University of Economics, Prague

Faculty of Informatics and Statistics

Department of Information and Knowledge Engineering

Study programme: Applied Informatics

Specialization: Knowledge Technology

Fractal application in data compression

MASTER THESIS

Student: Bc. Petr Dušák

Thesis supervisor: prof. RNDr. Jiří Ivánek, CSc.

Opponent :

2015

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze, 11. prosince 2015 . . . . . .

Bc. Petr Dušák

Abstrakt

Posláním Technology Transfer Programme Office je zvýšení prospěšnosti Evropské kosmické agentury pro obyvatelstvo, a to přenášením technologií vyvinutých pro vesmírný program. *Method and Apparatus for compressing time series*, volně přeloženo „Nástroj pro komprimaci časových řad“, je patentovaný kompresní algoritmus, jehož primárním cílem je komprimovat časové řady, které poskytují informace o stavu satelitů a kosmických sond. Je navržen tak, aby jeho výpočetní náročnost byla co nejnižší, protože výpočetní čas je na satelitech a sondách vzácnou komoditou. Patentovaný algoritmus je inspirován fraktály – metodami pro generování terénu. Konktrétně se jedná o metodu přesouvání středního bodu. Tato práce poskytuje základní přehled fraktálů, jejich aplikací a zabývá se modifikací patentovaného algoritmu. Cílem modifikace je dosažení vyšší komprese za cenu vyšší náročnosti na provedení komprese. Modifikovaný algoritmus je schopen dosahovat až o 25 % vyšší kompresi. Toto číslo je horní, empiricky naměřenou hodnotou. V rozsáhlém testu na telemetrických datech ze sondy Rosetta dosahoval modifikovaný algoritmus zlepšení přibližně 5 %.

Klíčová slova

Komprese dat, časové řady, fraktály

Abstract

The mission of the Technology Transfer Programme Office is to increase impact on a society by transferring technologies developed by the European Space Agency. *Method and Apparatus for compressing time series* is a patented compression algorithm designed to be efficient as its purpose is to run on deep space probes or satellites. The algorithm is inspired by a method for fractal terrain generation, namely the midpoint displacement algorithm. This work introduces fractals, their application and modifying the patented algorithm, in order to achieve greater compression. The modification lies in modifying the displacement mechanism. The modified algorithm is capable of reducing data up to 25 %, compared to the patented algorithm. The modification made the algorithm less efficient. In large-scale test, performed on Rosetta spacecraft telemetry, the modified algorithm achieved around 5 % higher compression.

Keywords

Data compression, time series, fractals

Table of contents

Preface

The basis of the thesis is a patent *Method and Apparatus for compressing time series* with which I worked during my internship at the European Space Agency (ESA) in Technology Transfer Office. One of my duties during the internship was an exploitation of ESA's software patents, mostly data compression. The invention is internally called *Fractal Resampling* and I will be using this designation as well.

During a discussion about *Fractal resampling*, when I was asking about statistical qualities, I was told that the algorithm could achieve better compression if certain modifications were made. I asked about some materials that would support this claim, but there were not any. I was told that with a time pressure to publish the patent, there was not enough time to test it. This thesis is set to answer the question if the patented algorithm can achieve better compression.

The thesis consists of three sections. The first section is dedicated to basic fractals and their most basic theory. The second chapter is devoted to application of fractals in the real world and how looking at things trough "fractal glasses" can improve traditional ways. The third is the last chapter and it is devoted to studying, testing and modifying the compression algorithm mentioned above.

1 Introduction to fractals

In this section, I will introduce the basics of fractals, ways to create them, some of their properties and give examples of some basic fractals. Specifically the Cantor set, the Koch curve, Julia sets and the Mandelbrot set.

A characteristic feature of fractals is a fine structure and self-similarity. It means that there are details at arbitrarily small scales and some fractals look the same at arbitrary scale. Complexity of fractals can be captured and described by a *fractal dimension*. It attempts to quantify complexity of fractals by measuring the rate at which increased detail becomes apparent. It indicates the complexity of the fractal and amount of space it occupies when viewed at high resolution. [1]

Fractals are a relatively new branch of mathematics and set theory and they found application in areas where methods of classical geometry and calculus could not cope with irregularity [2]. It was recognized early in the 20th century that studying irregularity or fragmentation cannot be satisfied with defining dimensions as a number of coordinates. [3]

1.1 Fractal definition

"A fractal is by definition a set for which the Hausdorff-Besicovitch dimension strictly exceeds the topological dimension." [3]

This is how Bernoit Mandelbrot defined a fractal in his book *The Fractal Geometry of Nature* from 1982.

Two types of dimensions were mentioned in the first quote. The Hausdorff-Besicovitch dimension, denoted as D, and the topological dimension, denoted as D_T. Both dimensions are in Euclidian space $\mathbb{R}^E$. In span of $\mathbb{R}^E$, $0 < D, D_T < E$. D_T is always an integer, but D does not have to be an integer. For all of Euclidian objects $D = D_T$. However most of the objects that were mentioned in the book that Mandelbrot wrote satisfied $D > D_T$ [1], but there was no term that would describe such objects. That led him to create the term *fractal* [3].

The topological dimension is the most common definition of a dimension. It is always expressed as a whole number. Points have $D_T = 0$, curves $D_T = 1$, surface of an object $D_T = 2$ and $D_T = 3$ is for spatial objects.

[1] Hurewicz and Wallman proved that the topological dimension cannot be greater than the Hausdorff-Besicovitch dimension [28].

The Cantor set (introduced in section 1.3.1) has $D = 0.631$, which is a rather strange number that does not fit into traditional geometry and it cannot be explained by the topological dimension. As if the set or object was something less than a line, a 1-dimensional object. The Koch curve (introduced in section 1.3.2) has $D = 1.261$, which would indicate that it is something more than a curve, but less than a plane. How the dimensions were calculated is explained in the relevant sections.

1.1.1 Hausdorff-Besicovitch dimension

The Hausdorff-Besicovitch dimension was defined by Felix Hausdorff and it was later extended by Abram S. Besicovitch. It is a mathematical way to express a dimension of an object.

In order to calculate the Hausdorff-Besicovitch dimension, a knowledge of the Hausdorff measure is required. It is a measure that defines a covering set of an object. As the measure decreases, the set that is covering the object is getting smaller. With a smaller covering set, it is possible to cover the object more precisely. When the size of the covering set decreases, the resulting measure of the object increases. The measure is either zero or infinity, depending on the covering set. The Hausdorff-Besicovitch dimension is the critical value of the Hausdorff measure when it goes from 0 to ∞.

A good example is in section 2.1, where this idea is described on measuring coastline length of Great Britain and how its length depends on size of a ruler.

1.1.2 Calculating the dimension

Self-similarity method

The self-similarity method is probably the simplest method to calculate the dimension of a fractal. I will demonstrate the method on a simple square when dividing it into smaller squares.

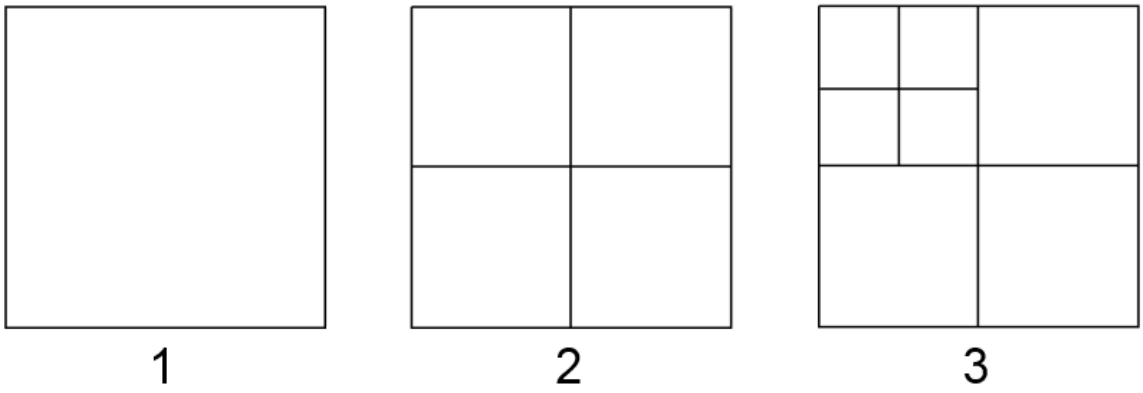


Figure 1 Self-similarity method demonstration

All sides of a square, staring from the left in Figure 1, are divided by one-half, which creates four smaller squares as can be seen in the second step. Dividing each side is then repeated, on the four squares that on in second step, creating 16 smaller squares.

Step	Ratio	Number
1	1	1
2	0.5 = ½	4
3	0.25 = ¼	16

How the dividing procedure influences the number of squares, can be written as

$$\left(\frac{1}{r}\right)^{D} = N;\ D = \frac{\log N}{\log \frac{1}{r}} \tag{1}$$

where D is the dimension, r is ratio and N is number of squares.

For this particular case the result is $D = 2$, which was expected, because a square is a Euclidean object. There are objects like the Hilbert curve, shown in Figure 2, which a space fitting curve with the same dimension as a square.

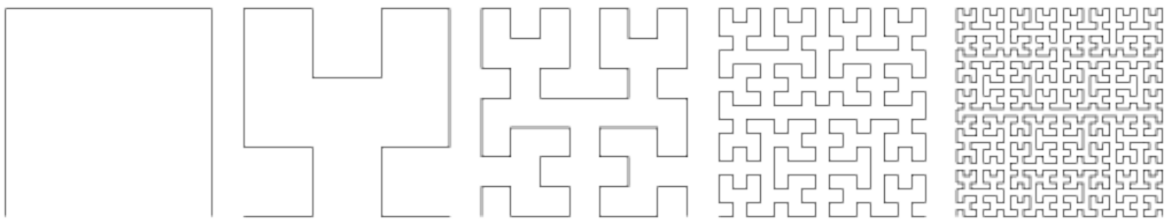


Figure 2 Hilbert curve [4]

Box-counting method

Another way to calculate the dimension is the Box-counting method. The Box-counting dimension is estimating how much space a fractal fills when examined at small scales and it is particularly useful when dealing with curves [1].

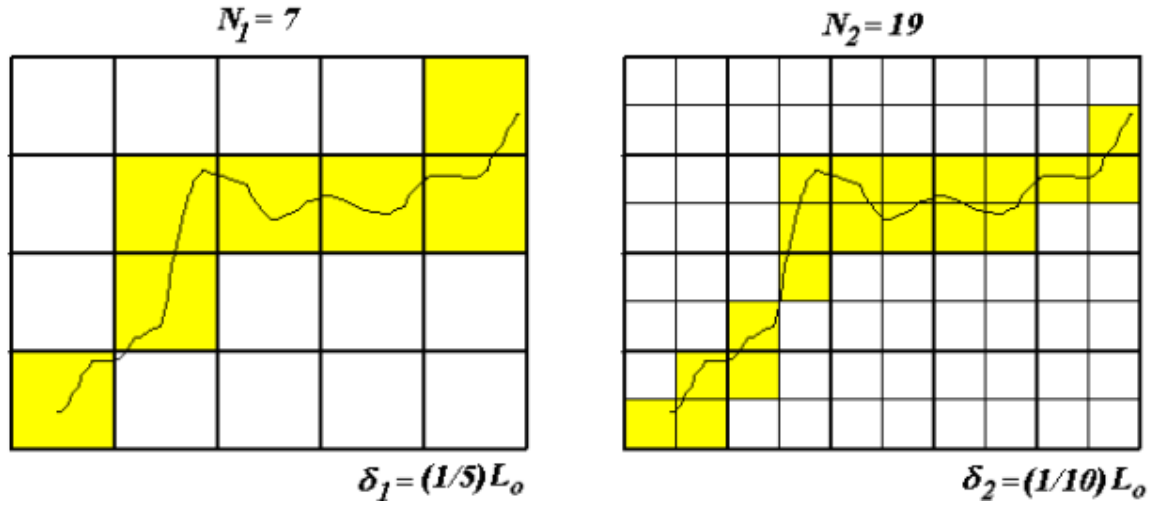


Figure 3 Box-counting method example [5]

In Figure 3 is an example of the Box-counting method, where N is a number of the boxes that are needed to cover the curve, L_0 is a length of the whole grid and l_0 is a length of a box.

The grid that recovers the object is divided into $n_k = L_0/\delta_k$ boxes of an equal side δ_k. How many of these boxes recovering the object is counted [5]. The process would continue and in each step, the boxes would be smaller than in the previous step.

The fractal dimension is then obtained through following formula:

$$D = -\frac{\ln N}{\ln\left(\frac{l_0}{L_0}\right)} \tag{2}$$

The dimension of the curve with a box size $\delta_2 = \frac{1}{10}L_0$ is $D = -\frac{\ln 19}{\ln\left(\frac{1}{10}\right)} = 1.27$.

1.1.3 Properties

The properties are defined loosely and it is not necessary for a fractal to have, or follow, all of them. [6].

1. The fractal has a fine structure, which means it has details on arbitrarily small scales.
2. The fractal is too irregular to be described in traditional geometrical language, both locally and globally.
3. The fractal has some form of self-similarity, approximate or statistical.
4. Usually, the fractal dimension of the fractal is greater than the topological dimension.
5. In most of the cases, the fractal is defined in a very simple way.

1.2 The Feedback Machine

Feedback machines have the ability to transform something very simple into something complex. That is achieved through iterations. A good example is the Koch curve that is described in section 1.3.2. It starts as a simple line and within a few iterations a snowflake-like shape emerges. I am going to use an example from a book, *Chaos and Fractals – New Frontiers of Science*, where it is called The Feedback Machine [2].

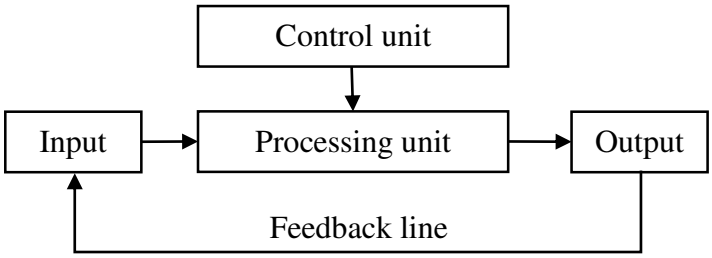


Diagram 1 The Feedback Machine

A feedback machine can be something simple as pointing two mirrors against each other and observing seemingly infinite tunnel. In this case, there is not much control over the result.

Pointing a camera on a screen that is displaying an output from the camera is a better example that is aligned with Diagram 1. The camera is the input and the screen is the output. The processing unit is electronics inside the camera and screen. All that can be controlled on both ends. On the input side, there are parameters like focal length, apertures, focus or position of the camera. On the output side, there are parameters like brightness, contrast or refresh rate.

Basic feedback process

One-Step Machines

One-Step Machines are characterized by a formula $x_{n+1} = f(x_n)$, where $f(x)$ is any function of x [2]. In one-step machines the output depends only on the input.

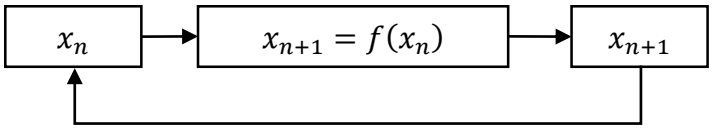


Diagram 2 One-step feedback machine

It is a powerful mathematical tool and it can be traced back to the Babylonian civilization [7], where it was used for calculating, or to be more precise, approximating a square root of a number by the method of mean. The idea is quite simple and to find the square root of a number and can be written as the following formula

$$x_{n+1} = \frac{x_n + \frac{a}{x_n}}{2} \tag{3}$$

where a is a number of which the square root is going to be calculated.

For $a = 5$, the sequence would look like this

$$x_1 = \frac{x_0 + \frac{5}{x_0}}{2} = \frac{5 + \frac{5}{5}}{2} = 3\ ;\ x_2 = \frac{x_1 + \frac{5}{x_1}}{2} = \frac{3 + \frac{5}{3}}{2} = 2.3\overline{3} \tag{4}$$

With each step, the approximation is approaching $\sqrt{a}$. The third step would yield 2.2360, which can be considered as a sufficient approximation.

Other feedback machines

There are other variations of feedback machines. The Two-Step Feedback machine which could be written as $x_{n+1} = f(x_n, x_{n-1})$. Fibonacci numbers are good example of Two-Step Feedback machine. A Fibonacci number can be generated by a function $f(x_n, x_{n-1}) = x_n + x_{n-1}$.

There can be the One-Step Feedback Machine with more variables, the function can yield a random result or a memory can be added to the feedback machines. The feedback machine with memory is a typical computer or a smartphone.

1.3 Basic fractals

1.3.1 Cantor set

The Cantor set was introduced by a German mathematician Georg Cantor in 1883. The Basic Cantor set, displayed in Figure 4, is a set of an infinite number of points in a unit interval [0, 1].

It starts with the interval $E_0 = [0,1]$. The set is split into thirds and the middle[2] third is removed. This will create two intervals $E_{1,L} = \left[0, \frac{1}{3}\right]$ and $E_{1,R} = \left[\frac{2}{3}, 1\right]$, where $E_{1,L}$ denotes the left segment of the first step and $E_{1,R}$ denotes the right segment of the first step. The next step is to remove the middle third from segments $E_{1,L}$ and $E_{1,R}$ and again, from each segment, two new segments are created. The segment $\left[0, \frac{1}{3}\right]$ is split into $\left[0, \frac{1}{9}\right]$ and $\left[\frac{2}{9}, \frac{1}{3}\right]$. The segment $\left[\frac{1}{3}, 1\right]$ is split into $\left[\frac{2}{3}, \frac{7}{9}\right]$ and $\left[\frac{8}{9}, 1\right]$.

Figure 4 Cantor Set

The most upper set E_0, in this case depicted as a line, is the zero order step and continues all the way to the E_n. The whole set is $F = \bigcap E_n$, where F is the fractal and E_n is n[th] step. Each step has 2^n segments. F is the set of points that is infinitely often. It means that a certain points, for example $\frac{1}{3}$, will remain in the set. No matter how many times the removal process is repeated.

It is possible to show properties listed in section 0 on the Cantor set [6]:

1. F is self-similar. It is clear that the part of F in the interval $\left[0, \frac{1}{3}\right]$ and the part of F in $\left[\frac{2}{3}, 1\right]$ are both geometrically similar to F, scaled by a factor $\frac{1}{3}$. Again, the parts of F in each of the four intervals of E_2 are similar to F but scaled by a factor $\frac{1}{9}$, and so on. The Cantor set contains copies of itself at many different scales.

[2] This set is sometimes referred as the middle-third Cantor set

2. The set F has a *fine structure*; that is, it contains detail at arbitrarily small scales. The more enlarged the picture of the Cantor set is, the more gaps become apparent to the eye.
3. Although F has an intricate detailed structure, the actual definition of F is very straightforward.
4. F is obtained by a recursive procedure. The construction is consisted of repeated removing of the middle thirds segment.
5. The geometry of F is not easily described in classical terms: it is not the locus of the points that satisfy some simple geometric condition, nor is it the set of solutions of any simple equation.
6. It is difficult to describe the local geometry of F – near each of its points are a large number of other points, separated by gaps of varying lengths.

Dimension

The dimension of the Cantor set is easily calculated through the Self-similarity method.

In each iteration, the set is split into three segments, hence ratio, or factor, $r = \frac{1}{3}$ and two new segments are created. Then the dimension is $D = \frac{\log N}{\log \frac{1}{r}} = \frac{\log 2}{\log 3} = 0.6309.$

1.3.2 Koch curve

A Swedish mathematician Helge von Koch introduced the Koch curve in 1904. It begins as a line, which is split into three segments. Just like in the Cantor set, the middle segment is removed. The next step is a construction of an equilateral triangle as shown in Figure 5.

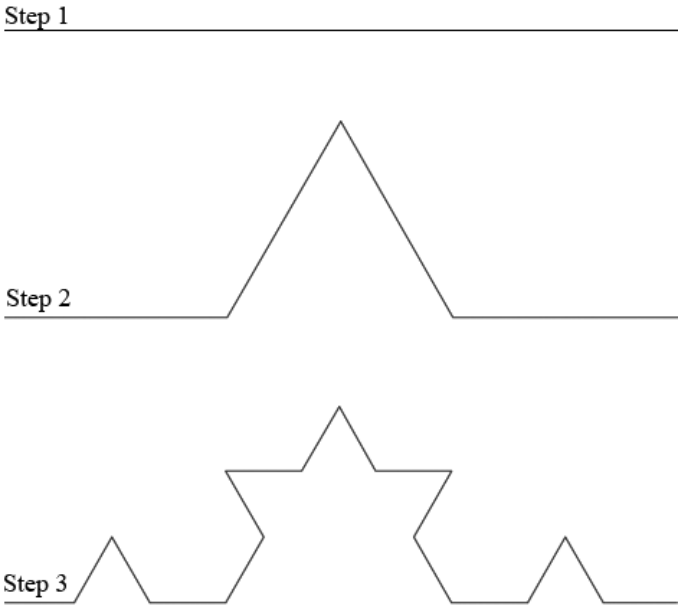


Figure 5 Koch curve

Koch Snowflake

The Koch snowflake is a variation of the Koch curve, where the starting point are three Koch curves forming an equilateral triangle. In Figure 6 is the process of creating Koch snowflake. In six steps, the object gets more and more dense. If I were to add a seventh or eighth step, the sides of what once was a triangle would get so dense that, it would appear as solid lines. The structure is self-similar therefore, it does not matter at which step we would zoom in, because the resulting image or given segment would be same all the time.

It has a few interesting properties. It has an infinite perimeter, but the area is a finite number. Each line is growing by

$$A * \left(\left(3 * \frac{1}{3}\right) - \frac{1}{3} + \frac{2}{3}\right) = \frac{4}{3}A \tag{5}$$

where A is the length of the initial curve. One segment is removed and two segments are added. For the second step, the length is $4^2 * A * \frac{1}{3^2}$ and finally for the kth step, the length is $\frac{4^k}{3^k} * A$.

If I were to draw a circumscribed circle to the triangle in Figure 6. The number of iterations is irrelevant, because the snowflake would not reach the circumscribed circle. It suggests that the area is finite. Area S of Koch snowflake is given by a formula

$$S = \frac{3\sqrt{2}}{5} * A^2 \tag{6}$$

where A is the length of the triangle's side [8].

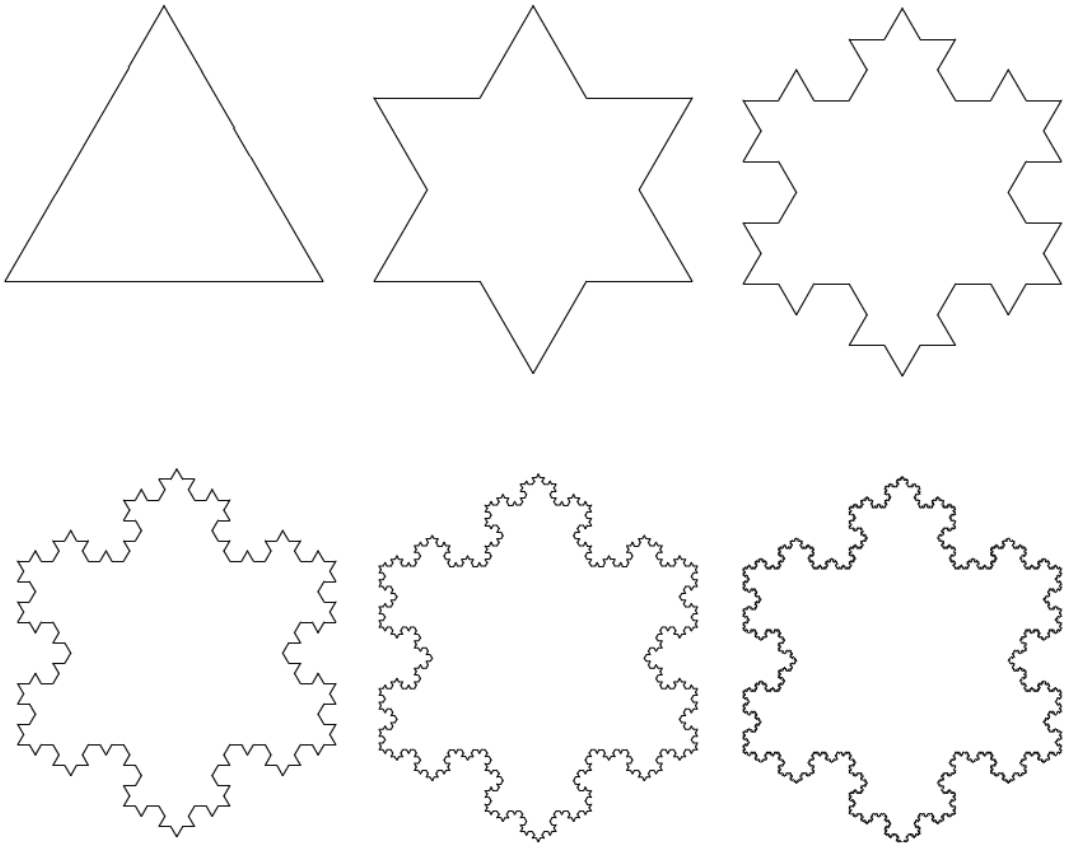


Figure 6 Koch snowflake[3]

Dimension

The dimension of the Koch curve can easily be calculated through the Self-similarity method. During each iteration, the curve is split into three segments, hence ratio $r = \frac{1}{3}$. One segment is removed and two are added, therefore the new length of the segment is $N = 4$.

Then the dimension is $D = \frac{\log N}{\log \frac{1}{r}} = \frac{\log 4}{\log 3} = 1.261.$

[3] Created on https://scratch.mit.edu/projects/3102566/

1.3.3 Mandelbrot set

The Mandelbrot was discovered in 1979 by Benoit Mandelbrot, a Polish-born, French and American scientist. The Mandelbrot set can be referred to as M-set.

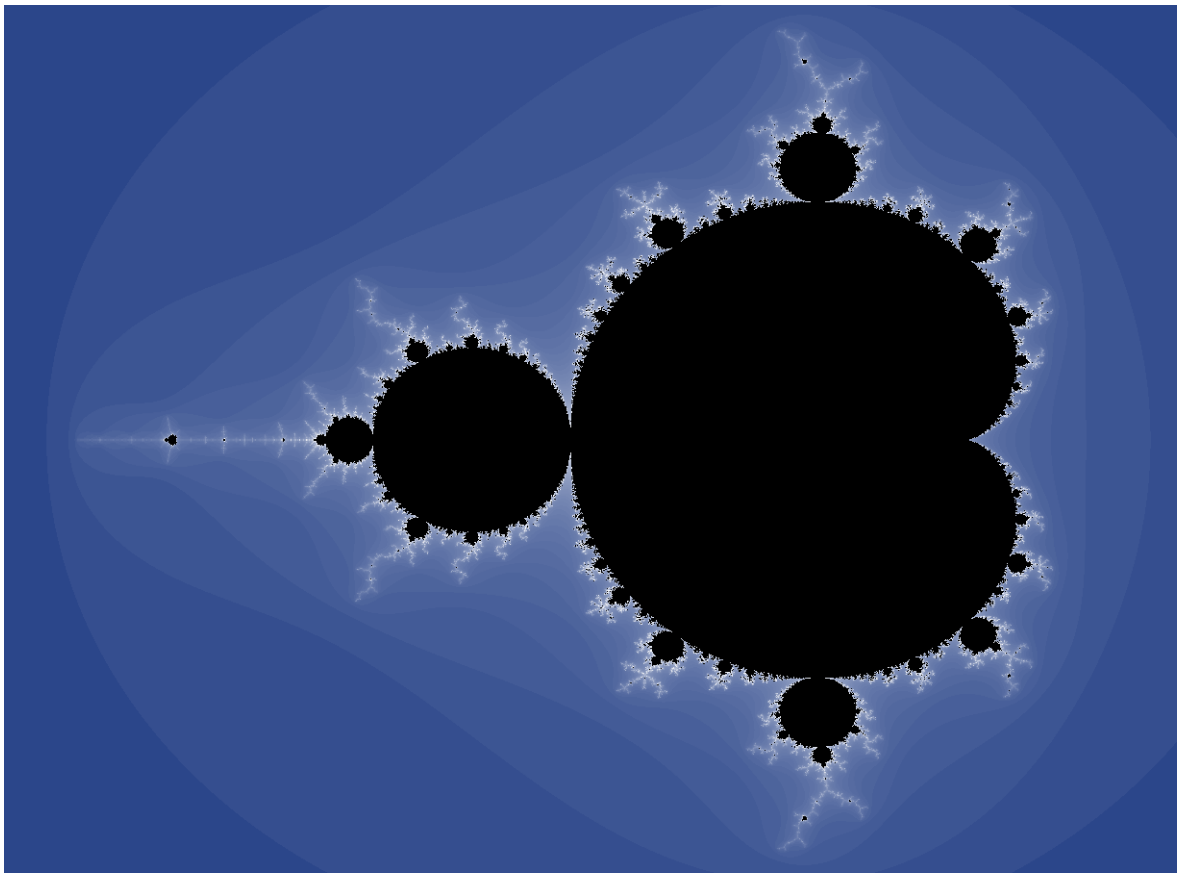


Figure 7 The Mandelbrot set [source: math.utah.edu]

Introduction

This is perhaps the most interesting fractal. It is difficult to describe, because it looks like many things that are very common in nature, but it is something unique, complex and yet created from one, very simple, formula (7).

$$f(z) = z^2 + c \tag{7}$$

where *c* is a magnitude or absolute value of a complex number that is defined as

$$|c| = \sqrt{a^2 + b^2} \tag{8}$$

A complex number is in format $a + bi$, where a and b are real numbers and i is the imaginary unit; a and b are coordinates on the complex plane. Real numbers are on horizontal axis and imaginary units are on the vertical axis.

The formula $f(z) = z^2 + c$ is creating something which is difficult to interpret. From a mathematical point of view, it is well defined and there are no unknowns. However, there are unknowns from a semantical point of view. It is a relatively young discovery that came with computer age and it has been rising more questions than it is able to answer. One of the hardest questions could be "*Why is this part of product of the formula resembling a bug, neural network or anything that comes to mind?"* Certain areas were even given names, shown in Figure 8 and Figure 9.

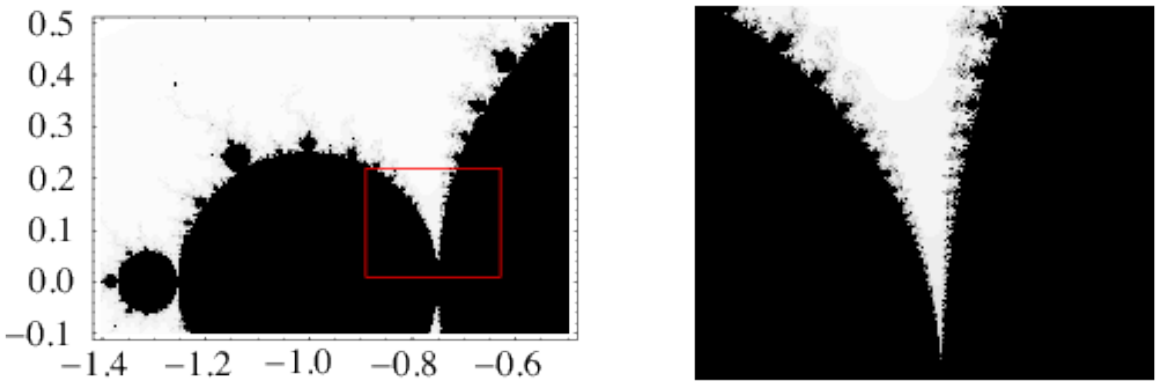


Figure 8 Sea Horse valley – centred on -0.75+0.1i [9]

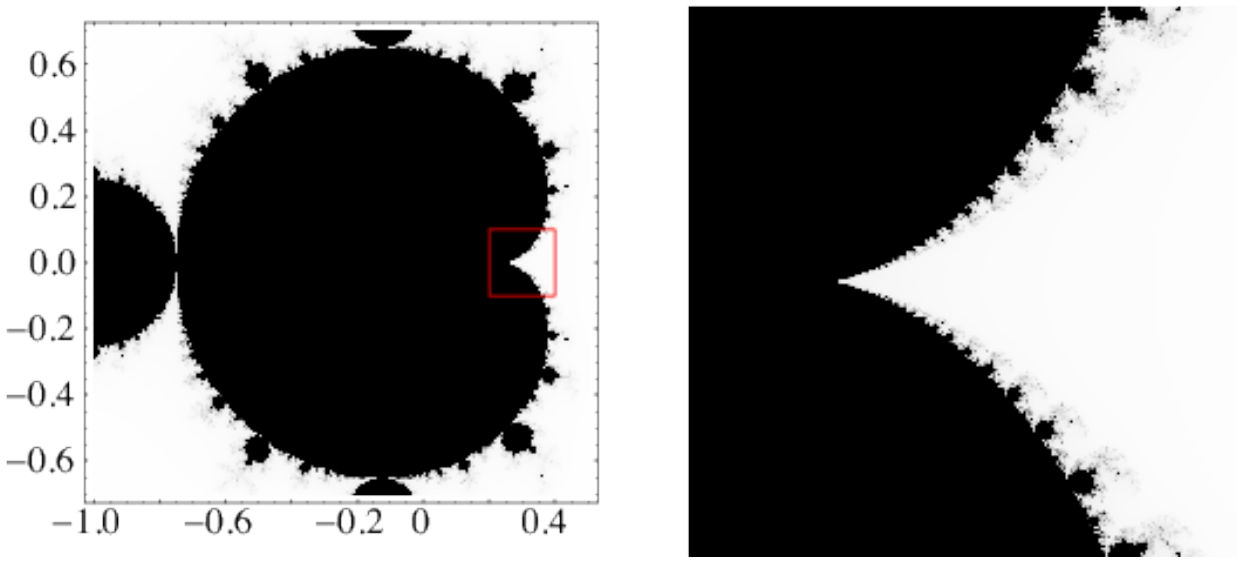


Figure 9 Elephant Valley – centred on 0.3+0i with size approximately 0.1+0.1i [9]

Construction

The Mandelbrot set exist in the realm of complex numbers. It is a visual representation of an iterated function depicted on the complex plane. Function

$f(z) = z^2 + c$ can be viewed as a tool to move between points on the complex plane. Figure 10 shows a span of the Mandelbrot set across the complex plane.

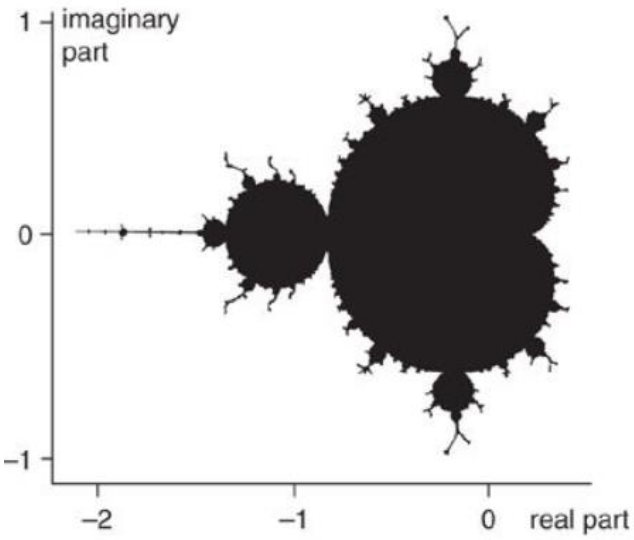


Figure 10 Span of the Mandelbrot set in the complex plane

It concerns about the value produced by $f(z) = z^2 + c$. A complex number c is presented to the function and the iteration starts with $z = 0$. There are two possible outcomes of these iterations:

1. The size of the number is limited by a circle with radius 2 with center at $[0, 0]$.
2. The number grows to infinity.

The Mandelbrot set is a set of complex numbers that are abiding the first outcome.

Colours are arbitrary, but a general unwritten rule is to use black colour for bounded points. Other colours are assigned according to the number of iterations it needs to reach a certain distance.

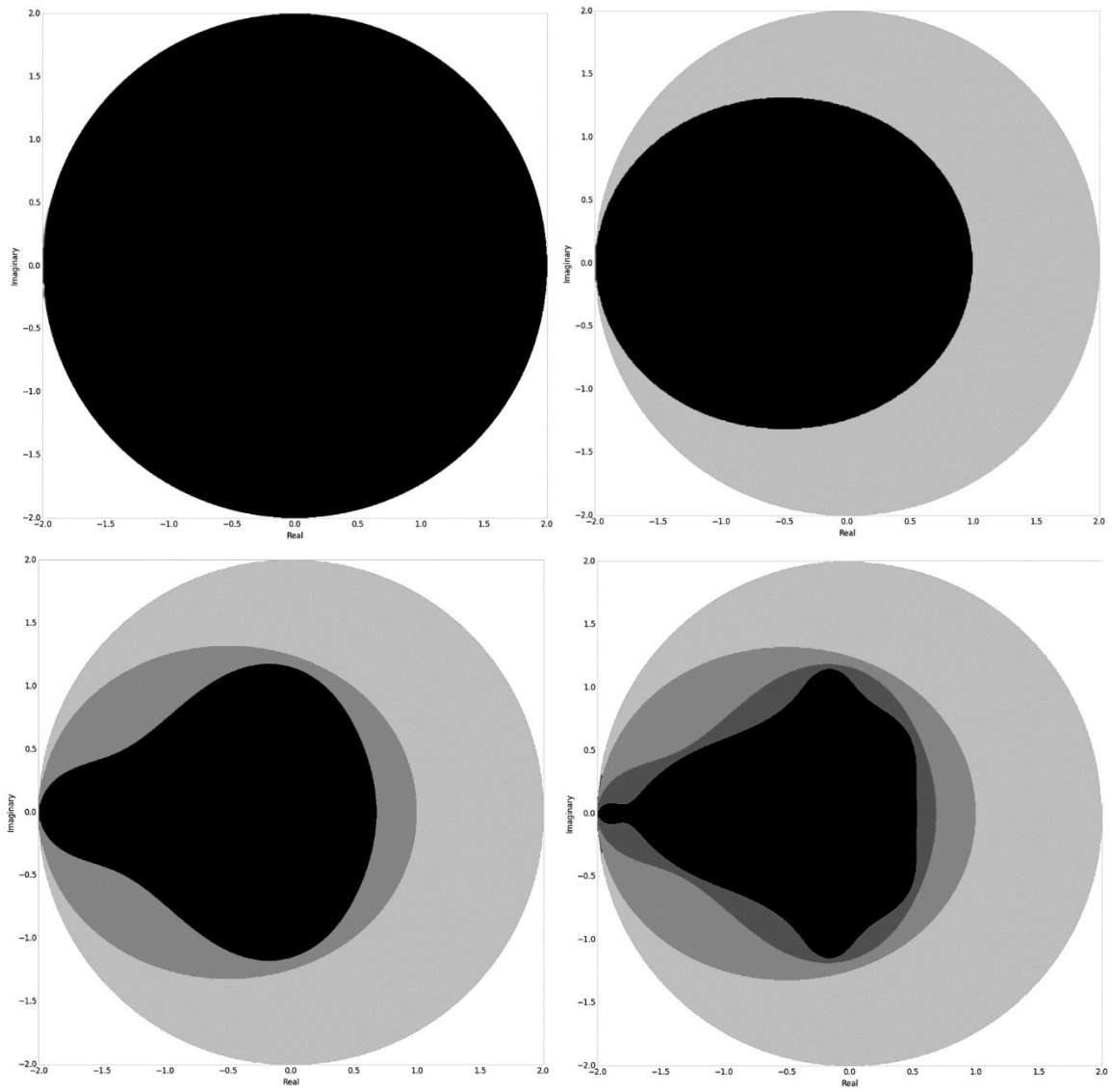


Figure 11 First four iterations of Mandelbrot set

The Iteration starts with a circle with radius 2. All points within this circle are part of a $set\ A$. All points whose first iteration through function $f(z) = z^2 + c$ fall outside of $set\ A$, when $f(z) > 2$, are added to a $set\ B$. For the next iteration $set\ A = A - B$. This process is repeated and as the number of iteration grows, the M-set will start showing more and more details. The first four iterations of the Mandelbrot set are shown in Figure 11.

In Table 1 are some handpicked values and their iterations. For demonstration purposes, I used only the real numbers, with their imaginary part set to $0i$. The real numbers in an interval $(-2, 1)$ will not grow to infinity. This interval can be spilt furthermore into $x = (-2, 0]$ and $y = [0, 1)$. If a number from the interval y is iterated through $f(z) = z^2 + c$, the result will converge to a constant. This is different for numbers that in the interval x, because the resulting number can wander between the starting and ending point of the interval x.

Iteration	Absolute value of a complex number						
0	-2	-1,75	-1	0,2	0,3	1	2
1	2	1,3125	0	0,24	0,39	2	6
2	2	-0,02734	-1	0,2576	0,4521	5	38
3	2	-1,74925	0	0,26635776	0,50439441	26	1446
4	2	1,309884	-1	0,270946456	0,554413721	677	2090918
5	2	-0,0342	0	0,273411982	0,607374574	458330	4,37194E+12
6	2	-1,74883	-1	0,274754112	0,668903873	2,10066E+11	1,91138E+25
7	2	1,308406	0	0,275489822	0,747432391	4,41279E+22	3,65339E+50
8	2	-0,03807	-1	0,275894642	0,85865518	1,94727E+45	1,3347E+101
9	2	-1,74855	0	0,276117854	1,037288717	3,79186E+90	1,7815E+202

Table 1 Sample of values from generating the Mandelbrot set

Dimension

It was shown that the boundary of the Mandelbrot set has the Hausdorff-Besicovitch dimension of two. The proof is based on a study of bifurcation of parabolic periodic points. [10]

This was the Mandelbrot set and as it was with previously mentioned fractals, there is much more to cover. There are whole books dedicated to Mandelbrot set and there is a lot to learn about this fractal.

The next section is about Julia sets, which were actually predicted before the Mandelbrot set was discovered. In a chronological order, it would make sense to put Julia sets before the Mandelbrot set, but Julia sets are closely related to the Mandelbrot set and it is easier to explain it after the Mandelbrot set was introduced.

1.3.4 Julia sets

Julia sets are named after a French mathematician Gaston Julia who published[4] them in 1918. This topic was brought back to the light with discovery of the Mandelbrot set and advances in information technology.

Just like the Mandelbrot set, Julia sets lies on the complex plane and are generated by iterating a polynomial function $f(z) = z^2 + c$, where *c* is a magnitude or absolute value of a complex number that is defined as $|c| = \sqrt{a^2 + b^2}$. The complex number is in a format

[4] G. Julia, *Mémoire sur l'iteration des fonctions rationnelles*, Journal de Math. Pure et Appl. 8(1918) 47–245

$a + bi$, where a and b are real numbers and i is the imaginary unit. The values a and b are coordinates on the complex plane.

Construction

The simplest Julia set has a function $f(z) = z^2$, which is a circle with a radius $|z| = 1$. The function $f(z) = z^2 + c$ is actually describing a way or path, on the complex plane. In other words f_0 is a set of coordinates for the next point on the complex plane. The path is shown in Figure 12. The coordinates of f_0 are passed to f_1 which is again a set of coordinates and so forth. It can also be written as a sequence:

$$z \to z^2 + c \to (x^2 + c)^2 + c \to ((x^2 + c)^2 + c)^2 + c \to \cdots \quad (9)$$

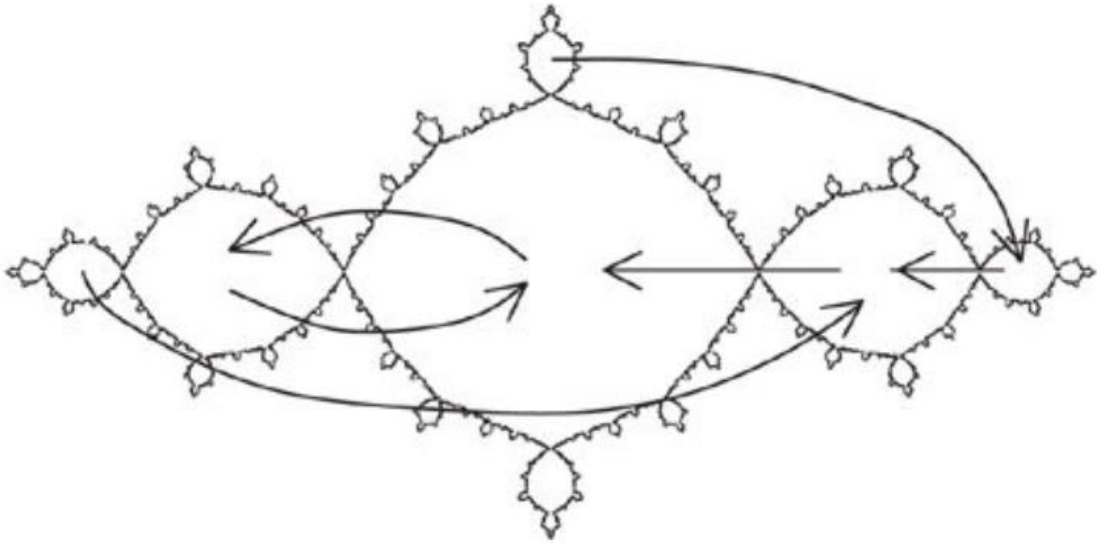


Figure 12 Traversing across complex plane [1]

The sequence has one of the following properties [2]:

- The sequence becomes unbounded, which means that the elements of the sequence leave any circle around the origin
- Alternatively, the sequence remains bounded, which means there is a circle around the origin, which is never left by the sequence.

The Mandelbrot set is constructed by iterating numbers from the zero ($z = 0$), but Julia sets are iterated from an arbitrary number.

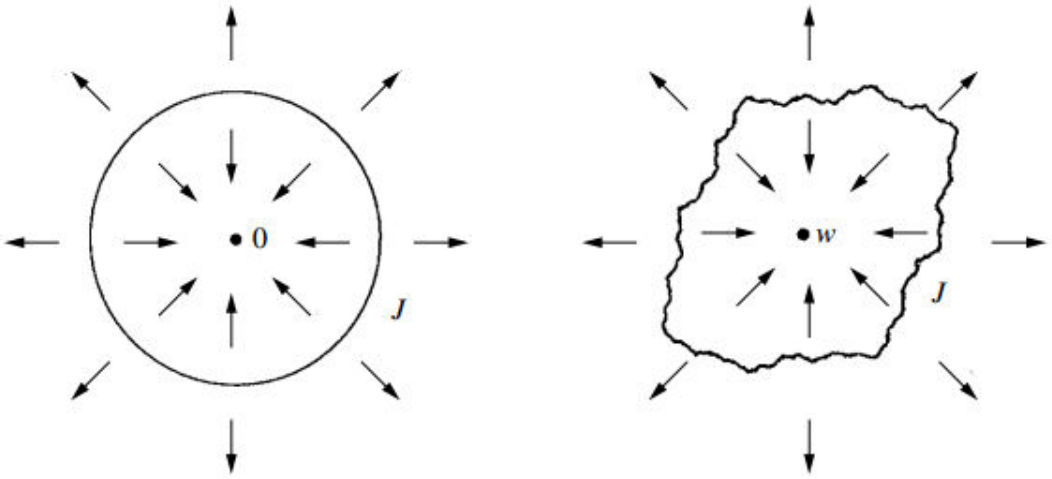


Figure 13 Boundary line of Julia sets [6]

The collection of points that leads to the first kind of behaviour is called the *unbounded set* for c. The collection of points that leads to the second kind of behaviour is called the *bounded set* for c; illustrated in Figure 13.

Connected vs disconnected

There are two types of Julia sets, *connected Julia sets* and *disconnected Julia sets*. This is the part, where the Mandelbrot set comes into the play. If a point is in the Mandelbrot set, then it is the connected Julia set. It is very easy to distinguish between those two categories. The disconnected sets are composed of islands. While the connected Julia sets are simply one continuous object. Examples are shown in Figure 14.

Connected set Disconnected set

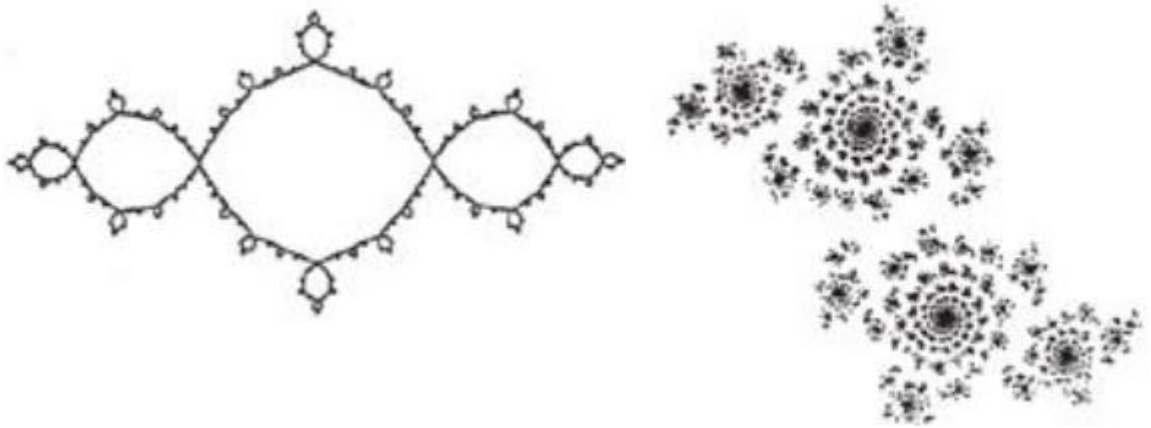


Figure 14 Connected vs disconnected set [1]

The Mandelbrot set can be used as a guide map of Julia sets. The complex numbers from various locations of the Mandelbrot set are producing Julia sets with certain attributes and shapes, shown in Figure 15.

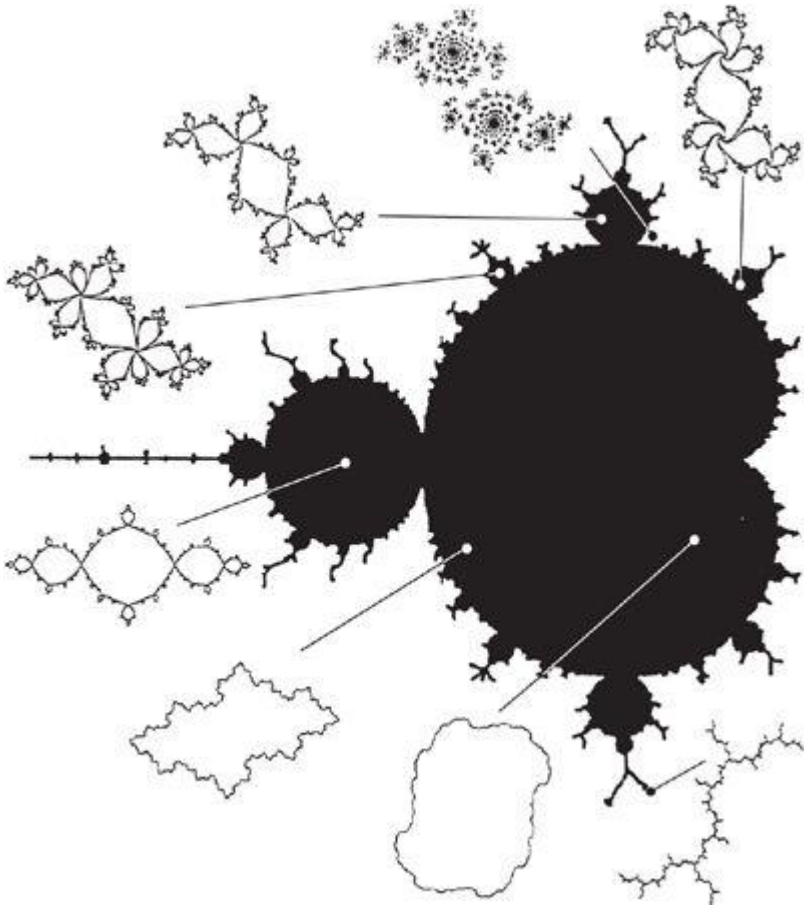


Figure 15: Relationship of Mandelbrot set and Julia sets [1]

Dimension

It was shown that the dimension of Julia sets is the same as the Mandelbrot set. The Hausdorff-Besicovitch dimension of Julia sets is two and it is based on the same study as the dimension of the Mandelbrot set [10].

2 Fractals in real world

In the previous chapter, I introduced fractals more from a theoretical point of view. This chapter envelops fractals, fractal-like structures and their applications.

2.1 Coastlines and landscapes

Lewis Richardson was the first one who presented, in a quantitative form, an observation that a length of a coastline depends on a scale at which it is measured. It is known as the Coastline Paradox or Richardson effect. This paradox says that a length of a coastline depends on a ruler length [11]. This idea was furthermore developed by Mandelbrot, who published a paper *How Long is the Coast of Britain?*, where he pointed out that Richardson's calculations essentially said that the coastline had a dimension D = 1.25 and it was valid over a wide range of scales. The article demonstrated that the fractional dimension was appropriate for describing a natural feature. It played a key role in convincing scientists that such notions could be used to study real phenomena of an irregular nature. [1] [12]

In Figure 16 is an example of measuring the length of the Britain's coastline. It is apparent that the area of Great Britain is a finite number. It is only going to get more and more precise as the ruler length decreases.

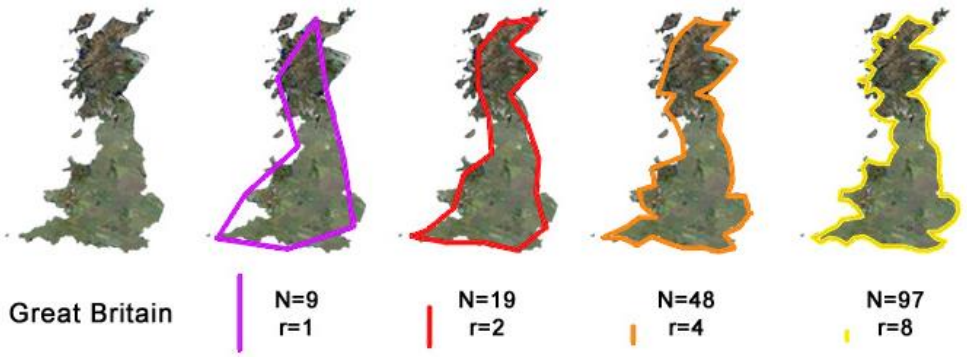


Figure 16: Measuring the length of the coastline of Great Britain [13]

Fractals are also being used to generate landscapes, mountains or even whole planets. There are several methods to generate such objects. One of the simplest method is the *midpoint displacement algorithm*, which is able to generate landscape silhouettes in 2D. Other methods are based on the Poisson faulting, Fourier filtering, Successive random additions, or summing band-limited noises [3] [14]. To keep things simple I will not go into detail here, as these methods form a whole separate topic [15].

Midpoint displacement algorithm

I will introduce the midpoint displacement algorithm, because it is used in chapter 3 for time series compression.

The process starts with a line. Let us denote the line as $\overline{AB}$. The first step is adding a point to the middle of the line, hence a *midpoint*. I will denote the point as X. The midpoint travels, or is displaced, by a predefined distance. Randomly up or down from its original location. That creates two lines, $\overline{AX}$ and $\overline{XB}$. The process is repeated on both segments and by repeating this process, a terrain is generated, in this case a 2-dimensional landscape. The process is shown in Figure 17.

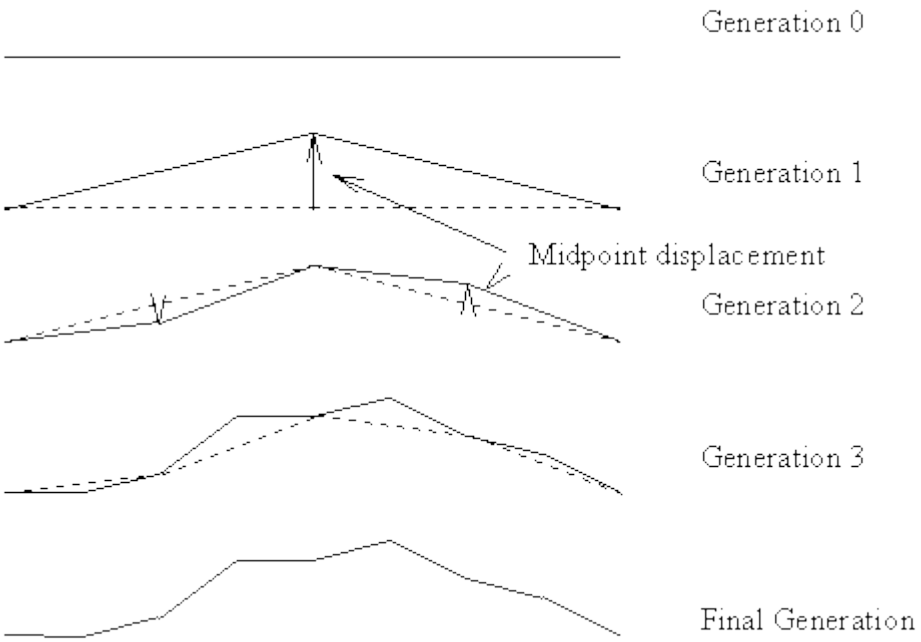


Figure 17 Midpoint displacement [16]

2.2 Turbulent fluids

A turbulent liquid or gas is one that behaves in a non-smooth swirling fashion. Often when a tap is first turned on, water emerges in a smooth stream, but then it breaks up into a gushing, irregular and turbulent flow. Turbulent fluids can be difficult to control, predict and often have a violent association. Despite having been studied intensively by scientists for hundreds of years, turbulence is far from understood. [1]

When results of turbulence simulations are plotted, they appear like a space-fitting curve. In Figure 18 is such a simulation and it is apparent that it is not filling up the whole image evenly and that can be exploited.

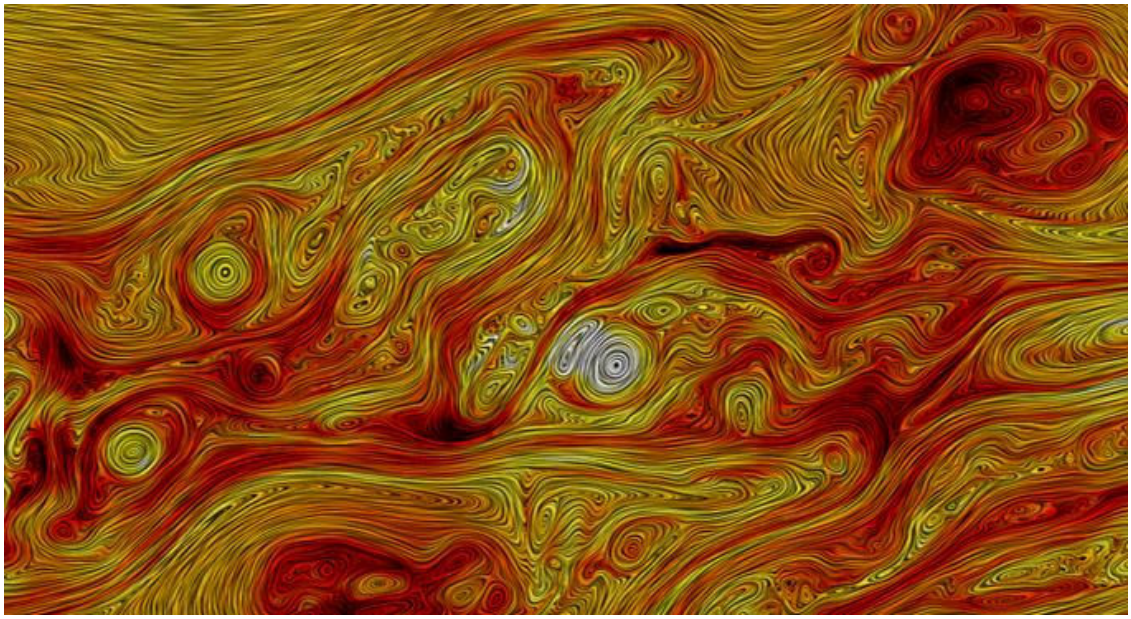


Figure 18 Simulating turbulence [spaceweather.com]

Simulating turbulences in rocket engine

GPU[5] *to Mars* was presented in 2015 on GTC (GPU Technology Conference) by SpaceX. They presented an interesting way to simulate Mars rocket engines that SpaceX developed. By using a GPU for calculations and using some of the fractals' properties to compress data in CFD (Computational Fluid Dynamics) simulations.

A challenging task is to capture a complete system on a uniform grid of the simulation. The CFD models are three-dimensional and the number of points can reach up to 10^{18}. If one point on the grid held 1kB of information, the grid would consist of yottabytes of data. There are ways to address this issue. The model can be simplified or a part of the simulation can be excluded completely. It reduces the complexity, but that will not simulate the whole system. Another possibility is lowering resolution.

The fractal features are allowing focusing computational resources to interesting parts of the grid. Similarly, like compressing images, but in this case turbulences are compressed.

[5] Graphics Processing Unit

The key in this scenario is to do all the computing on compressed data without decompressing them.

Fractal grid

Fractals have structure on all scales and that can be used to create adaptive grids and focus the resources on certain areas. In Figure 19 is a grid from the CFD simulation. It is actually just the grid, without any objects in it. The dark lines are vortexes, turbulences and shockwaves that formed around a capsule during a re-entry simulation. The darker areas are actually dense grid boxes. The grid boxes are adaptive and they get denser in areas where the turbulences emerge. This way it is possible to have a coarser grid in empty and uninteresting areas.

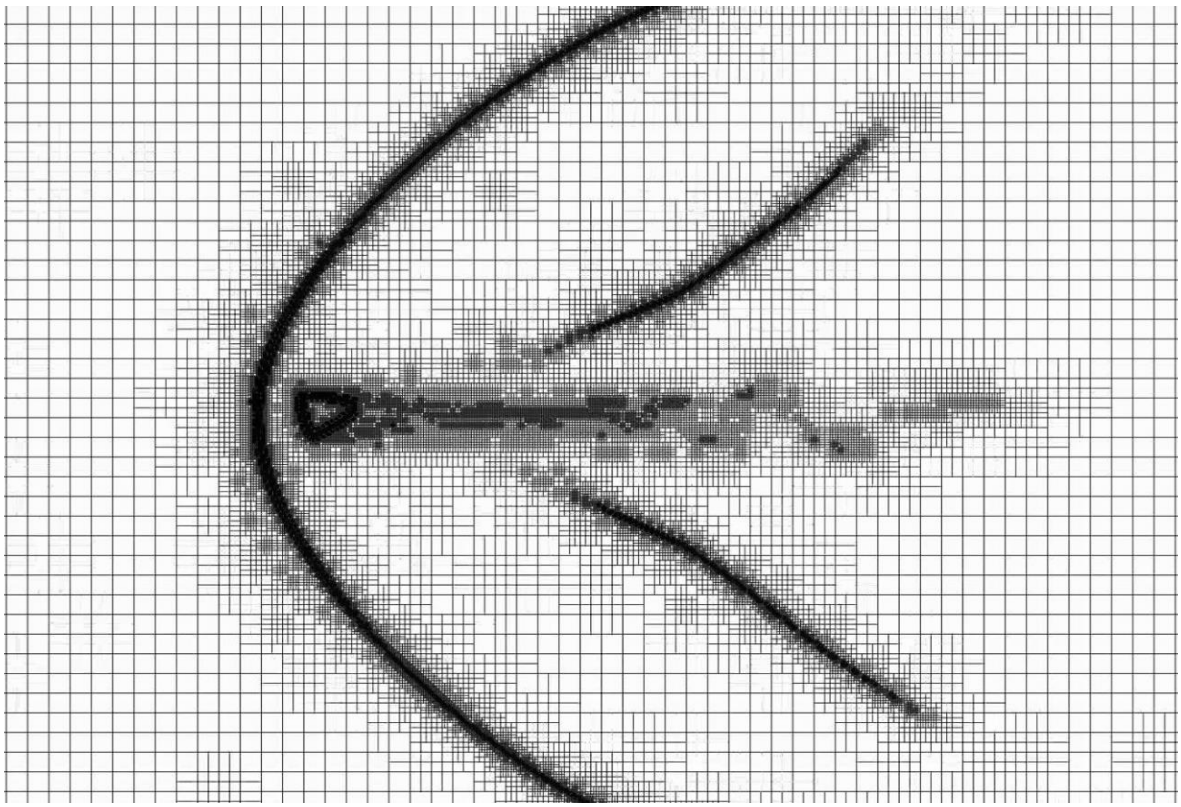


Figure 19 Shockwave forming around capsule during re-entry

2.3 Fractals in finance

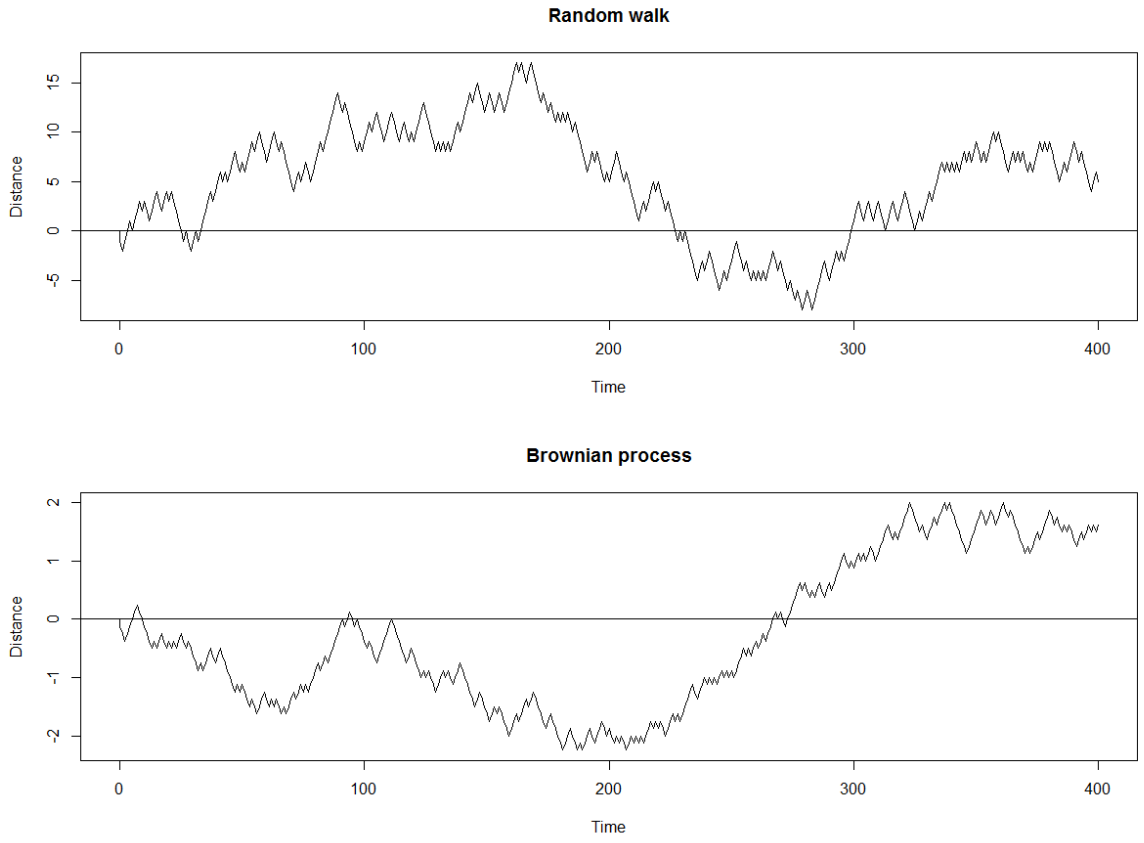
A random walk in Euclidian space is a process where an object moves randomly one unit, either backwards or forwards with a same probability ($p = 0.5$) every second or any

other uniform unit. With equal chances traveling either backwards or forwards, the object will not get too far from its origin. Mathematically, the random walk can be written as:

$$X_t = X_{t-1} + a_t \tag{10}$$

where a_t is the irregular component or noise that bears characteristics of white noise [17].

With more frequent traveling, for example, one step would have a length $\frac{1}{8}$ and occurred every $\frac{1}{64}$ of a second. A plotted random walk would keep same appearance on a larger scale, but on a smaller scale, an additional irregularity becomes present in the plot. By taking very rapid and small steps, the plot of the random walk takes on a fractal form, called the *Brownian* or *Wiener process*. [1]



Plot 1 Comparison of Random walk and Brownian process

Many real world phenomena depend on frequent random events. Time series like stock prices or exchange rates are probably the closest to the Brownian process. It was proposed by Louis Bachelier to study financial data as the Brownian process. It makes sense to view financial data as the Brownian processes, because, for example, the share price is generated in a similar way. The price of a share results from many individual investors estimating future value of an assets using information available to them at the time. Such information arrives randomly and involves numerous factors such as news or rumours of the behaviour

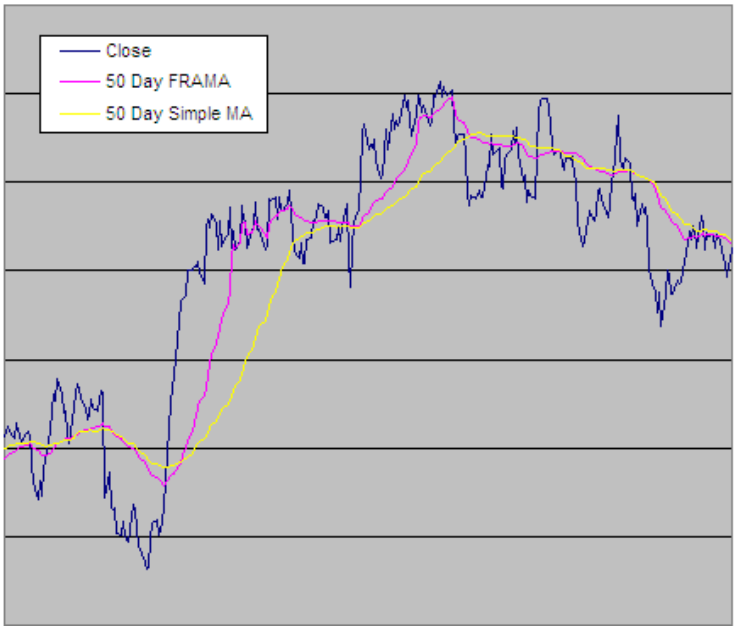
of governments, companies, banks, etc. With virtually instantaneous trading, the share prices are determined by a combination of a large number of small, seemingly random, upwards or downward steps in a very short time intervals. A consequence of that is a plot of the share prices looks in many ways very similar to the Brownian process.

The thought of looking at financial data as a fractal led to development of models like FRAMA. It is an acronym for *Fractal Adaptive Moving Average*. As the second half of the name suggests, FRAMA is a moving average. It identifies the fractal dimension in financial data and uses the dimension to adapt the smoothing period of an exponential moving average (EMA).

To determine the fractal dimension of a generalized pattern, the pattern is covered with N number of small objects of several various sizes s [18].

$$\frac{N_2}{N_1} = \left(\frac{s_1}{s_2}\right)^D ; D = \frac{\log \frac{N_2}{N_1}}{\log \frac{s_1}{s_2}} \tag{11}$$

The dimension is then used to modify the exponent of the exponential moving average. It rapidly follows significant changes in prices, but becomes very flat in congestion zones. A comparison of FRAMA and SMA is in Plot 2.



Plot 2 FRAMA compared to the Simple Moving Average [19]

2.4 Fractals of the human body

There are several fractal-like structures in or on our bodies. For example, the respiratory system, nervous system, blood supply system, neural system or even the skin and the way the skin cells are arranged.

Lungs are an excellent example of a natural fractal organ. They are like a tree that is hanging upside down and sharing same branching pattern as a tree. Not just the pattern, but also the purpose is similar – respiration [13].

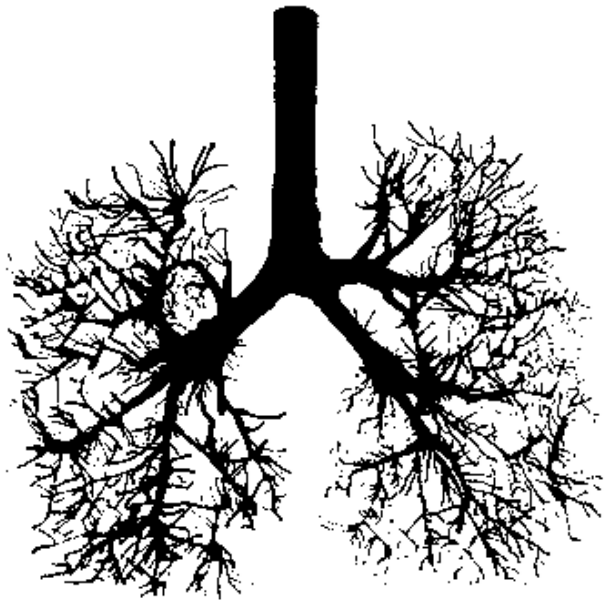


Figure 20 Human lungs [source: cargocollective.com/mattrobinsonuk]

In the respiratory system, the windpipe splits into two bronchial tubes leading into the two lungs. These tubes split into narrower tubes, which continue to split repeatedly until, after about 11 levels of branching, they reach numerous very fine tubes called bronchioles. These end in microscopic thin-walled sacs called alveoli. A lung contains around 400 million very closely spaced alveoli. The amount of gas that can be exchanged through the lungs in a mammal is directly proportional to their total surface area. Although the volume of a pair of human lungs is only about 4 - 6 litres, the surface area of the same pair of lungs is between 50 and 100 square meters. [1] [13]

2.5 Data compression

Data compression can be divided into two categories. The first category is lossless compression and the second one is lossy compression. Lossless compression involves algorithms that are reducing data needed to describe information, but without removing or

degrading the information itself. A common application of this type of the compression is a document compression. The compressed file is then reconstructed to the original form without losing any information.

On the other hand, lossy compression can achieve much higher compression ratios, but the reconstructed data are only an approximation of the original data. The most common application of lossy compression is a part of our daily lives when we are watching videos, listening music or browsing pictures. Imagine a picture of a landscape with blue sky. A 24-bit palette, also known as True colour, has $2^{24} = 16\,777\,216$ colour variations. Lossy compression algorithms takes an advantage in an inability of a human eye to distinguish subtle changes in the colour variations and replaces, for example, five different gradients of blue with one that does not dramatically change the reconstructed image.

The fractal self-similarity can be used for lossy data compression. The first research in this area was done by Michael Barnsley in late 1980s [1]. The idea behind these techniques is to find simple patterns and describe complex objects with it.

2.5.1 Image compression

Compressing images as fractals offer not only a theoretically high compression ratio, but also it preserve more information in the images. The fractal image compression is looking for parts in the image that are similar to other parts of the same image.

The fractal image compression has a following workflow. The image is divided into small and large blocks. The large blocks are called *domain blocks* and the small blocks are called *range blocks.* The domain blocks are then used, as a description of the range blocks. Each range block is matched with the domain block. That would not work without transformations, thus transformations like rotating, scaling, brightness and contrast adjustments are used on the domain blocks in other to get the best possible match with the range block [20].

Information is stored in so-called fractal codes that contain following [21]:

- The translation done on the domain blocks to match the position of their associated range blocks
- The transformations done on the domain blocks
- The colour, brightness, contrast or any other adjustments

No pixels from the original image are stored. Only the mathematical functions, that were used to produce the fractal code, are stored. The downside of this process is the matching procedure that can be very demanding process.

The decompressing of images from the fractal code is much easier, because the fractal code is like a cookbook full of recipes. In this case, it is full of the functions that will produce

the approximated image. Compared to the traditional image compression algorithms like JPEG, the image that have been compressed with the fractal compression algorithm, can be decompressed into larger a resolution than the original image without getting too pixelated.

The compression process, which is effectively creation of a fractal, is also known as the Iterated Function System (IFS). This is based on the same principle as *The Feedback Machine*, which is described in the first chapter. The decompression of the image is then enveloped by the *Collage Theorem* that says:

If the image you want to get is called L, then you need to find functions f such that $F(L) = L$. Then no matter what initial image you start with, if you iterate F, you will "eventually" get L, where "eventually" means you will get closer and closer to it and after a while your image will be indistinguishable from L. [22] [23]

2.5.2 Wavelet compression

This algorithm is a follow up to section 2.2. I will describe a one-dimensional version, but in CFD (Computational Fluid Dynamics), it is used in a three-dimensional space. It is also worth mentioning that the 1D version does not look like a fractal, but the 3D adaptation does, shown in Figure 21. The grid in Figure 19 is compressed by this algorithm.

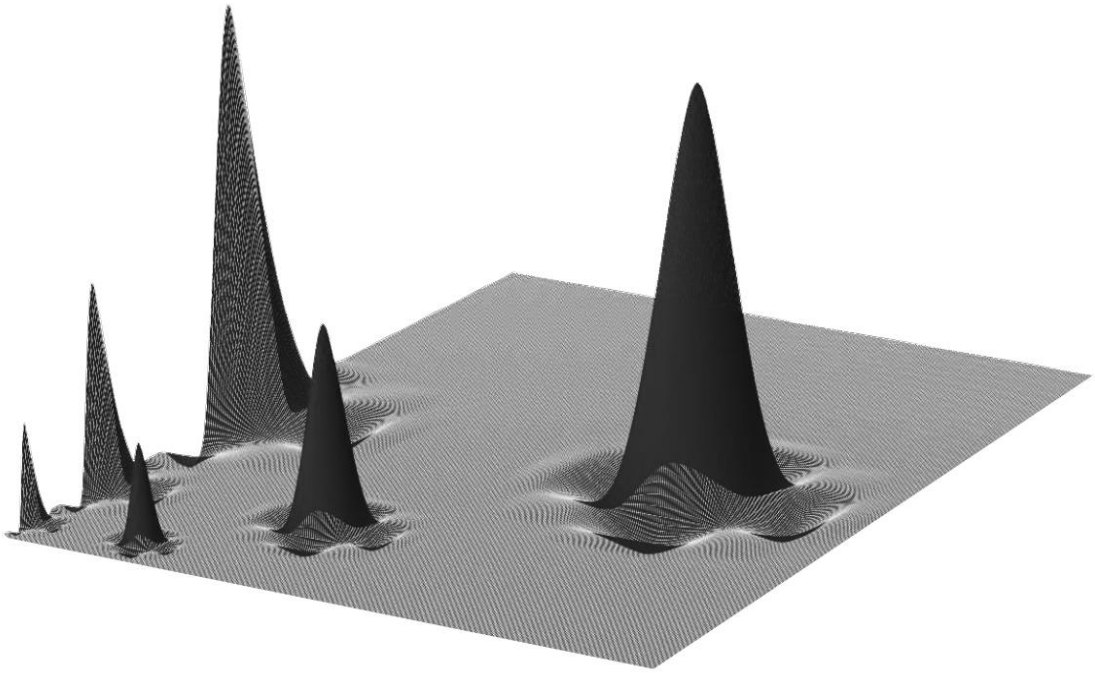


Figure 21 Local fractal basis [24]

The steps of 1D wavelet compression are following:

a) Data are regularly sampled and every other sample is thrown out, but not discarded. What remains is a $set\ A$ and the rest is a $set\ B$.

b) A linear interpolation is assumed between neighbouring samples of the $set\ A$. In-between those two samples used to be another sample, which is now in the $set\ B$. The shortest distance from the interpolated line to the sample from $set\ B$ is calculated. This is depicted in Figure 22. If the distance between the line and the sample is too great sample is returned to the $set\ A$.

These two steps are repeated until there are only two samples left in the $set\ A$.

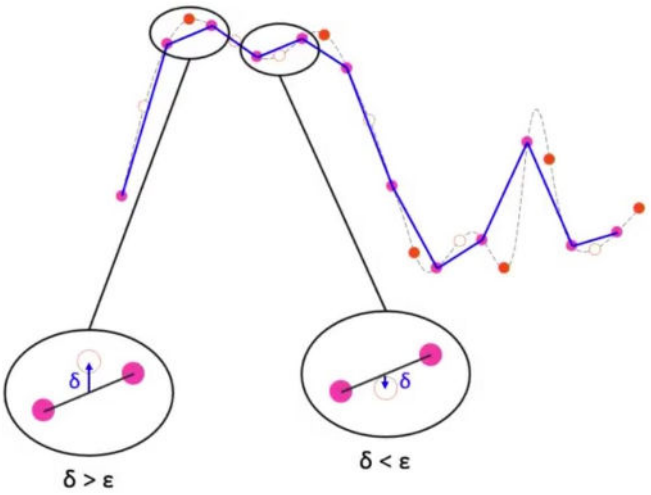


Figure 22 Wavelet compression

3 Fractal Resampling

The Fractal Resampling was developed at the European Space Operation Centre (ESOC) and it has been patented[6] as *Method and Apparatus for compressing time series* in 2013.

3.1 Introduction

As the name of the patent suggests, it addresses time series compression. The invention is inspired by methods for fractal terrain generation that are used in computer games and scientific simulations. It uses the midpoint displacement algorithm to compress time series. The midpoint displacement algorithm is described in section 2.1.

The term resampling refers to the fact that the invention selects certain samples from the original time series to achieve data compression. It allows transmitting a reduced data set, i.e., the selected data points, in the same format as the original time series as a contrast to compression methods that have to encode the underlying data samples [25]. Furthermore, it enables to perform calculations on the data without decompressing it.

The primary purpose of the invention is compression of housekeeping telemetry of remote systems. A remote system can be a satellite or deep space probe. The invention can remove background noise from the telemetry, but unlike other resampling methods, it preserves peaks, spikes and any extreme values.

If a remote system has large number parameters, it is usually not possible to achieve high sampling rates on all of them due to a downlink capacity. Limited downlink capacity requires compromises. For example Rosetta spacecraft, on whose telemetry I will be conducting my tests, has around 16 000 parameters that are being recorded.

The sampling rate can be modified and trade-offs can be made. Lowering the sampling rate of some parameters can be traded for a higher sampling rate of others, in order to match the bandwidth capacity. It is not an ideal scenario, because there might be short-lived events, which could occur outside of the sampling intervals. The lower sampling rate is increasing a risk of completely missing out these events or capturing only a part of it. The fractal resampling addresses this issue by eliminating samples with only background noise and keeping samples only around areas where waveforms are present.

[6] Patent public number: US 2013/0212142 A1

3.2 Algorithm

First, I will introduce two terms that are being used here.

- Sample – A sample is a pair of two figures, in format [time, value].
- Maximum allowed error – A maximum allowed error, denoted as ε, is a way to control quality of the compressed data. As it decreases, the amount of information kept in the data increases. It must be defined before the compression process begins. I refer to it as predetermined maximum error or maximum error, for short. There is a whole section devoted to this term later on.

The algorithm has following steps [26]:

1. The first and last samples of the original time series are included in the list of displaced points.
2. Linear interpolation is assumed between the start and end samples determined in first step.
3. For every point in the original time series that corresponds to the current segment, the absolute error between its actual value and the corresponding linearly interpolated one is determined.
   a) If this error is equal or above the maximum allowed error (ε) this segment need to be displaced. The displacement consists of adding the middle point to the list of displaced points and applying step 2 to both the left and right side of the displacement (e.g. left = (start, displaced point), right = (displaced point, end))
   b) If the error is lower than the maximum allowed error (ε) no displacement is needed

It is important to understand how the compression is achieved. The compressed time series is actually a list of the displacement points. The algorithm does not remove any samples from the original time series, but it leaves out certain samples during the compression process. The compression process is actually a reconstruction of the original time series. It starts as a line, from the first and last samples of the original time series, which is being aligned by adding midpoints into it.

The algorithm was designed in a way that it uses as little resources as possible. Once the first absolute error, between the actual value and the corresponding linearly interpolated line, is greater or equal to ε, the segment will be split into two. For example, in a segment with 1000 samples, if the 10^{th} sample will not be close enough to the interpolated line, the loop stops at the 10^{th} observation. What is the in the rest of the segment is irrelevant, because the segment will be split into two segments.

Example of a sample elimination

I will demonstrate how the algorithm achieves the compression on a simple time series with 8 samples labelled from A to H. The whole process is depicted in Figure 23.

1. The samples A and H are added to the list of displacement points.
2. The samples A and H are linearly interpolated.
3. For the samples from B to F are calculated the shortest distances from the interpolated line $|AH|$. The red dashed line in Figure 23 is marking out a limit, which has been introduced as the maximum predetermined error.

    a) The first sample, B, is too far from the interpolated line. After checking the distance of the B, there is no need to check the other samples and the line $|AH|$ will be displaced. If a length of a segment is an odd number, then the length is modified by adding or subtracting 1, only for the midpoint search, the segment is not modified in any way [25]. There are only two possible outcomes for a segment, it is left alone or it is displaced.

    b) The sample E is selected as the displacement point and it is added to the list of displacement points. The list is now 3 samples long.

In Figure 23, in the step 2, the process is repeated, but this time with lines $|AE|$ and $|EH|$. No samples are eliminated in the step 2, because either of the segments did not have all the samples within the boundaries of maximum error. The samples C and G are added to the list of displacement points. The list is now 5 samples long.

In Figure 23, in the step 3, the process is repeated, but this time with lines $|AC|, |CE|, |EG|$ and $|GH|$. From these four segments, only in the segment $|CE|$ are all samples, in this case only one sample, within the boundaries. It is the sample D and it is the only point that is omitted from the original time series. Samples B and G are added to the list of displacement points. The list is now 7 samples long.

The list of displacement points is 7 samples long and it is the compressed time series.

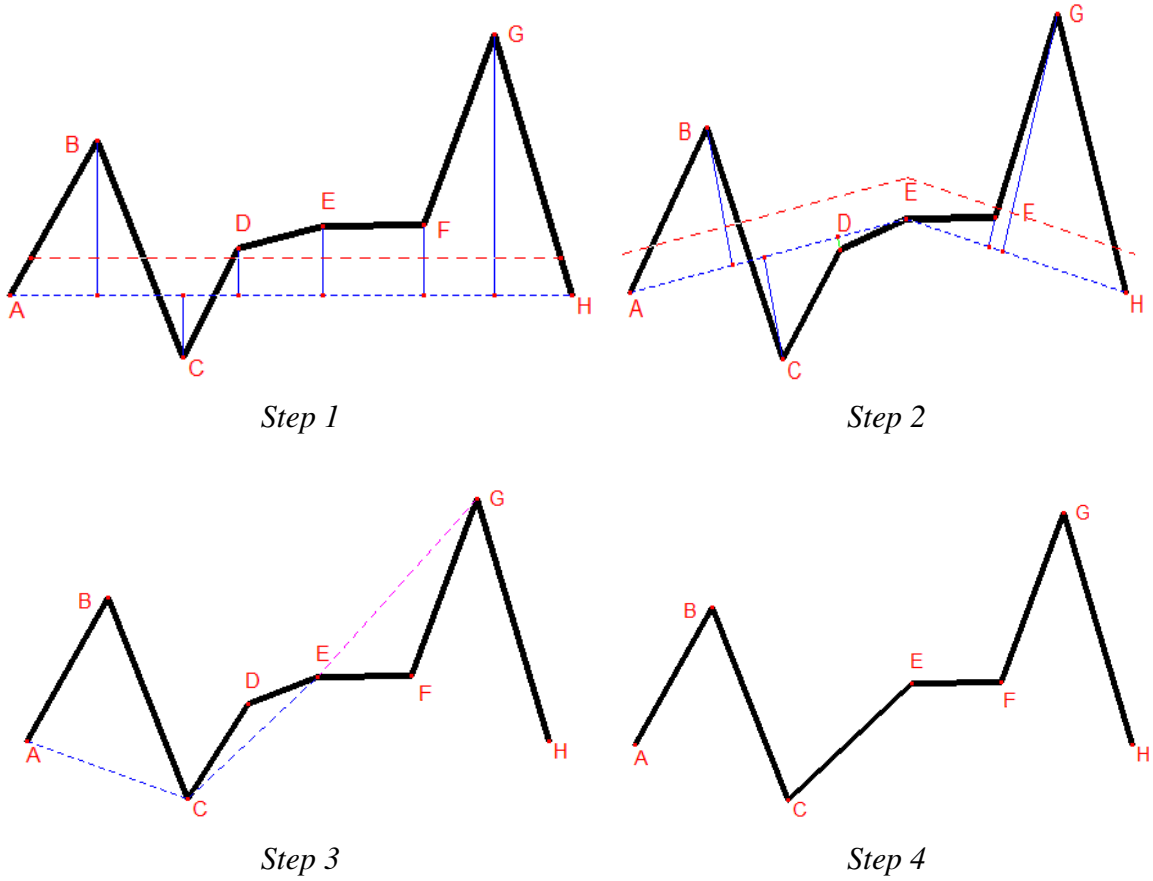


Figure 23 Point elimination process

3.2.1 Displacement point modification

The mission of the Technology Transfer Office Programme is to bring technologies, developed for space programs, to terrestrial applications. One of the areas is software inventions. It was already mentioned that the algorithm has to run efficiently and that it uses as little resources as possible. Resources, such as computational power or electric power, are not a limiting factor for the most devices that are being used in everyday life.

To distinguish these two algorithms I refer to them as *the original algorithm* and *the modified algorithm*.

The question I set to answer in this thesis is what the original algorithm can do, when it is slightly modified. The modification lies in changing the displacement point mechanism. In the original algorithm, it is always the middle point of a segment. With the modification, any point in the segment could be used as the displacement point.

There are two expected outcomes of this:

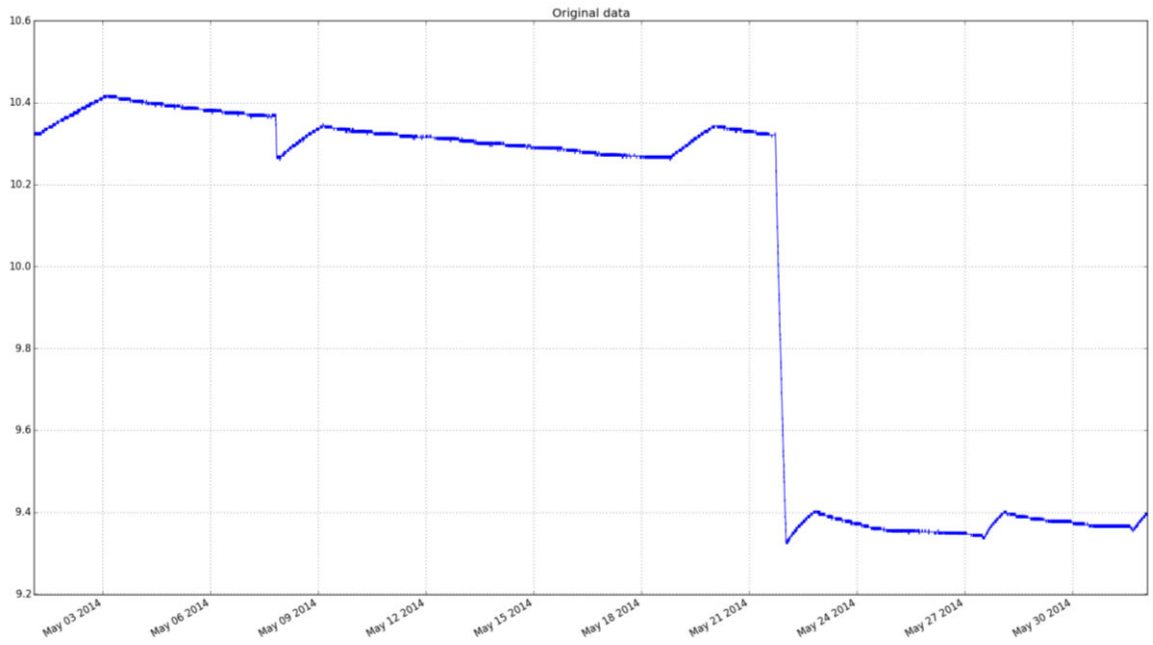
1. Increased load on a system that is compressing the data
2. Higher compression

I will quickly return to Figure 23 and the example of the sample elimination. In step 3a, the modified algorithm will not stop after checking the distance of the point B. It will record the distance and its index and continue checking other samples. When it is finished with all the samples from B to F, the sample with the greatest distance from the interpolated line is chosen as the displacement point. In the first step, it is the point G. The following steps are the same, but of course with different segments.

The modified algorithm could be used for archiving already existing time series repositories.

3.2.2 Demonstration of the patented algorithm

To demonstrate how the original algorithm compresses data, I will use a telemetry[7] of a tank pressure measured in bar[8]. Raw data consists 43 623 samples, which is shown in Plot 3.

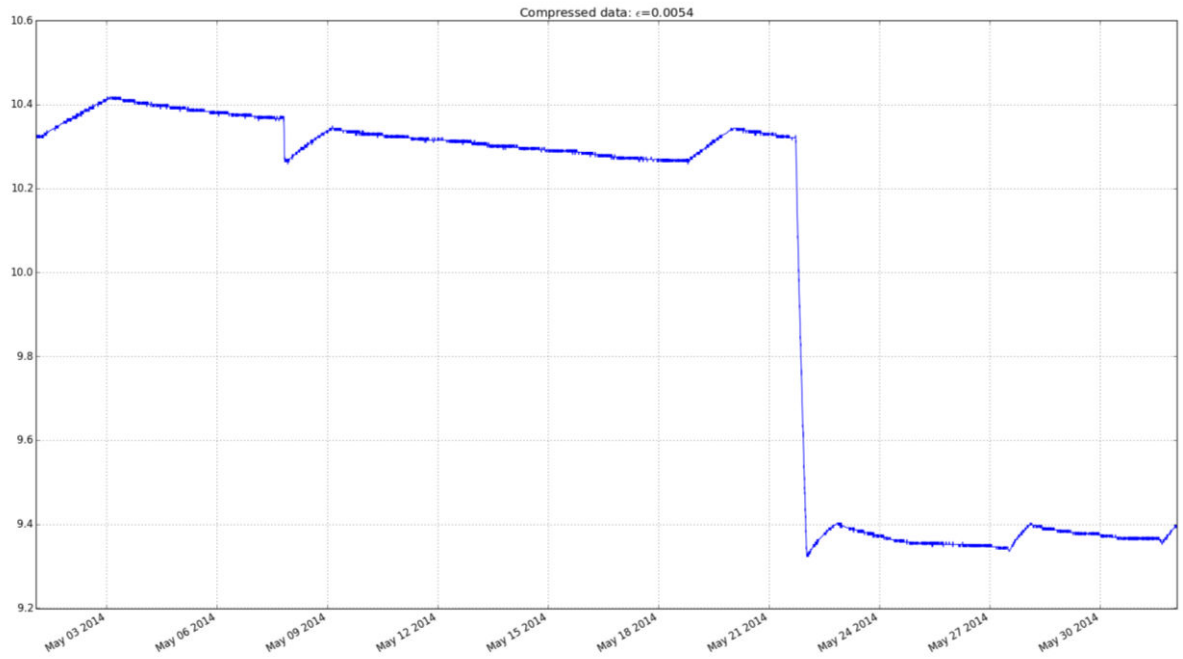


Plot 3 Uncompressed data

[7] Parameter NAAD0331 from May 2014

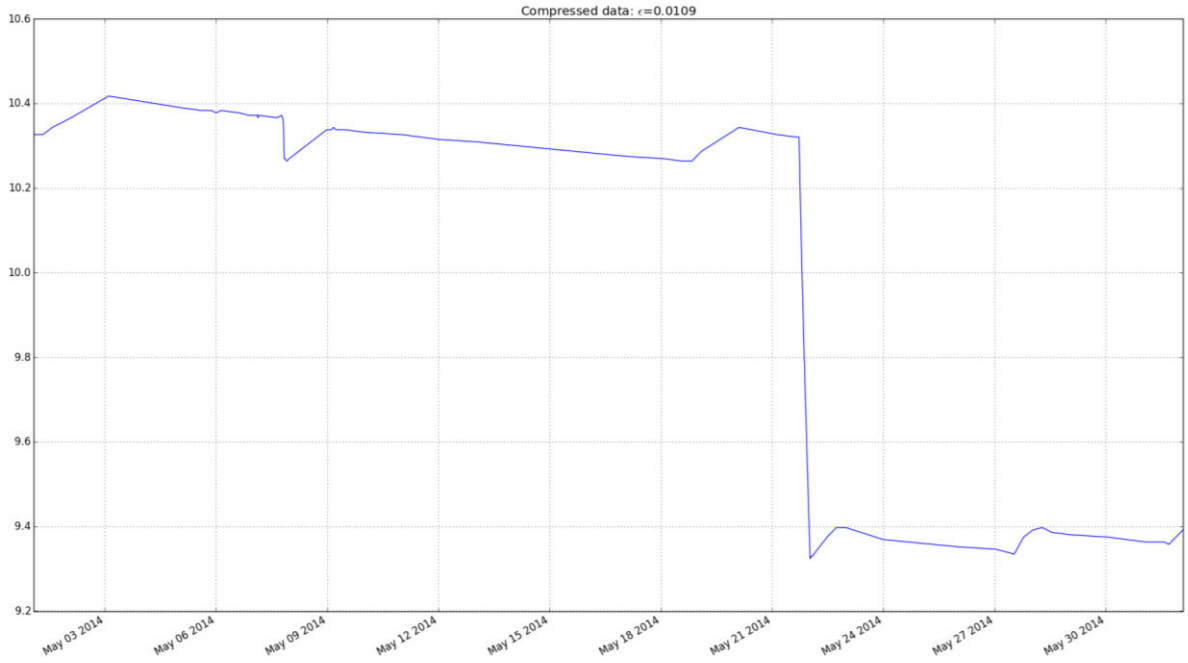
[8] Bar is metric, but not SI unit of pressure. 1 bar is about equal to the atmospheric pressure on Earth at sea level [8]. 1 bar = 100 000 Pa [7]

When the time series is compressed with maximum error $\varepsilon = 0.0054$ [bar], which is equal to 0.5% of the time series' range. The compressed time series, shown in Plot 4, consists of 6636 samples. Comparing it to the raw data, it is evident that there was a reduction in samples. To be precise, 36 987 samples were omitted which translates into 84.8% of the original data and the compression ratio 6.573. The mean squared error of the compressed time series, using 15.2% of original data, is 4.61×10^{-6}. A quick glance at the plotted data reveals that the compression is not good enough. There is still quite a lot of noise, which is not giving any useful information.



Plot 4 Compressed data with 0.5% relative error

With $\varepsilon = 0.0109$ [bar], which is 1% of the time series' range, the compressed time series now consists of only 77 samples. The compressed time series, in Plot 5, is now much cleaner after omitting 43 546 samples, which is 99.82% of the original data and the compression ratio 566.5. The most, if not all, important information is still in the data. The mean squared error of the compressed time series, using only 0.18% of the original data, is 8.77×10^{-6}.

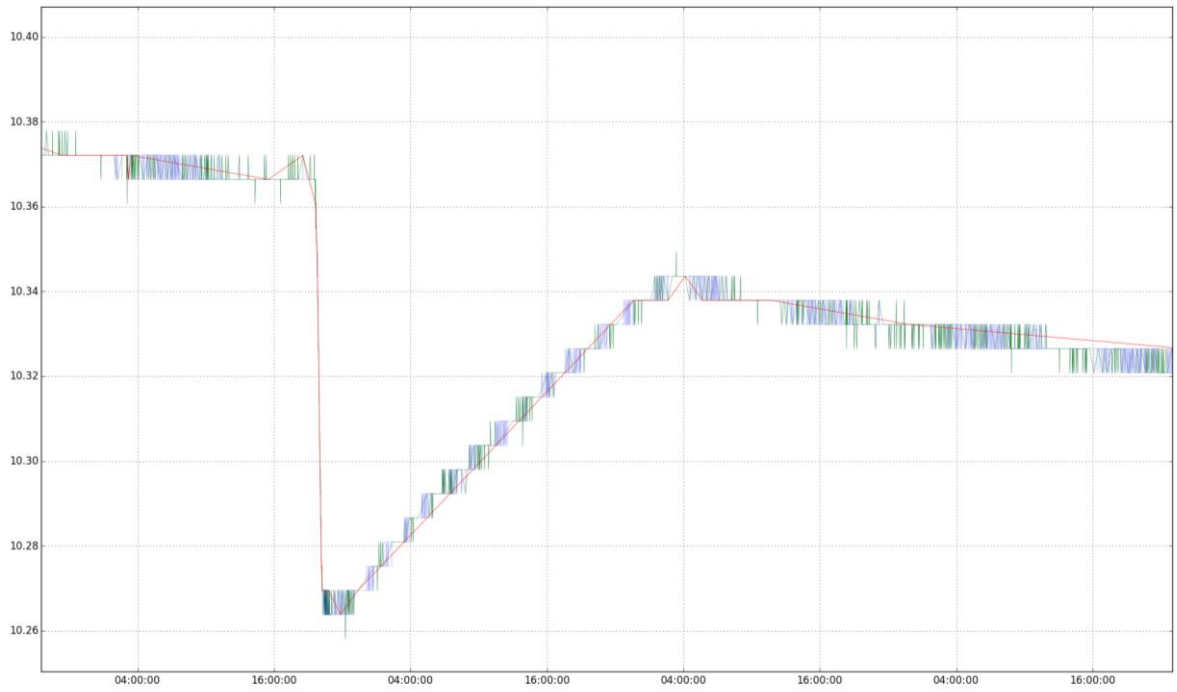


Plot 5 Compressed data with 1% relative error

To provide a better perspective, in Plot 6 is a detail with three of previously mentioned time series overlapped in one plot. It also show which samples have been omitted. The plot is composed of three layers.

- The first layer (blue) is the original time series
- The second layer (green) is the original time series compressed with 0.5% predetermined relative error
- The third layer (red) is the original time series compressed with 1% predetermined relative error

A layer with higher number is covering up the previous layer. The second layer covers an area that it used and effectively showing removed samples. From a perspective of the second layer, everything that is coloured in blue was omitted.



Plot 6 Direct comparisons. Blue line: original data, Green line: 0.5% compression, Red line: 1% compression

3.2.3 Predetermined error settings

The predetermined error should be a typical error threshold for the observed system, wherein fluctuations within the error threshold correspond to background noise and/or fluctuations that do not indicate a particular event or abnormality, and, thus, are not required for further analysis. [25]

"*By Way of example, if the second predetermined error value is set to Zero, then all the data points of the time series sequence determined by the selection value are outputted.*" [25]

The idea behind the quote is to run the algorithm with two predetermined error values. The first one is higher in order to return a "thumbnail" of the time series. The thumbnail is a very coarse approximation of the original time series and if it shows something unusual, or something worth investigating, then the second predetermined error is used. The second one is the predetermined error that is used for the compression of the time series that would be actually examined. If it is set to zero, all samples are retrieved without any compression.

The predetermined error has to be set for each parameter. This is probably the biggest drawback of the original algorithm, because it requires a certain knowledge of a system and it could be a very time consuming procedure.

I am using two types of a predetermined error. The first type is an absolute maximum predetermined error when it is a matter of one certain parameter. The second type is a relative maximum predetermined error, which I am using during comparison of both algorithms, globally for all parameters. The relative error is calculated from the time series' range.

3.2.4 Statistical measurements

Both algorithms have the same upper bound for their statistical quantities like the minimum, maximum, range, average, variance, standard deviation, and mean squared error.

Statistical quantity	Upper bound with predefined error ε
Any sample	ε
Maximum	ε
Minimum	ε
Range	$2 * \varepsilon$
Geometric average	ε
Geometric variance	$6 * \varepsilon$
Geometric standard deviation	$6 * R * \varepsilon$
Mean square error	ε^2

where ε is the maximum absolute error and R is the range of the time series.

The average, variance and standard deviation need to be geometric, because it is necessary to adjust for the irregular sampling rate.

The way these bounds are calculated and mathematical proofs can be found in document[9] [27].

[9] https://www.slideshare.net/secret/fIf4NgaDLru2qQ

3.3 Test method

I have two months of telemetry data from Rosetta spacecraft. The complete set has about 16 000 parameters, from which I have available about 10 000 for testing. After further selection, excluding some of the parameters that did not have any samples or not enough samples, I used about 2 100 parameters. I did not include parameters which had less than 1000 samples and for the measurements I used first 10 000 samples of each parameter.

I picked March 2010 and May 2014 for testing. March 2010, because it was a relatively uneventful month while Rosetta was traveling through the Solar System and May 2014, because this was a relatively eventful month as the spacecraft started approaching the comet 67P/Churyumov–Gerasimenko. In May 2014, Rosetta performed a series of complex manoeuvres to reduce the distance between itself and the comet from around 20 million km to 100 km [28].

I will be processing each telemetry parameter as if it was sampled with an irregular sampling rate, because some of the parameters were showing small sampling irregularity (1-2 milliseconds).

The telemetry data are confidential and they will not be available for repeating the experiment. When I mention a specific parameter, I will refer to it, for example, as NAAD0331 – tank pressure (bar).

3.3.1 Comparing original and modified algorithm

Comparing the algorithms requires working with the original time series, denoted as X, from which is the compressed time series, denoted as X', created. In order to compare the compressed and original time series, changes to the compressed time series are necessary, because $X' \subseteq X$. It is expected that X' will have fewer samples than X. For any measurements, it is necessary to create another time series, denoted as X'', which will represent the compressed time series X'. X'' will have same length as X. All three time series are in Figure 24.

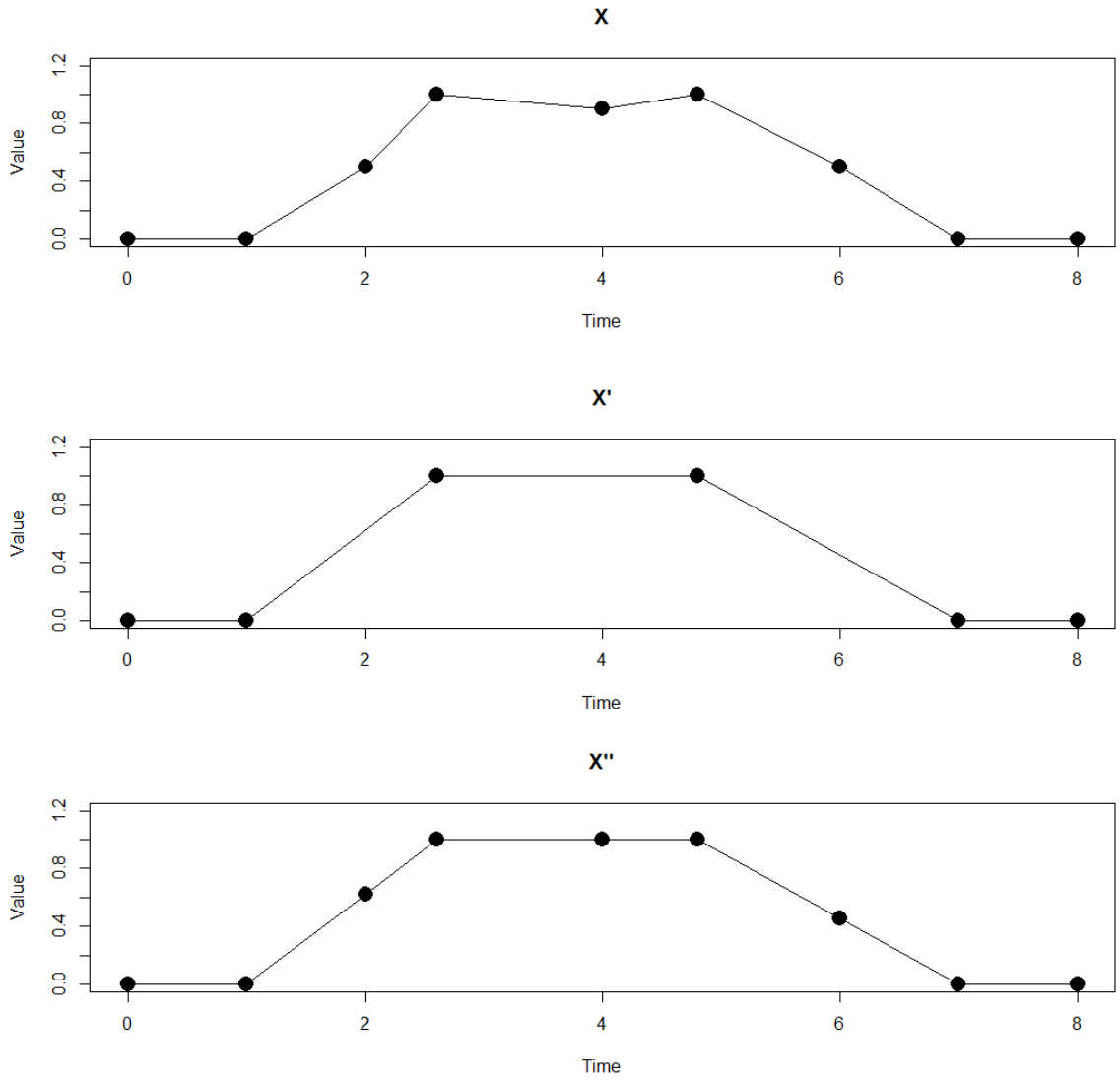


Figure 24 Time series that are used during comparison

The series X'' starts as the series $X'$and additional points are added to X'' in order to match the length of X. From samples X'_t and X'_{t+1} is taken the time component. A range from these time components is created. X'_t is the starting point of the range, denoted as t_1. X'_{t+1} is the ending point of the range, denoted as t_2. In X', t_1 and t_2 are neighbouring points, but in X, there might be samples in that range which were omitted during the compression.

A set of time components, which I denote as $T = \big(X(t_1), X(t_2)\big)$ is created. T is the set of timestamps, because the value (the recorded value) component is irrelevant. X'_t and X'_{t+1} are linearly interpolated and for each point of T is calculated a value, that lies on the interpolated line. The calculated value will be later compared to the actual value that was recorded by a telemetric recording device.

3.3.2 Implementation

The algorithms were implemented in Python 3.4 and subsequent data analysis was performed in R.

It was much more intuitive to implement the algorithms in a recursive way. I was aware of possible complications, especially the recursion depth and indeed some of the parameters reached recursion depth and eventually I ended up using an iterative way.

In a real deployment scenario, it would come down to specifics of a given system, but it would be possible to run the recursive algorithm. I was testing it on time series with lengths up to 300 000 samples, therefore it did not come as a surprise that the recursive way did not work. Figure 25 shows that the compression is significant after obtaining only 20 samples. It is not necessary to compress the whole time series at once, however compressing the whole time series leads to the best results.

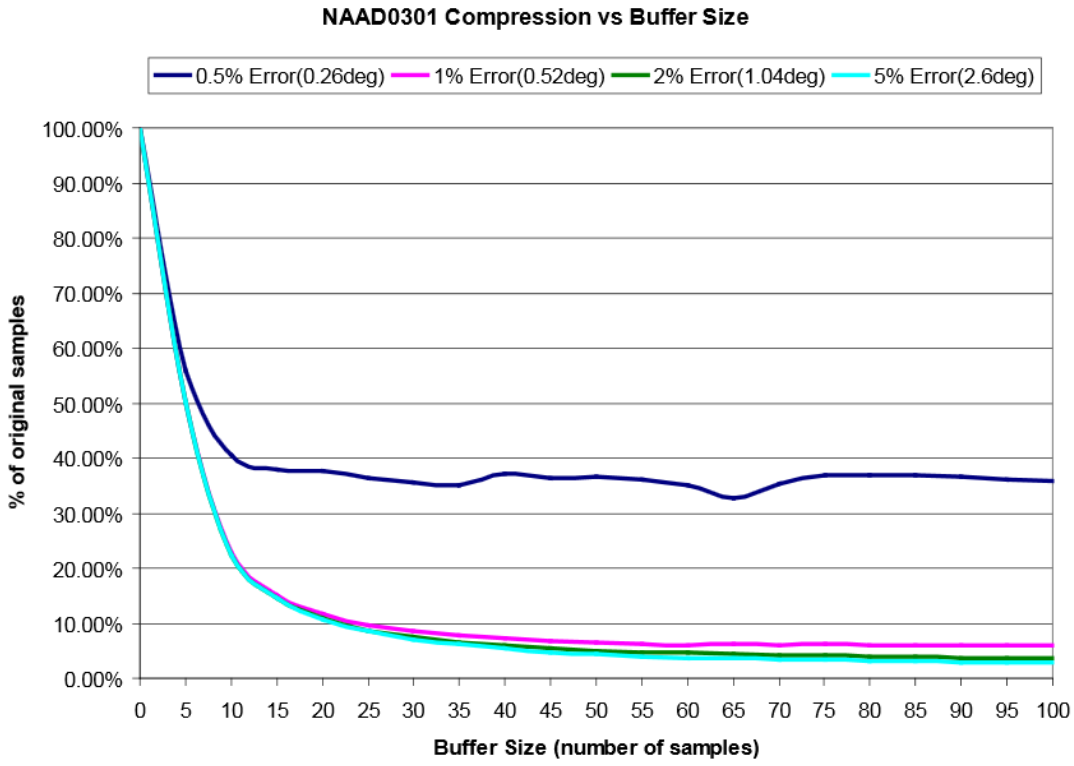


Figure 25 Buffer size and compression capabilities [29]

3.3.3 Metrics

This section describes metrics that I used for evaluating the algorithms.

Compression ratio

The compression ratio, or the data compression ratio, is commonly used to quantify the reduction in the data. It is defined as:

$$CR = \frac{S_U}{S_C} \tag{12}$$

where S_C is the compressed size and S_U is the uncompressed size. I will be using lengths of the compressed and uncompressed time series to determine the compression ratio. A time series with 100 samples that was compressed to 20 samples has the compression ratio 5:1. In tables, I will not use the "5:1" notation and if there is a number 5 in the compression ratio column, it means the ratio 5:1.

Savings

Savings, or space savings, is a relative measure. As the name suggests, it indicates how much space was saved with the compression. It is defined as:

$$Sv = 1 - \frac{S_C}{S_U} \tag{13}$$

where S_C is the compressed size and S_U is the uncompressed size. I will be using lengths of the compressed and uncompressed time series to determine the space savings. The result is in an interval $(0,1)$. When the result is multiplied by 100, it can be presented as percentage.

Mean Squared Error (MSE)

To measure a quality of the approximation of the original time series, the Mean Squared Error was calculated. The mean squared error is defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(X_i - X_i'')^2 \tag{14}$$

where n is the length of the vector of the values, X is the vector of the actual (recorded) values and X'' is the vector of the approximated values.

Relative Absolute Error and Root Relative Squared Error

In order to have another measure, similar to the mean squared error – Relative Absolute Error (RAE) and Root Relative Squared Error (RRSE) [30] has been calculated.

$$RAE = \frac{\sum_{i=1}^{n}|X_i'' - X_i|}{\sum_{i=1}^{n}|\bar{X} - X_i|} \tag{15}$$

$$RRSE = \frac{\sum_{i=1}^{n}(X_i'' - X_i)^2}{\sum_{i=1}^{n}(\bar{X} - X_i)^2} \tag{16}$$

where X_i'' is the value from the compressed time series, X_i is the actual (recorded) value and $\bar{X}$ is the average of actual values.

After reviewing the results, I started questioning their validity, because these metrics can be sensitive to outliers. Something I did not notice nor foreseen before running the computations on a larger scale. The problem is the average in the denominators. I modified these metrics and computed RAE and RRSE with a median, instead of the average. Revised metrics are be following

$$RAE_{med} = \frac{\sum_{i=1}^{n}|X_i'' - X_i|}{\sum_{i=1}^{n}|\tilde{X} - X_i|} \tag{17}$$

$$RRSE_{med} = \frac{\sum_{i=1}^{n}(X_i'' - X_i)^2}{\sum_{i=1}^{n}\left(\tilde{X} - X_i\right)^2} \tag{18}$$

where $\tilde{X}$ is the median of actual values.

Operation Count

Let us look back at section 3.2 and the algorithm steps, specifically step 3 – the step where the distance of a sample from the interpolated line is calculated. When the distance is calculated, the operation count is incremented by 1.

I will continue with the example that is on Figure 23.

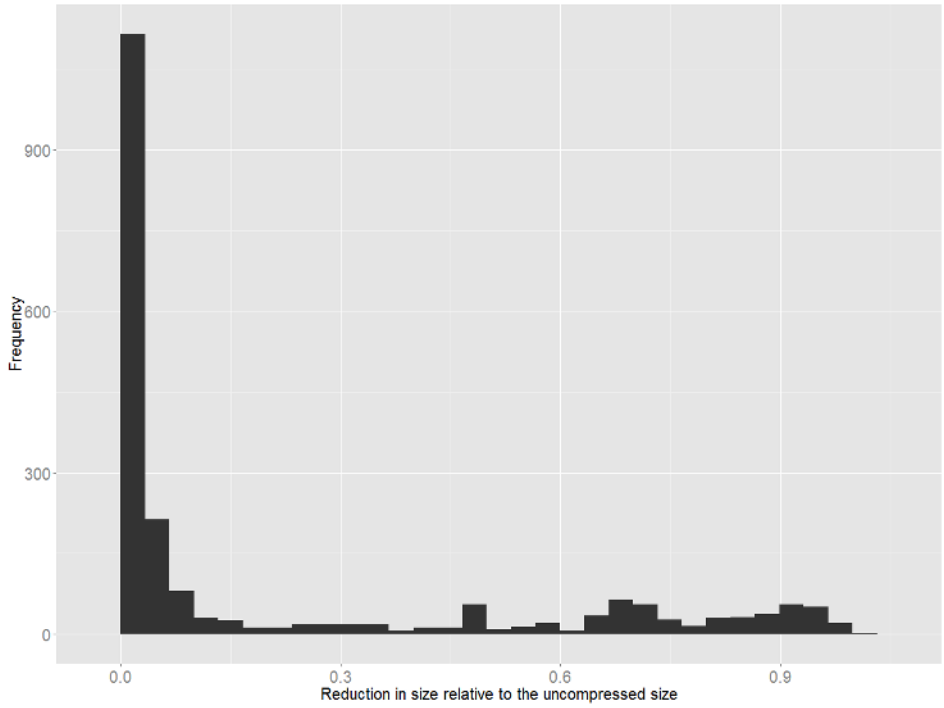
- In case of the original algorithm, after the first step, the counter *c* have value $c = 1$, because only one sample, B, was checked.
- In case of the modified algorithm, after the first step, the counter *c* have value $c = 6$. It is possible to predetermine the value of c, because by the definition, the whole segment is checked, then $c = l - 2$, where l is the length of the segment.

This is a crucial metric, because it will tell how much more operations, or checks, the modified algorithm needed to compressed the data.

3.4 Results

I can confirm that removing the stopping mechanism from the original algorithm leads to a greater compression ratios, but it comes with a greater demand on resources. Tests were conducted relative predetermined errors 0.5%, 1%, 2%, 5%, 10%, 20%, 25%, 50% and 90%.

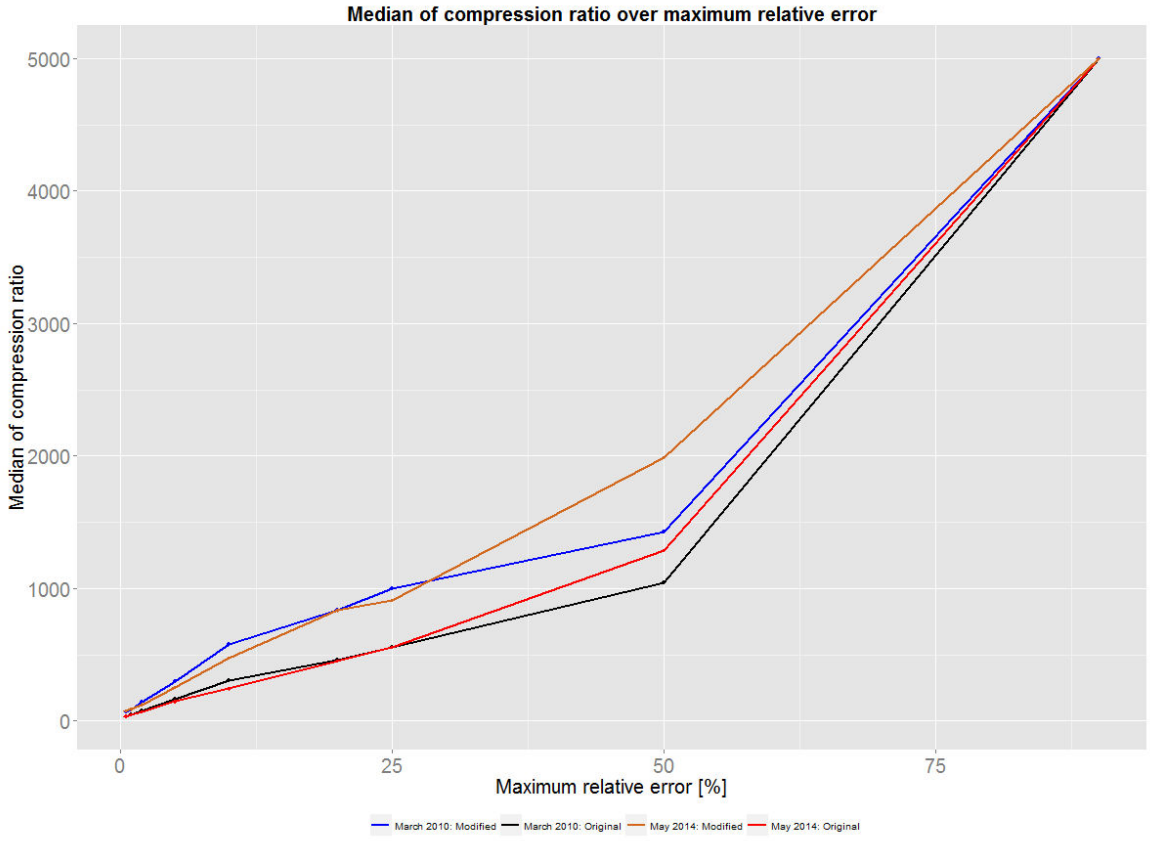
In this section, I will present the comparison of the original and modified algorithm. I decided to use a median over average as aggregation function. Plot 7 shows reduction in the data relative to uncompressed size with the relative error $\varepsilon = 0.5\%$. The distribution would shift to the right with increased relative error.



Plot 7 Reduction in the data relative to uncompressed size

3.4.1 Compression and costs

The algorithms, on a large-scale test, are closest at the relative error $\varepsilon = 0.5\%$. The gap widens until $\varepsilon = 50\%$ and from that point, both algorithms' compression ratios begin to close again, shown in Plot 8.



Plot 8 Compression ratio

Intuitively, with the increasing relative error, the number of sample in the compressed time series is converging to two. Two is the minimal number of samples of a compressed time series. The last point in the Plot 8, 90% relative error, is very close to that limit and that is the reason why all four lines are coming together.

In Table 2 are numerical results of the whole test. The designation v1 and v2 comes from my implementation, where I used number 1 to run the original algorithm and number 2 to run the modified algorithm. The relative difference is obtained as $\frac{v2-v1}{v1}$ and absolute difference as $v2 - v1$.

	Compression ratio				Relative difference	
	March 2010		May 2014		March	May
ε	v1	v2	v1	v2		
0,5	1528,9	1585,5	1520,2	1582,7	3,70%	4,11%
1	1538,3	1599,1	1528,6	1594,5	3,95%	4,31%
2	1552,9	1620,2	1545,0	1616,8	4,33%	4,65%
5	1612,4	1686,4	1596,9	1686,5	4,59%	5,61%
10	1733,5	1825,3	1642,9	1754,5	5,29%	6,79%
20	1830,8	1953,7	1716,9	1862,9	6,72%	8,51%
25	1884,6	2009,8	1749,5	1903,4	6,64%	8,79%
50	2216,4	2312,4	2186,4	2306,5	4,33%	5,50%
90	3419,0	3417,3	3708,9	3728,9	-0,05%	0,54%

Table 2 Compression ratios on full set

The compression ratios are aggregated through average, which is the reason for such high values. I used an average, because I wanted to have a different perspective, opposed to the median, which I used in the charts above. Such high numbers are given by the amount of "flat lines" in the set of tested parameters. Each parameter is limited to 10 000 samples, which capped the compression ratio to 5 000. When a compressed time series is only few samples long, one sample can make a huge difference in the compression ratio, but in a compressed time series with 5 000 samples, one sample will have much smaller impact on the compression ratio.

	Total savings				Absolute difference	
	March 2010		May 2014		March	May
ε	v1	v2	v1	v2		
0,5	0,77	0,81	0,77	0,81	4,02%	4,19%
1	0,78	0,82	0,79	0,83	3,80%	3,93%
2	0,81	0,84	0,82	0,85	3,48%	3,41%
5	0,84	0,87	0,85	0,88	2,96%	2,98%
10	0,87	0,90	0,87	0,90	2,57%	2,59%
20	0,90	0,92	0,91	0,93	1,95%	1,70%
25	0,92	0,93	0,92	0,94	1,46%	1,31%
50	0,98	0,98	0,98	0,98	-0,06%	-0,02%
90	0,99	0,99	0,99	0,99	-0,39%	-0,30%

Table 3 Savings on the full set

Savings values are more representative than compression ratios. The results are somewhat consistent with the compression ratios in Table 2. On average, the modified algorithm achieves about 4% higher data compression. An interesting situation occurred at 50% and 90% relative error, because the original algorithm outperformed the modified algorithm.

The difference in the compression ratios between March and May is there due to the different flight phase of the spacecraft.

Plot 9 Number of operations

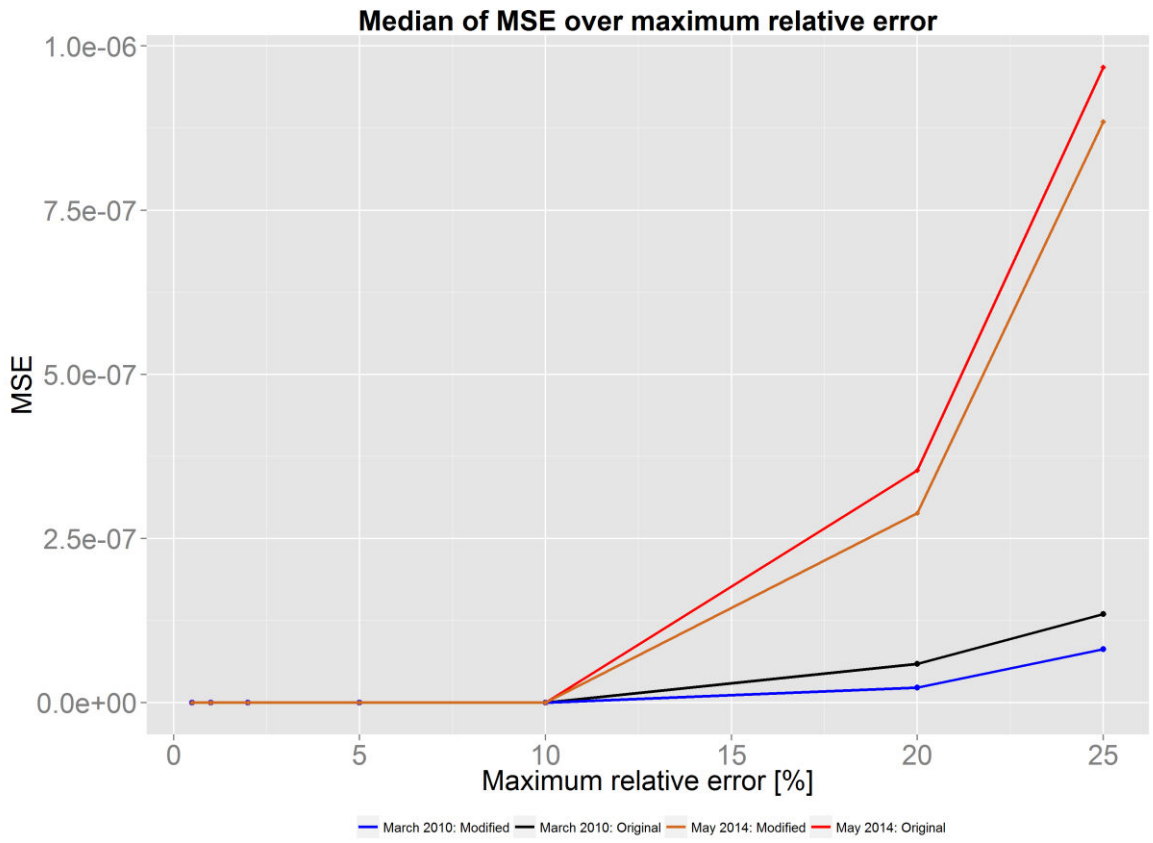
Plot 9 is giving out information about the increased demand of the resources. The original algorithm is running, efficiently, with almost a constant intricacy. On the other hand, the modified algorithm has a rather exponential progress.

3.4.2 Quality of compressed data

This section is about the quality of the approximation that is the compressed time series providing. I removed values for 50% and 90% relative error, because those values are not that relevant to the outcome.

MSE

The mean squared error is defined in section 3.3.3. From the definition, it is obvious that if $MSE = 0$, then the compressed time series will provide the same information as the original time series, but using less samples – this the ideal scenario.



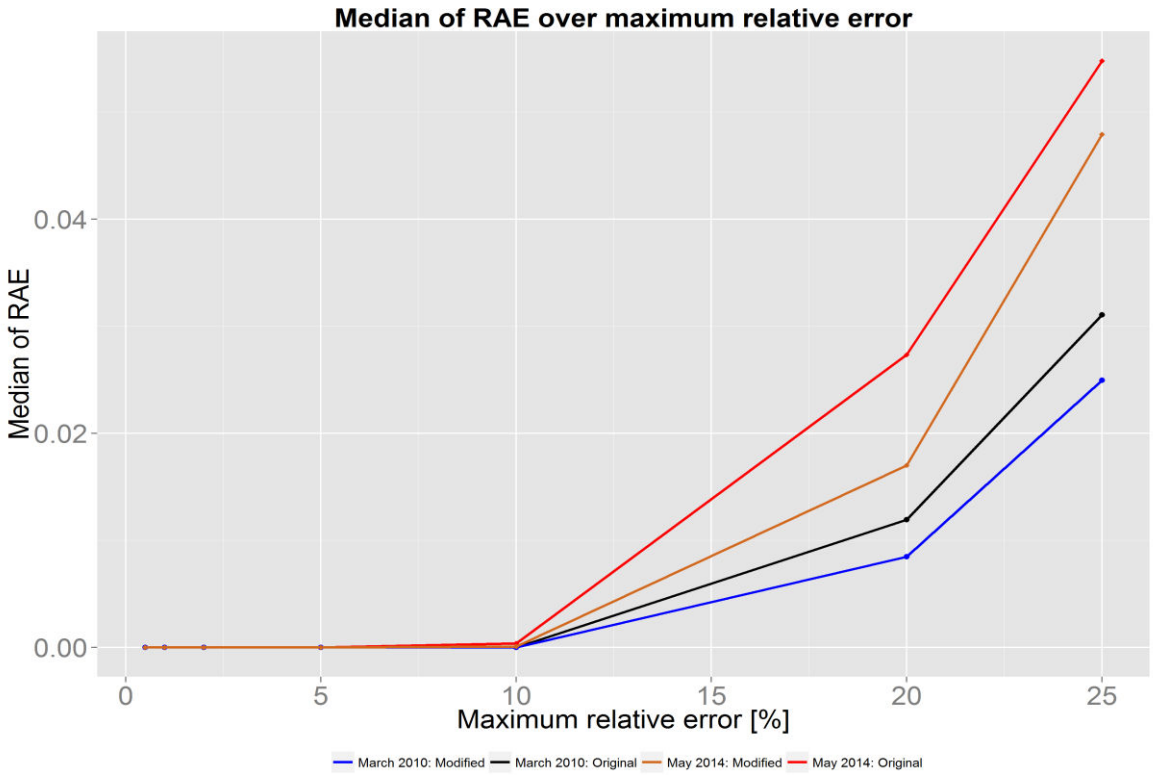
Plot 10 Media of MSE over maximum relative error

Plot 10 is showing rather interesting values up to 10%. The reason is the median, which I used and the majority of MSEs were zeros. The vast majority of MSEs ended up in a range $(0, 1)$ and a very little number of MSEs reached up to 7×10^{13}.

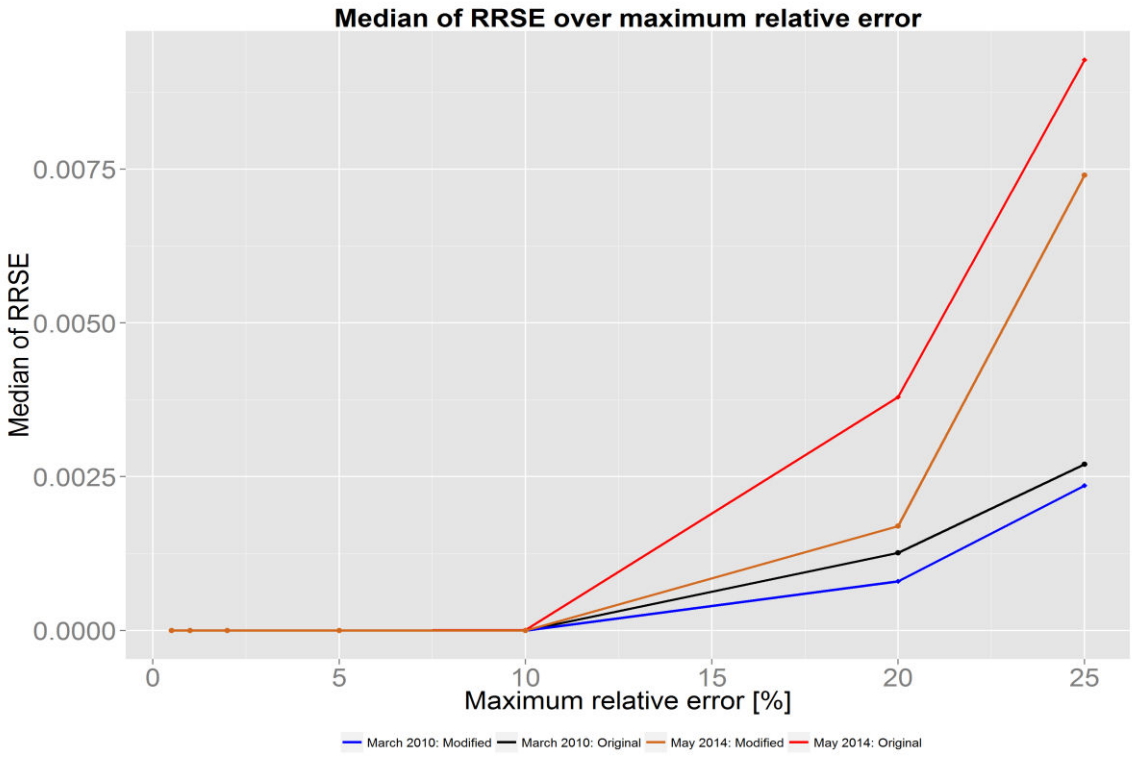
After the first batch of tests, I realized that I have to measure it differently and eliminate absolute values.

Relative absolute error

The relative measures are defined in section 3.3.3. Both results are very similar to the mean squared error until 10%. The information loss is minimal and the modified algorithm is showing slightly better results. To explain the mysterious range $(0.5\%, 10\%)$ I created histograms, shown in Plot 13, which will explain the zero values in this range.

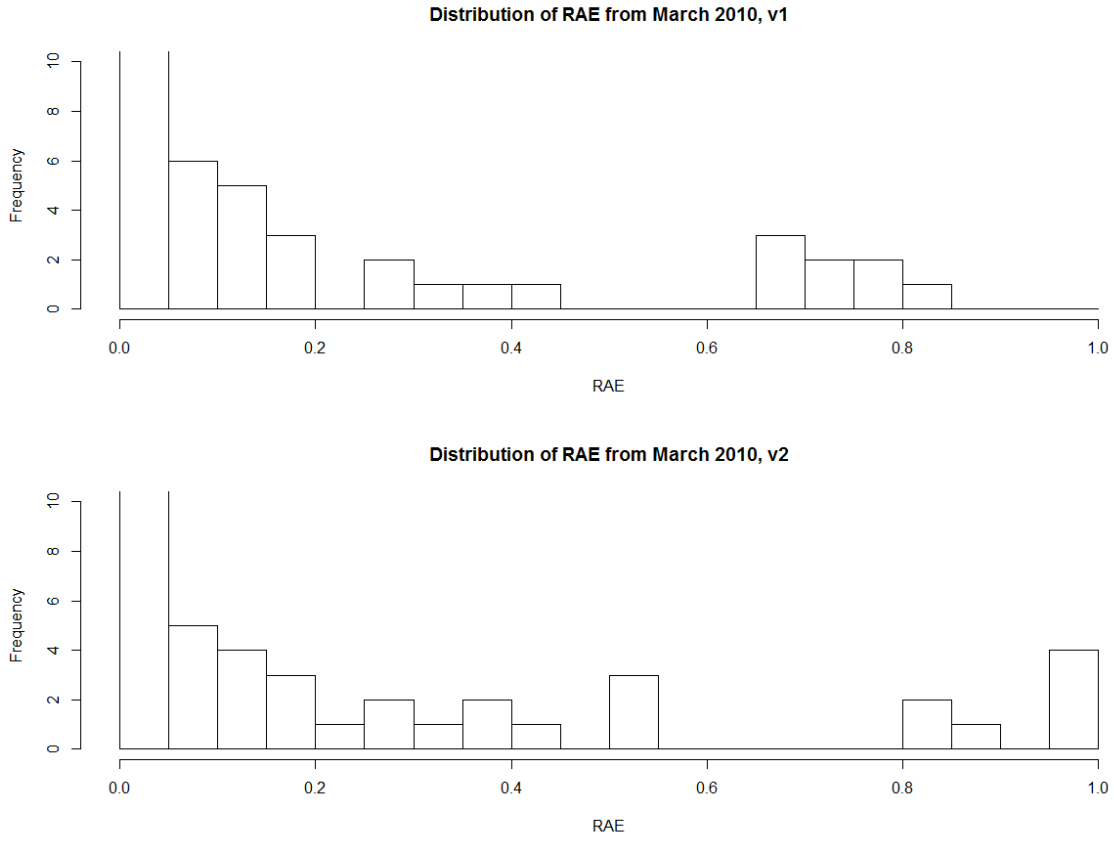


Plot 11 Relative absolute error



Plot 12 Relative root squared error

The columns of both histograms are 5% wide and the y-axis is limited to 10 samples, otherwise the number of samples in the first column would scale down the remaining columns. The first column in the first histogram has around 2080 samples and the remaining 30 are distributed in range $(0.05, 1)$. The situation is similar in the second histogram where the first column has about the same number of samples, but a different distribution of the remaining samples.



Plot 13 Distribution of RAE

3.4.3 Additional evaluation

I decided to reanalyse the results after testing the algorithms as a lossless compression, in section 3.5, where I used three categories to divide the parameters. The categories were low, medium and high. The reason for creating those categories was to help with an interpretation of the results and better insight into them.

The low category contains parameters, which were not that compressible, mostly because of their volatility and their compression ratio very close to 1:1. On the opposite side are the parameters from the high category. A large number of them had the range close to 0, which means that the whole time series was represented by only two samples. That pushed the compression ratio extremely high, because as the length of the time series grows,

the compression ratio is growing with it, because the time series was still represented by only two samples – the first and last sample from the original data.

It was a battle between the low and high category. The results were skewed to one of the categories, depending which one had more members and the answer was somewhere in the middle. Therefore, I decided to revise the results on a different set of parameters.

I looked at the data from the lossless compression, which I have obtained for the original algorithm, selected parameters with the range higher than 0, thus eliminating some of the "flat lines", and parameters with the compression ratio at least 1.1:1. I refer to this subset as *the medium subset*. The relative difference of compression ratios is obtained as $\frac{v2-v1}{v1}$ and absolute difference of savings as $v2 - v1$.

	Compression ratio				**Relative difference**	
	March 2010		May 2014		March 2010	May 2014
ε	v1	v2	v1	v2		
0,5	110,7	216,0	109,7	231,3	95,1%	110,9%
1	124,8	235,9	121,2	250,1	89,0%	106,3%
2	148,4	268,4	136,7	278,8	80,8%	104,0%
5	249,3	379,2	217,9	382,0	52,1%	75,3%
10	464,9	623,2	294,8	494,4	34,0%	67,7%
20	619,5	831,1	413,9	668,0	34,2%	61,4%
25	707,8	919,5	467,6	730,9	29,9%	56,3%
50	1200,0	1369,6	1204,0	1419,0	14,1%	17,9%
90	2784,0	2766,2	3225,9	3269,2	-0,6%	1,3%

Table 4 Compression ratio of medium subset, aggregated by an average

	Total savings				**Absolute difference**	
	March 2010		May 2014		March 2010	May 2014
ε	v1	v2	v1	v2		
0,5	0,730	0,791	0,754	0,818	6,1%	6,5%
1	0,745	0,802	0,770	0,830	5,7%	6,0%
2	0,770	0,821	0,793	0,845	5,0%	5,2%
5	0,812	0,852	0,824	0,866	4,0%	4,2%
10	0,839	0,874	0,844	0,880	3,5%	3,6%
20	0,867	0,896	0,872	0,900	2,9%	2,8%
25	0,883	0,906	0,888	0,911	2,3%	2,3%
50	0,964	0,966	0,965	0,968	0,2%	0,3%
90	0,987	0,980	0,987	0,982	-0,7%	-0,5%

Table 5 Total savings of medium subset, aggregated by an average

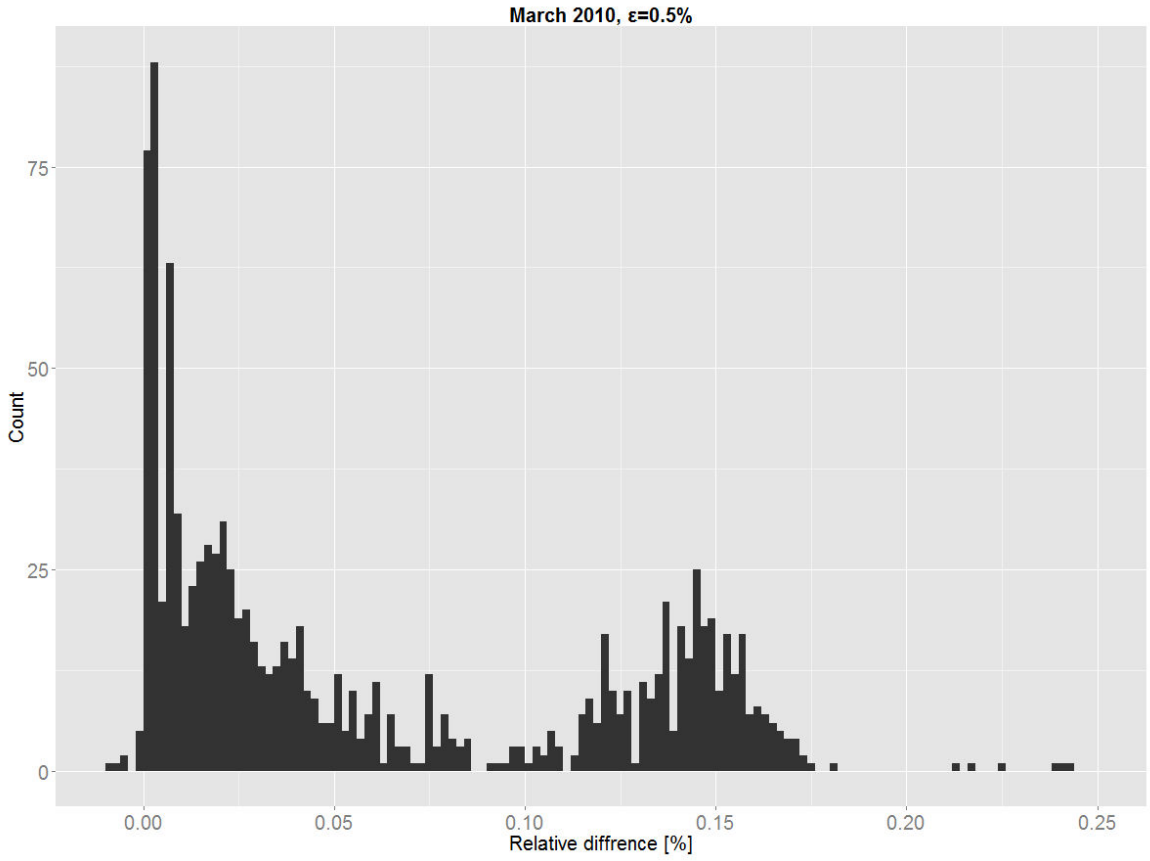
I will start with Table 5. The results are similar to the results in Table 3, but around 2% higher. The modified algorithm achieving around 6% higher compression and this advantage is disappearing as the relative error grows. Six percent is not a number that would persuade

me to use the modified algorithm. Table 4 is telling a rather interesting story of those two extra percent that were achieved on the medium subset. The numbers are much smaller compared to the numbers in Table 2, but that is not the relevant information here. It is the difference between the algorithms. On the full set, the two tables were telling almost the same story, but not in this case. In the medium subset, there are parameters that were compressed much more by the modified algorithm. In addition, the numbers were aggregated by mean, which most likely pushed the compression ratios of modified algorithm so high. A time series with 10 000 samples that compressed to 40 samples, will have $CR = 250$. However, a time series that compressed to 5000 samples will have $CR = 2$. In absolute numbers, the difference of 20 samples is negligible in the longer compressed time series.

Plot 14 is providing extra context to Table 5 and Table 7. It is a histogram of differences between algorithms. The difference *Df* is calculated as:

$$Df = \frac{l_{c,v1} - l_{c,v2}}{l_u} \tag{19}$$

where $l_{c,v1}$ is the length of the compressed time series by the original algorithm, $l_{c,v2}$is the length of the time series compressed by the modified algorithm and l_u is the length of the uncompressed time series.



Plot 14 Relative difference between algorithms

The histogram shows two "islands". The left one has clearly more members and is closer to the 0. The right one is the cause for the high difference in the compression ratios. The situation is similar with different values of ε. I will not show another 9 similar histograms.

3.4.4 Deployment scenario

In this scenario, I tried to set the maximum allowed error for each parameter individually. In the real deployment scenario, technicians should do this. I have a metadata file, which tells me what the parameter is, but it does not give any clues to what a typical error of the given system is. I used trial and error method. I set the error value, plotted the compressed time series and reaped those two steps until I felt that I am not compressing the original time series too much. I used same categories as I used in section 3.5. The relative difference is obtained as $\frac{v2-v1}{v1}$

		Compression ratio			
CR	**Parameter**	**v1**	**v2**	**Difference**	**Relative error**
High	NACW0G0W	625,0	2500,0	300,0%	0,7
	NACW0R07	1111,1	1111,1	0,0%	1
	NACG0009	5,4	8,7	60,4%	3
	NACW0F01	625,0	625,0	0,0%	50
	NACW152E	833,3	2500,0	200,0%	50
Medium	NAAD0432	42,4	25,1	-40,7%	0,7
	NAAD0431	42,0	34,2	-18,5%	1,6
	NCNAT101	38,2	56,2	47,2%	5
	NAAD0331	117,6	93,5	-20,6%	4
	NACW0K0G	113,6	44,2	-61,1%	8
Low	NPWD1274	28,6	16,9	-41,0%	55
	NAAD0503	2,0	2,4	21,3%	20
	NACW0G0J	8,6	5,8	-32,8%	30
	NPWD1434	67,6	17,5	-74,2%	30
	NAAD0603	41,5	25,4	-38,7%	40

Table 6 Test with manual error setting

Table 6 shows something I noticed before, but without realizing its true meaning. I do not want to generalize the results based on a set of 15 parameters. However, the modified algorithm, in some cases, yields the compressed time series with a lower compression ratio than the original algorithm. It means that the modified algorithm can use lower error setting, thus increasing quality of the compressed time series, while achieving the same compression as the original algorithm.

This is a rather contrasting and baffling result, because all the tests I ran and evaluated were with the same error setting and yet the modified algorithm came out as the winner, by about 6%. According to these results, it should lose.

3.5 Lossless compression

From the beginning, I was looking at these algorithms as lossy compression algorithms. The patent does not even mention lossless possibility. Unfortunately, the idea to test it as a lossless compression algorithm came very late and I did not have time to perform measurements on such a scale as I did for the other predetermined errors.

In order to obtain at least some measurements, I handpicked a few parameters and compared only those. I looked at the compression ratios from the previous measurements, for $\varepsilon = 0.5\%$, and picked 5 parameters, which had compression ratio above 500, around 20 and below 1.5. Only the first 10 000 samples of each parameter were used. The relative difference of compressed lengths is obtained as $\frac{v2-v1}{v1}$ and absolute difference $v2 - v1$.

		Compressed length		**Difference**		**Number of operations**
CR	Parameter	v1	v2	Absolute	Relative	Relative
High	*NACW0G0W*	17	4	13	76,47%	-108,3%
	NACW0G0K	19	4	15	78,95%	-108,2%
	NACG0009	4617	2394	2223	48,15%	-1650,8%
	NACW0F01	16	16	0	0,00%	-70,8%
	NACW152E	15	4	11	73,33%	21,0%
Medium	*NAAD0432*	7426	6546	880	11,85%	-1483,1%
	NAAD0431	5650	4828	822	14,55%	-1172,1%
	NCNAT101	318	193	125	39,31%	-715,7%
	NACW1401	109	68	41	37,61%	-166,3%
	NMDAT101	716	599	117	16,34%	-230,3%
Low	*NPWD1274*	9992	9974	18	0,18%	-1559,3%
	NAAD0503	9953	9877	76	0,76%	-2281,8%
	NACW0G0J	10000	10000	0	0,00%	-1660,8%
	NPWD1434	9848	9679	169	1,72%	-1767,0%
	NAAD0603	9885	9838	47	0,48%	-1530,5%

Table 7 Lossless compression comparison

The results in Table 7 are showing a strong dependency on the data that are being compressed. In some cases, in the low category, the modified algorithm achieved the same results as the original algorithm, but need much more operations to produce the compressed time series. The situation with demand on the resources is similar in the other two categories, where the modified algorithm needed much more operations to achieve compression.

The modified algorithm needed further adjustment in the step when checking the distances of samples from interpolated line. If all the distances were equal, it used the first sample as the displacement point. I borrowed an idea from the original algorithm and used the midpoint as the displacement point for cases like this.

I hesitate to make a definitive conclusion. I encountered difficulties with the speed of the modified algorithm that I was not able to eliminate in time. I started experimenting with CUDA[10], but as I mentioned above, this was a late idea and I could not produce a working implementation in time.

Nevertheless, the selection of parameter in Table 7 represents a fairer set. Fairer in a way that each category is equally represented and each parameter had 10 000 samples. On average, the lossless modified algorithm eliminates about 7 % more samples. In addition, the way I picked the parameters in Table 7 motivated me to produce an additional set of plots and results for the medium category, but on a larger scale.

I tested the data from March 2010 and May 2014 with the original algorithm. The original algorithm had no issues with speed there. The algorithm is blisteringly fast even in the lossless setting. This was anticipated as, because Plot 9 has already showed that it is performing almost equally regardless of the setting of the maximum error. The data from March have compression ratio $3.03: 1$ and the data from May 2014 $2.97: 1$.

3.6 Future work

I do not think that there are many improvement possibilities on the original algorithm. It is efficient and fast, but there are areas, which could ease up setting up the predetermined errors for larger number of parameters.

The modified algorithm gave me much more things to think about, because it caused me much more troubles. These troubles came in scenarios when I was testing larger volumes of data. The biggest problem was the overall speed. Just to give an idea, the original algorithm needed about 4 minutes to compress a set of 18 539 737 samples. The modified algorithm needed 97 minutes and that number would rise as the predetermined error decreases. These numbers are for relative error $\varepsilon = 10\%$.

[10] Compute Unified Device Architecture – technology from NVidia enabling parallel computation by using GPUs

Here is the list of areas, which might lead to a higher compression speed:

- Exploring options whether it would be possible to infer some information about a segment from the preceding segment, i.e. segments with low variance in data could be skipped without the necessity to check it.
- Ranking the segments in a queue, by length or measuring some other statistical quantities. Then, for example, processing shorter segments first. This could improve memory usage, but I donot think it would lead to the speed increase.
- Exploring ways to traverse a (random) tree-like structure that is being created during the compression and thorough exploration of Cantor set theory in relation to binary trees

3.7 Conclusion

The expectations were that the modified algorithm will achieve a better compression, but it will take its toll on the performance. The question was. How much will it improve and what will be the marginal benefit?

The modified algorithm can achieve up to 25 % higher compression. However, this number heavily depends on the data. In large sets of time series, of Rosetta's telemetry, the modified algorithm achieved only 4 % to 6 % higher compression.

The modified algorithm is achieving better results, compared to the original algorithm, as the predetermined maximum error decreases, reaching the peak when the algorithms turn into lossless algorithms. The heavy dependency on the data remains and the modified algorithm needs significantly more resources to compress a time series.

There are several ways to interpret the results, either by looking at the whole set of parameter or dividing it into intervals. Both ways will show different numbers, but similar conclusions. I will close this work with three points:

- The modified algorithm can achieve up to 25% higher compression than the original algorithm
- The modified algorithm can achieve same compression as the original algorithm with lower setting of the maximum predetermined error
- The modified algorithm needs significantly more resources to compress time series

References

[1] K. Falconer, Fractals - A very short Introduction, Oxford University Press, 2013.

[2] H.-O. Peitgen, H. Jürgens and D. Saupe, Chaos and Fractals - New Frontiers of Science, 2 ed., New York: Springer, 2004.

[3] B. B. Mandelbrot, The Fractal Geometry of Nature, W. H. Freeman and Co., 1982.

[4] Wolfram, "Hilbert Curve," [Online]. Available: http://mathworld.wolfram.com/HilbertCurve.html.

[5] L. M. Alves, "Foundations of Measurement Fractal Theory for the Fracture Mechanics," in *Applied Fracture Mechanics*, 2012.

[6] K. Falconer, Fractal Geometry: Mathematical Foundations and Application, John Wiley & Sons, Ltd, 2003.

[7] D. G. Allen, "Babylonian Mathematics," College Station, 2002.

[8] S. Khan, "Area of Koch snowflake (part 2) - advanced," Khan Academy.

[9] Wolfram Research, Inc., "Mandelbrot Set," Wolfram Research, Inc., [Online]. Available: http://mathworld.wolfram.com/MandelbrotSet.html.

[10] M. Shishikura, "The Hausdorff dimension of the boundary of the Mandelbrot set and Julia sets," 12 April 1991. [Online]. Available: http://arxiv.org/pdf/math/9201282v1.pdf.

[11] Wolfram MathWorld, "Coastline Paradox," Wolfram, [Online]. Available: http://mathworld.wolfram.com/CoastlineParadox.html.

[12] B. B. Mandelbrot, "How long is the coast of Britain?," *Science,* no. 156, pp. 636-368, 1967.

[13] Fractal Foundation, "Fractal Foundation Online Course - Chapter 1 - FRACTALS IN NATURE," Fractal Foundation, [Online]. Available: http://fractalfoundation.org/OFC/OFC-10-4.html.

[14] B. B. Mandelbrot, "Fractal landscapes without creases and with rivers," in *The Science of Fractal Images*, Springer-Verlag New York, 1988, p. Appendix A.

[15] F. K. Musgrave, “Methods for Realistic Landscape Imaging,” Yale University, 1993.

[16] J. K. Johnson, “A study of generative systems for modeling natural phenomena,” [Online]. Available: http://valhallawebdesign.com/Thesis/chapter_4.html.

[17] j. Arlt and M. Arltová, Ekonomické časové řady, Professional Publishing, 2009.

[18] J. F. Ehlers, “Fractal Adaptive Moving Averages,” *Stocks & Commodities V. 23:10,* pp. 81-82.

[19] ETF HQ, “Fractal Adaptive Moving Average (FRAMA)”.

[20] A. Jacquin, “Image coding based on a fractal theory of iterated contractive image transformations,” *Image Processing, IEEE Transactions* , pp. 18-30, 1 1992.

[21] A. Berggen, “Fractal Image Compression,” [Online]. Available: http://www.uio.no/studier/emner/matnat/ifi/INF5080/v05/undervisningsmateriale/mkt14a-FractalCompression.pdf.

[22] M. Barnsley, Fractals everywhere, San Diego: Academic Press Professional, Inc., 1988.

[23] S. S. Snyder, “Fractals and the Collage Theorem Expository Paper,” July 2006. [Online]. Available: http://scimath.unl.edu/MIM/files/MATExamFiles/Snyder_MAT_Exam_ExpositoryPaper.pdf.

[24] A. Licht and S. Jones, “GPUs TO MARS,” in *SIGGRAPH*, Los Angeles, 2015.

[25] J.-A. Martinez-Heras, T. F. Ferreira Francisco and A. Donati, “METHOD AND TELEMETRIC DEVICE FOR RESAMPLING TIME SERIES DATA”. United States Patent US 2013/0212142 A1, 15 August 2013.

[26] J.-A. Martínez-Heras, T. Francisco and A. Donati, “More Observability for Less Bandwidth ...Where’s the Trick?,” in *SpaceOps*, Stockholm, 2012.

[27] P. Nabais, “Fractal Resampling Mathematical Proofs,” European Space Operations Centre, Darmstadt, 2013.

[28] European Space Agency, “Rosetta timeline,” 25 9 2015. [Online]. Available: http://www.esa.int/Education/Teach_with_Rosetta/Rosetta_timeline. [Accessed 20 7 2015].

[29] J. A. Martínez Heras and A. Donati, “Fractal Resampling: Time Series Archive Lossy Compression with Guarantees,” PV2013, European Space Agency, Frascati, 2013.

[30] J. N. Kok, “Evaluation – next steps Lift and Costs,” Leiden Institute of Advanced Computer Science, Leiden.

[31] Bureau International des Poids et Mesures, “The International System of Units (SI),” STEDI MEDIA.

[32] Wikipedia, “Bar (unit),” [Online]. Available: https://en.wikipedia.org/wiki/Bar_%28unit%29#cite_note-BIPMSI-1. [Accessed 10 10 2015].

[33] N. Lesmoir-Gordon, Director, *The Colours of Infinity.* [Film]. 1995.

[34] National Institute of Standards and Technology, “CODATA VAlue: Planck lenght,” National Institute of Standards and Technology, [Online]. Available: http://physics.nist.gov/cgi-bin/cuu/Value?plkl.

[35] W. Hurewicz and H. Wallman, “Dimension Theory,” *Princeton Mathematical Series, v. 4.,* p. 165, 1941.

List of plots, figures and tables