$$\tilde{x}_0 = \text{GON} + \frac{deviation}{2^{15}}$$

$$= 0.015625 + 0.00787353515625 = 0.02349853515625$$

8. Limiting the output precision to 8 bits, by the following operation is performed [*floor* ($\tilde{x}_0 \times 2^8$)]/ $2^8$, thus producing $\tilde{x}_0 = 0.0234375$, which is exactly the value of $x_0$ considered as initial condition to generate the sequence for this case.

As can be noticed, if two codes $\mathcal{P}_4(T,x)$ = 10000 and $\mathcal{Q}_4(T,x)$ = 10001 are considered, for $\mathcal{P}_4(T,x)$, since the first '1' from the LSB is the MSB, the difference is 0 and therefore, the code can be directly converted to the initial condition. This can also be verified from the GON plot in Fig. 4.3 where the GON of the maximum value (represented by ‘1’ followed by all ‘0’s in Gray code) does not deviate from the ideal case. As for $Q_4(T,x)$, since the first ‘1’ from the LSB side is the LSB, all the subsequent symbols approaching the MSB are used for calculating the difference, and is given by ‘−D’. Thus, the differences are generated and the GON values are adjusted accordingly. Finally, the algorithms described in sections 4.4 and 4.5 are tested for performance and are the results are discussed in next chapter.

# 5 RESULTS AND DISCUSSION

The algorithms developed for the initial condition estimation are first tested through simulated results in MATLAB. A tent map circuit was also developed based on the circuit developed by Campos et al. [42] and was implemented for further testing by Dutta which can be found in [28]. The symbolic sequences were recovered for all the initial conditions approximately corresponding to 8 and 16-bit precision values. The aforementioned algorithms were then tested for performance. The results are shown for input conditions with 8-bit precision. All the implemented real circuit results are converted for both 8 and 16-bit precision outcomes.

## 5.1 Simulation results

The algorithms are first tested using all the symbolic sequences generated by non-ideal TM and the effectiveness of both the methods are shown separately in sections 5.1.1 and 5.1.2 for the interval arithmetic algorithm described in 4.4.1 and the deviation adjustment algorithm (4.5.1) respectively. The performance is observed by measuring the deviation of the outcomes from the initial conditions generated by dividing the state space *I* into equally spaced points of 8-bit resolution. The percentage error for various scenarios as well as the bit error is measured, in order to realise the applicability of both the methods in successfully developing an ADC. Both the methods show sufficient coincidence between the initial condition.

### 5.1.1 Results for interval arithmetic algorithm

The interval arithmetic method is performed on a data set of 8-bit resolution and iterated through a map with parameter $\mu = 0.75$. First, the percent error between the converted values and the initial conditions are charted and plotted in Fig. 5.1. The initial conditions were generated for 8-bit resolution and also calculated using a sequence of 8 symbols. As the method is tested by increasing the number of symbols to 12 and 16 bits to test the performance, the percent error improved as can be seen in Fig. 5.2 and Fig. 5.3 respectively.

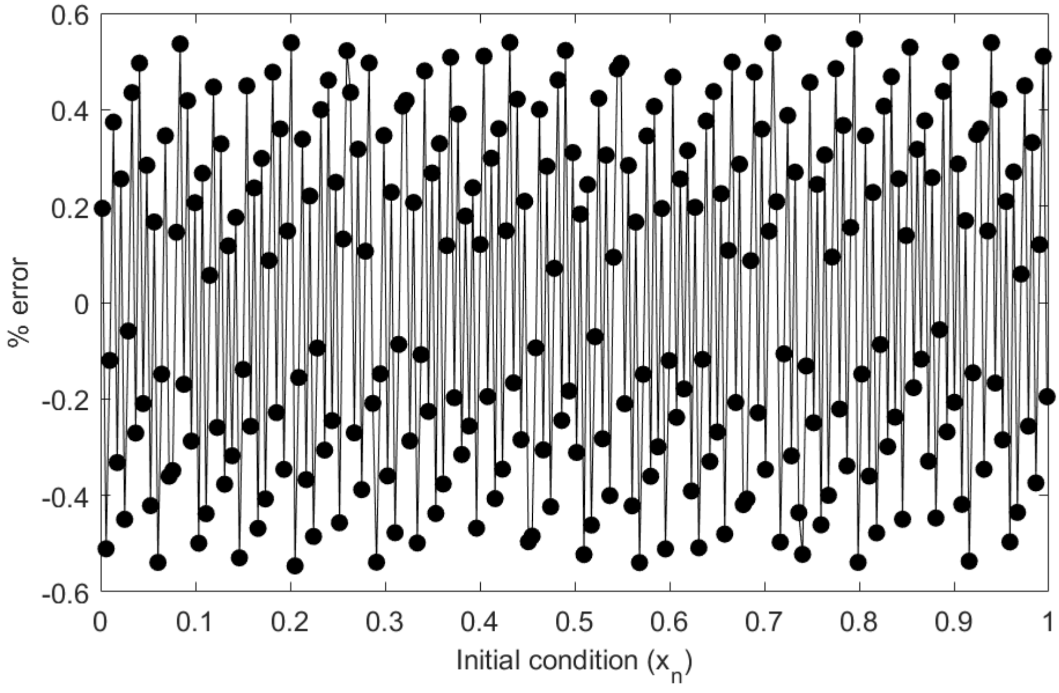


Fig. 5.1 Percent error using 8 symbols

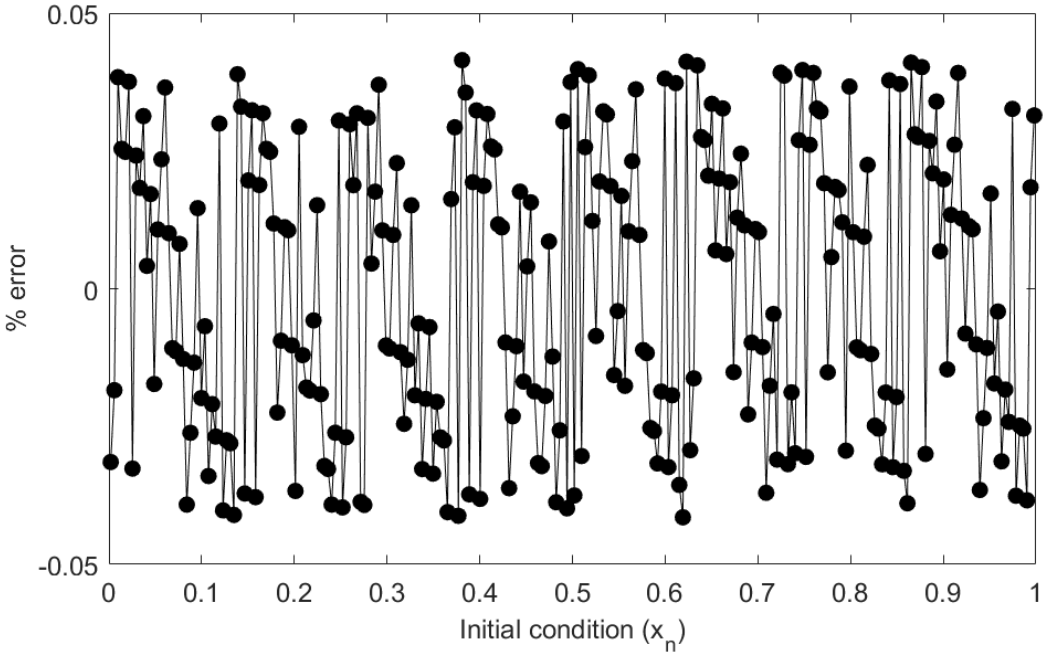


Fig. 5.2 Percent error using 12 symbols

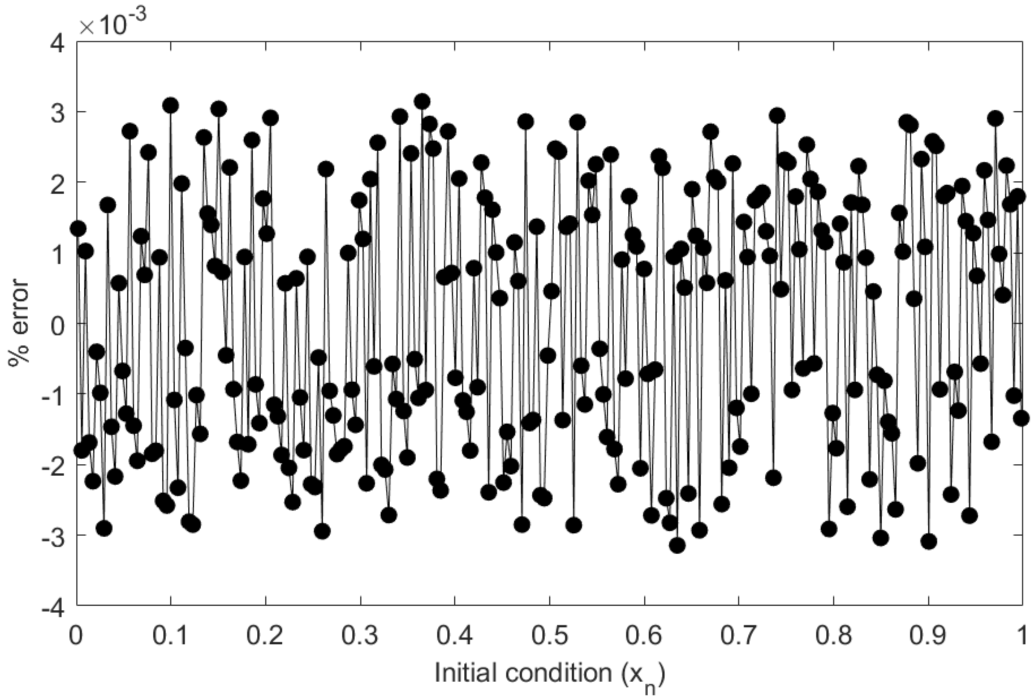


Fig. 5.3 Percent error using 16 symbols

It can be seen that, there is a gradual improvement in the results with increasing the number symbols used for running the algorithm. The error goes from ±0.6% to about ±0.04% and then a little over ±0.003% for 8, 12 and 16 bits, showing a tendency in increasing improvement with the bits.

In order to observe whether this is a valid trend, the method is tested for a range of bit-length of the symbolic sequence used to perform the estimation. It is observed that the maximum absolute error percentage gradually diminished exponentially (Fig. 5.4) as the number of bits used is increased linearly from 1 to 50. To better realise the error beyond 10-bit estimation, the logarithm of the errors are calculated and plotted against the number of bits used in Fig. 5.5. It is also observed that from 10 bits onwards, the spread of the error is condensed and uniform. It must be noted, however, that the presence of noise in a real setting will influence the exponential improvement, which would entail further work to recover the performance.

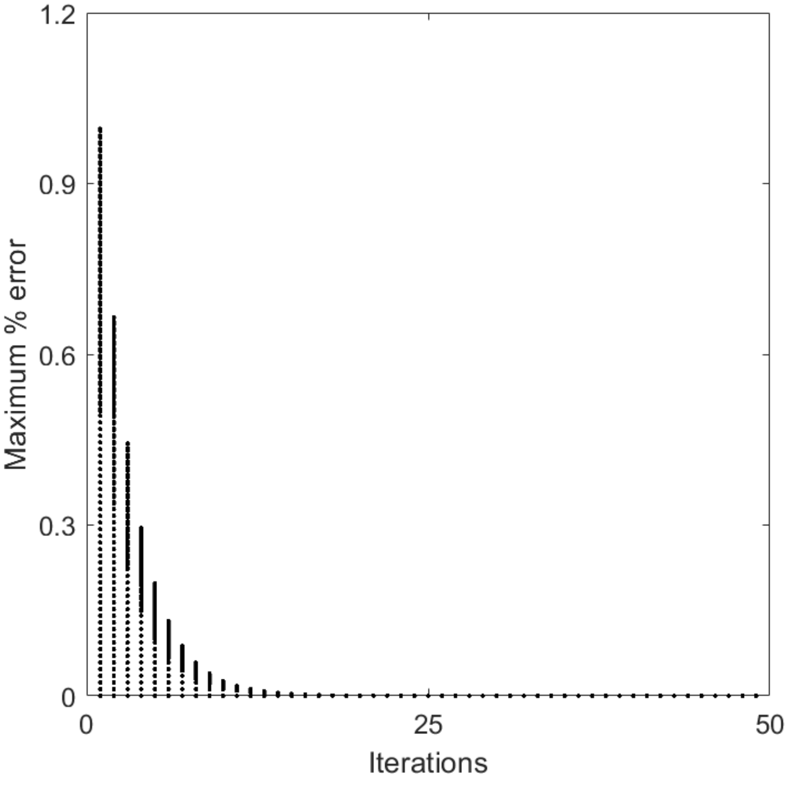


Fig. 5.4 Exponential reduction with increasing symbols

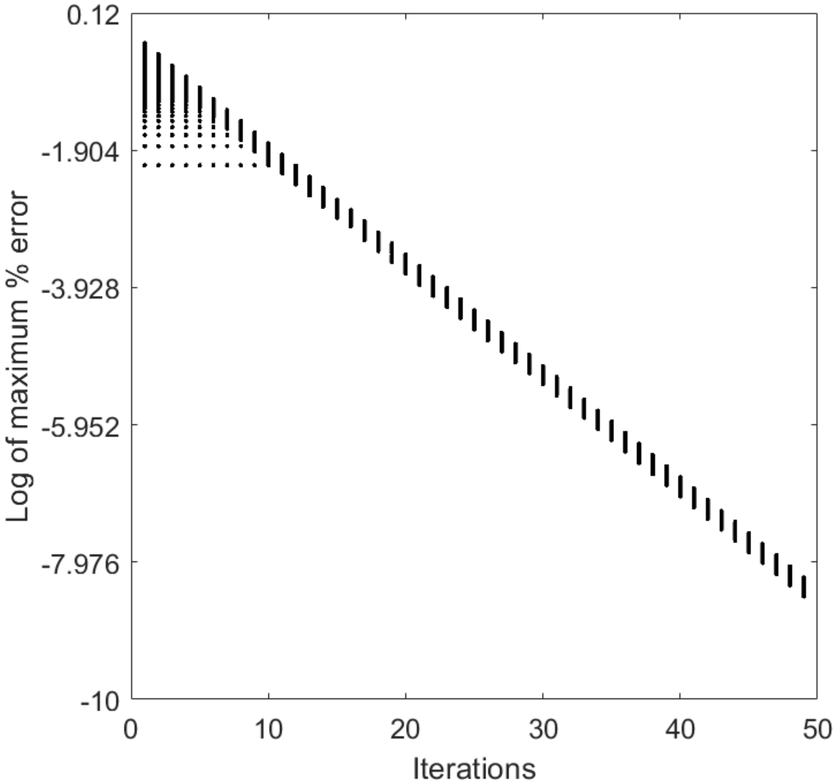


Fig. 5.5 Logarithmic view of the exponential reduction

The error also diminished when compared over a range of parameters as can be seen from the logarithmic value of the absolute errors over a selection of parameters in Fig. 5.6. For parameters 0.75, 085 and 0.95, the logarithmic maximum % error are plotted and can be seen that the errors improve as the parameters approach the ideal parametric value. The improvement also increases as the parameter approaches the ideal value. Thus, the improvement of the case with $\mu$ = 0.95 over $\mu$ = 0.85 is better than that of $\mu$ = 0.85 over $\mu$ = 0.75.

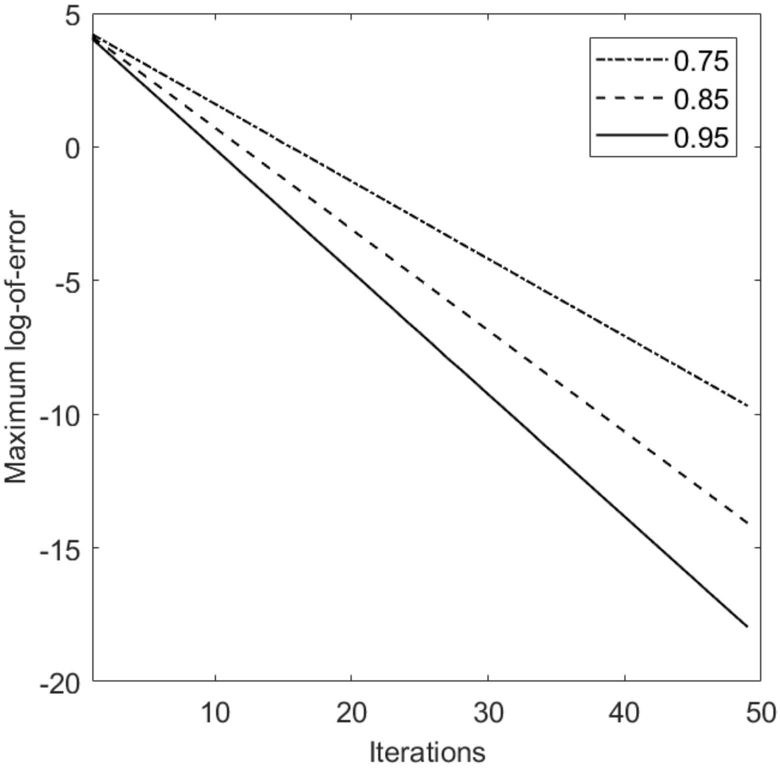


Fig. 5.6 The maximum log-of-error for multiple parameters

From Fig. 5.6, it can be observed that relaxing the parameter value from the ideal does not introduce a drastic error in the recovery and therefore can be afforded while implementing the TM function in a real circuit. Such results encouraged the successful implementation of a physical TM and observe the performances in the later stages, which are also detailed in [28]. As a result, using cheaper components may be a viable option to reduce the cost while not compromising any major setback in the performance.

In order to observe how the algorithm performs over the entire range of chaotic behaviour, the same observation is conducted for a parametric range of $\mu$ = [0.5,1] where the conversion is performed with 8, 16 and 32 bits. The logarithmic maximum % error can be seen in Fig. 5.7 and can be seen that they do not show the linear relationship with the parameter. Increasing the number of symbols for the conversion also shows diminished error over a range of parameters.

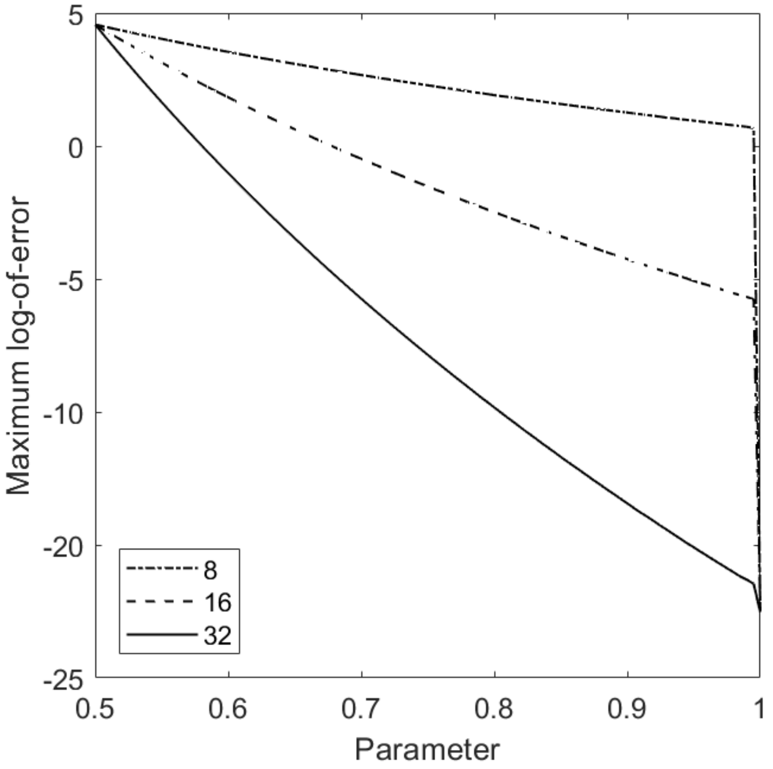


Fig. 5.7 The maximum log-of-error using 8, 16 and 32 symbols

In order to observe how the algorithm performed in terms of bit accuracy, the initial condition is generated by dividing the state space *I* into a set of initial conditions of 8-bit resolution and sorted into a histogram with interval (bin size) of same as the step size of the initial conditions. The input data can be viewed in Fig. 5.8 where it can be seen that the frequency count of each bin is one, i.e. every bin consists of one input data. The data set is iterated through a non-ideal TM whose parameter is estimated to be $\mu$ = 0.9527. The symbolic sequences are generated and the GON is calculated for the initial conditions and a histogram plot is generated as also shown in Fig. 5.8. As can be seen, points have deviated from their actual values and overlapped with some neighbouring points, leaving substantial gaps among the bins.

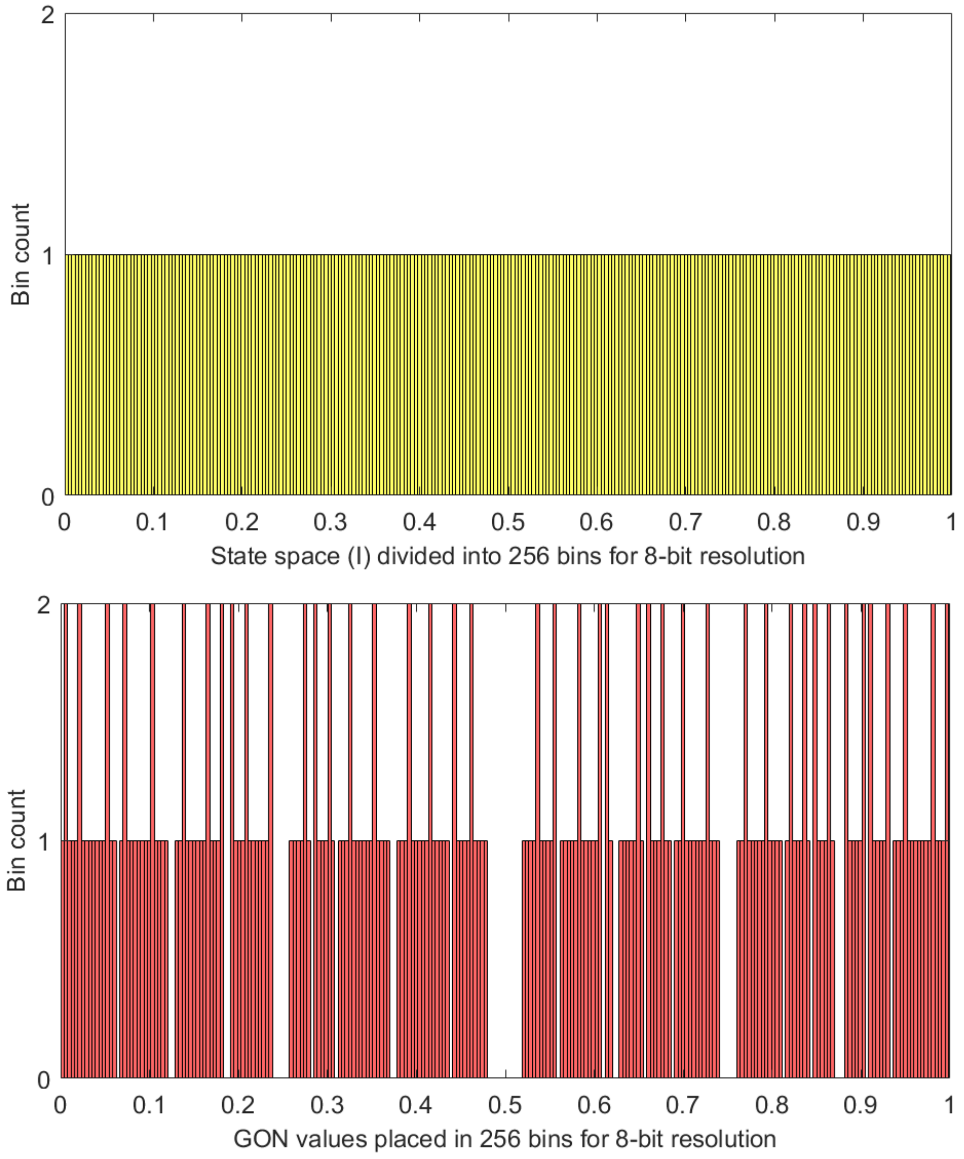
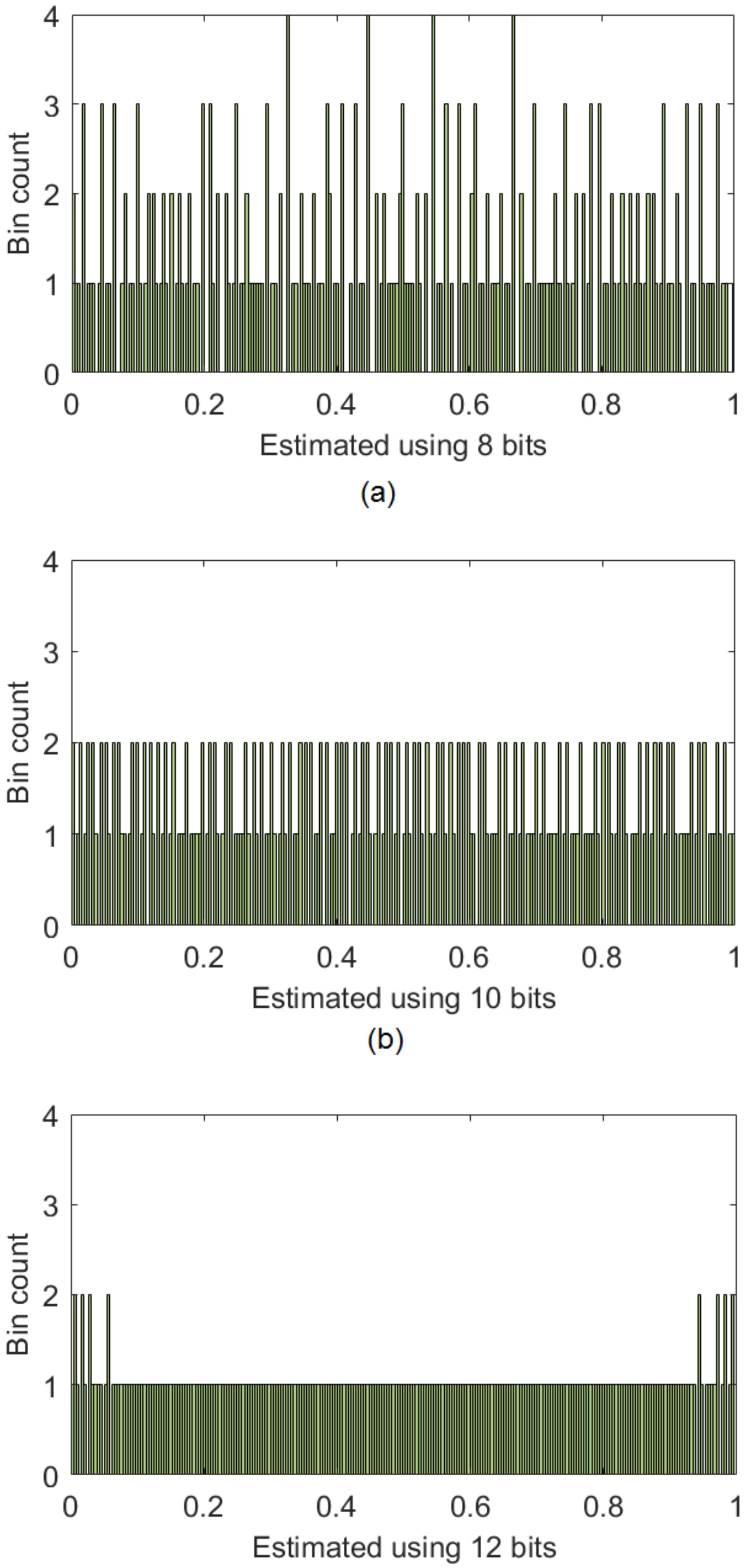


Fig. 5.8 Histogram of the data set and the calculated GON

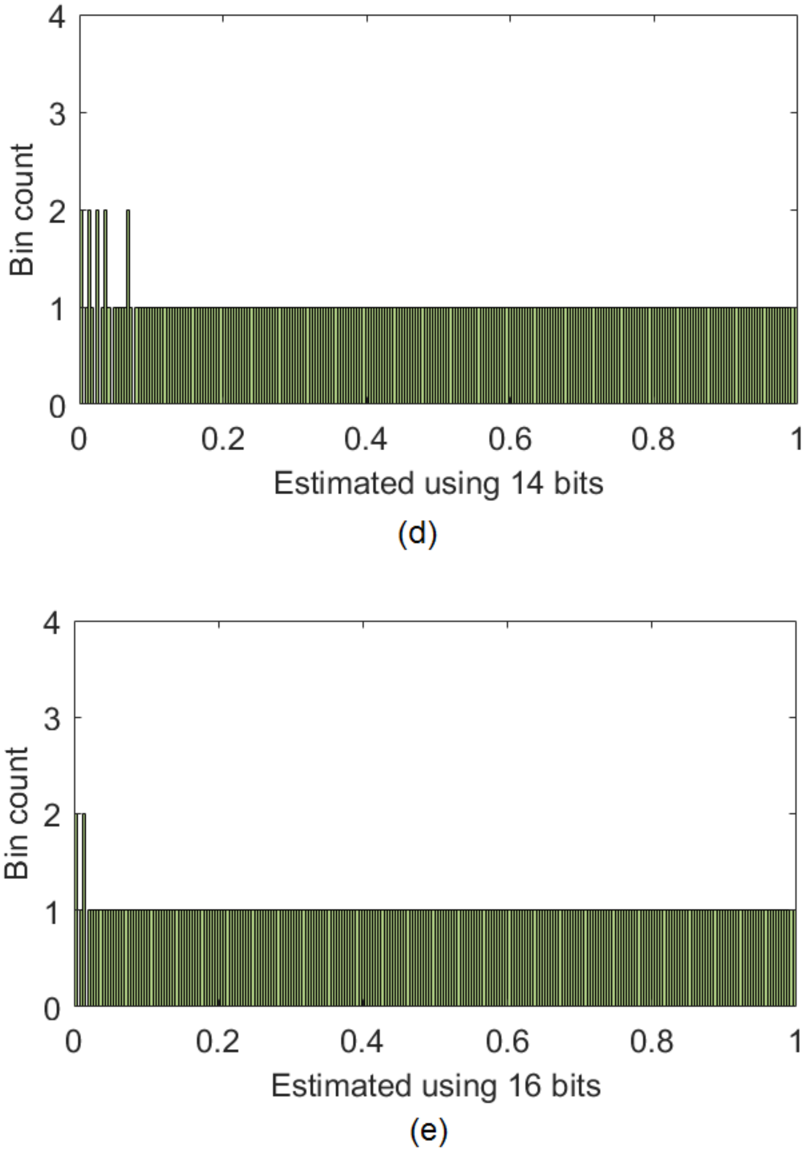
As can be seen from Fig. 5.9, to observe how the estimation method is performing, the algorithm has been carried out for the same set of symbolic signatures but considering varying number of bits; the performance has gradually improved with the increased number of symbols considered.


Bin count
0
1
2
3
4
0
0.2
0.4
0.6
0.8
1
Estimated using 8 bits
Bin count
0
1
2
3
4
0
0.2
0.4
0.6
0.8
1
Estimated using 10 bits
Bin count
0
1
2
3
4
0
0.2
0.4
0.6
0.8
1
Estimated using 12 bits

(a)

(b)

(c)

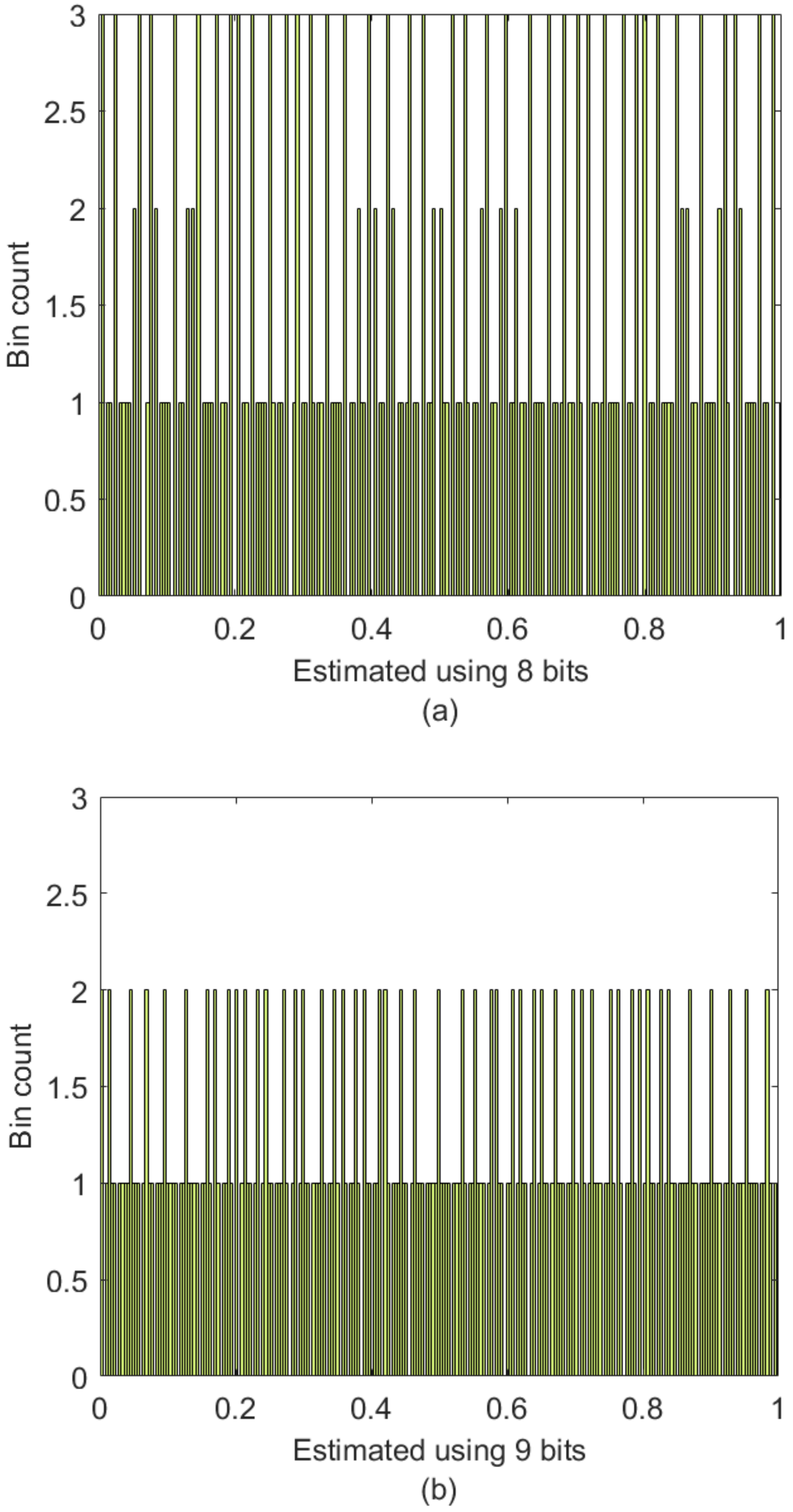


(d)

(e)

Fig. 5.9 Histograms for interval arithmetic method

The algorithm shows promising outcomes when the number of bits considered are increased to 12, and by 16 bits, only two estimated initial conditions are misplaced in the wrong bins. Even after increasing the number of symbols, however, some errors are still observed, especially for the initial conditions with lower magnitude. Since the initial conditions were generated by dividing the state space into 8-bit resolution dataset, the points represent the edges of the bins in the histogram. Since signals are not likely to be exactly on the edges of the intervals, the initial condition is better represented if they are shifted by half a step (Fig. 5.10).



(a)

(b)

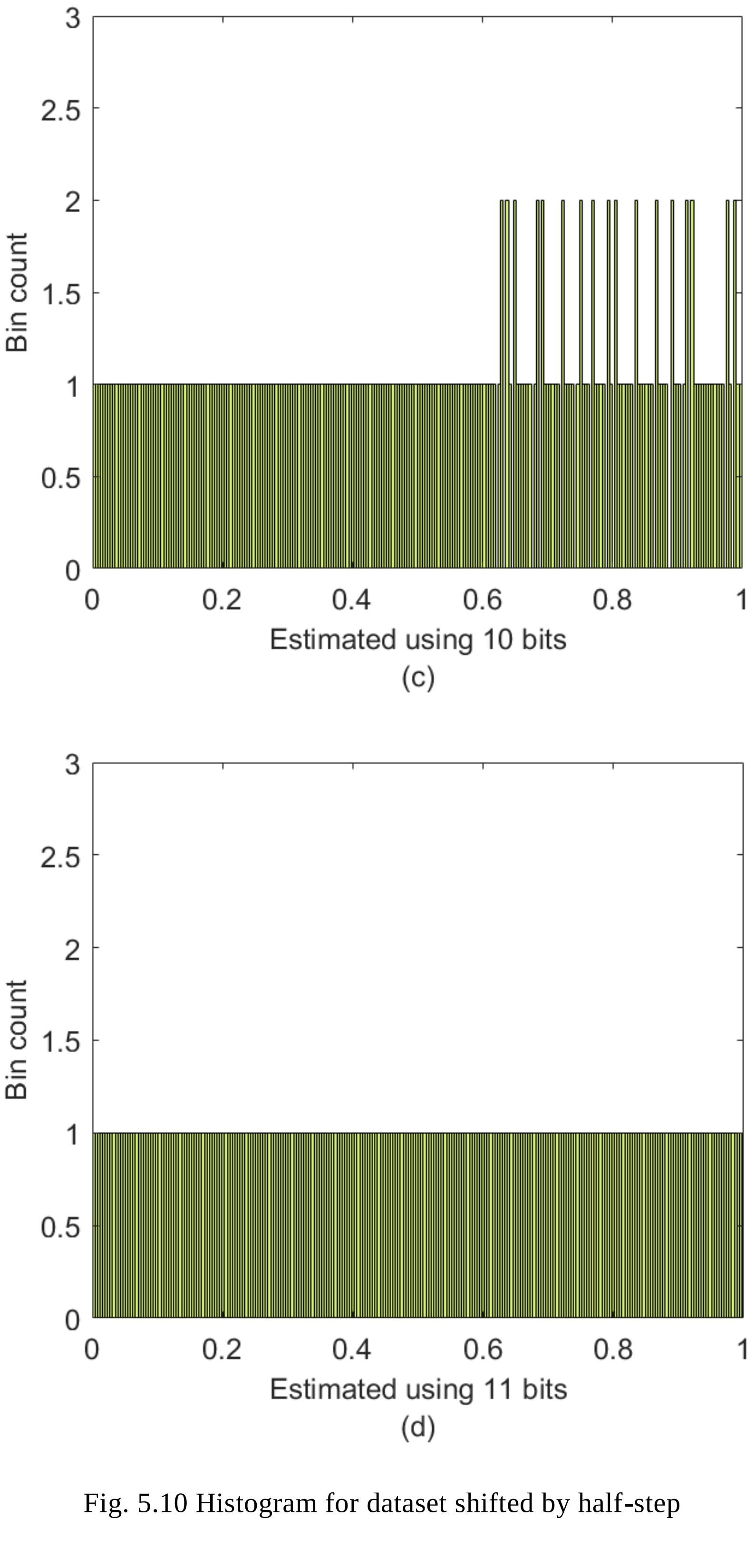


Fig. 5.10 Histogram for dataset shifted by half-step

Using such data points, when the histograms are generated, the outcome is much more improved By the time 11 bits are used, all the estimated initial conditions are uniformly restored to the correct bins. Next, the performance of the deviation adjustment algorithm is tested.

### 5.1.2 Results for deviation adjustment algorithm

The deviation adjustment method shows similar performance as the interval arithmetic algorithm, and the percentage error is calculated between the set of estimated outcomes and the initial condition as shown in Fig. 5.11. The percentage error appears to lie within -0.9% and 1.2% (absolute band of 2.1%), being slightly higher than the interval arithmetic algorithm. By increasing the number of bits in sequence considered for the estimation, the error is seen to be improving. Using 12 bits, Fig. 5.12 is generated, and it can be seen that the error band has improved drastically and lies between -0.22% and -0.12%, i.e. within 0.1% absolute percent error band.

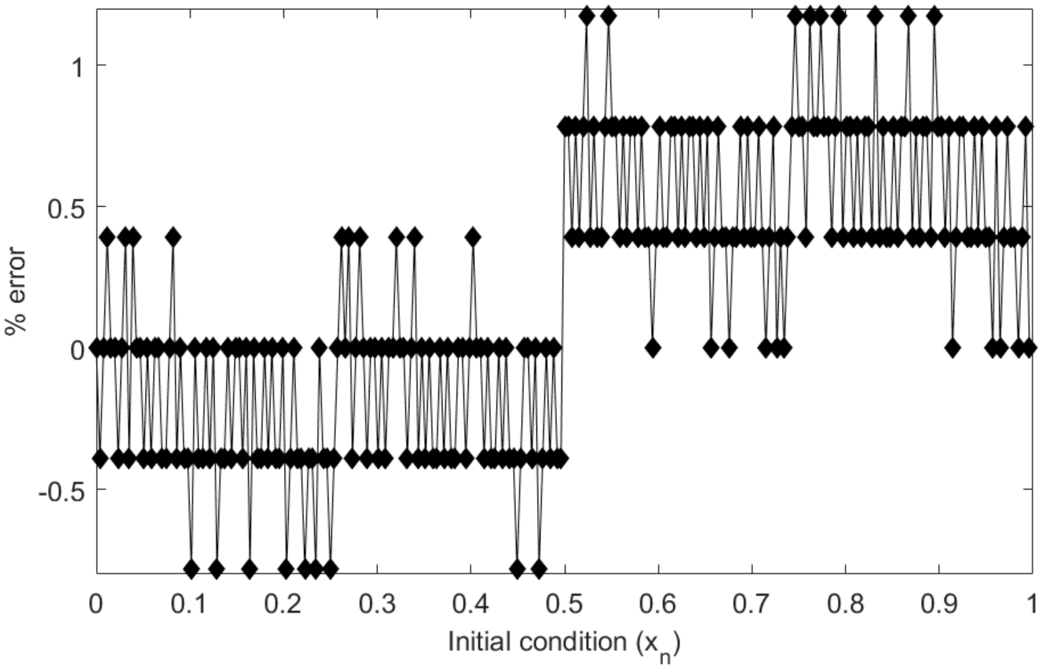


Fig. 5.11 Percent error using 8 symbols

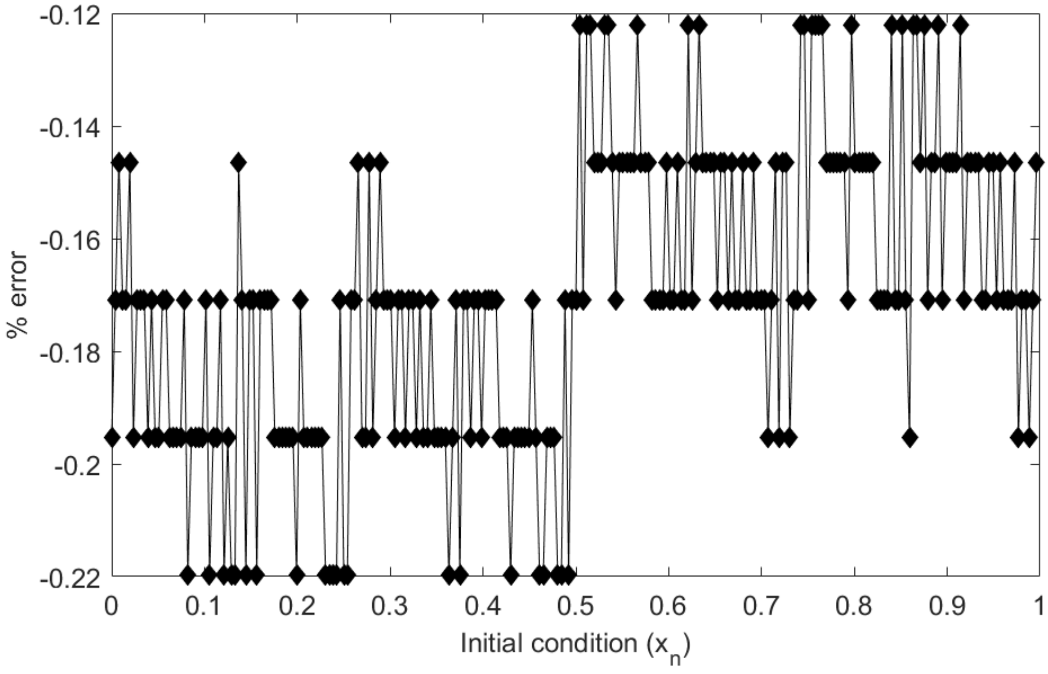


Fig. 5.12 Percent error using 12 symbols

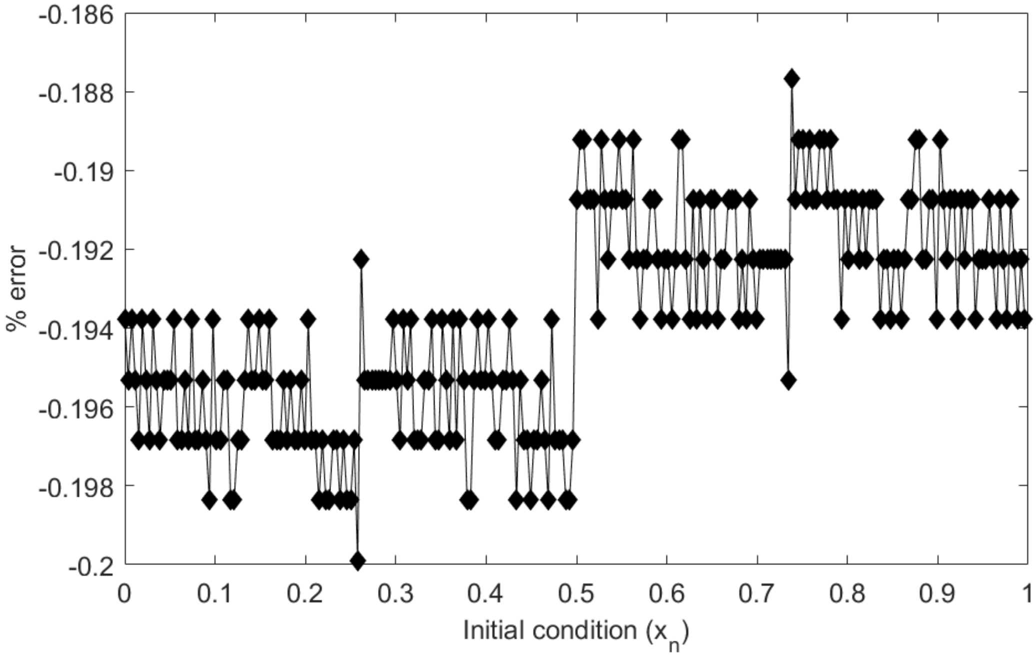


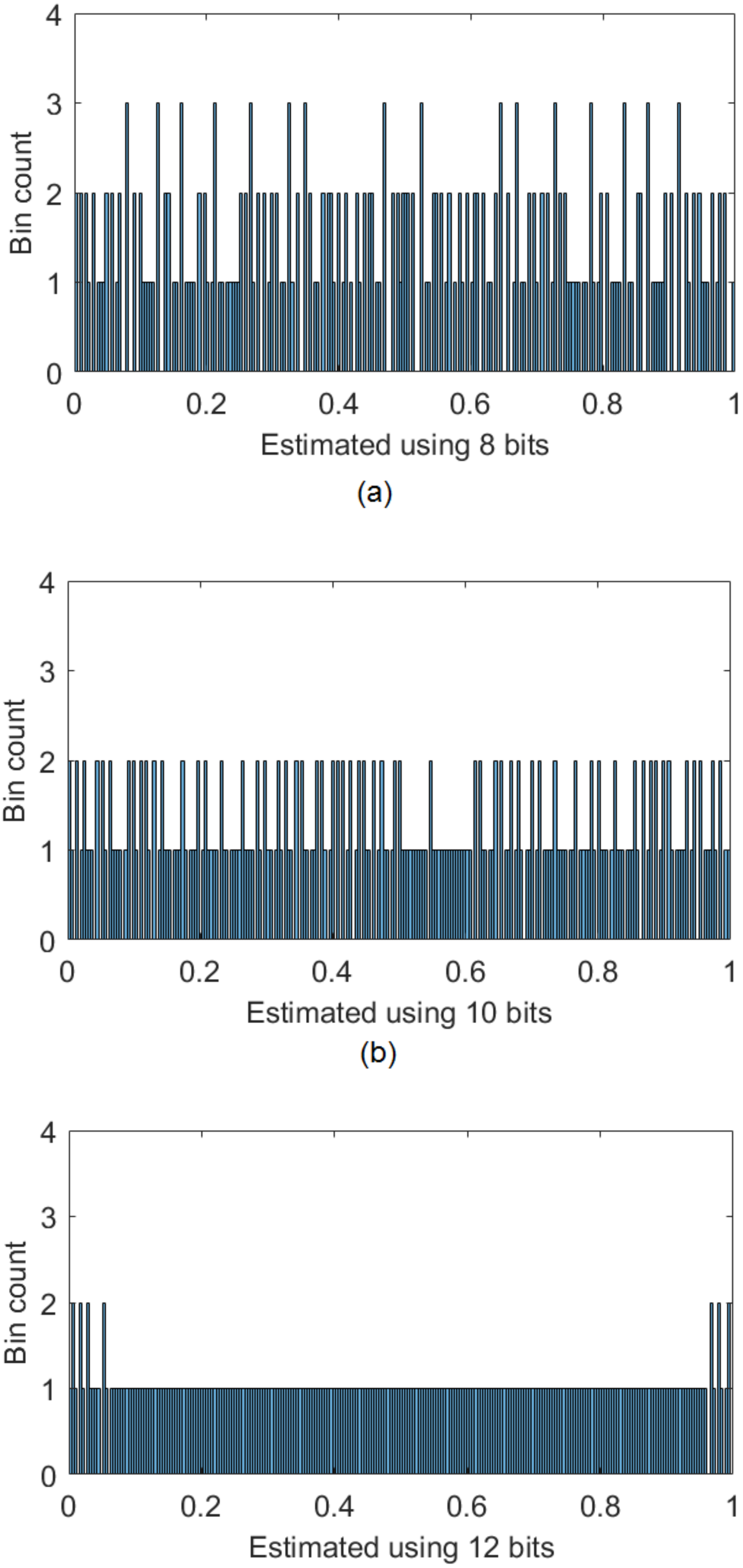
Fig. 5.13 Percent error using 16 symbols

When 16-bit long sequences are considered (Fig. 5.13), the error is reduced to a band of ~0.0.12% (within -0.188% and -0.2%). Both 12 and 16-bit results show improvement over the interval arithmetic method. However, for the deviation adjustment method, there is a tendency of the error bands to lie predominantly in one of the halves about ‘0’, i.e. the spread of the error is not uniform about ‘0’. Also, in both cases of 12 and 16-bit sequences, the error bands are entirely in the negative half.

For this method, the parameter value as well as the initial condition generated, must be limited to the same bit accuracy as the resolution. This is because the method is based on adjusting for the individual deviation in symbol for every iteration resulting from the change in symbolic sequence due to a reduced parameter. Therefore, the deviation that account for more than half the step size (i.e. affects the resolution) must attribute sufficiently for the change. Since the parameter dictates the magnitude of the deviation value for each iterate, the parameter must also be calculated similarly. However, this does not pose a challenge because in signal measurement applications, the bit accuracy need not exceed the resolution being considered.

In order to observe the bit accuracy, a histogram similar to the results for the interval arithmetic algorithm in section 5.1.2 is produced for the estimated outcomes of 8-bit precision with a bin size for the graph also measured in 8-bit accuracy.

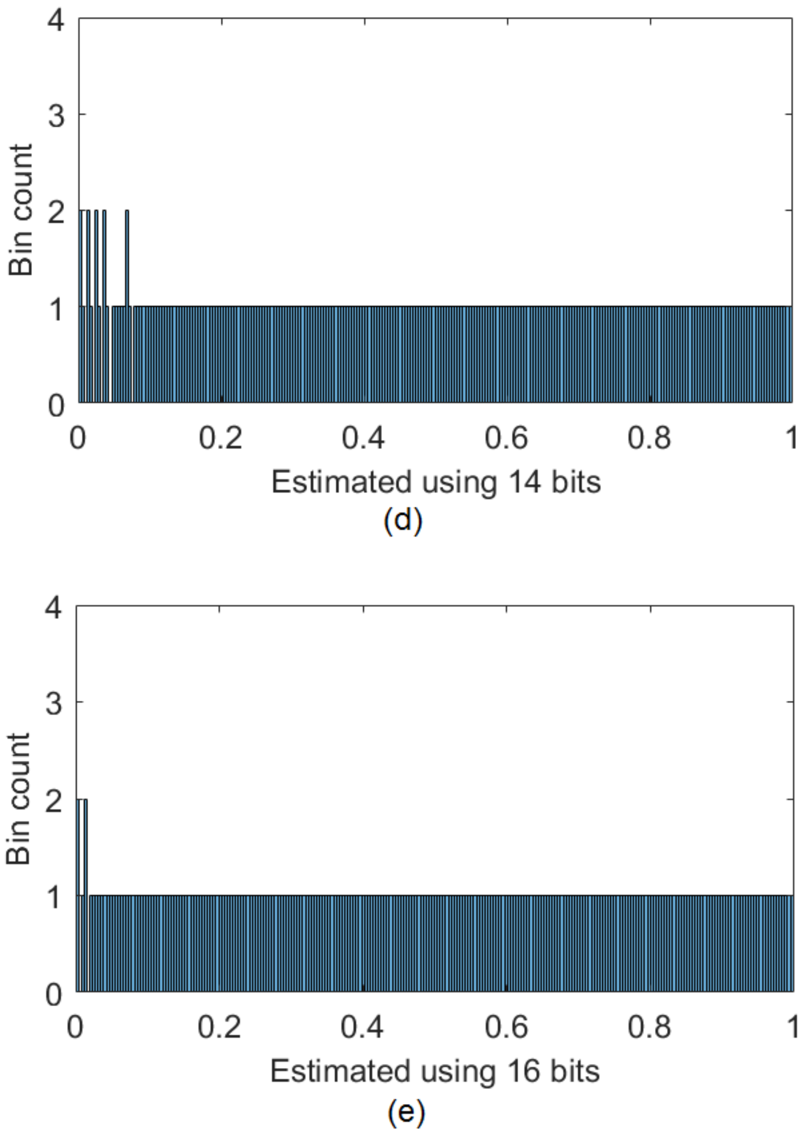
As before, compared to the GON values calculated as per section 2.3, although the points are better spread out over the state space, there are more than one-bit error for certain estimated outcomes (Fig. 5.14).


Bin count
0
1
2
3
4
0
0.2
0.4
0.6
0.8
1
Estimated using 8 bits
Estimated using 10 bits
Estimated using 12 bits

(a)

(b)

(c)



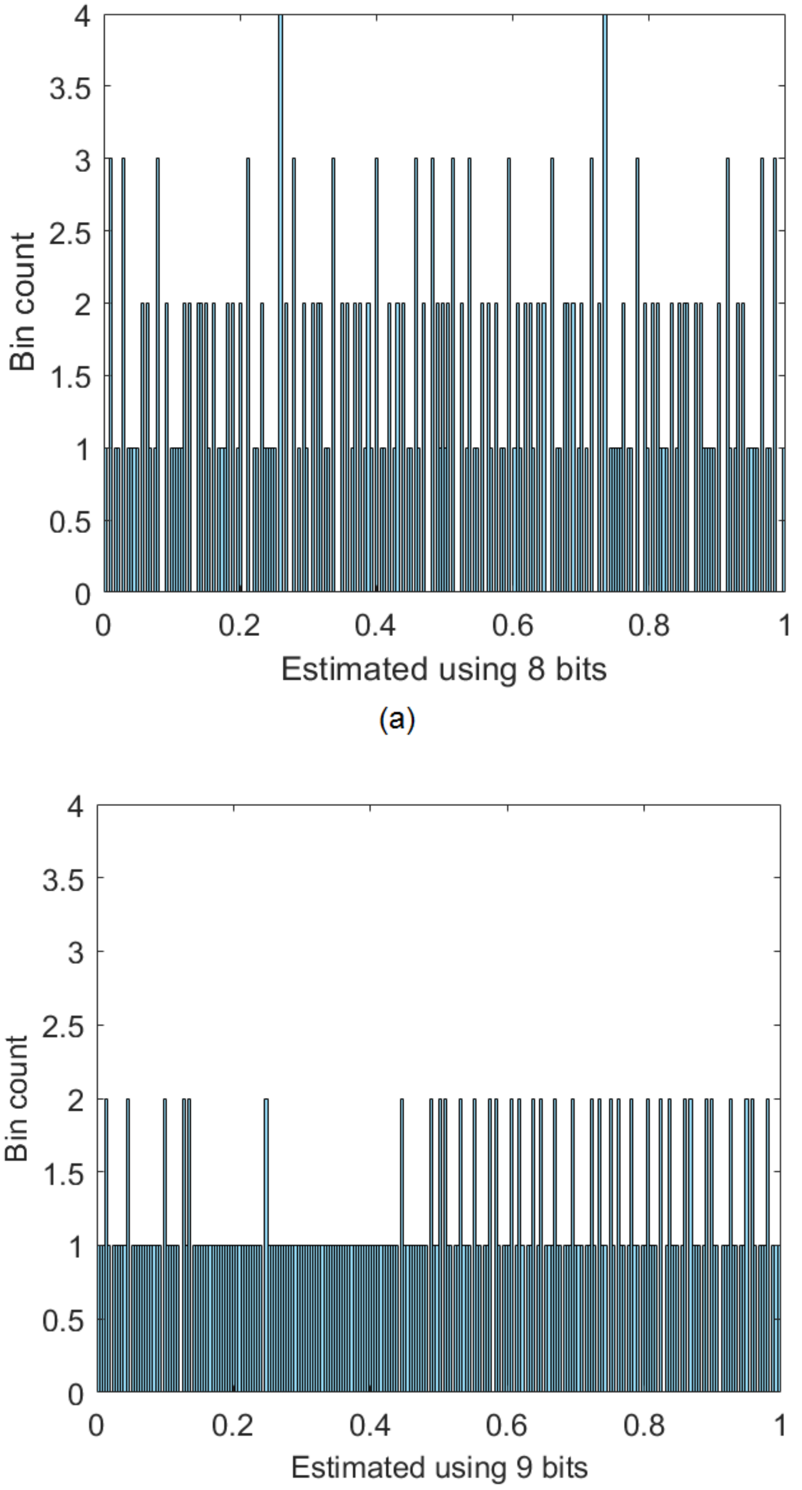
(d)

(e)

Fig. 5.14 Histograms for deviation adjustment method

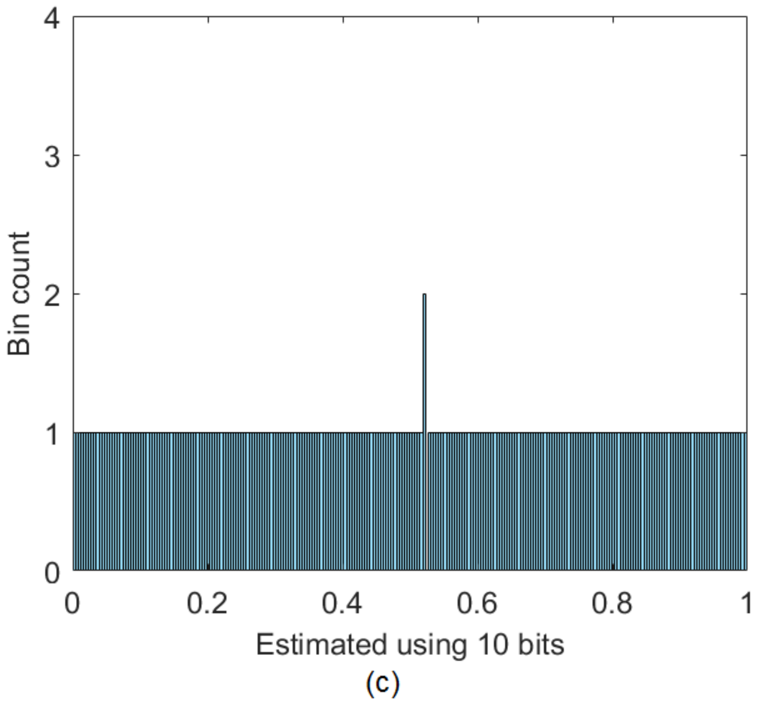
The test is also run for increasing number of symbols engaged in the conversion from 8 to 16 bits with an increment of two bits. It is observed in Fig. 5.14 that the success is achieved for increasing higher number of symbols than the intended bit precision. This is due to the fact that the cumulative effect of the reduced height map is reflected more as the iteration progresses further. As a result, the deviation due to the parameter is increasingly accounted for by a longer symbolic sequence.

Like the previous method, the current method is also tested using data points incremented by half step size. When the histograms are generated (Fig. 5.15), the outcome is again much more improved, as expected.



(a)

(b)



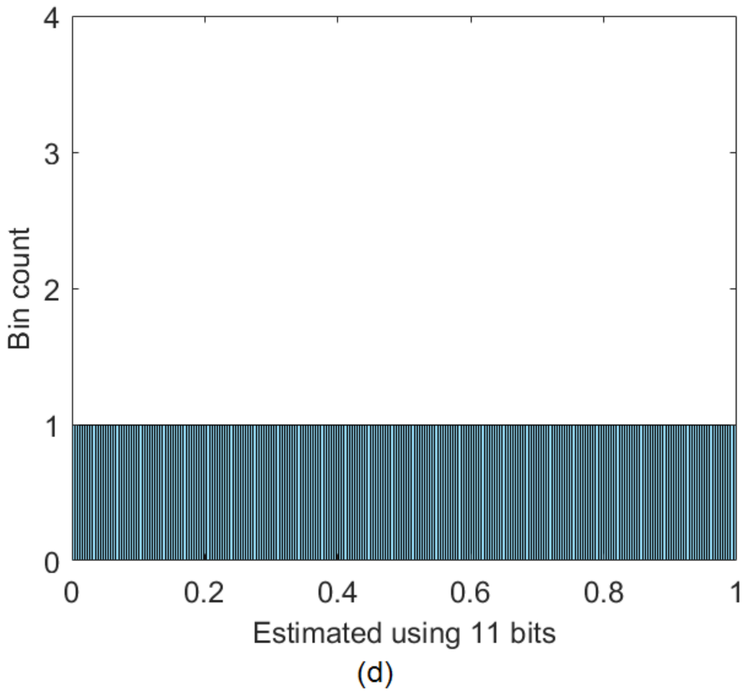


Fig. 5.15 Histograms for dataset shifted by half-step

In order to compare how the methods performed with respect to other initial condition estimation previously mentioned in Chapter 2, the results have been compared with the outcomes generated by 8-bit symbolic sequences for a number of chosen methods. The results are shown in Fig. 5.16.

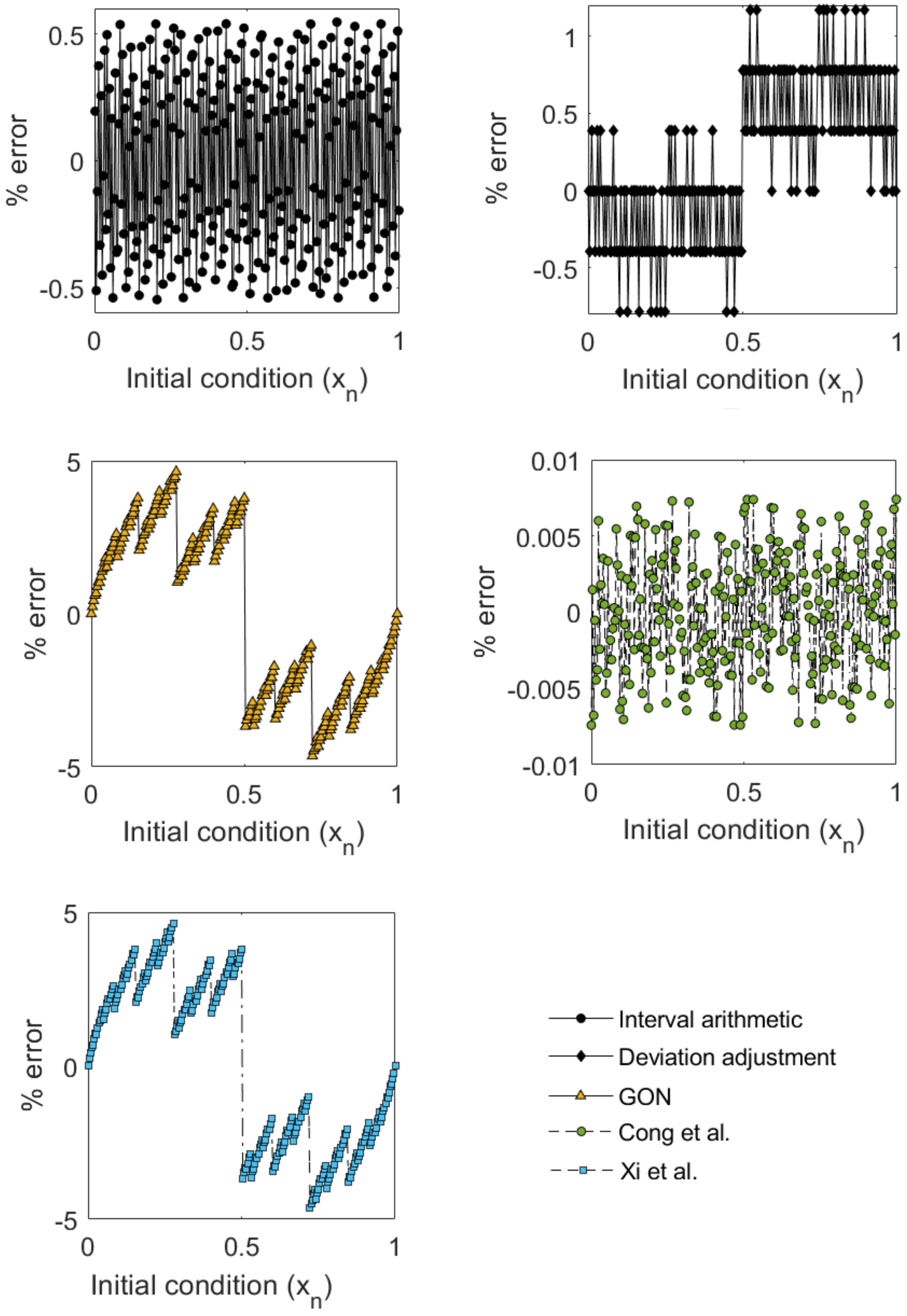


Fig. 5.16 Comparison of various estimation techniques

The methods by both Kennedy [12] and Kapitaniak et al. [13] produce results equivalent to GON, as both the methods essentially convert using a base of two, the methods are therefore not shown separately in the figure. It can be seen that the performance of the presented algorithms is only excelled by the method developed by Cong et al. [24] However, the added advantage in the interval arithmetic method is that the computation for the initial condition can be pipelined alongside the symbol collection in the circuit, thus speeding up the process of conversion. On the other hand, the deviation method can sufficiently reduce the processing because it employs a simple calculation to estimate the generic LUT data, and the deviation measurement equation is easily updated with the deviation.

Finally, the presence of noise is also simulated with the help of MATLAB. The MATLAB function “awgn(.)” is used to simulate the noise in the circuit. When the noise is added to every iterative step, as it would occur in a real circuit, the symbolic signatures of the initial conditions get corrupted. It is seen that as the effect of noise builds up over the iteration, the cumulative effect on the symbolic dynamics result in random bits getting flipped, i.e. the 1’s become 0’s or vice versa. Expectedly, as the noise is increased, more symbols are flipped. Also, as the iterations progress, the number of symbols flipped across the state space escalates. To emulate the possible range of noise in real situation, the values are chosen to be such that the signal to noise ratio (SNR) is between 0 to 40 dB. To observe the behaviour, a count of flipped bits cross the state space is recorded for every iterate for each noise level. The count is normalised and plotted against the number of iterations to demonstrate the average impact of noise on the iterates. However, for the chosen range of noise, the number of flipped symbols become steady after a certain number of iterations. This phenomenon can be observed in Fig. 5.17.

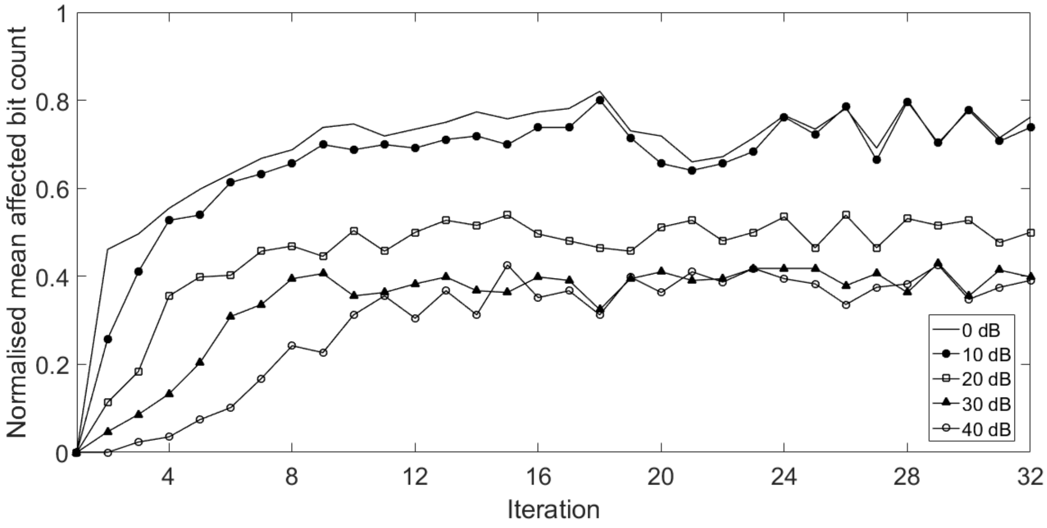


Fig. 5.17 Effect of noise on symbolic outcomes

In order to verify whether the presented algorithms can also estimate the initial condition from real hardware data vulnerable to the noisy environment, a physical circuit has been implemented on which the algorithms were tested. The test results are shown in the following section.

## 5.2 Implementation

Having seen that both the methods performed satisfactorily in the simulated environment of MATLAB, the next step was to test the performance using symbolic trajectories generated physically implemented hardware version of the map function. The electronic circuit of a TM by Campos et al. [42] has been adapted by Dutta [28] and implemented as shown in Fig. 5.18 where the relevant blocks are shown: the TM circuitry, the comparator for the symbols and the sample and hold (S/H) for the feedback. The physical hardware of the implemented circuit is shown in Fig. 5.19.

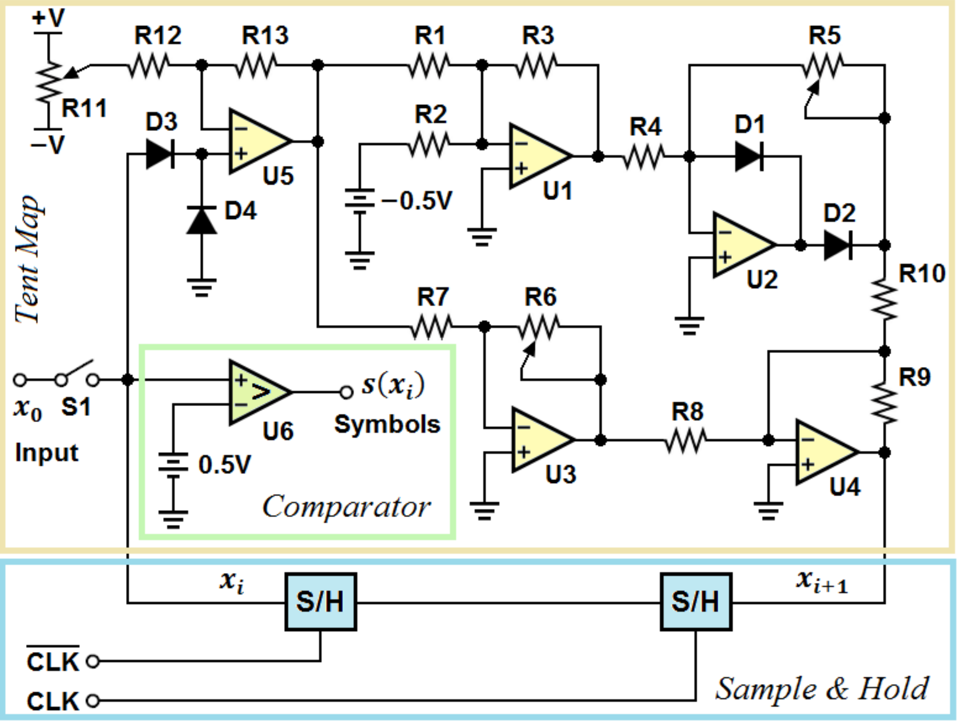


Fig. 5.18 Schematic diagram adapted from the TM circuit [42].

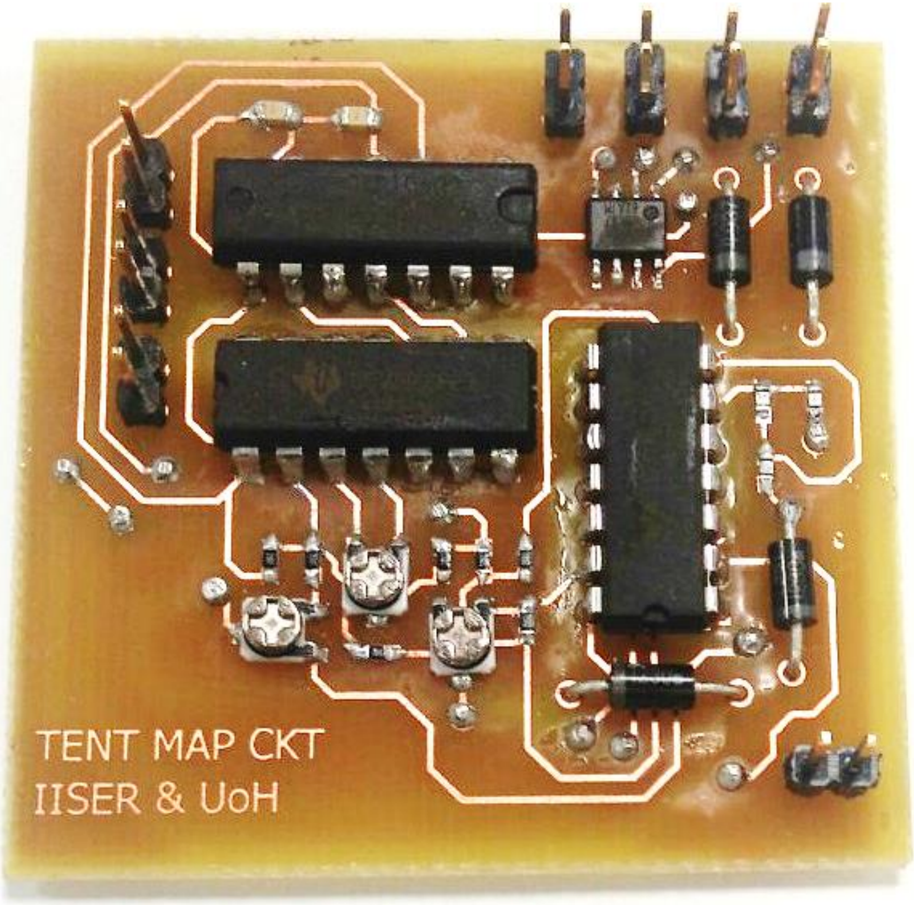


Fig. 5.19 Physical hardware of the TM function [28].

The map was tested through an oscilloscope where the relative height of the map function is shown with respect to the full-scale ramp input of [0,1] (Fig. 5.20). The TM is fed through channel 1 (blue) and for comparison, the ramp input is shown using channel 2 (pink). As can be seen, the imperfection in the hardware components has resulted in the map height to be limited to approximately $\mu$ = 0.81.

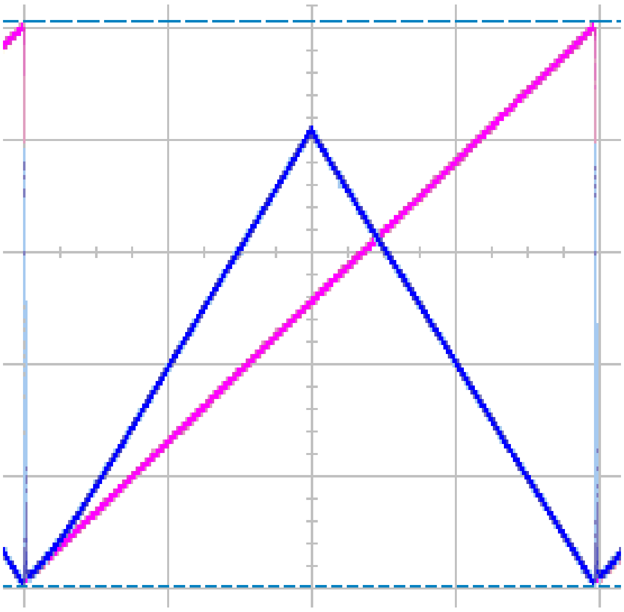


Fig. 5.20 Image capture of the map on oscilloscope

To verify that the map can successfully perform iterations through the feedback system, the circuit is tested for function using 200mV and 518mV. The input signals are iterated 16 times and the symbolic sequences are also generated. The real as well as the symbolic trajectories are observed in the oscilloscope and can be seen from the image capture of the time series shown in Fig. 5.21. Channel 1, set at 200mV per division (mV/div) shows the real trajectory while the symbolic signature is viewed through Channel 2 (5V/div).

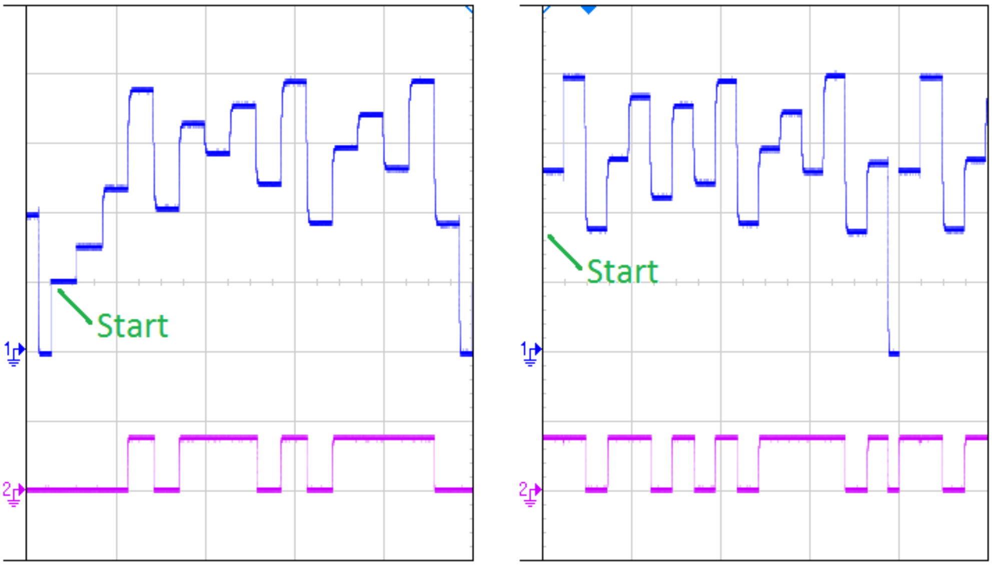


Fig. 5.21 Time series for 200mV (left), and 518mV (right)

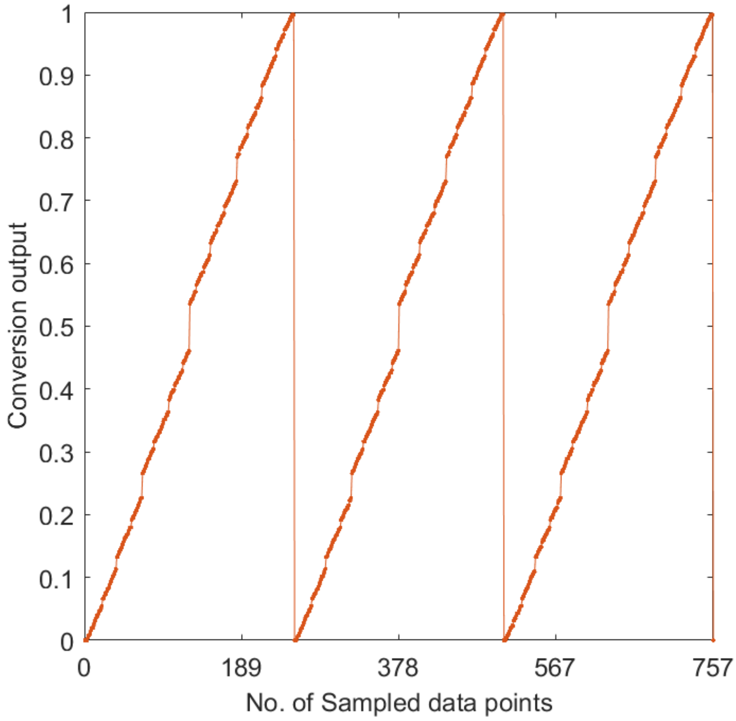


Fig. 5.22 GON of a 3-cycle ramp

In order to determine how the retrieved trajectories perform, when used for calculations in the digital domain, a 3-cycle ramp is run at 10mHz and 8-bit symbolic trajectories have been utilised to run the GON function. The resulting values have been imported and plotted through MATLAB and can be seen in Fig. 5.23.

Once the functionality of the circuit is confirmed to be true to the TM behaviour, the samples of the initial conditions and the respective 8 bits as well as 16 bits long symbolic sequences are collected for a one-cycle ramp (data can be found in Appendix X with half-step-size shifted. One by one, both the algorithms have been run on the same set of symbolic sequences, and the outcome is compared with the original input.

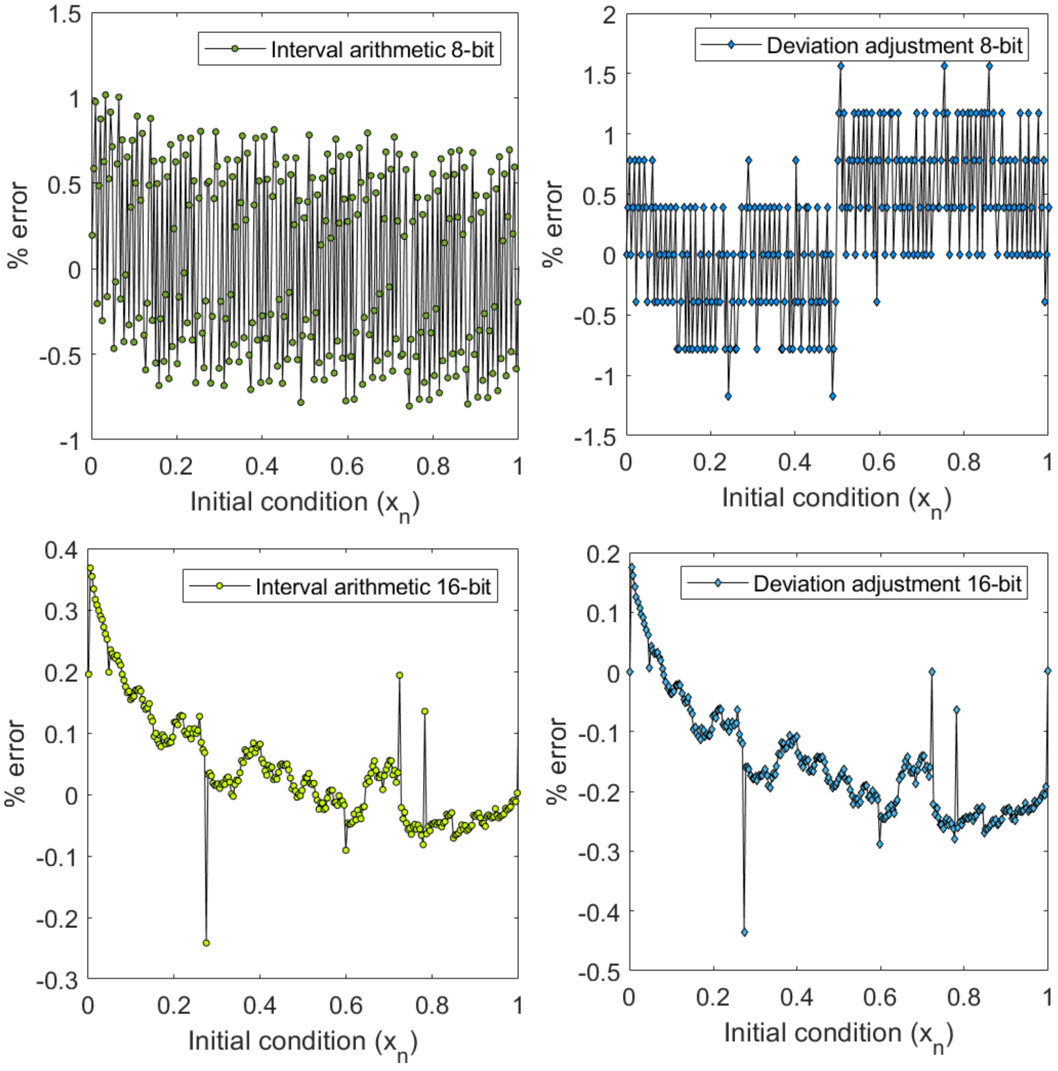


Fig. 5.23 The 8-bit and 16-bit % error using both the algorithms

The data is first experimented with the interval arithmetic algorithm and percent error is generated for both 8 and 16-bit long sequences. Fig. 5.23 and Fig. 5.24 show the outcomes. For the percent error, there is a general trend of skewness in the error which seem to be higher for the values near zero in both the cases of symbolic length (Fig. 5.23). This is because the lower valued initial conditions have been more readily affected by the system noise. Similar observations are shown for the deviation adjustment algorithm. The percent error for the deviation adjustment method using 8-bit is seen to be slightly higher in case of the practical implemented TM. Although the 16-bit error plots look identical with equal ranges, the deviation adjustment method produces more negative error than the interval arithmetic method.

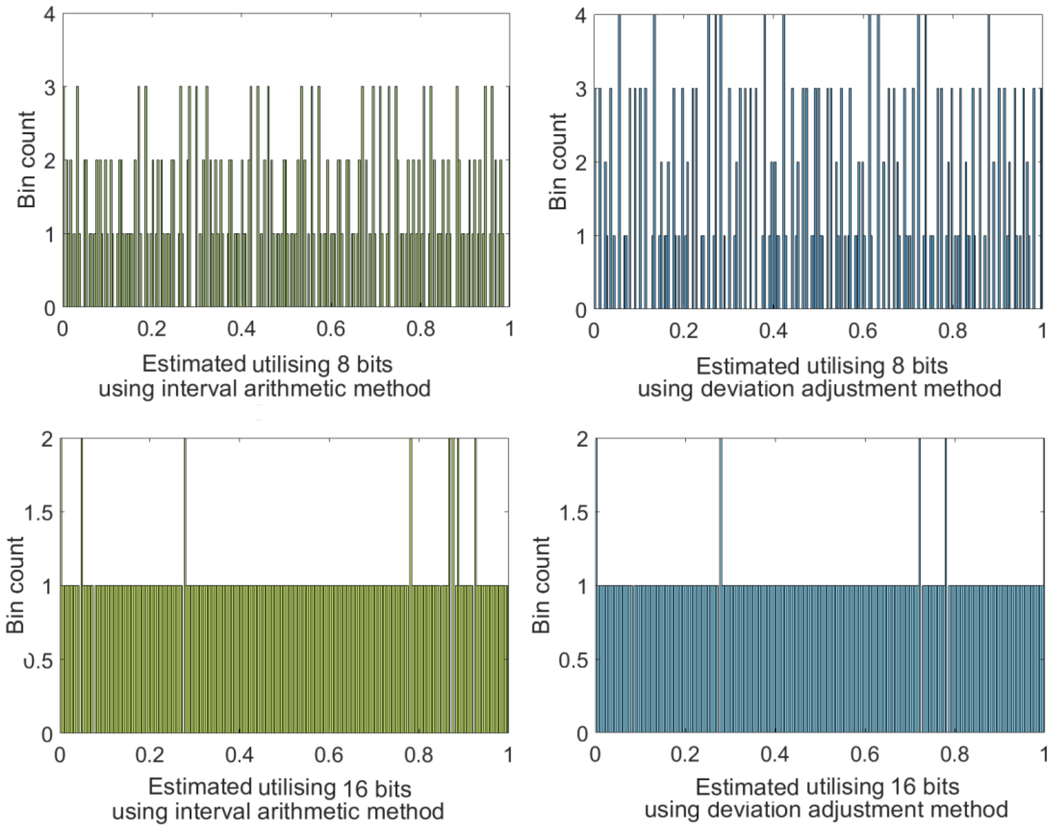


Fig. 5.24 The 8-bit and 16-bit histograms using both the algorithms

Next, the bit errors are observed by plotting the histograms for all the four cases as before (Fig. 5.24). For 8-bit the outcome again appears to be better for the interval arithmetic method. However, for 16-bit, the deviation adjustment method shows improved spread, although the percent error showed no noticeable difference. This is because, although the absolute errors were similar, the some of the results produced by the interval arithmetic methods must have been on the edge of the histogram bins and therefore got misplaced.

Both the methods, therefore, can be successfully utilised for the implementation of the ADC. The milestones achieved through the methods as possible approaches in this work for the proposed implementation a chaos based ADC is summarised in the next chapter and possible future directions are also indicated.

# 6 CONCLUSION AND FURTHER WORK

The principles of measurement systems and several analogue to digital conversion (ADC) techniques have been discussed. Each of the approaches is dedicated to enhancing certain aspects of the ADC such as superior quality of signal quantisation i.e. better resolution, accuracy and precision, also power optimisation and design level complexities are that are significant factors that are dealt differently for different approaches. From the review of each of such techniques e.g. working principle, constituent system, power and resource consumption it is clearly seen that each aspect of optimisation involves certain trade-offs. Due to this, optimising every aspect of an ADC is probably challenging as trying to enhance the quantisation can increase the amount of hardware required to fulfil the approach which might be resource consuming in terms of power and chip area and adding to the design complexity. Otherwise to keep the resource and power aspects optimum some accuracy and precision is sacrificed which makes choosing the right approach optimally and organisation of the entire approach very crucial.

Given that the chaotic systems are governed by simple mathematical rules and processes, such systems are easy to implement in the physical platform. As chaotic dynamics is iterative in nature, a single functional block is reused to produce the long-term trajectories through feedback mechanisms, this prevents involving any additional components. Therefore, chaotic maps to be used as measurement system involve very little amount resources to complete the entire system. Since chaotic dynamics is sensitive to initial condition, use of chaos in measurement applications has been proven feasible and promising.

It was demonstrated that chaotic maps can be utilised as a quantisation unit for signal detection. The signal to be measured is input to the chaotic function as initial condition and iterated several times depending on the amount of information required to define the input. Each TM iteration involves stretching or folding which is responsible for the partitioning of the state space. As the iterations are continued more partitions are generated hence more intervals are created and accordingly the dynamics produced by a certain initial condition can be used to identify the originating interval of the input with a reasonable accuracy. Knowing that the itinerary of chaotic dynamics can be treated as a unique signature of an initial condition it can be utilised to back track the initial condition. Applying symbolic dynamics to chaos is even more advantageous as plenty of resources can be saved just by introducing a binary symbolic structure to the chaotic itineraries depending on a threshold. As the dynamics is produced for a desired number of iterations/bits, the symbolic sequence for the initial condition is converted to real values to identify the real signal present as an input.

The physical implementation of the chaotic system is subject to several non-idealities. One such non-ideality in the circuit realisation of the tent map is the parametric imprecision. When TM circuit is used for signal quantisation as an ADC, the parametric imprecision is responsible for the loss of correspondence between the initial condition and the symbolic sequence as the partitions generated by the non-ideal parameter is shifted from their ideal positions. Due to the shifted partitions the symbolic sequence also differs from sequence that is ideally generated to define an initial condition. Therefore, correspondence between the intervals and the symbols have been thoroughly studied in order to understand the consequences of shifted partitions. The knowledge of non-ideal parameter value is found to be relatable to the shift of partitions. The amount of shift introduced by the non-ideal parameter is found to be proportional to the amount of change in the parameter value. Hence using the correct parameter value and applying interval arithmetic actual position of partitions are retrieved and accordingly a method has been formulated through which correct interval for each of the symbol in the sequence can be chosen, thus the interval of the initial condition can be properly identified.

It has been shown that the partitioning of the state space on every iteration results into creation of sub-intervals. Evidently, if the number of iterations are increased and more partitions and therefore bits of information is generated that can be utilised to further back track or narrow down the interval of initial condition with a reasonable accuracy. The higher is the number of bits more are the divisions in the state space thus the intervals become narrower and thereby increasing resolution of the originating signal.

Due to iterative dynamics a single chaotic map is utilised repeatedly to produce the dynamics, hence the cost for generating greater number of bits to enhance precision is negligible compared to other ADC architectures e.g. flash ADC whose number of comparators doubles per bit whereas for the chaotic ADC a single comparator is reused to generate the symbols on every iteration. Therefore, from the perspective of resource consumption the technique described appears to be promising.

## 6.1 Future directions

The methods devised in the work are open to several future possibilities of further related as well as independent research areas that can be pursued. Apart from successfully developing the ADC as a device – which requires more adjustments of the algorithms in terms of addressing other hardware implementation related issues – the methods can also be utilised in other independent directions, the most important being cryptography and cryptanalysis.

### 6.1.1 ADC implementation

In order to implement the ADC as a stand-alone device, certain other factors need to be taken care of. Like the parametric divergence, the critical point of the TM is also subject to deviations introduced by circuit imprecisions and perturbations.

**Shift of the critical point**

Further research is required in the direction of addressing the shift of the critical point in the implemented circuitry. The use of skew TMs [49] as an alternative might be useful once the feasibility is tested. Since the parameter of the skew TM is determined by the location of the critical point of the map, it provides the advantage of addressing only a single variable that defines the structure of the map completely. Therefore, modifications in the existing methods for a skew TM is the first step to pursue in terms of ADC implantation.

**Implementation as a stand-alone device**

Once these issues are successfully tested in the simulation domain, a skew TM can be implemented in real circuitry and the estimation algorithms can finally be executed on a field programmable gate array (FPGA) as a stand-alone device. This will benefit the measurement system with more robust conversion approach as a single source of error needs to be dealt in case of skew TM on the contrary to the two independent sources of errors that would have been present in simple TM.

**Deviation adjustment through bits**

Additionally, the deviation adjustment method can be improvised for easier implementation through an FPGA. In this case, the research might be focussed in a way such that the deviation can be used to determine the correct (ideal) symbolic sequence, and the symbols might be directly adjusted to generate equivalent binary symbols. As a result, the use of floating number library in the FPGA might be substantially reduced.

### 6.1.2 Application in Cryptography

Other possible applications of initial condition estimation and trajectory analysis can be explored. One such application could be cryptography where information is protected and preserved through cryptic means. TM is widely used in cryptography because of the robust chaos generated for a wide range of parameter values. The meaningful information is mutated by transforming it through the chaotic function that makes the information appear as random data to the unintended observants. Then for further utilisation and retrieval of information the encrypted data needs to be reinstated which is ideally analogous to tracking back from the current iterate (encrypted data) to the initial condition (original data). The parameter of the map is used as an encryption key that is utilised during decryption. For a chaotically encrypted information, such a robust and efficient initial condition estimation algorithm as proposed in this work can be applied. However, there are several methods and approaches available for encryption, e.g. a single chaotic map or coupled chaotic maps can be utilised as the encrypting function and the encrypted outcomes can both be symbolic or real valued. Depending on the approach and the level of complexity demanded as an outcome, further research is necessary for the modification of the proposed approach in this work so that it can be dedicated for retrieval of the encrypted data. Possible area of encryption is image processing, data storage and communication.

### 6.1.3 Application in communication

Communication systems is another promising area where messages are encrypted from the originating end and decrypted at the receiving end. To maintain unhindered speed the encryption-decryption processes are implemented in the hardware domain. Such an electronic hardware usually contains chaotic maps which generate semirandom data that can be sent as an encrypted message for the original message input as initial condition to the chaotic system. The parameter is saved and sent as a secured key to the receiver so that the receiver can retrieve the original message from the encrypted message by backtracking. With the proposed algorithm such backtracking can be more efficient and speedy. Hence, through further research the proposed technique can be dedicated to solving similar problems like initial condition estimation into a wide range of areas.

# REFERENCES

[1] Ingraham, R. L. (1991). *A Survey of Nonlinear Dynamics (Chaos Theory)*. World Scientific Publishing Company

[2] Poincaré, H. (1993). *New methods of celestial mechanics*. Los Angeles, CA: Tomash

[3] Baker, G. L., & Gollub, J. P. (1990). *Chaotic Dynamics: An Introduction*. Cambridge University Press

[4] Gleick, J. (1988). *Chaos: making a new science*. Penguin

[5] Lorenz, E. (n.d.). Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences*, *20*, 130-141

[6] Gilmore, R., and Lefranc, M. (2002) "Discrete Dynamical Systems: Maps," *The Topology of Chaos*, 1st ed. New York, NY, USA: JW&Sons, 40-53

[7] Strogatz, S. H. (2000). *Nonlinear dynamics and chaos: With applications to physics, biology, chemistry and engineering*. Cambridge, Mass: Westview

[8] Radwan, A. G., Abd-El-Hafiz, S. K., & AbdElHaleem, S. H. (2014). *An image encryption system based on generalized discrete maps.* Paper presented at the 2014 21st IEEE International Conference on Electronics, Circuits and Systems (ICECS), Marseille, 283-286. 10.1109/ICECS.2014.7049977

[9] Kurian, A. P., & Puthusserypady, S. (2006). Secure Digital Communication using Chaotic Symbolic Dynamics. *Turkish Journal of Electrical Engineering, 14*(1), 195-207

[10] Maggio, G. M., & Galias, Z. (2002). Applications of symbolic dynamics to differential chaos shift keying. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, 49*(12), 1729-1735. 10.1109/TCSI.2002.805701

[11] Rogers, A., Keating, J., Shorten, R., & Heffernan, D. M. (2002). Chaotic maps and pattern recognition – the XOR problem. *Chaos, Solitons and Fractals, 14*(1), 57-70. 10.1016/S0960-0779(01)00181-3

[12] Kennedy, M. P. (1995). a nonlinear dynamics interpretation of algorithmic a/d conversion.*International Journal of Bifurcation and Chaos, 5*(3), 891-893. 10.1142/S0218127495000685

[13] Kapitaniak, T., Zyczkowski, K., Feudel, U., & Grebogi, C. (2000). Analog to digital conversion in physical measurements. *Chaos, Solitons and Fractals, 11*(8), 1247-1251. 10.1016/S0960-0779(99)00003-X

[14] Metropolis, N., Stein, P. R., & Stein, M. L. (1973). On finite limit sets for transformations on the unit interval. *Journal of Combinatorial Theory, Series A, 15*(1), 25-44. 10.1016/0097-3165(73)90033-2

[15] Bollt, E., Standford,T., Lai, Y., and Życzkowski, K. (2001). What symbolic dynamics do we get with a misplaced partition? on the validity of threshold crossings analysis of chaotic time-series. *Physica D: Nonlinear Phenomena, 154*(3-4), 259-286. 10.1016/S0167-2789(01)00242-1

[16] Arroyo, D., & Alvarez, G. (2014). Application of gray codes to the study of the theory of symbolic dynamics of unimodal maps. *Communications in Nonlinear Science and Numerical Simulation, 19*(7), 2345. 10.1016/j.cnsns.2013.11.005

[17] Berberkic, S. (2014) Measurement of small signal variations using one-dimensional chaotic maps. *Doctoral thesis*, University of Huddersfield

[18] Berberkic, S., Mather, P., & Bromley, R. (2014). *G.B. Patent No. WO/2014/191732*.

[19] Banerjee, S., Yorke, J. A., & Grebogi, C. (1998). Robust chaos. *Physical Review Letters, 80*(14), 3049-3052. 10.1103/PhysRevLett.80.3049

[20] Schweizer, J., & Schimming, T. (2001). Symbolic dynamics for processing chaotic signals. I. noise reduction of chaotic sequences. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, 48*(11), 1269-1282. 10.1109/81.964416

[21] Marteau, P. F., & Abarbanel, H. D. I. (1991). Noise reduction in chaotic time series using scaled probabilistic methods. *Journal of Nonlinear Science, 1*(3), 313-343. 10.1007/BF01238817

[22] Litovski, V., Andrejevic, M., & Nikolic, M. (2006). Chaos based analog-to-digital conversion of small signals. Paper presented at the *2006 8th Seminar on Neural Network Applications in Electrical Engineering*, Belgrade, Serbia & Montenegro, 173-176. 10.1109/NEUREL.2006.341205

[23] Xi, C., Yong, G. and Yuan, Y. (2009). A Novel Method for the Initial-Condition Estimation of a Tent Map. *Chinese Physics Letters, 26*(7), pp. 078202 - 1–3. 10.1088/0256-307X/26/7/078202

[24] Cong, L., Xiaofu, W., & Songgeng, S. (1999). A general efficient method for chaotic signal estimation. *IEEE Transactions on Signal Processing, 47*(5), 1424-1428. 10.1109/78.757236

[25] Collet, P., and Eckmann, J. P. (1980). Typical Behavior for One Map. *Iterated Maps on the Interval as Dynamical Systems*, 1st ed. Boston, MA, USA: Birkhäuser Basel, 7–22

[26] Amigó, J. M., Elizalde, S., & Kennel, M. B. (2008). Forbidden patterns and shift systems.*Journal of Combinatorial Theory, Series A, 115*(3), 485-504. 10.1016/j.jcta.2007.07.004

[27] Wu, X., Hu, H., & Zhang, B. (2004). Parameter estimation only from the symbolic sequences generated by chaos system. *Chaos, Solitons and Fractals, 22*(2), 359-366. 10.1016/j.chaos.2004.02.008

[28] Basu, R., Dutta, D., Banerjee, S., Holmes, V., & Mather, P. (2017). An Algorithmic Approach for Signal Measurement Using Symbolic Dynamics of Tent Map. *IEEE Transactions on Circuits and Systems I: Regular Papers , PP*(99), 1-10. doi: 10.1109/TCSI.2017.2773202

[29] Sheingold, D. H., & Analog Devices. (1986). *Analog-digital conversion handbook* (3rd ed.). London;Englewood Cliffs;: Prentice-Hall.

[30] Walden, R. H. (1999). Analog-to-digital converter survey and analysis. *IEEE Journal on Selected Areas in Communications, 17*(4), 539-550. 10.1109/49.761034

[31] Robert, M., Savaria, Y., & Wang, C. (2004). Analysis of metrics used to compare analog-to-digital converters. Paper presented at the *The 2nd Annual IEEE Northeast Workshop on Circuits and Systems, 2004. NEWCAS 2004*, 301-304. 10.1109/NEWCAS.2004.1359091

[32] Mather, P. J. (n.d.). *6. Mixed-Signal Circuit Structures*. Lecture presented at NIE2203 Electronics 2 Lecture in University of Huddersfield, Huddersfield

[33] Bashir, S., Ali, S., Ahmed, S., & Kakkar, V. (2016). Analog-to-digital converters: A comparative study and performance analysis. Paper presented at the *2016 International Conference on Computing, Communication and Automation (ICCCA)*, Noida, 999-1001. 10.1109/CCAA.2016.7813861

[34] El-Chammas, M., and Murmann, B. (2011). A 12-GS/s 81-mW 5-bit Time-Interleaved Flash ADC With Background Timing Skew Calibration *IEEE Journal of Solid-State Circuits, 46*(4), 838-847, 10.1109/JSSC.2011.2108125

[35] Black, W. C. and Hodges, D. A. (1980). Time interleaved converter arrays, *IEEE Journal of Solid-State Circuits*, *15*(6), 1022-1029, 1980. 10.1109/JSSC.1980.1051512

[36] Silva, J., Moon, U., Steensgaard, J., & Temes, G. C. (2001). Wideband low-distortion delta-sigma ADC topology. *Electronics Letters, 37*(12), 737. 10.1049/el:20010542

[37] Barot, N. *Successive Approximation Analog to Digital Converter* (Unpublished doctoral dissertation). San Jose State University

[38] Zhu, D., Sifleet, T, Nunnally, T and Huang, Y. (n.d.). *Analogue to Digital Converters* Lecture presented at Mechatronics Course Lecture in Georgia Tech University [online: http://ume.gatech.edu/mechatronics_course/ADC_F08.pdf]

[39] Kimura, H., Matsuzawa, A., Nakamura, T., & Sawada, S. (1993). A 10-b 300-MHz interpolated-parallel A/D converter. *IEEE Journal of Solid-State Circuits, 28*(4), 438-446. 10.1109/4.210026

[40] Nauta, B., & Venes, A. G. W. (1995). A 70-MS/s 110-mW 8-b CMOS folding and interpolating A/D converter. *IEEE Journal of Solid-State Circuits*, *30*(12), 1302-1308. 10.1109/4.482155

[41] Suneel, M. (2006). Electronic circuit realization of the logistic map. *Sadhana, 31*(1), 69-78. 10.1007/BF02703801

[42] Campos-Cantón, I., Campos-Cantón, E., Murguía, J. S., & Rosu, H. C. (2009). A simple electronic circuit realization of the tent map. *Chaos, Solitons and Fractals*, *42*(1), 12-16. 10.1016/j.chaos.2008.10.037

[43] Wang, L., & Kazarinoff, N. D. (1987). On the universal sequence generated by a class of unimodal functions. *Journal of Combinatorial Theory, Series A*, *46*(1), 39-49. 10.1016/0097-3165(87)90075-6

[44] Álvarez, G., Romera, M., Pastor, G., & Montoya, F. (1998). Gray codes and 1D quadratic maps. *Electronics Letters*, *34*(13), 1304. 10.1049/el:19980950

[45] Arroyo, D., Alvarez, G., & Amigó, J. M. (2009). Estimation of the control parameter from symbolic sequences: Unimodal maps with variable critical point. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, *19*(2), 023125-023125-9. 10.1063/1.3155072

[46] Dinu, A. and Vlad, A. (2014). The Compound Tent Map and the Connection between Gray Codes and the Initial Condition Recovery. *U.P.B. Sci. Bull.*, 76 (1), A, 17-28

[47] Dutta, D., Basu, R., Banerjee, S., Holmes, V., & Mather, P. (2018). Parameter estimation for 1D PWL chaotic maps using noisy dynamics. *Nonlinear Dynamics*, 1-15

[48] Martinez-Gonzalez, R. F., Diaz-Mendez, J. A., & Vazquez-Medina, R. (2016). Algorithm for implementing 32-bits represented bernoulli map using an 8-bits microcontroller. Paper presented at the 1-6. 10.1109/IESummit.2016.7459773

[49] Wang, K., Pei, W., Hou, X., Shen, Y., & He, Z. (2009). Symbolic dynamics approach to parameter estimation without initial value. *Physics Letters A, 374*(1), 44-49. 10.1016/j.physleta.2009.10.021

# APPENDIX 1

Contributions to journal publications associated with the research.

**List of articles**

1.1 An Algorithmic Approach for Signal Measurement Using Symbolic Dynamics of Tent Map

1.2 Parameter estimation for 1D PWL chaotic maps using noisy dynamics

## Appendix 1.1

This article has been published in the Transactions in Circuits and Systems – I on 7 December 2017 and can be found online at: 10.1109/TCSI.2017.2773202

# An Algorithmic Approach for Signal Measurement Using Symbolic Dynamics of Tent Map

R Basu, D. Dutta, S. Banerjee, V. Holmes and P. Mather

R. Basu, D. Dutta, V. Holmes and P. Mather are with the Engineering and Technology Department, School of Computing and Engineering, University of Huddersfield, Queensgate, Huddersfield, W. Yorks., UK, HD1 3DH (email: rajlaxmi.basu@hud.ac.uk, dhruba.dutta@hud.ac.uk, v.holmes@hud.ac.uk, p.j.mather@hud.ac.uk).

S. Banerjee is with the Department of Physical Sciences, Indian Institute of Science Education & Research, Kolkata, Mohanpur Campus, Nadia-741246, India (email: soumitro@iiserkol.ac.in).

**Reference**

Basu, R., Dutta, D., Banerjee, S., Holmes, V., & Mather, P. (2017). An Algorithmic Approach for Signal Measurement Using Symbolic Dynamics of Tent Map. *IEEE Transactions on Circuits and Systems I: Regular Papers* , *PP*(99), 1-10.

## Appendix 1.2

This article has been published in the Nonlinear Dynamics on 18 September 2018 and can be found online at: 10.1007/s11071-018-4538-x

# Parameter estimation for 1D PWL chaotic maps using noisy dynamics

D. Dutta, R Basu, S. Banerjee, V. Holmes and P. Mather

D. Dutta, R. Basu, V. Holmes and P. Mather are with the Engineering and Technology Department, School of Computing and Engineering, University of Huddersfield, Queensgate, Huddersfield, W. Yorks., UK, HD1 3DH (email: rajlaxmi.basu@hud.ac.uk, dhruba.dutta@hud.ac.uk, v.holmes@hud.ac.uk, p.j.mather@hud.ac.uk).

S. Banerjee is with the Department of Physical Sciences, Indian Institute of Science Education & Research, Kolkata, Mohanpur Campus, Nadia-741246, India (email: soumitro@iiserkol.ac.in).

**Reference**

Dutta, D., Basu, R., Banerjee, S., Holmes, V., & Mather, P. (2018). Parameter estimation for 1D PWL chaotic maps using noisy dynamics. *Nonlinear Dynamics*, 1-15.

# APPENDIX 2

MATLAB codes developed for experimentation as well as documentation.

**List of codes**

2.1 Dataset generator for both algorithms

2.2 Bifurcation diagram generator for:

2.2.1 Bitshift Map
2.2.2 Logistic Map
2.2.3 Tent Map

2.3 Implementation of algorithms

2.3.1 Interval arithmetic algorithm
2.3.2 Deviation adjustment algorithm

## Appendix 2.1

MATLAB code for dataset generation is presented.

```
format long

iteration = 16;     % number of iterations the sequence is generated up to
partition = 0.5;    % map partition
M = 2*partition;    % Map height
power = 8;          % divition generated as 2^power

increment = (1/(2^power)); % size of each point or increment in the dataset

x = increment/2;  % initiate the points at half stepsize
x_max = 1;        % final value of x after increments
xNew = x;         % copy first point of the date set for iteration

N = ((x_max-x)/increment);     % calculating number of increment steps
N = ceil(N);                   % rounding up N to the next higher integer

GON = zeros(N,1);              % Gray ordering number array
Newfinal_result = zeros(N,iteration); % stores all x for all steps
Data_set_8bit = zeros(N,1);           % initial condition data set
Newfinal_gray = zeros(N,iteration);   % stores all op for all steps
Newfinal_bin1 = zeros(N,iteration);   % stores all bin-op for all steps

aNew = 0.9054 ;
slopeNew = 1+aNew;                 % sets the peak height of the tent map
```

```
for i = 1:N                     % runs a for loop for increment steps

    x2 = xNew;        % copy data point or initial condition for iteartion

    for n = 1:iteration            % runs a for loop for the iterations

        if x2 <= partition            % condition for when x < 0.5
            yNew = slopeNew*x2;       % evaluating iterate x_n+1
            opNew = 0;                % symbolic output '0' when x < 0.5
        elseif x2 > partition         % condition for when x >= 0.5
            yNew = slopeNew*(M-x2);   % evaluating iterate x_n+1
            opNew = 1;                % symbolic output '1' when x >= 0.5
        end

        Newfinal_result(i,n) = x2;      % store the iterate as trajectory
        Newfinal_gray(i,n) = opNew;     % store the symbol as symbolic seq.

%       x2 = awgn(yNew,100);              % adding gaussian noise
        x2 = yNew;                      % copying the x_n = x_n+1 for next
                                         % iteration
    end
    xNew = (xNew + increment);         % increasing x by one step size for
                                         % the next initial condition
end

for row = 1:N                     % for N initial conditions
    for col = 1:iteration         % Converting the Gray code into binary
        if col == 1
            Newfinal_bin1(row,col) = Newfinal_gray(row,col);
        elseif col > 1
            Newfinal_bin1(row,col) = bitxor(Newfinal_gray(row,col),Newfinal_bin1(row,col-1));
        end
    end
    %----------------- converting the binary sequence into real values
    for col = 1:iteration
        GON(row,1) = (GON(row,1)+(Newfinal_bin1(row,col)*(2^(-(col)))));
    end

end

%-----Limiting the bit accuracy of the initial conditions up to 8 bit

Data_set_8bit = Newfinal_result(:,1).*(2^power);
Data_set_8bit = floor(Data_set_8bit);
Data_set_8bit = Data_set_8bit.*(2^(-power));

% histogram(Data_set_8bit(:,1),(2^power),'EdgeColor','none','FaceColor','yellow');
%
% % set(gca,'xlim',[0 1]);
% set(gca,'ylim',[0 2]);
% % axis square;

% histogram(Data_set_8bit(:,1),256);
% hold on
% histogram(GON(:,1),256);
```

# Appendix 2.2

MATLAB codes for bifurcation diagrams are presented.

## Appendix 2.2.1

**Code for generating bifurcation diagram of Bitshift Map (BM)**

```
pre_trap = 100;    % iterations to ensure point enters trapping region
trap = 80;    % iterations used for generating bifurcation
x = zeros(trap,1);    % array to store iterates for each parameter value

for r = 0.5:0.00025:1    % running through parameter values from 0.5 to 1

    x(1) = r;    % initialise x at the maximum value

    % ----------initial iterates are eliminated---------------------------
    for n = 1:pre_trap    % bitshift map run but iterates not stored
        if x(1) <= 0.5
            x(1) = 2*r*x(1);
        elseif x(1) > 0.5
            x(1) = (2*r*x(1))-1;
        end
    end

    % ----------bifurcation diagram generated-----------------------------
    for n = 1:trap-1    % bitshift map run and iterates stored for plotting
```

```
        if x(n) <= 0.5
            x(n+1) = 2*r*x(n);
        elseif x(n) > 0.5
            x(n+1) = (2*r*x(n))-1;
        end
    end

    % ---------iterates plotted for specific parameter--------------------
%     plot(r*ones(trap,1), x, 'k.', 'markersize', 3);
    plot(r*ones(trap,1), x, 'k.', 'markersize', 7); % increased markersize
    hold on;
end

% -------------plot limits and axes---------------------------------------
xlabel('Parameter (μ)');
ylabel('Iterates (x_n)');
% set(gca, 'xlim', [0.5 1]);
set(gca, 'xlim', [0.999 1]);   % to observe a narrow region of parameter
axis square;
hold off;
```

## Appendix 2.2.2

### Code for generating bifurcation diagram of Logistic Map

```
pre_trap = 100;    % iterations to ensure point enters trapping region
trap = 80;    % iterations used for generating bifurcation
x = zeros(trap,1);    % array to store iterates for each parameter value

for r = 0.5:0.00025:1    % running through parameter values from 0.5 to 1

    x(1) = r;    % initialise x at the maximum value

    % ----------initial iterates are eliminated---------------------------
    for n = 1:pre_trap    % logistic map run but iterates not stored
            x(1) = 4*r*x(1)*(1 - x(1));
    end

    % ----------bifurcation diagram generated-----------------------------
    for n = 1:trap-1    % logistic map run and iterates stored for plotting
        x(n+1) = 4*r*x(n)*(1 - x(n));
    end

    % ---------iterates plotted for specific parameter--------------------
    plot(r*ones(trap,1), x, 'k.', 'markersize', 3);
    hold on;
end

% -------------plot limits and axes---------------------------------------
xlabel('Parameter (μ)');
```

```
ylabel('Iterates (x_n)');
set(gca, 'xlim', [0.5 1]);
axis square;
hold off;
```

## Appendix 2.2.3

### Code for generating bifurcation diagram of Tent Map

```
pre_trap = 100;    % iterations to ensure point enters trapping region
trap = 80;    % iterations used for generating bifurcation
x = zeros(trap,1);    % array to store iterates for each parameter value

for r = 0.5:0.00025:1    % running through parameter values from 0.5 to 1

    x(1) = r;    % initialise x at the maximum value

    % ----------initial iterates are eliminated---------------------------
    for n = 1:pre_trap    % tent map run but iterates not stored
        if x(1) <= 0.5
            x(1) = 2*r*x(1);
        elseif x(1) > 0.5
            x(1) = 2*r*(1 - x(1));
        end
    end

    % ----------bifurcation diagram generated-----------------------------
    for n = 1:trap-1    % tent map run and iterates stored for plotting
        if x(n) <= 0.5
            x(n+1) = 2*r*x(n);
        elseif x(n) > 0.5
            x(n+1) = 2*r*(1 - x(n));
        end
    end

    % ---------iterates plotted for specific parameter--------------------
    plot(r*ones(trap,1), x, 'k.', 'markersize', 3);
    hold on;
end

% -------------plot limits and axes---------------------------------------
xlabel('Parameter (μ)');
ylabel('Iterates (x_n)');
set(gca, 'xlim', [0.5 1]);
axis square;
hold off;
```

# Appendix 2.3

MATLAB codes for the initial condition estimation are presented.

## Appendix 2.2.1

**Code for estimating initial condition through interval arithmetic**

```
format long
iteration = 12;               % setting number of itertions
Parameter_Mu = slopeNew/2;    % setting parameter for estimation

A = 0;                   % initialising lower bound
B = 0;                   % initialising upper bound
Delta = 0;               % scaled interval size initialised
l = 0;                   % size of the interval initialised
alpha = 0;               % odd even counter variable initialised
% N = 41;
% Symbols_Gray = xlsread('30bit_real_symbols_IPfreq10mHz_sampfreq_0.8mHz.xlsx');
Symbols_Gray = Newfinal_gray;  % copying generated grey code for estimation
X0_Dash_Array = zeros(N,1); % estimated initial condition array initialised
Diff = zeros(N,1);   % error or difference between the actual and
                      % estimated initial condition
X0_Dash = 0;       % single initial condition estimate variable initialised

for j = 1:N      % for N initial conditions
        for i = 1:iteration % for i iteration of each initial condition
```

```
          alpha = alpha + Symbols_Gray(j,i); % count number of 1s odd/even
          if i == 1                 % if the first symbol
             if Symbols_Gray(j,1) == 1 % is 1 then the primary half interval
                A = 0.5;           % is mirrored with lower bound = 0.5
                B = 0;              % and upper bound = 0
             else
                A = 0;             % other wise keeping primary half
                B = 0.5;            % unmirrored
             end
          else
             if rem(alpha,2) == 0 % if no. of '1's in the sequence is even
                A = A;            % lower bound unchanged
                B = A + Delta;    % upper bound shifted to lower bound +
                                    %scaled interval size
             else                 % if no. of '1's in the sequence is odd
                A = B - Delta;    % lower bound is shifted to upperbound -
                                    % delta
                B = B;            % upper bound is unchanged
             end
          end
          l = B - A;     % determine the length of newly formed interval
          Delta = l/(2*Parameter_Mu); % size of the interval scaled
                                       % proportional to mu
        end

% first symbol is not due to the result of TM iteration therefore orienting
% the final estimated point is necessary and therefore scaled accordingly
% and again the interval is unmirrored for the range 0.5-1 (with first
% symbol as 1)
        if rem(alpha,2) == 0    % if no. of '1's in the sequence is even
           if Symbols_Gray(j,1) == 1 % if the first symbol is 1
```

```
            X0_Dash = 1 - (A/Parameter_Mu); % unmirror the interval
                                             %  and scale down by mu
         else                      % if the first symbol is 0
            X0_Dash = (A/Parameter_Mu); % leave the orientation unhanged
                                         % scale down by mu
         end
      else
         if Symbols_Gray(j,1) == 1
            X0_Dash = 1 - (B/Parameter_Mu);
         else
            X0_Dash = (B/Parameter_Mu);
         end
      end
      X0_Dash_Array(j,1) = X0_Dash;     % store the estimated result
      A = 0;                            % reset all variables for the
      B = 0;                              % for the next new estimation
      Delta = 0;
      l = 0;
      alpha = 0;
      X0_Dash = 0;
end


Diff(:,1)= (Newfinal_result(:,1) - X0_Dash_Array(:,1))*100;
hold on
plot(Newfinal_result(:,1),Diff(:,1),'kO-','Markersize',7,'markerfacecolor',[0,0,0]);
% X0_Dash_Array(:,:) = X0_Dash_Array(:,:).*2^iteration;
% X0_Dash_Array(:,:) = floor(X0_Dash_Array(:,:));
% X0_Dash_Array(:,:) = X0_Dash_Array(:,:)./2^iteration;
% Error = Data_set_8bit - X0_Dash_Array;
% figure
% subplot(2,1,1)       % add first plot in 2 x 1 grid
% histogram(GON(:,1),(2^power),'EdgeColor','k','FaceColor','r');
% title('conventionally corrected (bin to dec)')
% set(gca,'xlim',[0 1]);
% set(gca,'ylim',[0 4]);
% % axis square;
% % hold on
%
%
% subplot(2,1,2)       % add second plot in 2 x 1 grid
% histogram(X0_Dash_Array(:,1),(2^power),'EdgeColor','k','FaceColor','b');
% title('correction algorithm with limited bit precision')
set(gca,'xlim',[0 1]);
% set(gca,'ylim',[-0.02 0.02]);
% axis square;
% % hold on
```

## Appendix 2.2.2

### Code for estimating initial condition through deviation adjustment

```
format long

power = 8;                      % power of 2 to generate number of divisions
first = 0;                      % if else identifier
second = 0;                     % if else identifier

Gain = 1.905242919921875;       % Gain used for estimation
iteration = 16;                 % number of bits considered for estimation
Bit_Accuracy = iteration;       % Bit accuracy set to number of iterations

% Newfinal_gray = zeros(N,iteration);   % stores all op for all steps
Gray_inverse = zeros(N,iteration);     % stores the inverted GON
Newfinal_bin = zeros(N,iteration);     % stores binary of Newfinal_gray
GON = zeros(N,1);                      % binary to decimal array initialise
INT = zeros(N,1);                      % binary to integer

Weighting = zeros(1,iteration); % A,B,C,...ordered by position of iterates
Eq1D = zeros(1,iteration);      % equation array
Equation = zeros(N,iteration);   % stores equation coefficients
                                 % corresponding to each row
Sum_Equation = zeros(N,1);       % Solved resut of equation array
Estim_init_con = zeros(N,1);     % Estimated initial condition array
```

```
%----------weighting register definition-------------------
for col = 1:iteration
    Weighting(1,col) = (2^15)*((Gain^(-col)) - (2^(-col)));
                % scaling up the weights by 15 (or number of iterations)
end

%-----G->B->D----LSB first---------------------------------
for row = 1:N

%-----------gray to binary---------------------------------
    for col = 1:iteration
        if col == 1
            Newfinal_bin(row,col) = Newfinal_gray(row,col);
        elseif col > 1
            Newfinal_bin(row,col) = bitxor(Newfinal_gray(row,col),Newfinal_bin(row,col-1));
        end
    end

%-----------binary to decimal------------------------------
    for col = 1:iteration
        GON(row,1) = (GON(row,1)+(Newfinal_bin(row,col)*(2^(-(col)))));
    end
    INT(row,1) = (2^15)*(GON(row,1)); % real valued decimals are converted
                                      % to integer

%-----------inverting gray code - LSB first----------------
    for col = 1:iteration
        Gray_inverse(row,iteration-col+1) = Newfinal_gray(row,col);
    end

end
```

```
%--------generating difference wrt gray code (LSB first)-------
for row = 1:N                  % for each input of the data set
  for col = 1:iteration        % for each symbol in the sequence
    if first==0 && Gray_inverse(row,col)==0 % no operation on the fist bit
    end
    if first==1             % after the first '1' is detected
        if Gray_inverse(row,col) == 0 && second == 0 %if next symbol is '0'
            Eq1D(1,1) = 1;          % fill first cell of eqn array with '1'
        elseif Gray_inverse(row,col) == 1 && second == 0
            Eq1D(1,1) = -1;      %if next symbol is '1' then fill '-1'
        end

        if second == 1           % process starts for filling the Eq1D reg
            Eq1D(1,2:iteration) = Eq1D(1,1:iteration-1); % shift cell by 1
            Eq1D(1,1) = 0;                   % clear first cell
            if Gray_inverse(row,col-1) == 1
                Eq1D(1) = 1;   % if previous symbol is 1 fillup 1 in the
                                % current cell
            end
            if Gray_inverse(row,col) == 1
                Eq1D = -Eq1D;  % if current symbol is 1 then change sign
                                % of the entire equation
            end
        end
        second = 1;               % flag to indicate that the bit (2nd bit)
                                   % after the first '1' is reached
    end
    if Gray_inverse(row,col)==1
        first = 1;                  % first '1' has been detected
    end
  end
  Equation(row,:) = Eq1D;   % copy the Eq1D into the matrix of equations
  Eq1D(1,:) = 0;            % reset Eq1D for next row
  second = 0;               % reset second for next row
  first=0;                  % reset first for next row
end

%--------estimating the initial condition from equation--------------------
for row = 1:N
    for col = 1:iteration
         Equation(row,col) = Weighting(1,col) * Equation(row,col);
    end
    Sum_Equation(row,1) = sum(Equation(row,:));
    Estim_init_con(row,1) = (Sum_Equation(row,1) + INT(row,1))*(2^(-15));
end

% --------------- Bit-accuracy limited to iterations ----------------------

Estim_init_Limit_bit = Estim_init_con(:,1).*(2^Bit_Accuracy);
Estim_init_Limit_bit = floor(Estim_init_Limit_bit);
Estim_init_Limit_bit = Estim_init_Limit_bit.*(2^(-Bit_Accuracy));
Error = (Data_set_8bit - Estim_init_Limit_bit)*100;



% --------------- Plotting styles and techniques --------------------------
figure
subplot(2,1,1)       % add first plot in 2 x 1 grid
histogram(GON(:,1),(2^power),'EdgeColor','k','FaceColor','r');
title('conventionally corrected (bin to dec)')
% set(gca,'xlim',[0 1]);
```

```
%%set(gca,'ylim',[0 4]);
% axis square;
% hold on


subplot(2,1,2)       % add second plot in 2 x 1 grid
histogram(Estim_init_Limit_bit(:,1),(2^power),'EdgeColor','k','FaceColor','b');
title('correction algorithm with limited bit precision')
% set(gca,'xlim',[1 256]);
%%set(gca,'ylim',[0 4]);
% axis square;
% hold on

% subplot(3,1,3)       % add second plot in 2 x 1 grid
% histogram(Estim_init_Limit_bit(:,1),256,'EdgeColor','k','FaceColor','g');
% title('correction algorithm')
% % set(gca,'xlim',[1 256]);
% set(gca,'ylim',[0 4]);
% % axis square;
% % hold on

%
figure
% subplot(2,1,1)       % add first plot in 2 x 1 grid
plot(Data_set_8bit(:,1),Error(:,1));
% title('error using 12 bits')
% set(gca,'xlim',[1 256]);
% % set(gca,'ylim',[-0.005 0.005]);
% axis square;
% % hold on
% % plot(Data_set_8bit(:,1),pos_half_LSB(:,1));
% % plot(Data_set_8bit(:,1),neg_half_LSB(:,1));
% % plot(Data_set_8bit(:,1),pos_full_LSB(:,1));
% % plot(Data_set_8bit(:,1),neg_full_LSB(:,1));
% % hold off;
%
% subplot(2,1,2)       % add second plot in 2 x 1 grid
% plot(Data_set_8bit(:,1),Error(:,2));
% title('error using 13 bits')
% % set(gca,'xlim',[1 256]);
% set(gca,'ylim',[-0.0003 0.0008]);
% % axis square;
% hold on
% plot(Data_set_8bit(:,1),pos_half_LSB(:,2));
% plot(Data_set_8bit(:,1),neg_half_LSB(:,2));
% plot(Data_set_8bit(:,1),pos_full_LSB(:,2));
% plot(Data_set_8bit(:,1),neg_full_LSB(:,2));
% hold off;
%
% figure
% subplot(3,1,1)       % add second plot in 2 x 1 grid
% plot(Data_set_8bit(:,1),Error(:,3));
% title('error using 14 bits')
% % set(gca,'xlim',[1 256]);
% set(gca,'ylim',[-0.0003 0.0004]);
% % axis square;
% hold on
% plot(Data_set_8bit(:,1),pos_half_LSB(:,3));
% plot(Data_set_8bit(:,1),neg_half_LSB(:,3));
% plot(Data_set_8bit(:,1),pos_full_LSB(:,3));
% plot(Data_set_8bit(:,1),neg_full_LSB(:,3));
% hold off;
```

```
%

% subplot(3,1,2)       % add second plot in 2 x 1 grid
% plot(Data_set_8bit(:,1),Error(:,4));
% title('error using 15 bits')
% % set(gca,'xlim',[1 256]);
% set(gca,'ylim',[-0.0003 0.0004]);
% % axis square;
% hold on
% plot(Data_set_8bit(:,1),pos_half_LSB(:,4));
% plot(Data_set_8bit(:,1),neg_half_LSB(:,4));
% plot(Data_set_8bit(:,1),pos_full_LSB(:,4));
% plot(Data_set_8bit(:,1),neg_full_LSB(:,4));
% hold off;
%
% subplot(3,1,3)       % add second plot in 2 x 1 grid
% plot(Data_set_8bit(:,1),Error(:,5));
% title('error using 16 bits')
% % set(gca,'xlim',[1 256]);
% set(gca,'ylim',[-0.0003 0.0004]);
% % axis square;
% hold on
% plot(Data_set_8bit(:,1),pos_half_LSB(:,5));
% plot(Data_set_8bit(:,1),neg_half_LSB(:,5));
% plot(Data_set_8bit(:,1),pos_full_LSB(:,5));
% plot(Data_set_8bit(:,1),neg_full_LSB(:,5));
% hold off;

% subplot(2,2,3)       % add second plot in 2 x 1 grid
% plot(Data_set_8bit(:,1),Error(:,6));
% title('error using 15 bits')
% % set(gca,'xlim',[1 256]);
% set(gca,'ylim',[-0.002 0.002]);
% % axis square;
% hold on
% plot(Data_set_8bit(:,1),pos_half_LSB(:,6));
% plot(Data_set_8bit(:,1),neg_half_LSB(:,6));
% plot(Data_set_8bit(:,1),pos_full_LSB(:,6));
% plot(Data_set_8bit(:,1),neg_full_LSB(:,6));
% hold off;
% %
% % %--------------
% %
% subplot(2,2,4)       % add second plot in 2 x 1 grid
% plot(Data_set_8bit(:,1),Error(:,7));
% title('error using 16 bits')
% % set(gca,'xlim',[1 256]);
% set(gca,'ylim',[-0.002 0.002]);
% % axis square;
% hold on
% plot(Data_set_8bit(:,1),pos_half_LSB(:,7));
% plot(Data_set_8bit(:,1),neg_half_LSB(:,7));
% plot(Data_set_8bit(:,1),pos_full_LSB(:,7));
% plot(Data_set_8bit(:,1),neg_full_LSB(:,7));
% hold off;
%
% subplot(3,1,2)       % add second plot in 2 x 1 grid
% plot(Data_set_8bit(:,1),Error(:,8));
% title('error using 15 bits')
% % set(gca,'xlim',[1 256]);
% set(gca,'ylim',[-0.002 0.004]);
% % axis square;
% hold on
% plot(Data_set_8bit(:,1),pos_half_LSB(:,8));
% plot(Data_set_8bit(:,1),neg_half_LSB(:,8));
% plot(Data_set_8bit(:,1),pos_full_LSB(:,8));
% plot(Data_set_8bit(:,1),neg_full_LSB(:,8));
% hold off;
%
% subplot(3,1,3)       % add second plot in 2 x 1 grid
% plot(Data_set_8bit(:,1),Error(:,9));
% title('error using 16 bits')
% % set(gca,'xlim',[1 256]);
% set(gca,'ylim',[-0.002 0.004]);
% % axis square;
% hold on
% plot(Data_set_8bit(:,1),pos_half_LSB(:,9));
% plot(Data_set_8bit(:,1),neg_half_LSB(:,9));
% plot(Data_set_8bit(:,1),pos_full_LSB(:,9));
% plot(Data_set_8bit(:,1),neg_full_LSB(:,9));
% hold off;
```

# APPENDIX 3

Input dataset generated from the implemented hardware used for the initial condition estimation using both the algorithms are presen ted. 30 symbols were collected, of which 16 symbols have been used.

Table A. 1 Input signal with the symbolic sequences

| Input signal | Symbolic sequence | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.001953125000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.005859375000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0.009765625000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0.013671875000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 0.017578125000 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0.021484375000 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0.025390625000 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0.029296875000 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0.033203125000 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0.037109375000 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0.041015625000 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0.044921875000 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.048828125000 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0.052734375000 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0.056640625000 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0.060546875000 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0.064453125000 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0.068359375000 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0.072265625000 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0.076171875000 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0.080078125000 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0.083984375000 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0.087890625000 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0.091796875000 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0.095703125000 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0.099609375000 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0.103515625000 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0.107421875000 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0.111328125000 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0.115234375000 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0.119140625000 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0.123046875000 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0.126953125000 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0.130859375000 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0.134765625000 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0.138671875000 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0.142578125000 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0.146484375000 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |

| | |
|---|---|
| 0.150390625000 | 0 0 0 1 0 0 1 1 1 0 1 0 1 0 0 1 1 1 0 1 1 1 0 0 1 0 1 0 0 1 |
| 0.154296875000 | 0 0 1 1 0 0 1 1 0 1 0 1 1 0 1 1 0 0 1 0 1 1 0 1 0 1 1 0 1 1 |
| 0.158203125000 | 0 0 1 1 0 0 1 0 1 0 0 1 1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 1 1 0 |
| 0.162109375000 | 0 0 1 1 0 0 1 0 0 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 0 0 |
| 0.166015625000 | 0 0 1 1 0 1 1 0 1 1 1 1 1 1 0 1 0 1 1 1 0 1 1 1 0 1 0 0 1 0 |
| 0.169921875000 | 0 0 1 1 0 1 1 1 1 1 1 0 0 1 1 1 1 0 1 1 0 1 0 1 0 1 0 1 0 1 |
| 0.173828125000 | 0 0 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 |
| 0.177734375000 | 0 0 1 1 0 1 0 1 1 1 0 0 1 0 1 0 1 1 1 0 1 1 0 0 1 1 1 0 1 1 |
| 0.181640625000 | 0 0 1 1 0 1 0 0 1 0 1 1 0 0 1 0 1 1 1 0 1 1 1 1 0 0 1 1 0 1 |
| 0.185546875000 | 0 0 1 1 1 1 0 0 1 0 1 0 1 1 0 0 1 0 1 1 0 0 1 0 0 1 1 1 0 0 |
| 0.189453125000 | 0 0 1 1 1 1 0 1 1 1 1 1 0 1 0 1 0 0 1 1 1 1 0 1 0 1 0 0 1 0 |
| 0.193359375000 | 0 0 1 1 1 1 1 1 0 0 1 1 0 1 0 1 1 1 1 1 0 1 0 1 1 0 0 1 0 1 |
| 0.197265625000 | 0 0 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 1 0 1 0 1 1 1 |
| 0.201171875000 | 0 0 1 1 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 0 1 0 1 1 1 0 1 0 1 0 |
| 0.205078125000 | 0 0 1 1 1 1 1 0 0 1 1 0 1 0 0 1 0 1 0 0 1 0 1 0 1 1 0 1 0 1 |
| 0.208984375000 | 0 0 1 1 1 0 1 0 1 0 1 0 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 |
| 0.212890625000 | 0 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 1 |
| 0.216796875000 | 0 0 1 1 1 0 1 1 0 0 1 1 0 1 1 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 |
| 0.220703125000 | 0 0 1 1 1 0 0 1 1 1 1 1 0 1 0 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 |
| 0.224609375000 | 0 0 1 0 1 0 0 1 0 1 0 0 1 1 1 0 1 0 0 1 1 1 0 1 1 0 0 1 0 1 |
| 0.228515625000 | 0 0 1 0 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 0 1 1 1 0 1 0 0 1 0 1 |
| 0.232421875000 | 0 0 1 0 1 0 1 1 1 0 1 0 1 1 1 0 0 1 0 0 1 0 0 1 0 1 0 0 1 1 |
| 0.236328125000 | 0 0 1 0 1 0 1 0 1 1 0 1 0 0 1 1 1 1 1 0 0 1 1 1 1 0 1 1 0 0 |
| 0.240234375000 | 0 0 1 0 1 1 1 0 0 1 0 0 1 0 1 0 0 1 1 0 1 0 0 1 1 1 0 0 1 1 |
| 0.244140625000 | 0 0 1 0 1 1 1 1 1 0 0 1 0 1 0 0 1 0 1 0 1 0 0 1 0 1 0 0 1 0 |
| 0.248046875000 | 0 0 1 0 1 1 1 1 0 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 |

| | |
|---|---|
| 0.251953125000 | 0 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 1 0 1 1 |
| 0.255859375000 | 0 0 1 0 1 1 0 1 1 0 1 1 1 1 0 1 0 0 1 0 0 1 0 1 0 1 1 1 1 1 |
| 0.259765625000 | 0 0 1 0 1 1 0 0 1 1 0 1 0 1 0 1 1 1 1 1 1 0 1 0 0 1 1 1 0 1 |
| 0.263671875000 | 0 0 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 0 1 1 |
| 0.267578125000 | 0 0 1 0 0 1 0 1 0 1 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 1 1 |
| 0.271484375000 | 0 0 1 0 0 1 1 1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 0 |
| 0.275390625000 | 0 1 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 1 1 1 1 1 0 1 1 1 1 1 1 0 |
| 0.279296875000 | 0 1 1 0 0 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 0 1 1 1 |
| 0.283203125000 | 0 1 1 0 0 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 0 1 0 1 1 1 1 1 1 0 |
| 0.287109375000 | 0 1 1 0 0 1 0 1 0 1 0 1 1 1 1 1 0 0 1 1 1 0 1 1 1 1 1 0 1 0 |
| 0.291015625000 | 0 1 1 0 0 1 0 1 1 0 0 1 1 1 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 |
| 0.294921875000 | 0 1 1 0 1 1 0 0 1 1 1 0 1 0 1 1 0 0 1 1 0 1 0 1 1 1 0 1 1 0 |
| 0.298828125000 | 0 1 1 0 1 1 0 1 1 1 1 0 1 0 1 0 1 1 1 1 1 1 1 0 1 0 1 0 1 1 |
| 0.302734375000 | 0 1 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 1 1 0 0 1 1 1 1 0 1 0 1 0 |
| 0.306640625000 | 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0 1 0 0 1 |
| 0.310546875000 | 0 1 1 0 1 1 1 0 1 0 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1 0 1 |
| 0.314453125000 | 0 1 1 0 1 1 1 0 0 1 1 1 1 1 0 1 1 0 1 1 0 0 1 0 1 0 1 1 1 1 |
| 0.318359375000 | 0 1 1 0 1 0 1 0 1 1 1 0 0 1 0 1 1 1 1 1 0 0 1 1 1 1 0 0 1 0 |
| 0.322265625000 | 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 |
| 0.326171875000 | 0 1 1 0 1 0 1 1 0 0 1 1 1 1 1 0 1 0 1 0 1 1 1 1 0 0 1 0 1 1 |
| 0.330078125000 | 0 1 1 0 1 0 0 1 1 1 1 1 0 0 1 1 0 1 0 1 1 1 1 0 1 0 1 0 0 1 |
| 0.333984375000 | 0 1 1 1 1 0 0 1 1 1 0 0 1 0 1 0 1 0 1 1 1 1 0 1 0 0 1 1 1 1 |
| 0.337890625000 | 0 1 1 1 1 0 1 1 0 1 1 0 0 1 1 1 1 0 0 1 1 1 1 1 0 1 1 0 1 0 |
| 0.341796875000 | 0 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 0 0 1 0 0 1 0 1 0 1 0 0 1 1 |
| 0.345703125000 | 0 1 1 1 1 0 1 0 1 1 1 1 1 1 1 0 1 1 0 1 0 0 1 1 1 1 0 1 1 1 |
| 0.349609375000 | 0 1 1 1 1 1 1 0 0 1 0 1 0 1 0 1 0 0 1 0 1 1 1 1 1 0 0 1 0 0 |

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.353515625000 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0.357421875000 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 0.361328125000 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0.365234375000 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0.369140625000 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0.373046875000 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0.376953125000 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0.380859375000 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0.384765625000 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0.388671875000 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0.392578125000 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0.396484375000 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0.400390625000 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0.404296875000 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0.408203125000 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0.412109375000 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0.416015625000 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0.419921875000 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0.423828125000 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0.427734375000 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0.431640625000 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0.435546875000 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0.439453125000 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0.443359375000 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0.447265625000 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0.451171875000 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |

| | |
|---|---|
| 0.455078125000 | 0 1 0 1 1 0 1 0 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 0 1 1 |
| 0.458984375000 | 0 1 0 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 0 1 0 1 0 0 1 0 1 0 1 1 |
| 0.462890625000 | 0 1 0 1 1 0 1 1 0 0 1 1 1 0 1 1 1 0 1 0 0 1 0 1 1 1 1 0 0 1 |
| 0.466796875000 | 0 1 0 1 1 0 0 1 1 1 0 1 0 0 1 0 0 1 0 1 1 1 1 1 1 1 1 0 0 1 |
| 0.470703125000 | 0 1 0 0 1 0 0 1 1 1 0 0 1 0 1 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 |
| 0.474609375000 | 0 1 0 0 1 0 1 1 0 1 1 0 0 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 0 1 |
| 0.478515625000 | 0 1 0 0 1 0 1 1 1 0 1 0 0 1 0 1 0 1 1 1 1 1 1 0 1 1 0 1 1 0 |
| 0.482421875000 | 0 1 0 0 1 0 1 0 1 1 1 1 0 1 0 1 0 0 1 1 1 1 0 0 1 1 1 1 0 1 |
| 0.486328125000 | 0 1 0 0 1 1 1 0 0 1 0 1 1 0 1 1 0 1 1 0 1 0 1 1 1 0 1 1 1 1 |
| 0.490234375000 | 0 1 0 0 1 1 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1 1 1 0 1 0 1 1 0 0 |
| 0.494140625000 | 0 1 0 0 1 1 1 1 0 1 0 1 1 0 1 0 1 1 1 0 1 1 0 0 1 0 1 0 1 0 |
| 0.498046875000 | 0 1 0 0 1 1 0 1 0 1 1 1 1 0 1 1 1 0 1 0 1 0 0 1 0 1 0 1 0 1 |
| 0.501953125000 | 1 1 0 0 1 1 0 1 0 1 1 1 0 1 0 0 1 1 1 0 1 0 1 0 0 1 1 1 1 0 |
| 0.505859375000 | 1 1 0 0 1 1 1 1 0 1 0 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 0 |
| 0.509765625000 | 1 1 0 0 1 1 1 1 1 0 0 1 0 1 1 1 0 0 1 1 0 1 0 0 1 0 1 1 1 1 |
| 0.513671875000 | 1 1 0 0 1 1 1 0 0 1 0 0 1 1 1 1 1 0 0 1 1 1 0 1 0 1 1 0 0 1 |
| 0.517578125000 | 1 1 0 0 1 0 1 0 1 1 0 1 0 1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 |
| 0.521484375000 | 1 1 0 0 1 0 1 1 1 0 1 0 1 1 1 0 0 1 1 0 1 1 1 0 0 1 0 0 1 0 |
| 0.525390625000 | 1 1 0 0 1 0 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 1 1 1 |
| 0.529296875000 | 1 1 0 0 1 0 0 1 0 1 0 0 1 1 1 1 1 1 0 1 1 1 1 0 0 1 1 0 1 0 |
| 0.533203125000 | 1 1 0 1 1 0 0 1 1 1 1 1 0 1 1 1 0 0 1 0 1 1 0 1 1 1 1 1 0 1 |
| 0.537109375000 | 1 1 0 1 1 0 1 1 0 0 1 1 0 1 0 0 1 0 1 0 0 1 0 0 1 0 1 1 1 1 |
| 0.541015625000 | 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 0 1 0 0 1 |
| 0.544921875000 | 1 1 0 1 1 0 1 0 1 0 1 0 0 1 1 1 0 1 1 1 1 0 0 1 0 1 0 0 1 0 |
| 0.548828125000 | 1 1 0 1 1 1 1 0 0 1 1 1 1 0 1 0 0 1 0 1 0 1 1 1 1 0 0 1 0 0 |
| 0.552734375000 | 1 1 0 1 1 1 1 0 1 0 1 1 1 0 1 0 1 0 1 0 1 0 0 1 1 1 0 1 1 0 |

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.556640625000 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0.560546875000 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0.564453125000 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0.568359375000 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0.572265625000 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0.576171875000 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0.580078125000 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0.583984375000 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0.587890625000 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0.591796875000 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0.595703125000 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0.599609375000 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0.603515625000 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0.607421875000 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0.611328125000 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0.615234375000 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0.619140625000 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0.623046875000 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0.626953125000 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0.630859375000 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0.634765625000 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0.638671875000 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0.642578125000 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0.646484375000 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0.650390625000 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0.654296875000 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |

| | |
|---|---|
| 0.658203125000 | 1 1 1 1 1 0 1 1 1 0 1 0 1 0 0 1 0 1 1 0 0 1 0 1 0 1 0 0 1 0 |
| 0.662109375000 | 1 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 |
| 0.666015625000 | 1 1 1 1 1 0 0 1 0 1 0 1 1 0 1 0 0 1 0 0 1 1 0 1 0 0 1 1 1 1 |
| 0.669921875000 | 1 1 1 0 1 0 0 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 1 0 1 1 |
| 0.673828125000 | 1 1 1 0 1 0 0 1 0 0 1 1 1 0 1 0 1 0 0 1 0 1 0 1 1 0 0 1 0 1 |
| 0.677734375000 | 1 1 1 0 1 0 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 |
| 0.681640625000 | 1 1 1 0 1 0 1 0 1 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 0 1 1 1 1 1 |
| 0.685546875000 | 1 1 1 0 1 1 1 0 0 1 1 0 1 0 1 1 0 0 1 1 1 0 1 0 1 1 0 1 1 0 |
| 0.689453125000 | 1 1 1 0 1 1 1 0 1 0 1 0 1 1 0 1 0 0 1 1 1 1 1 0 1 0 1 1 0 0 |
| 0.693359375000 | 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 1 0 1 1 1 0 1 0 1 1 1 0 1 0 1 |
| 0.697265625000 | 1 1 1 0 1 1 1 1 0 0 1 1 1 1 0 0 1 0 0 1 0 0 1 1 0 1 0 0 1 1 |
| 0.701171875000 | 1 1 1 0 1 1 0 1 1 1 1 1 0 1 1 0 1 1 0 0 1 1 1 0 1 0 1 1 1 1 |
| 0.705078125000 | 1 1 1 0 1 1 0 0 1 0 1 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 1 0 |
| 0.708984375000 | 1 1 1 0 0 1 0 0 1 0 1 1 0 0 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 0 |
| 0.712890625000 | 1 1 1 0 0 1 0 1 1 1 0 0 1 1 1 0 1 1 0 1 1 0 1 0 0 1 1 1 0 1 |
| 0.716796875000 | 1 1 1 0 0 1 1 1 0 1 1 0 0 1 0 0 1 0 0 1 0 0 1 0 1 0 1 1 1 1 |
| 0.720703125000 | 1 1 1 0 0 1 1 1 1 0 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 |
| 0.724609375000 | 1 1 1 0 0 1 1 0 1 0 1 1 0 1 0 0 1 1 1 1 1 0 0 1 1 1 1 0 0 1 |
| 0.728515625000 | 1 0 1 0 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0 1 1 1 1 0 1 1 0 1 |
| 0.732421875000 | 1 0 1 0 0 1 0 1 0 0 1 0 1 1 0 0 1 0 0 1 0 1 0 1 1 1 0 1 1 0 |
| 0.736328125000 | 1 0 1 0 0 1 0 1 1 1 1 0 0 1 0 0 1 0 0 1 0 1 1 0 1 1 0 0 1 1 |
| 0.740234375000 | 1 0 1 0 0 1 0 0 1 1 1 1 1 1 1 1 0 1 1 0 0 1 0 1 0 1 1 1 0 1 |
| 0.744140625000 | 1 0 1 0 1 1 0 1 1 0 0 1 0 0 1 0 0 1 0 1 1 1 0 0 1 0 1 1 1 1 |
| 0.748046875000 | 1 0 1 0 1 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 1 1 0 0 1 0 0 1 0 |
| 0.751953125000 | 1 0 1 0 1 1 1 1 0 1 1 1 0 1 1 0 1 0 1 0 0 1 0 1 1 0 1 1 1 1 |
| 0.755859375000 | 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 1 1 1 0 1 1 1 0 1 0 1 1 0 1 |

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.759765625000 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0.763671875000 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0.767578125000 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0.771484375000 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0.775390625000 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0.779296875000 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0.783203125000 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0.787109375000 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0.791015625000 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0.794921875000 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0.798828125000 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0.802734375000 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0.806640625000 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0.810546875000 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0.814453125000 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 0.818359375000 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0.822265625000 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0.826171875000 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0.830078125000 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 0.833984375000 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0.837890625000 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0.841796875000 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0.845703125000 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0.849609375000 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0.853515625000 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0.857421875000 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |

| | |
|---|---|
| 0.861328125000 | 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 0 1 1 0 0 1 1 1 0 1 |
| 0.865234375000 | 1 0 0 1 0 1 1 1 0 1 0 0 1 1 0 1 0 1 1 1 0 0 1 1 1 1 0 1 1 0 |
| 0.869140625000 | 1 0 0 1 0 1 0 1 0 1 1 0 1 1 0 0 1 1 0 1 0 0 1 0 1 1 0 1 0 1 |
| 0.873046875000 | 1 0 0 1 0 1 0 1 1 0 1 0 0 1 0 0 1 1 1 1 0 1 1 1 0 1 0 1 0 1 |
| 0.876953125000 | 1 0 0 1 0 1 0 0 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 0 1 1 0 0 |
| 0.880859375000 | 1 0 0 1 1 1 0 1 1 0 0 1 0 0 1 1 1 1 1 1 1 1 1 0 1 0 1 1 0 0 |
| 0.884765625000 | 1 0 0 1 1 1 0 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 |
| 0.888671875000 | 1 0 0 1 1 1 1 1 0 1 1 1 0 1 1 0 1 0 1 1 1 0 0 1 0 0 1 1 0 1 |
| 0.892578125000 | 1 0 0 1 1 1 1 1 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 0 1 0 |
| 0.896484375000 | 1 0 0 1 1 1 1 0 1 1 0 1 1 0 0 1 1 0 1 0 1 1 1 0 1 0 1 0 1 1 |
| 0.900390625000 | 1 0 0 1 1 0 1 0 1 1 0 1 1 0 1 1 1 0 0 1 1 1 0 1 0 1 0 1 1 1 |
| 0.904296875000 | 1 0 0 1 1 0 1 1 1 0 1 1 0 1 0 1 1 1 1 0 0 1 1 1 1 1 0 1 0 1 |
| 0.908203125000 | 1 0 0 1 1 0 1 1 0 1 1 1 0 1 0 1 1 1 0 1 1 1 0 0 1 1 1 0 1 1 |
| 0.912109375000 | 1 0 0 1 1 0 0 1 0 1 0 1 0 1 1 1 1 0 1 1 1 1 1 1 0 1 1 0 1 0 |
| 0.916015625000 | 1 0 0 0 1 0 0 1 1 1 1 0 1 1 1 1 0 0 1 1 1 0 1 1 1 0 1 1 1 1 |
| 0.919921875000 | 1 0 0 0 1 0 0 1 0 0 1 0 1 1 0 1 0 1 0 1 1 1 1 1 1 1 0 0 1 0 |
| 0.923828125000 | 1 0 0 0 1 0 1 1 1 1 0 1 1 0 1 0 1 1 1 0 0 1 1 0 1 0 0 1 1 0 |
| 0.927734375000 | 1 0 0 0 1 0 1 0 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 |
| 0.931640625000 | 1 0 0 0 1 0 1 0 0 1 1 1 0 1 0 1 0 1 0 1 1 0 0 1 1 0 1 0 0 1 |
| 0.935546875000 | 1 0 0 0 1 1 1 0 1 1 1 0 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1 |
| 0.939453125000 | 1 0 0 0 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 1 1 0 1 1 0 1 1 1 1 1 |
| 0.943359375000 | 1 0 0 0 1 1 1 1 0 0 1 1 1 0 0 1 1 1 0 1 0 1 0 1 1 0 1 0 1 1 |
| 0.947265625000 | 1 0 0 0 1 1 0 1 1 1 0 1 0 1 0 0 1 0 1 1 1 1 0 1 0 0 1 0 1 1 |
| 0.951171875000 | 1 0 0 0 1 1 0 0 1 0 1 1 1 0 1 1 1 1 1 0 1 0 1 1 1 0 1 0 1 1 |
| 0.955078125000 | 1 0 0 0 0 1 0 0 1 0 1 1 1 0 1 0 1 1 1 1 1 1 0 1 1 1 1 0 0 1 |
| 0.958984375000 | 1 0 0 0 0 1 0 1 1 1 0 1 0 1 0 0 1 0 1 0 1 1 1 0 1 1 1 1 0 1 |

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.962890625000 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0.966796875000 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0.970703125000 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0.974609375000 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0.978515625000 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0.982421875000 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0.986328125000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0.990234375000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0.994140625000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0.998046875000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1.001953125000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# APPENDIX 4

Using the data from Appendix 3, the real valued trajectories and their corresponding symbolic sequences were plotted across the state space.

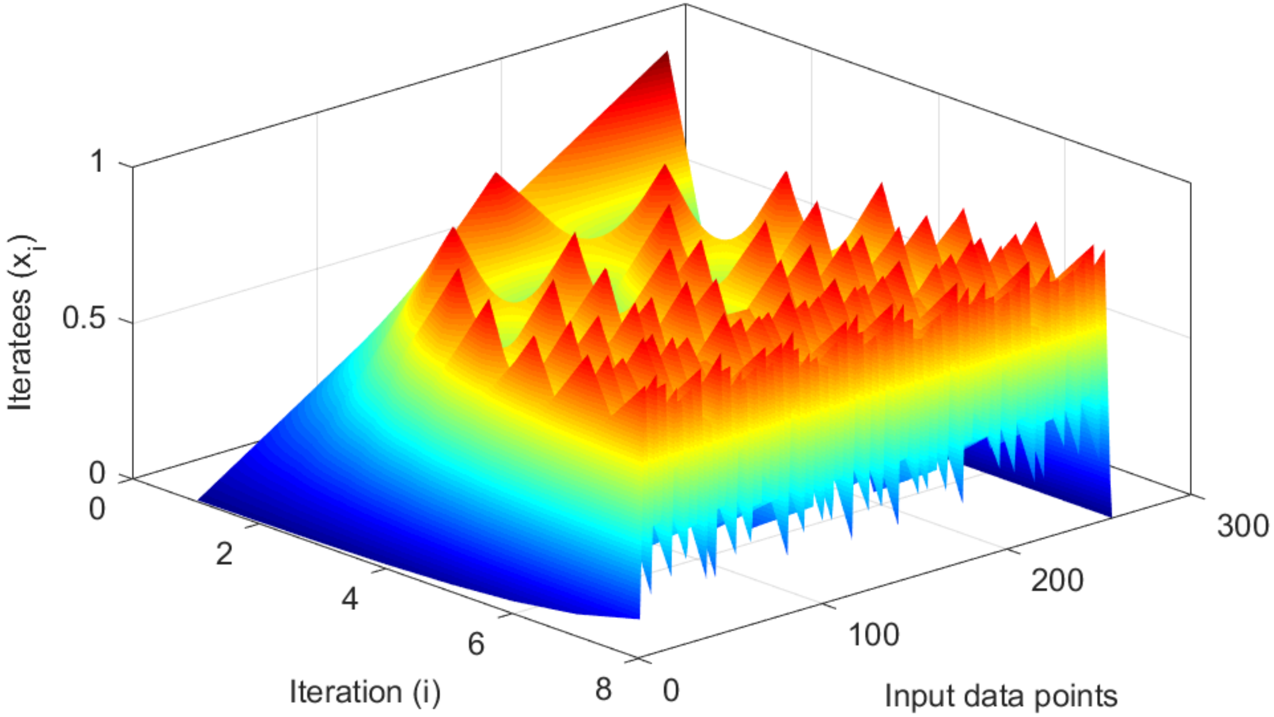


Fig. A. 1 3D view of the real iterates across the state space

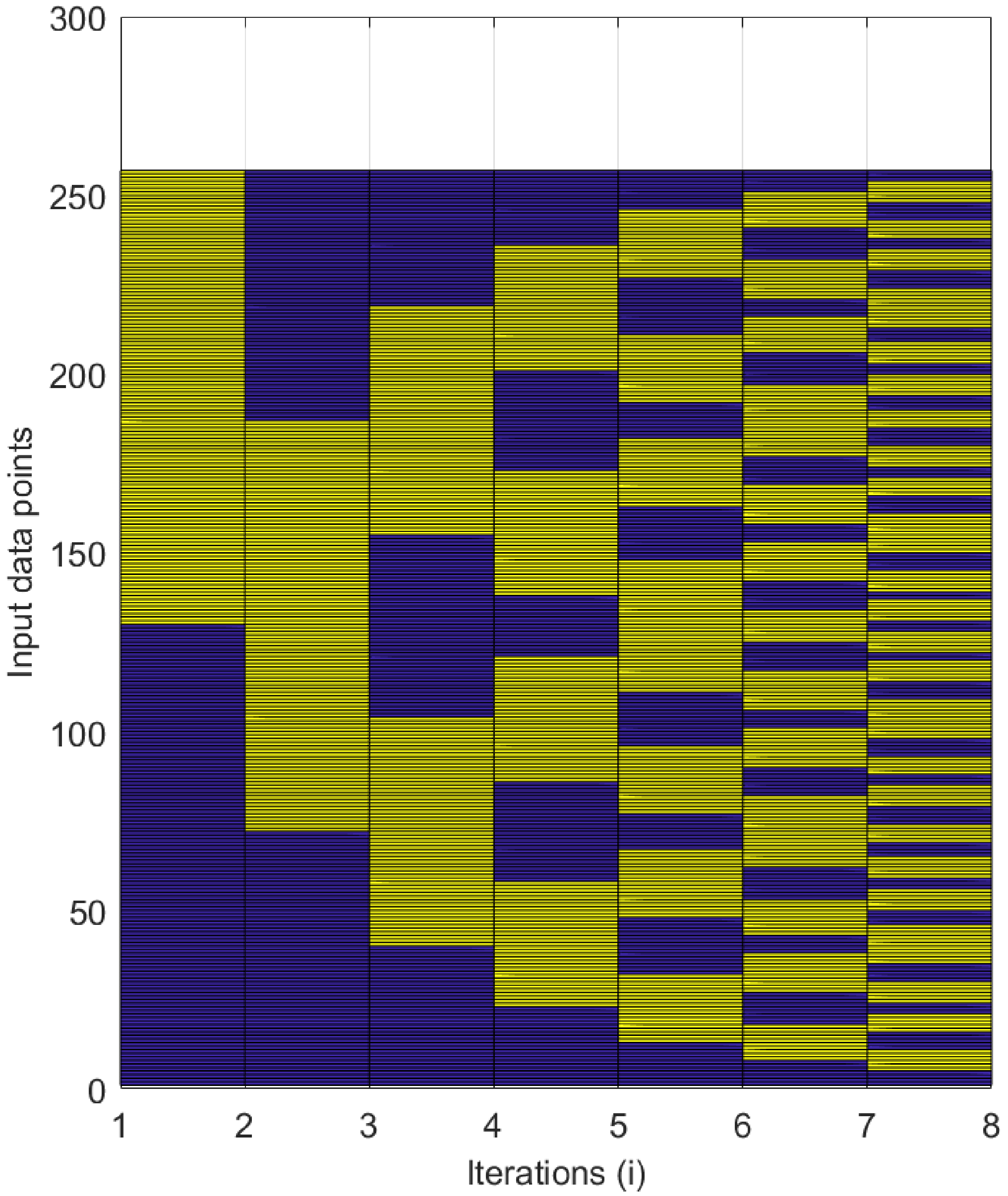


Fig. A. 2 Unequal interval partitioning of the real state space

# APPENDIX 5

The change in error over both parameter and number of symbols used to estimate the initial condition is viewed in a 3D plot. The plot combines the views of Fig. 5.6 and Fig. 5.7 in a surface plot.

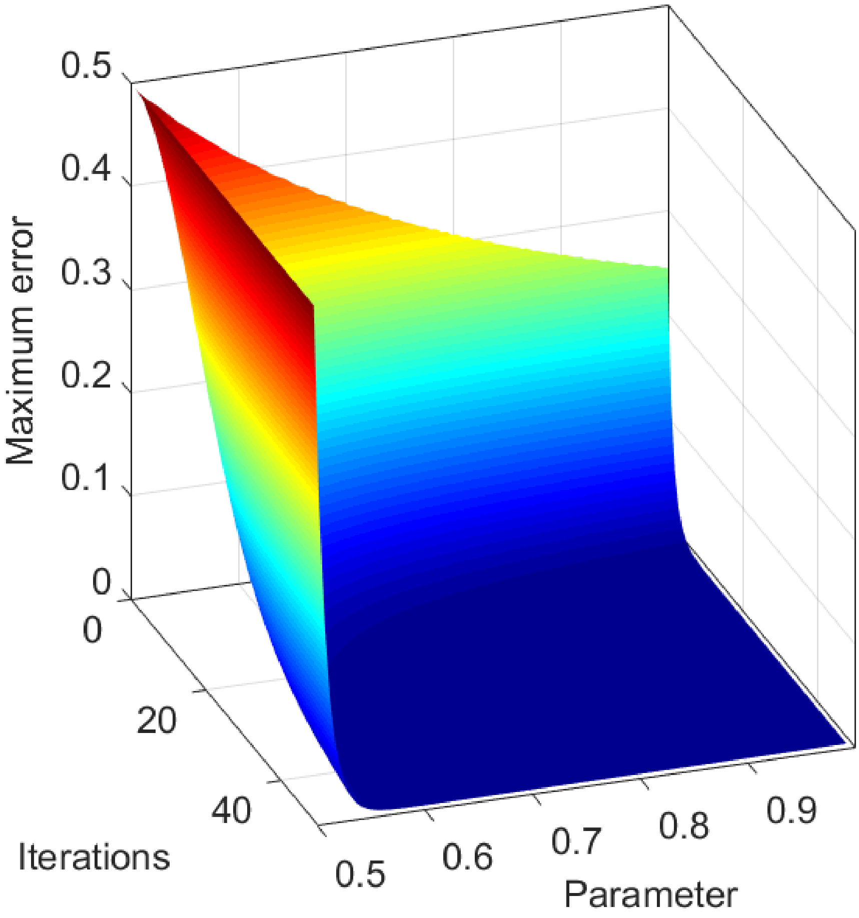


Fig. A. 3 A 3D view combining Fig. 6.6 and 6.7