TALLINN UNIVERSITY OF TECHNOLOGY
SCHOOL OF SCIENCE
DEPARTMENT OF CYBERNETICS

PLANETARY STATISTICAL TOPOGRAPHY AND THE POSSIBLE EXISTENCE OF ANCIENT MARTIAN OCEANS

MASTER THESIS

Student
Jürgen Rajasalu
212003YAFM
Supervisor
Jaan Kalda
Tenured Full Professor
Study program
Applied Physics

Tallinn 2023

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Jürgen Rajasalu .....
(signature)

Date: .....

The thesis adheres to all specified requirements

Supervisor: Jaan Kalda .....
(signature)

Date: .....

Abstract

This thesis analyses topographic data from Mars, Mercury, Earth, the Moon and a generated fractional Brownian surface (fBm) with the aim of inferring ancient water levels on Mars. Specifically, a number of scale-free characteristics of isolines, such as fractal dimensions and other scaling exponents, are computed for subsets corresponding to a fixed elevation and presented as functions of elevation. Chapter 2 provides a theoretical background to fractals and statistical topography, while Chapter 3 outlines the methods used to collect the data for analysis. The results are presented and examined in Chapter 4, followed by conclusions and suggestions for future improvements.

The results show that while Earth exhibits a sudden jump in fractal dimensions at the sea level, and decreased values of the fractal dimensions at elevations close to the sea level, no such jumps or decreased values of the fractal dimensions are observed for the other celestial bodies studied — Moon, Mercury and Mars. While for the Moon and Mercury the fractal dimensions remain almost constant over the whole range of altitudes, for Mars, the fractal dimensions tend to be smaller at high altitudes (above 3000 m) than elsewhere; this can be attributed to the planet's hemispheric dichotomy, often hypothesised to have been caused by a giant impact — all the high altitude areas are in the southern hemisphere.

Due to the limited availability of computing resources, the present study is based on downsampled elevation data. Further research is planned to provide more accurate results and a better understanding of ancient water levels on Mars and the geological processes that have shaped these planetary bodies. This will include the development of improved algorithms that are both more efficient and less sensitive to the finite-size effects that affect the values of the scaling exponents at the smallest available scales.

Table of Contents

1. Introduction

Understanding the distribution and availability of water beyond Earth has been a focus of astrobiological research for decades, given its fundamental role as a life-sustaining molecule. Apart from acting as a protective barrier against ultraviolet radiation, water, from a biological standpoint, has indispensable thermodynamic and chemical properties that facilitate the existence and proliferation of diverse organisms. Thus, any quest for signs of life beyond our planet logically commences by investigating the presence of this vital molecule.

Discoveries of extraterrestrial water in various forms, including water ice on Mars and potential subsurface oceans on Jupiter's moon Europa [1] and Saturn's moons Enceladus [2] and Ganymede [3], have kindled hope for extraterrestrial life. While celestial bodies such as Mercury and the Moon are incapable of sustaining liquid water due to their lack of a substantial atmosphere, Mars presents a different case. Presently, Mars possesses an atmosphere substantially thinner than Earth's, causing the average atmospheric pressure to fall just below the vapor pressure of water at its triple point. However, this may not necessarily have always been the case. Hypotheses suggest that with a potentially denser atmosphere and warmer climate in its geological past, Mars could have supported vast bodies of liquid water, even oceans [4] [5].

A fascinating topographical feature of Mars is the stark dichotomy between its northern and southern hemispheres, with a difference in elevation of up to 3 kilometres. This Martian dichotomy has given rise to numerous hypotheses for its origin, one of which proposes the existence of an ancient ocean [6]. A previous study [7] attempted to ascertain the levels of ancient Martian water using statistical topography methods, but the conclusions were both ambiguous and statistically insufficient, leaving room for further exploration and more rigorous analysis.

This thesis aims to build on the foundation of previous research while addressing its shortcomings. Using advanced statistical topography techniques and a comparative approach across different celestial bodies, the aim is to gain a more comprehensive understanding of the geological history of Mars. In particular, the objective is to determine, with greater certainty, whether an ancient ocean indeed existed on Mars. This research will not only contribute to our understanding of Mars' past climate conditions but will also have profound implications for the possibility of past life and future human habitation on the Red Planet.

2. Theoretical background

2.1 Fractals

To understand fractals, one should first know what they look like. A well known fractal called the Mandelbrot set can be seen on Figure 1, the landscape of an imaginary country on Figure 2 and the Von Koch curve on Figure 3. In a sense, a fractal is a set that can be characterized by fractional dimensionality. [8]

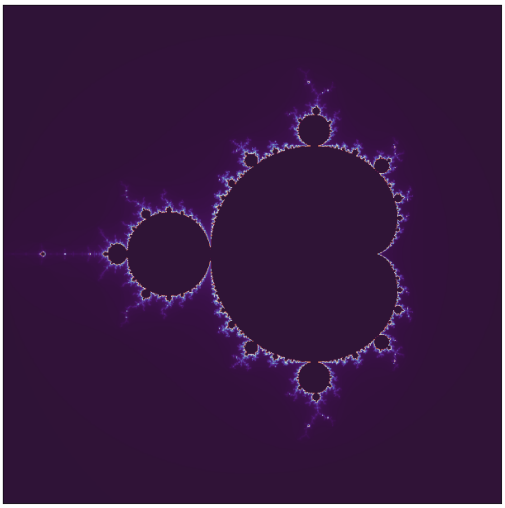

Figure 1. Mandelbrot set

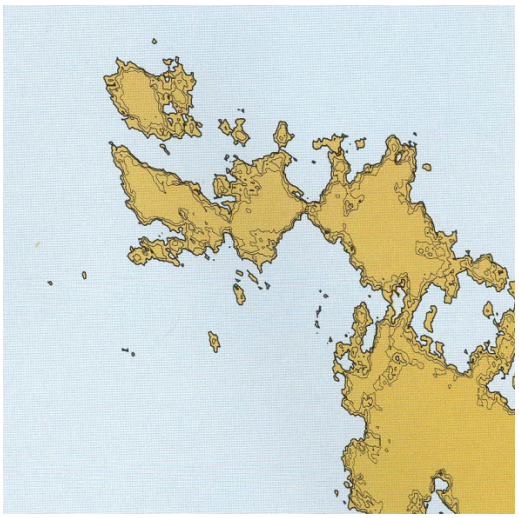

Figure 2. Fractal country [8]

Figure 3. Von Koch curve

Fractals exhibit detail at all scales and many have some degree of self-similarity i.e they are made up of parts that resemble the whole. This similarity doesn't have to be strictly geometrical, it can also be approximate or statistical. As classical geometry and calculus aren't suited to deal with fractals, new methods are required. The primary instrument

of fractal geometry is the fractal dimension. It is well known that a (smooth) curve is an example of a 1-dimensional object, while a surface is a 2-dimensional case. Fractals however have dimensions that are represented as fractions. For the Von Koch curve on Figure 1, it is $\log 4/\log 3 \approx 1.262$. This means that it is larger than 1-dimensional i.e having infinite length, but smaller than 2-dimensional i.e having zero area. [9]

One interpretation of these 'dimensions' is the following. A square can be thought about as being made up of four copies of itself that are scaled by a factor of $\frac{1}{2}$ (i.e a square with half the side length) and having the dimension $\log 4/\log \frac{1}{2} = 2$. In the same vain, the Von Koch curve is made up of four copies of itself that are scaled by a factor of $\frac{1}{3}$ with a dimension of $\log 4/\log \frac{1}{3} = \log 4/\log 3 \approx 1.26$. To generalize, a set that is made up of *m* copies of itself that are scaled by a factor *r* can be thought to have a dimension of $\log m/\log r$. This number is referred to as the *similarity dimension* of the set. [9]

Another definition for fractal dimension by Hausdorff Besicovitch goes like this: Let a fractal set *F* be covered by the "boxes" $U_1, U_2, ...$ (meaning $F \subset U_1 \cup U_2 \cup ...$) that have diameters (maximum linear size measured in *d* space) $\lambda_1, \lambda_2, ...$, respectively. $U(F, \lambda)$ represents the set of all possible coverings of F with $\lambda_i \leq \lambda$. The exterior *s*-dimensional measure" $M_s(F)$ can then be defined as

$$M_s(F) = \lim_{\lambda \to +0} \inf_{U(F,\lambda)} \sum_i \lambda_i^s. \tag{2.1}$$

Lastly, if $M_s(F) = 0$ for $s > D_H$, and $M_s(F) = \infty$ for $s < D_H$, then D_H is the "dimensional number," or the Hausdorff dimension of F:

$$D_H = \inf\{s : M_s(F) = 0\} = \sup\{s : M_s(F) = \infty\}. \tag{2.2}$$

[10]

One more way of defining the fractal dimension is the so called box counting dimension. If one were to cut a line into segments of length δ, then the number of these segments N is inversely proportional to δ for a line and δ^2 for a square.

$$N(\delta) \sim \frac{1}{\delta^D}, \tag{2.3}$$

where D represents the box counting dimension. [11]

Physial systems have a characteristic smallest length scale i.e the radius of an atom or molecule, denoted as R_0. To find a dimension here, one can imagine a linear chain of monomers. The number of these monomers in a chain of length $L = 2R$ is thus

$$N = (R/R_0)^1. \tag{2.4}$$

The asymptotic form of the relation between the number of particles and cluster size that

is measured by the smallest sphere of radius R that contains the cluster is given by

$$N = \rho(R/R_0)^D, \quad N \to \infty. \tag{2.5}$$

Here, ρ represents the number density. The exponent D is called the cluster dimension or the mass dimension and can be calculated by the following way

$$D_c = \frac{\ln N}{\ln R/R_0}. \tag{2.6}$$

[11]

It is important to realise that in theory, in the case of self-affine objects, all the different definitions for a fractal dimension give the same results. While in practise, the results may differ due to finite size effects or random deviations.

2.2 Statistical topography

Statistical topography is the study of the geometrical properties of iso-sets. These iso-sets are of a random potential $\psi(\boldsymbol{r})$ and can be either contour lines or surfaces. Coastlines are one such case. They exhibit patterns of irregularities and chaos between the border of sea and land nevertheless still having some fundamental uniformity. One example being the length of a coastline L, which depends on the resolution of the map λ as

$$L \propto \lambda^{1-D_1} \tag{2.7}$$

where D_1 is the "fractal dimension". These coastlines can be thought of as curves that connect points with the same elevation value, and are mathematically defined as iso-sets $z = \psi(x, y)$, where z is a fixed altit of the Earth's relief. [12]

By cutting through surfaces at a constant height, one can generate contour lines and islands with fractal geometry. The coastlines represented by the contour lines display fractal scaling and the length of the connected contour loops follows a power law distribution. Power law correlations are also found in the location of points on contours at the same height. The connected loops above the cutting height correspond to islands, the distribution of which also follows a power law. Furthermore, the same power law is present at all cutting heights, regardless of the fraction of area within islands. [13]

One important discipline related to the statistical topography is the percolation theory. This is essential, because the problem of statistical topography of random surfaces can be mapped to a percolation problem [14]. The percolation theory studies the formation and properties of clusters, particularly infinite clusters, as a function of the lattice site (or bond) survival probability η, cf. [10]. The percolation threshold η_c is the minimum site survival probability at which an infinite cluster appears, and near this threshold, the system

exhibits critical behavior and long-range correlations. The theory provides a means to understand how the statistical properties of random surfaces, such as their roughness, are related to the presence or absence of infinite clusters and their properties. What is particularly important in the context of the statistical topography is that there is a series of analytical exact results obtained for the percolation theory, cf. [10], that can be carried over to the statistical topography using the aforementioned mapping. It should be noted that percolation theory has been widely applied to study the behavior of porous materials, electrical and thermal conductivity in materials, and the spread of infectious diseases in populations, among others.

The basic idea of the mapping between the statistical topography and percolation problem can be outlined as follows [14]. To begin with, we build a lattice based on a landscape, provided as a function of two variables — the elevation as a function of longitude and latitude. Each local minimum of this function defines a lattice site, and the bonds are built as the steepest descent paths from each of the saddle points. One can imagine a scenario where all such bonds of this lattice survive for which the water level is higher than the elevation of the corresponding saddle point, and removed otherwise. Now, the fraction p of surviving bonds depends on the water level. Therefore, the statistical topography problem about the statistical properties of isolines becomes a percolation problem: up to what value of p=p_c will water be able to pass from one side of the system to the other, and what are the statistical properties of the percolation clusters? For surfaces with short-range correlations in elevation, there are only hills of a certain typical height and diameter. In that case, the probabilities for percolation at value p will converge to a sharp singularity at p_c as the system size increases. The transition that happens at p_c is a second order phase transition and is characterised by a set of critical exponents. Self-affine surfaces, however, are different, as their altitudes are very correlated over long distances. In this case, as the size of the system is increased, the distribution of percolation thresholds does not converge to a singularity at the percolation threshold. Meanwhile, the distribution of island sizes follows a power law that has a cutoff, which increases with system size. [15]

The height distribution function can be used to describe self-affine surfaces. In the case of such surfaces, height is invariant under rescaling $h(\mathbf{r}) \cong b^{-H} h(b\mathbf{r})$ where H represents the roughness (or Hurst) exponent ($0 \leq H < 1$ for rough surfaces). This means that for a self-affine surface, the surface height variance $\sqrt{\langle [h(\mathbf{x}) - \langle h \rangle]^2 \rangle}$ scales as L^H, where L stands for the size of the system with the average taken over x. To also take into account translational and rotational invariance of the surface, the structure function of the surface takes the form

$$C_2(\mathbf{r}) = \langle [h(\mathbf{x}) - h(\mathbf{x}+\mathbf{r})]^2 \rangle \sim |\mathbf{r}|^{2H}. \tag{2.8}$$

With this equation, one has a simple way of calculating the roughness exponent. To ascertain if a given surface is self-affine or multi-affine, one needs to measure instead the pth order structure function given by $C_p(\mathbf{r}) = \langle [h(\mathbf{x}) - h(\mathbf{x}+\mathbf{r})]^p \rangle$. The exponent hierarchy α_p varies linearly with p for a self-affine surface and nonlinearly with p for a multiaffine

surface. [16]

Since self-affine surfaces rescale differently depending on the direction, they are only fractals in a general sense. However, the level set of such a surface i.e the set of points where the surface intersects with the horizontal plane is a fractal object. By intersecting the plane at different heights, one gets statistically equivalent level sets, as the height fluctuations of a rough surface are unbounded. The contour loops that make up these level sets are expected to also be fractal, with a smaller fractal dimension than that of the whole level set (the union of all the contour loops of the same height). The size of these contour loops is only limited by the system size. It has been shown that the scaling of contour loops uniquely specifies the scaling of the associated self-affine rough surface. This is expressed in formulae giving the geometrical exponents in terms of the roughness exponent α . As a consequence, information about the out-of-plane fluctuations of the surface can be obtained by doing measurements solely on the level set. [17]

2.3 Brownian and fractional Brownian surfaces

This section provides an overview of the concept and properties of fractional Brownian surfaces, based on the review paper [10].

One simple natural example of a random process in nature is the Brownian motion of a particle due to the thermal agitation of molecules in an ambient medium. It is possible to approximate the velocity of Brownian motion, $d\mathbf{x}(t)/dt = \mathbf{v}_D(t)$, on a macroscopic time scale t as "white noise" i.e a Gaussian random function with zero mean and no covariance

$$\langle v_{Di}(t) v_{Dj}(t') \rangle = 2D_0 \delta_{ij} \delta(t - t'), \tag{2.9}$$

where D_0 represents the molecular diffusion coefficient. Thus, the coordinate of the Brownian particle, given by $\mathbf{x}(t) = \int_0^t \mathbf{v}_D(t')\mathrm{d}t'$, is also Gaussian, with an average $\langle \mathbf{x}(t) \rangle = 0$ and a covariance of $\langle x_i(t) x_j(t') \rangle = 2D_0 \delta_{ij} \min(t, t')$. It can be seen that $\mathbf{x}(t)$ does not represent a stationary random process due to the fact that its correlator is not a function of $(t - t')$. However, the delta variance is a function of $(t - t')$, which means that it can serve as a useful characteristic for this process.

Brownian motion is a suitable starting point for generating random fields. This is because by definition, the Brownian line-to-line function $\mathbf{B}(x)$ is a random function with Gaussian increments that have zero mean and a variance described by

$$\langle [B(x_1) - B(x_2)]^2 \rangle = b^2 |x_1 - x_2|. \tag{2.10}$$

This represents a self-affine fractal with a fractal dimension of $D = \frac{3}{2}$.

It is also possible to define a Brownian surface as the graph of a *Brownian plane-to-line*

function $B(x,y)$ i.e a Brownian line-to-line function of each of its arguments. The fractal dimension of such a surface can be shown to be $D = \frac{5}{2}$. A different cross section can also be made fro ma horizontal plane $B(x,y) = h = const$, with the fractal dimension, $D = \frac{3}{2}$, of the isoset $B(x,y) = h$.

Because the value $\frac{3}{2}$ for D considerably overestimates the measured fractal dimensions of natural coastlines, a generalization was introduced - a *fractional Brownian function* denoted by $B_H(\mathrm{x})$, where x is a d-dimensional argument. It is by definition a Gaussian random process with delta variance

$$\langle [B_H(x_1) - B_H(x_2)]^2 \rangle = b^2 |x_1 - x_2|^{2H}, \tag{2.11}$$

where the Hurst exponent H characterizes the spectrum of B_H. The particular case where $H = \frac{1}{2}$ represents the ordinary Brownian function $B(\mathrm{x})$ i.e $B(\mathrm{x}) = B_{1/2}(\mathrm{x})$. The fractional Brownian graphs for varying values of H are qualitatively similar, differing only by their degree of irregularity, which increases for decreasing H. The fractal dimension D of a graph can thus be defined by the measure of this irregularity. In the scaling range $\lambda < b^{1/(1-H)}$, the fractal dimension is $D = 2 - H$. In an infinite scaling range, the same dimension would refer to the horizontal cross section (isoset) of the fractional Brownian surface $z = B_H(x,y)$, with a fractal dimension that is unity greater i.e $D = 3 - H(\lambda < b^{1/(1-H)}$. A random potential with the delta variance and with $H = 0$, the resulting graph achieves the maximum allowed fractal dimension $D = d = 2$, that is the dense filling of the (x,y) plane by the short-scale oscillations of the potential.

One object, that is related to the function B_H, is the trajectory of fractional Brownian motion (or the fractional Brownian trail), which is given by $\mathrm{x} = B_H(t)$, in which every component of $B_H(t)$ is an independent fractional Brownian function of one-dimensional time. The fractional Brownian trail is a self-similar fractal as opposed to the fractional Brownian graph, which is self-affine. The time that is neccessary to pass the distance λ in a given box with the same size, is given by $t_\lambda \propto \lambda^{1/H}$. The covering number of such a trajectory traversed over the time period T would then be $N_\lambda \simeq T/t_\lambda \propto \lambda^{-1/H}$, for $H > 1/d$. With this, the fractal dimension of the fractal Brownian trail, that is embedded in a d-dimensional space, is given by

$$D = \min(1/H, d), \tag{2.12}$$

which for $H = \frac{1}{2}$ would give the standard Brownian motion with $D = 2$. This explains the finite probability of the return of a random walker to it's starting point in two dimensions and also the zero probability in three dimensions.

There are different ways for how the fractional Brownian function $B_H(x)$ can be obtained. One option is to generate it by its Fourier spectrum. Another possibility is to obtain it directly from $B(x)$ with fractional differentiation. or fractional integration: $B_H(x) = \hat{I}^{H-1/2} B(x)$.

The Riemann-Liouville fractional integral of the a-th order is given by

$$\hat{I}^{\alpha} f(x) = \frac{1}{\Gamma(\alpha)} \int_0^{\infty} x'^{\alpha-1} f(x - x') \mathrm{d}x', \qquad 0 < \alpha < 1. \tag{2.13}$$

By substituting f with $\mathrm{d}f/\mathrm{d}x$ in the integrand, one obtains the fractional derivative of order $1 - \alpha$.

The model of a fractional Brownian surface has been used to model Earth's relief. With the fractal dimension D, one can predict the distribution of islands (i.e their number with the size of order a) with

$$N_a \propto a^{-D}. \tag{2.14}$$

This distribution is in line with the empirical number-area rule or the Korčak law: the number of islands with area above A scales as A^{-k} for $k = D/2 = 1 - H/2$. For Earth, the average value for k is about 0.65 and corresponds to the Hurst exponent of 0.7. However, local measurements can show significant deviations from this power-law surface spectrum.

The fractional Brownian approach facilitates the linking of the relief spectrum with the fractal dimension of the whole isoset $B_H(x, y) = h$ and also with the distribution of islands.

2.4 Correlated percolation

In this section, an overview is provided about the numerical and analytical results regarding the correlated percolation problem, based on Ref. [18].

The percolation lattice does not have to necessarily be completely random, but can include some correlations. It can be described with an infinite set of random variables θ_i, that are unity at occupied sites and zero at empty sites. The correlations can then be characterized through the correlation function

$$c_{\theta}(x_i - x_j) = \langle \theta_i \theta_j \rangle - p^2, \tag{2.15}$$

where $p = \langle \theta_i \rangle$ represents the site occupation probability. Correlations can also be brought into the percolation model by allocating a random number $p_i \in [0, 1]$ (where $\langle p_i \rangle = p$) to each lattice site. The site values can then be calculated as

$$\theta_i = \Theta(p_i - r_i), \tag{2.16}$$

where $\Theta(x)$ denotes the Heaviside step function and $\{r_i\}$ are independent random variables, whcih are uniformly distributed in $[0, 1]$. The correlation function in this case is given by

$$c_p(x_i - x_j) = \langle p_i p_j \rangle - p^2. \tag{2.17}$$

By putting together the equations (2.16) and (2.15), one gets $c_{\theta}(a) = c_p(a) \equiv c(a)$. The

interesting parts of this are the algebraically decaying correlations

$$c(a) \propto |a|^{2H}, \qquad H \leq 0. \tag{2.18}$$

The scaling exponents are thought to be determined by the two-point correlation function, which means that a range of universality classes are determined by the Hurst exponent H. It is possible to show that for the case $H < -3/4$, the model belongs to the universality calss of uncorrelated percolation, while in the range $-3/4 \leq H \leq 0$, the scaling exponents are effected by the correlations. The conjecture is that short-range (local) variations in the percolation lattice do not influence the scaling exponents, implying that they solely depend on the long-range correlations represented through H, and not on the intricate structure of the percolation lattice. In the case $H > 0$, the short range fluctuations have less of an impact on the value of the random potential than the long range correlations and can thus be diminished by just scaling the model. As the scaling exponents are of interest in many different areas, one should calculate their values depending on the surface's roughness parameter H. Monte-Carlo simulations show that for uncorrelated percolation, different properties converge on the value $D_1 = 7/4$. While clear outcomes could not be obtained at $H = -0.75$ and $H = 0$, extrapolations seem to show that it terminates $7/4$ and 1 respectively. Another important consequence is the rejection of a well established conjecture that implies $D_1(H) = \frac{3}{2} - \frac{H}{2}$.

2.5 Geological landscapes

This section, which discusses the main differences between real geological landscapes and random Gaussian surfaces, is based on Ref, [19].

The creation of Earth's surface is a multifaceted process, influenced by various factors such as seismic and tectonic activities, erosion, sedimentation, and more. These factors can have different origins, like erosion resulting from meandering rivers, oceanic and atmospheric impacts, ice movement, avalanches, among others. In addition, the physical attributes of the ground cover a vast range. Developing a comprehensive, simplified mathematical model that accounts for this variety is virtually impossible. However, the scale-invariant characteristics of geological landscapes have been discovered to be remarkably universal, with self-affine properties and a Hurst exponent generally ranging from 0.7 to 0.9.

Recent experiments have shown that this self-affine behavior is not flawless, with the differential Hurst exponent decreasing as a function of scale, with a value of approximately 0.8, which is characteristic of smaller scales. Consequently, it is logical to anticipate a straightforward, universal, and resilient mechanism responsible for creating such surfaces. It can be demonstrated that this mechanism can be attributed to the interplay between erosion and tectonic activity.

Many geological landscape models primarily focus on the development of river networks, while a more inclusive approach considers erosion on a slope through a stochastic equation, resulting in direction-dependent exponents of approximately 0.63 and 0.83. The formation of rivers and erosion undoubtedly play a crucial role in shaping landscapes, but they cannot raise mountain heights. The only effort to incorporate tectonic processes into a robust self-affine model of Earth's surface was proposed by Mandelbrot. His model addressed roughening caused by tectonic activity, involving the random selection of a point within a polygon and drawing a randomly oriented "fault line" that splits the polygon into two sections, one of which is elevated by a unit height. This process is then repeated numerous times.

The Brownian growth of height differences is removed by normalizing the surface height. This produces a self-affine surface with a Hurst exponent of approximately 0.5. To address the discrepancy between the model and empirical values of 0.7 to 0.9, the model was generalized by replacing the Heaviside profile of the "fault" with a profile containing a singularity. Tectonic activity and fault formation, as portrayed in the Mandelbrot model, undoubtedly play a significant role in the evolution of Earth's surface. However, singular fault profiles lack physical justification, and there are no physical processes normalizing surface height to the number of faults. Erosion, instead, serves as the fundamental factor reducing height differences. Highly detailed erosion models are not suitable for revealing the most generic aspects of landscape roughening.

2.6 Ancient Mars

The complexity of the global topography on Mars reveals a rich history of different forces that have shaped the planet at different spatial scales, and encapsulates important information about its evolution. A map of Mars is shown in Figure 4. A key feature of the Martian crust is the pronounced difference between the northern and southern hemispheres, known as the Martian dichotomy. This dichotomy is expressed topographically by the presence of heavily cratered, elevated highlands in the southern hemisphere, which contrast with the lowland plains of the northern hemisphere. The cause of this hemispheric dichotomy remains largely unclear, but several theories have been proposed, including a single mega-impact or multiple impacts in the northern hemisphere, thinning of the northern hemisphere crust due to intense regional mantle convection, a differential rotation mechanism triggered by single-plume mantle convection, and an early stage of tectonic plate recycling. In addition, it has been suggested that a vast ocean once covered the northern hemisphere. [19]

The main evidence for this theory is the observation of several possible ancient shorelines, mainly found at the edges of the northern plains. Two of these river-like patterns can be traced uninterrupted for thousands of kilometres. However, this idea has faced strong opposition based on observations of significant long wavelength changes in elevation

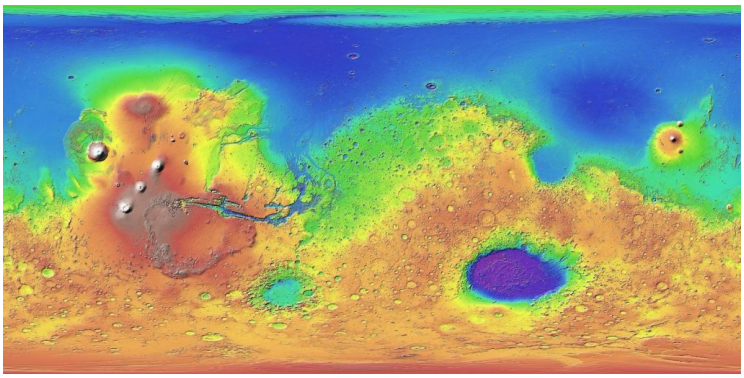

Figure 4. Mars MGS MOLA Global Color Shaded Relief [20]

along the supposed shorelines, which deviate by several kilometres and do not adhere to surfaces of equal gravitational potential, as would be expected for Earth's sea level. Despite these challenges, further research has continued the debate as to how these ancient shorelines could have been distorted to exhibit such remarkable variations in elevation today. [19]

3. Methods

3.1 Data sets analysed

In this thesis, the topographic data of four different celestial bodies and a fBm surface were analysed. The celestial bodies include Mars, Mercury, Earth and the Moon. Venus was also considered, but due to the only readily available data set having some artifatcs, it was omitted. The data sets that were not already with the resolution of 4 pixels per degree were converted to be so. This resolution was chosen as the amount of data was enough to get some meaningful results, but not too much so that running the code would have taken too long. The data set for the Earth comes from the National Oceanic and Atmospheric Administration (NOAA). All the other data sets are from the Planetary Data System (PDS).

3.1.1 Mars

The topographic data for Mars is from altimetry data observations acquired by the Mars Global Surveyor (MGS) Mars Orbiter Laser Altimeter (MOLA) instrument. The data products produced by MOLA include Aggregated Experiment Data Records, or AEDRs (raw data), Precision Experiment Data Records, or PEDRs (data from AEDRs with precision orbit corrections applied), and Experiment Gridded Data Records, or EGDRs (gridded products derived from PEDRs). Two different EGDRs are made, Initial Experiment Gridded Data Record (IEGDR) and Mission Experiment Gridded Data Record (MEGDR).

The data used for this thesis is a MEGDR product, a result of accumulated altimetry observations over the course of the whole MGS mission. This data is in the form of an image gridded at a resolution of 4 pixels per degree. The topographic map is computed as the planetary radius minus the areoid radius. The map is in simple cylindrical projection using the IAU2000 planetocentric coordinate system with east positive longitude and is in the form of a binary table with one row for each 0.25-degree latitude. The minimum and maximum topography observations for this data set are -8068 meters and 21134 meters. [21]

3.1.2 Mercury

The data for Mercury comes from Messenger Mercury Dual Imaging System Narrow Angle Camera and Mercury Dual Imaging System Wide Angle Camera. The dataset is composed of digital elevation models (DEM) of Mercury, created by multiple institutions involved in the MESSENGER mission. It includes numerous global DEM products and

certain region-specific DEM products. Each product features gridded map projections with varying resolutions. Global map products utilize simple cylindrical projections, with raster images ranging from 0 to 360 degrees positive east longitudes in the sample direction, and -90 to 90 degrees latitude extents from bottom to top in the line direction. The resolution of this data is 64 pixels per degree. The minimum and maximum topography observations for this data set are -10745 meters and 8827 meters respectively. [22]

3.1.3 Earth

In August 2008, the ETOPO1 Global Relief Model was developed by the National Geophysical Data Center (NGDC) of the National Oceanic and Atmospheric Administration (NOAA) as an improvement to the ETOPO2v2 Global Relief Model. ETOPO1 is available in "Ice Surface" and "Bedrock" versions and was generated from diverse global and regional digital datasets. These datasets were shifted to common horizontal and vertical datums, evaluated, and edited as needed. Data sources for ETOPO1 include NGDC, Antarctic Digital Database (ADD), European Ice Sheet Modeling Initiative (EISMINT), Scientific Committee on Antarctic Research (SCAR), Japan Oceanographic Data Center (JODC), Caspian Environment Programme (CEP), Mediterranean Science Commission (CIESM), National Aeronautics and Space Administration (NASA), National Snow and Ice Data Center (NSIDC), Scripps Institute of Oceanography (SIO), and Leibniz Institute for Baltic Sea Research (LIBSR). ETOPO1 is vertically referenced to sea level and horizontally referenced to the World Geodetic System of 1984 (WGS 84), with a resolution of 60 pixels per degree. The data set used in this thesis is the "Bedrock" version of cell registered binary data and is in simple cylindrical projection. The minimum and maximum topography observations for this data set are -10710 meters and 6572 meters. [23] [24]

3.1.4 The Moon

The data for the Moon came from the SELENE(KAGUYA) mission using the Laser Altimeter Instrument (LALT). The global topographic map of the Moon is obtained by interpolating elevation data in Lunar Global Topographic Data as a Time Series. The data set used has the resolution of 16 pixels per degree and using the simple cylindrical projection. The minimum and maximum topography observations for this data set are -8533 meters and 10715 meters. [25]

3.1.5 fBm surface

Fractional Brownian motions (fBm's) are a family of Gaussian random functions, that are defined by the following: B_t - ordinary Brownian motion, H - Hurst parameter $(0 < H < 1)$. Then fBm of the exponent H is a moving average of $dB(t)$, where past increments of $B(t)$ are weighted by the kernel $(t-s)^{H-\frac{1}{2}}$. The fundamental characteristic of fBm's is that the range of correlation between their increments can be considered unbounded. [26]

Normalized fractional Brownian motion has the property of having a Gaussian distribution for $t > 0$ and is the only Gaussian process with stationary increments that is self-similar [27]. In this thesis, fBm is used to check whether the algorithms that are used have been implemented correctly and whether or not they produce the results that are expected.

The specific method used is the Davies-Harte algorithm: Given that the autocovariance sequence (ACVS) of a stationary Gaussian process is known, then the steps to simulate a realisation of fBm of length N are the wollowing. First, for $k = 0, ..., 2N - 1$, one must compute

$$A_{k,N}(s) \equiv \sum_{j=0}^{N} s_j e^{-i\pi kj/N} + \sum_{j=N+1}^{2N-1} s_{2N-j} e^{-i\pi kj/N}. \tag{3.1}$$

Then, one is to check that $A_{k,N(s)} \geq 0$ for all k. Next, let the independent mean zero Gaussian random variables with unit variance be defined as $Z_0 ... Z_{2N-1}$. One must then compute the complex valued sequence given by

$$Y_k \equiv \begin{cases} \sqrt{2NA_{0,N}(s)} Z_0, & k = 0; \\ \sqrt{NA_{k,N}(s)}(Z_{2k-1} + iZ_{2k}), & 1 \leq k \leq N-1; \\ \sqrt{2NA_{N,N}(s)} Z_{2N-1}, & k = N; \\ \sqrt{NA_{k,N}(s)}(Z_{4N-1-2k} + iZ_{4N-2k}), & N+1 \leq k \leq 2N-1; \end{cases} \tag{3.2}$$

Finally, to simulate a realisation of the Gaussian process

$$X_t \equiv \frac{1}{2N} \sum_{k=0}^{2N-1} Y_k e^{i\pi kt/N}, \qquad t = 0, ..., N-1 \tag{3.3}$$

[28]

With this, one gets the 1D fBm functions.

The so called 4-vertex model can be used to create a 2D surface, as regarding the fractal dimension $D_1(H)$, the simulations show that for $0 \leq H \leq 1$, the 4-vertex model belongs to the same universality class as the isotropic Gaussian self-affine surfaces. For this model, height is given by the sum of two independent 1D fBm functions $\psi(x, y) = f(x) + g(y)$. This would work, however, the correlations would be strong and the surface would look unnatural. To fix this, more than two (8 in this case) fBm functions are added together instead. [29][18]

3.2 Methods used

3.2.1 Marching squares

Remote sensing image data extraction is a non-destructive approach that is frequently employed for observing and analyzing geographical features worldwide. This technique

allows for the results to be obtained without adversely affecting the study area, especially when carrying out geomorphological and environmental research in nearly unreachable locations. A wide variety of images are attainable, providing substantial spatial and temporal coverage at an affordable price. However, the raster methods used for linear feature segmentation in most commercial and open-source geographic information systems (GIS) will generate inherent vectorization along with related systematic errors. Consequently, their application in geomorphological studies, including estimating the lengths of geographical features such as perimeters, coastlines, and borders, is somewhat restricted.

Chain code methods were developed in order to code information about borders, which in turn allowed the analysis of geomteric figures in digital form. Chain code algorithms are used to create polygonal boundaries for objects with particular length and direction. A number code is given to each segment direction (Fig. 5). For a certain shape, different chain codes an be produced. There are two main cases that are based on 4- or 8-neighbour connectivity. To distinguish between them, one is named *outside pixel border* (OPB) (Fig. 5a) and the other *chain code* (CC) (Fig. 5b). OPB draws the frontier moving across consecutive boundary pixels, while CC draws the border by moving from the pixel center to another connected neighbour pixel center. The chain code perimeter formula is given by

$$P = n_e + \sqrt{2}n_o, \tag{3.4}$$

where n_e and n_o are the number of even and odd chain elements respectively. These methods however, are low-accuracy in some cases.

Another method, the crack code (also called mid crack) brings considerable improvement into the chain code methods by delineating the border segments that move along the pixel edge midpoints with 8-direction connectivities (Fig. 5c). Later on, another procedure, named the marching squares (MS) algorithm was developed (Fig. 6a), that was results wise the same as mid crack, but differed in execution methodology. The new procedure was to analyze the local properties of a 2 by 2 window in the center of four pixels called a bit quad (Fig. 6b). Every configuration of these bit quads is determined by the Euler number, with 16 combinations in total. [30]

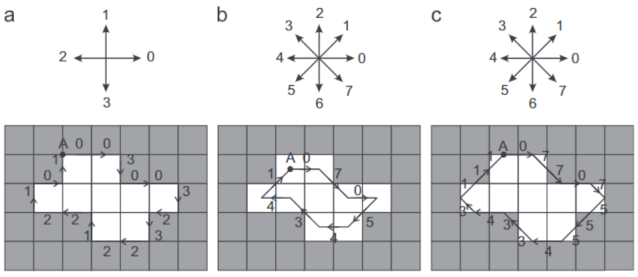

Figure 5. Chain code examples. Code numbers generated moving clockwise starting from corner A. (a) OPB, the sequence is 0030032322122101. (b) CC, the sequence is 07054341. (c) Mid-crack, the sequence is 0770755543343111. [30]

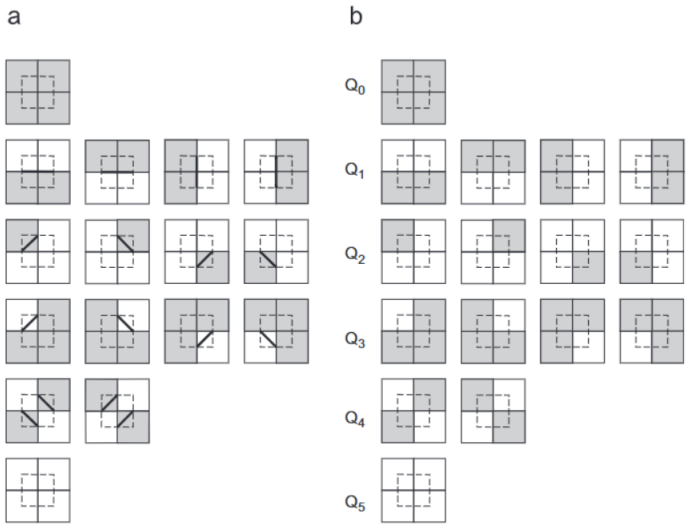

Figure 6. (a) Marching squares. (b) Bit quads configurations. [30]

3.2.2 Surveyor's formula

By surveying a plot of land, one gets data for the successive displacements that are required to cover the boundary of a simple plane polygon. To find the area of such a polygon, one could break it into triangles and use trigonometric methods. This is however very laborious. It is more efficient to introduce rectangular coordinates and change the displacement vectors from polar to rectangular so they could be added in order to get the coordinates for the vertices of the polygon. After that, a general formula can be used to express the area of the polygon as a function of its vertices. This formula is known as the surveyor's formula (also known as the shoelace formula and Gauss's area formula).

Let the vertices of a simple polygon, that are listed around the perimeter in a counter-clockwise order be (x_0, y_0), (x_1, y_1), ..., (x_{n-1}, y_{n-1}). The area of such a polygon can be calculated as:

$$A = \frac{1}{2}\left\{ \begin{bmatrix} x_0 & x_1 \\ y_0 & y_1 \end{bmatrix} + \begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \end{bmatrix} + ... + \begin{bmatrix} x_{n-2} & x_{n-1} \\ y_{n-2} & y_{n-1} \end{bmatrix} + \begin{bmatrix} x_{n-1} & x_0 \\ y_{n-1} & y_0 \end{bmatrix} \right\}, \tag{3.5}$$

where each edge of the polygon is represented by a 2 x 2 determinant in the formula. [31]

4. Results

4.1 Data collection

The contour lines for each of the 4 celestial bodies and a fractional Brownian surface (generated with the Davies Harte fractional Brownian surface (fBm) method [32]) were made by implementing the marching squares algorithm. Python was used for this due to it's ease of accessibility in the form of the Jupyter Notebook and the author's familiarity with it.

Contour lines were made starting from the lowest point of the data set up to the highest with a step size of 10 meters (every 100 meters for the Brownian surface). To the best of the author's and supervisor's knowledge, the study of the fractal dimension as a function of height is a new method that has not yet been used and is suitable for the purpose required. After the contour lines were gathered, a variety of different parameters were calculated for each contour including diameter, area and length. For area calculation, the surveyor's formula was used. Length calculation was further divided into multiple parts according to the scale of measurement, which extended from 1 up to 64 with increments of powers of two. Then, the total length of all the contours for each height was calculated. Also collected was the number of contours for each height. The code for this can be seen in Appendix 1.

Ideally, equation (2.7) should hold for all values of a for a self-affine curve. However, as this ideal case doesn't always hold, especially for the isolines of a real landscape, here we look at a diferential exponent, which is derived from the values of two neighbouring points (a and $2a$)

$$L(a,z) = \left(\frac{2a}{a}\right)^{1-D(z,a)} L(a,z), \tag{4.1}$$

where a denotes the measuring stick or resolution used and z is height. From this, the equation for $D(z,a)$ is

$$D(z,a) = \frac{\ln[L(a,z)/L(2a,z)]}{\ln 2} + 1 \tag{4.2}$$

With $D(z,a)$, two different fractal sets can be considered - $D_1(z,a)$ and $D_k(z,a)$, where D_1 represents $D(z,a)$ without taking new contours that appear when changing from a to $2a$ into account (denoted as D_h in subsection 2.4) and D_k represents the $D(z,a)$, where new contours are also taken into account. According to [13] and with the correction for D_1 from [18] taken into account, these fractal dimensions are connected to the Hurst parameter H

by the following equations

$$D_k = 2 - H \tag{4.3}$$

$$D_1 \approx \frac{3 - H}{2} + 0.064H(1 - H) \tag{4.4}$$

Equation (4.4) is an approximation coming from [18]. Similarly, the parameter k from Korčak's law can be calculated from

$$N(A > S, z) = N(S_0, z) \left(\frac{S_0}{S} \right)^{k(S,z)}, \tag{4.5}$$

where N denotes the amount of contours, A their area, and S the comparison area. From this equation, k can be expressed as

$$k(S, z) = \frac{\ln[N(A > S)/N(A > 2S)]}{\ln 2} \tag{4.6}$$

k can also be expressed as a function of D_k

$$k = \frac{D_k}{2} \tag{4.7}$$

For self-affine Gaussian surfaces equations (4.3, 4.4, 4.7) hold; however geological landscapes can depart significantly from Gaussianity [19]. Therefore, one of the working hypothesis was that departure from Gaussianty may help understanding the effect of oceans on the landscapes. Because of that, we constructed combinations of scaling exponents describing how far the landscape departs from Gaussianity and which should take a zero value for Gaussian self-affine surfaces, In particular the scaling exponents' anomalies ΔD and Δk were calculated, defined as as

$$\Delta D = (D_k - 1) - 2(D_1 - 1) \tag{4.8}$$

$$\Delta k = k - \frac{D_k}{2} \tag{4.9}$$

Here, Eq. (4.8) is based on the approximate relationship $D_1 \approx \frac{3}{2} - \frac{H}{2}$; this can be improved by using the more accurate approximation (4.4) [18] by introducing the "corrected" anomaly of the factal dimension ΔD_c:

$$\Delta D_c = D_1 - 1 - \frac{D_k - 1}{2} - 0.064(D_k - 1)(2 - D_k) \tag{4.10}$$

4.2 Data analysis

Now, the differences between the various planetary landscapes will be examined using the previously introduced characteristics. Firstly, the number of contours vs height, which can be seen in figure 7 and the lengths of those contours vs height in figure 8.

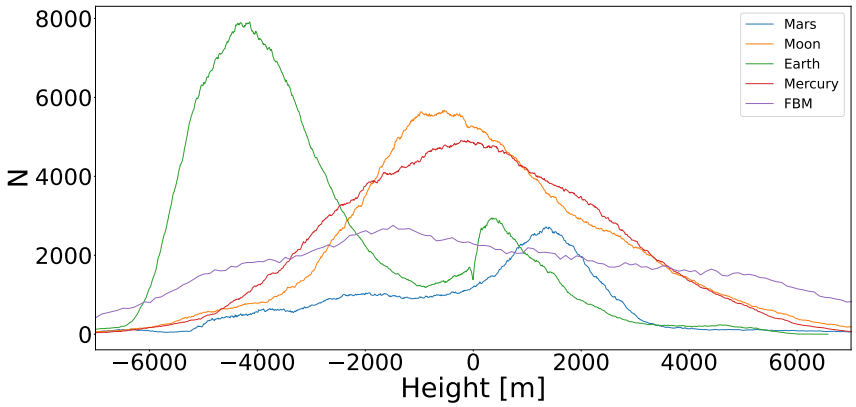

Figure 7. The total number of contours for a given height for Mars (blue), The Moon (orange), Earth (green), Mercury (red) and the FBM model (purple)

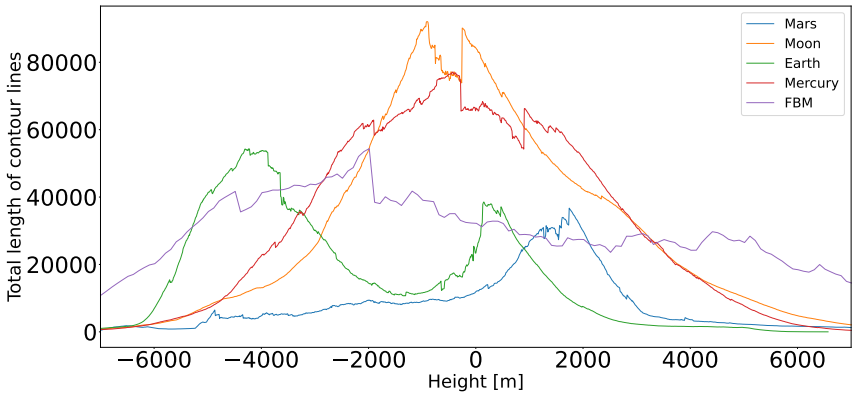

Figure 8. The total length of contours for a given height for Mars (blue), The Moon (orange), Earth (green), Mercury (red) and the FBM model (purple)

These two graphs behave similarly for the reason that more contours generally mean longer total length of contour lines. It is interesting to note that the graph for Earth has two peaks whereas all the others have one. For Earth, one peak is around and slightly above the sea level while the other peak is at the depths corresponding to the ocean floor, -4000m. Such a behaviour can be understood as follows. Ocean floor and altitudes near the sea level are mostly flat plains which can be characterised by small values of the local Hurst exponent and hence, with large values of the fractal dimension. This, in its own turn, results in longer isolines. Meanwhile, at the transition area, the continental slope, is characterised by steep gradients and hence, smaller fractal dimensions and smaller isoline lengths.

Another similar graph can be obtained by counting the amount of data points that are between two height values i.e how much an isoline shifts while changing height (or the difference in area of all of the contours when changing height). The graph for this can be seen in figure 9.

The biggest difference with the graph in figure 9 and the graphs in figures 7 and 8 is the emergence of a second peak for Mars at around -4000 meters. This can be ascribed to

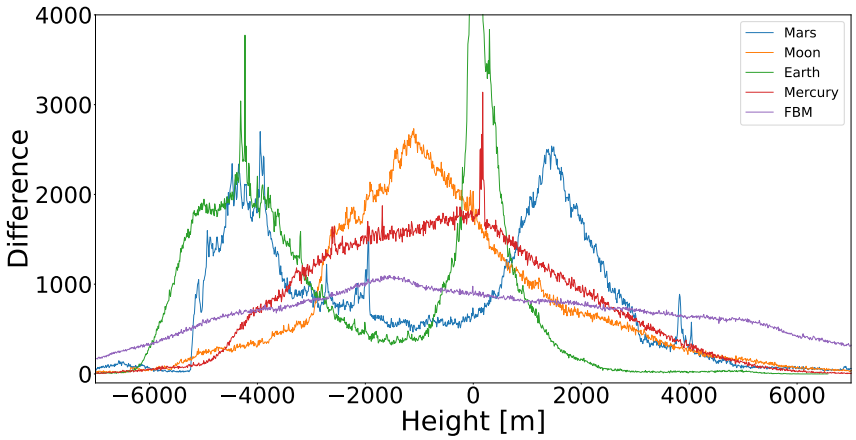

Figure 9. The difference of contour areas for different heights for Mars (blue), The Moon (orange), Earth (green), Mercury (red) and the FBM model (purple)

the Martian dichotomy: similarly to the oceanic floor on Earth, the northern hemisphere of Mars is more flat, hence small changes in the isoline altitude result in relatively big shifts in the isoline positions.

Before trying to study the results for the celestial bodies, one has to see if the methods used are working as intended by using them on a known subject - the fBm model. The 3D surface generated by fBm can be seen in figure 10. This surface was generated with a Hurst parameter value of 0.5. The height range is from -10188 to 16713.

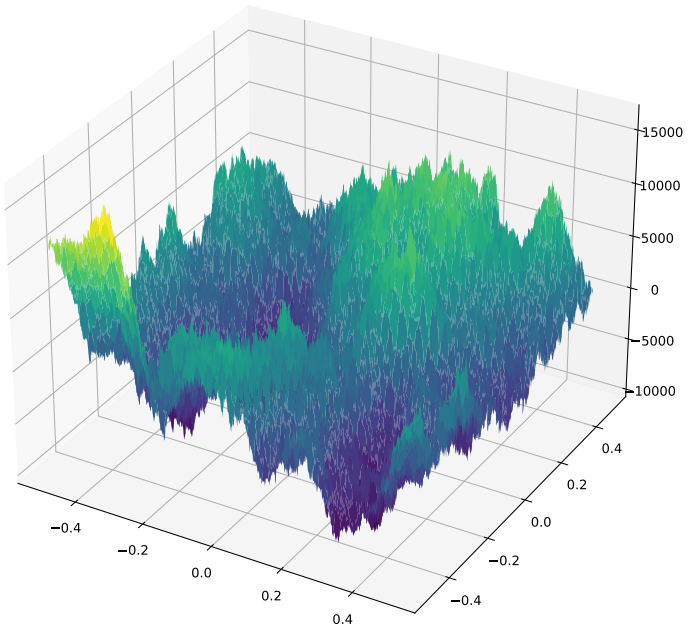

Figure 10. A Brownian surface generated by the Davies Harte FBM method [32]

The graphs for all the different parameters D_k, D_1, k, ΔD, ΔD_c and Δk can be seen in figures 11, 12, 13, 14, 15 and 16 respectively. According to equations (4.3), (4.4) and (4.6),

with the Hurst parameter $H = 0.5$, we expect to see values for D_k, D_1 and k to be 1.5, 1.28 and 0.75 respectively. From this it can be seen that firstly, the methods used underestimate the dimensions for the smallest scales (blur and yellow curves in Figs. 11–13). For larger scales, these values approach the expected results. This is the consequence of the so called finite size effect, where the exponent deviates from the correct results at the edges of the inertial range (i.e the range where the power law is expected to hold). The second observation is that the lines on the graphs are fluctuating due to statistical uncertainties and are almost horizontal (i.e. independent of the altitude), as expected. This gives us an opportunity to evaluate the amplitude of the random fluctuations for every exponent depending on the measurement scale by calculating their standard deviation. These values for each parameter can be seen in table 1. It can be concluded that in terms of statistical fluctuations, the best parameter with the smallest statistical fluctuations is D_1.

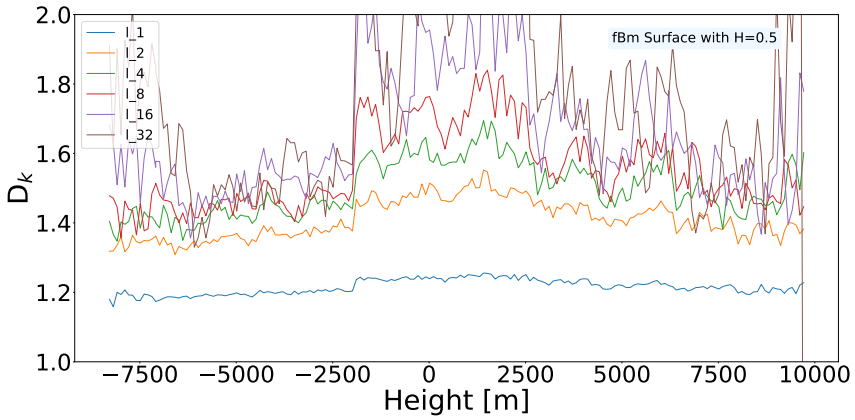

Figure 11. Parameter $D_k - 1$ for the fBm surface with $H = 0.5$

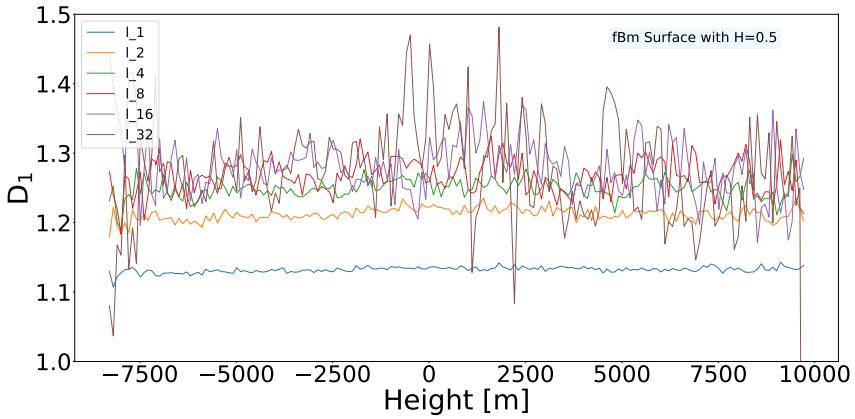

Figure 12. Parameter $D_1 - 1$ for the fBm surface with $H = 0.5$

The graphs for the fractal dimension D_k for the celestial bodies can be seen in figures 17, 18, 19 and 20. While the graphs for the Moon and Mercury seem to be more or less constant throughout the range of altitudes, with only some slight statistical fluctuations, the graphs for both Mars and Earth look different. For Mars the deviations from a horizontal line are significantly smaller than for the Earth, where there is clearly a big dip at around the height value of 0 meters. The behaviour for Earth can be explained by the presence of water and an atmosphere. Furthermore, all the different lines are quite close to each

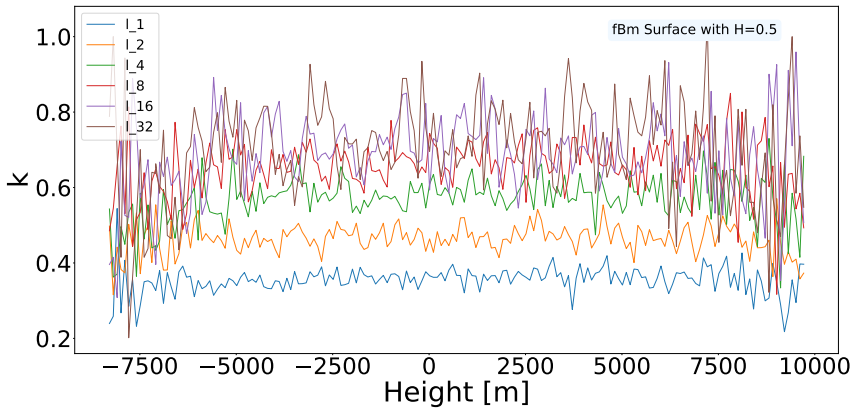

Figure 13. Parameter k for the fBm surface with $H = 0.5$

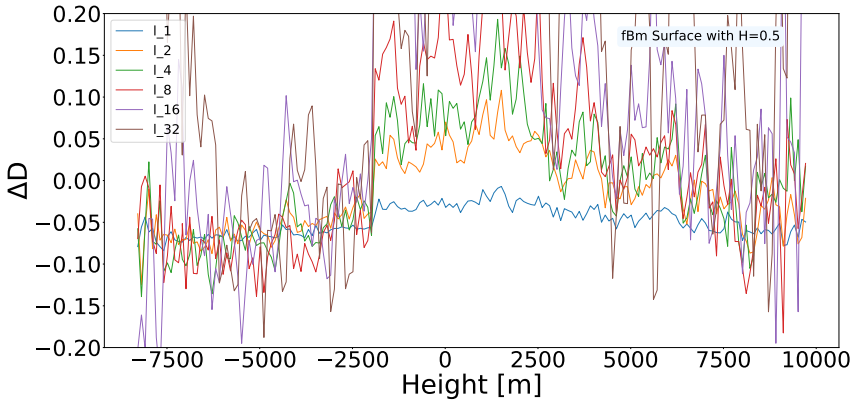

Figure 14. Parameter ΔD for the fBm surface with $H = 0.5$

Measuring stick / Parameter	1	2	4	8	16	32
D_k	0,08	0,23	0,26	0,34	0,39	0,47
D_1	0,04	0,09	0,10	0,13	0,15	0,14
k	0,14	0,21	0,27	0,28	0,38	0,35
ΔD	0,08	0,21	0,22	0,32	0,30	0,30
ΔD_c	0,05	0,12	0,13	0,18	0,17	0,19
Δk	0,14	0,21	0,29	0,30	0,39	0,34

Table 1. Population standard deviation calculated for each parameter at each measuring stick length for the fBm surface with H=0.5

other for Earth, but for other celestial bodies, they are at bigger intervals which means that there is a significant departure from self-affinity of these landscapes: for smaller scales, the surfaces are smoother; the origin of such behaviour is unknown and deserves further studies. Moreover, the fluctuations for all the graphs are bigger at the sides, due to the too short total length of the isolines meaning that statistical bases is insufficient. Because of that, the graphs have been cut off at very large and very small altitudes (<-6000 m and >6000 m).

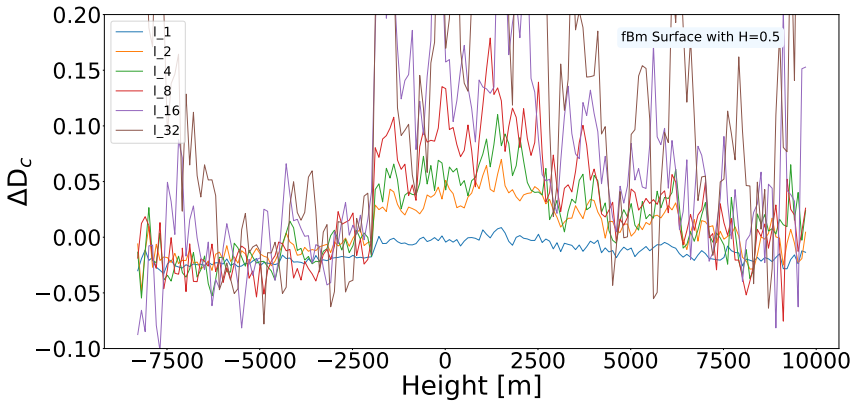

Figure 15. Parameter ΔD_c for the fBm surface with $H = 0.5$

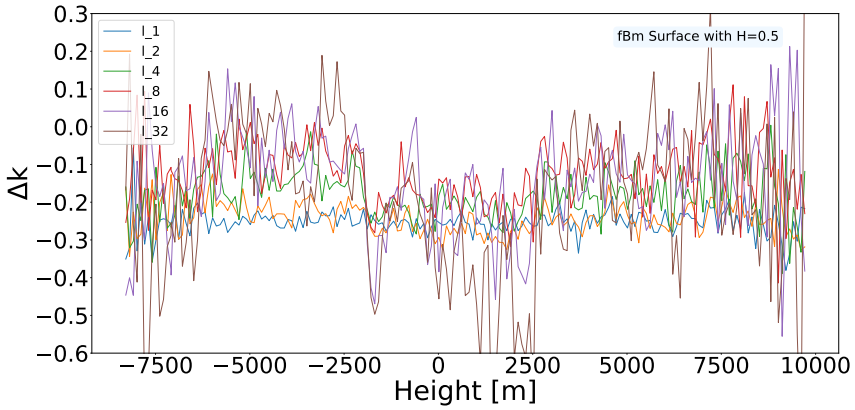

Figure 16. Parameter Δk for the fBm surface with $H = 0.5$

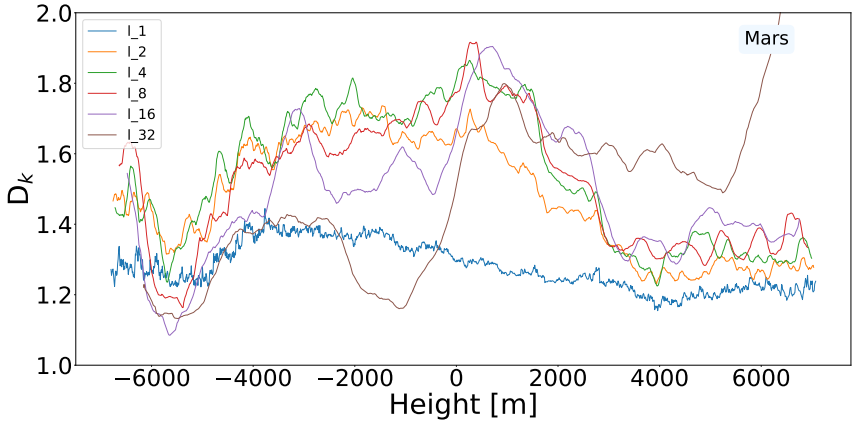

Figure 17. Parameter D_k for Mars for different measuring lengths

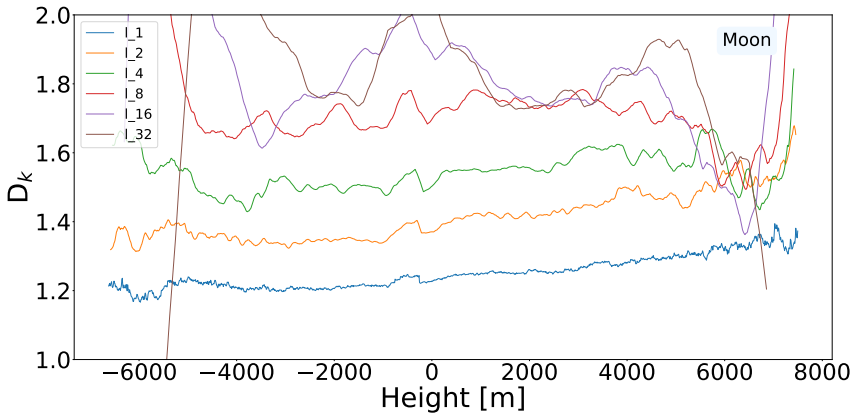

Figure 18. Parameter D_k for the Moon for different measuring lengths

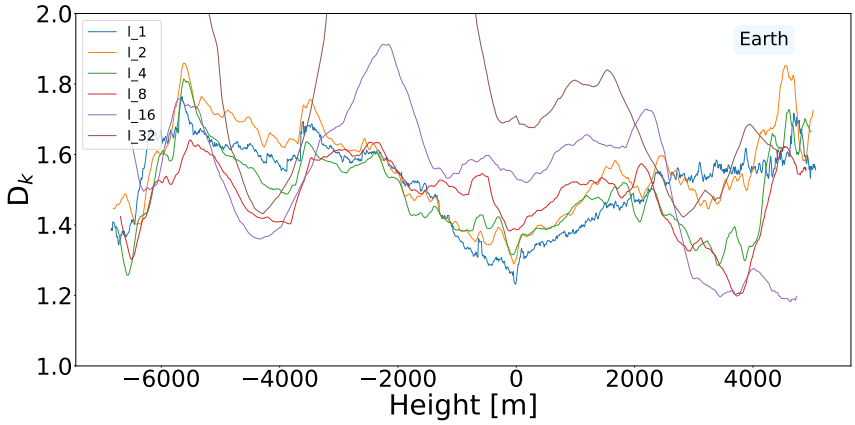

Figure 19. Parameter D_k for Earth for different measuring lengths

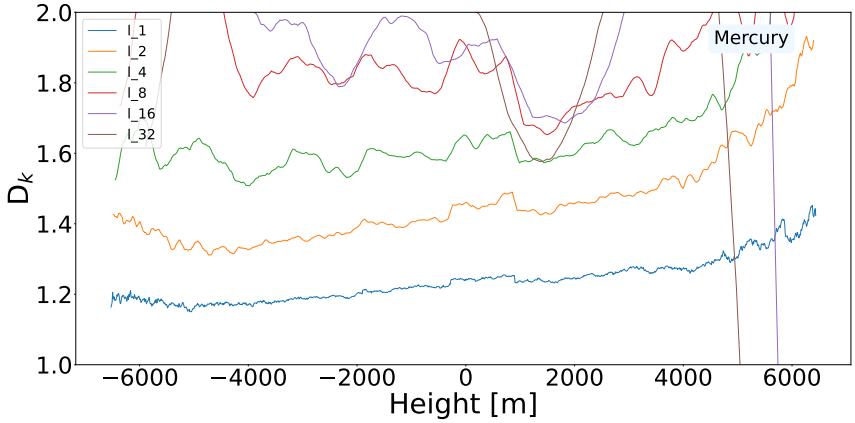

Figure 20. Parameter D_k for Mercury for different measuring lengths

The fractal dimension D_1, which represents individual connected loops can be seen in figures 21, 22, 23 and 24. The graphs look similar to the previous ones, except that now Mars is even smoother than before. The jumps in the graphs at the sea level for Earth are even more distinct here and are a clear indicator for the effect of the water level. If there were ever water on Mars, we would expect to see something similar — jumps of the curves or at least reduced values of D_1 at small elevation levels — on these graphs, but this not the case.

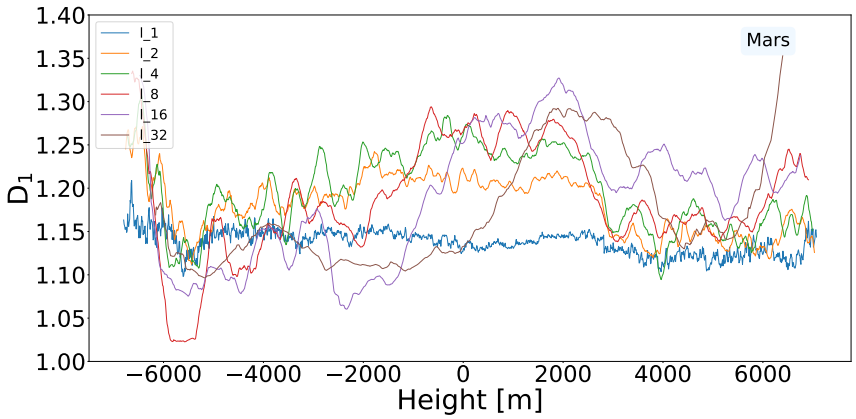

Figure 21. Parameter D_1 for Mars for different measuring lengths

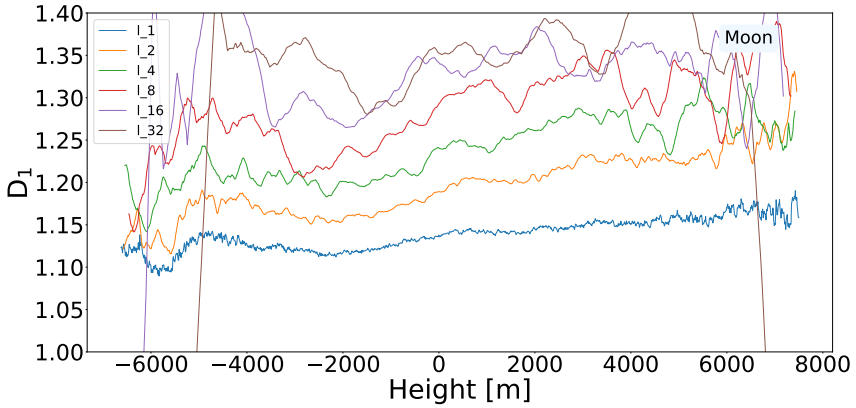

Figure 22. Parameter D_1 for the Moon for different measuring lengths

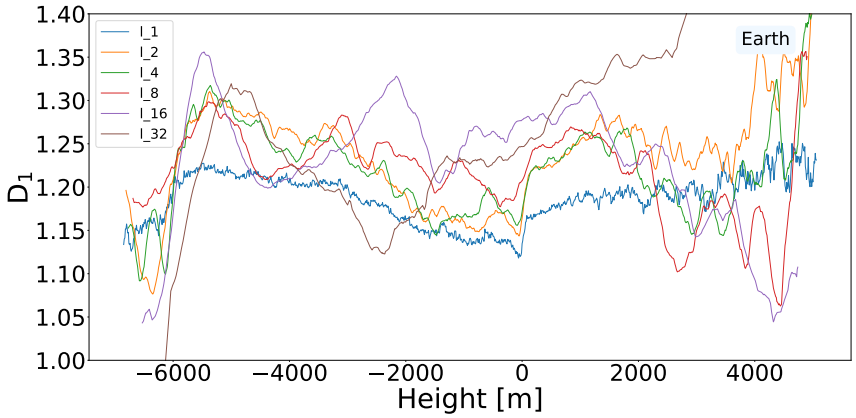

Figure 23. Parameter D_1 for Earth for different measuring lengths

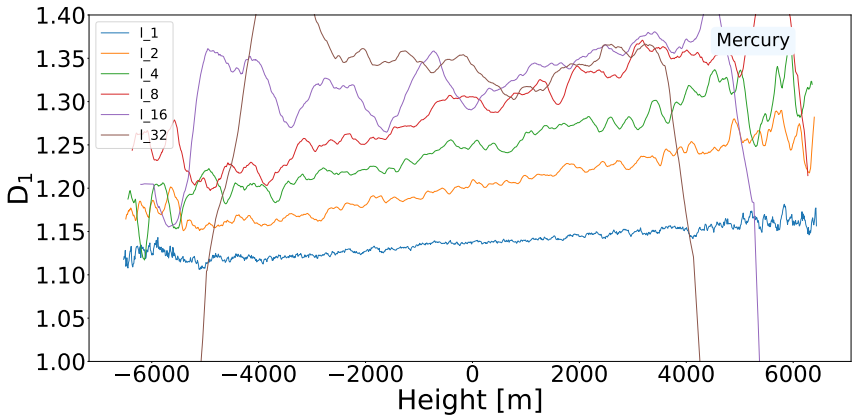

Figure 24. Parameter D_1 for Mercury for different measuring lengths

The graphs for the parameter k can be seen in figures 25, 26, 27 and 28. The same dip at height 0 meters for the Earth can not be observed here. However, interestingly the lines for the different length scales are all quite together for the Earth and not for the other celestial bodies. This might be due to geophysical processes on Earth and is another intriguing fact deserving further studies.

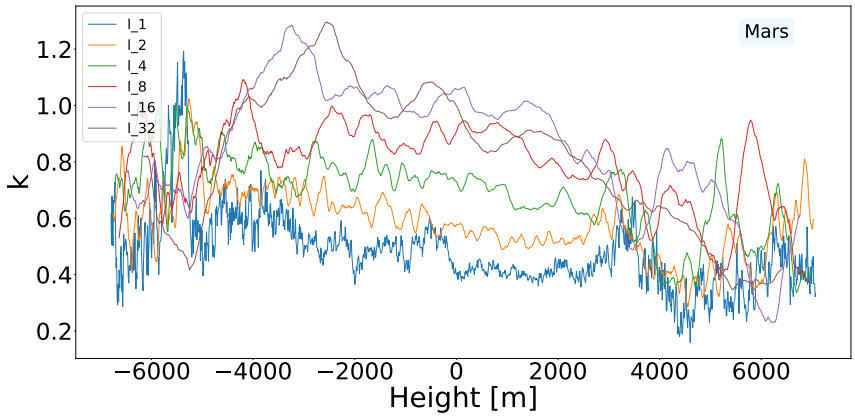

Figure 25. Parameter k for Mars for different measuring lengths

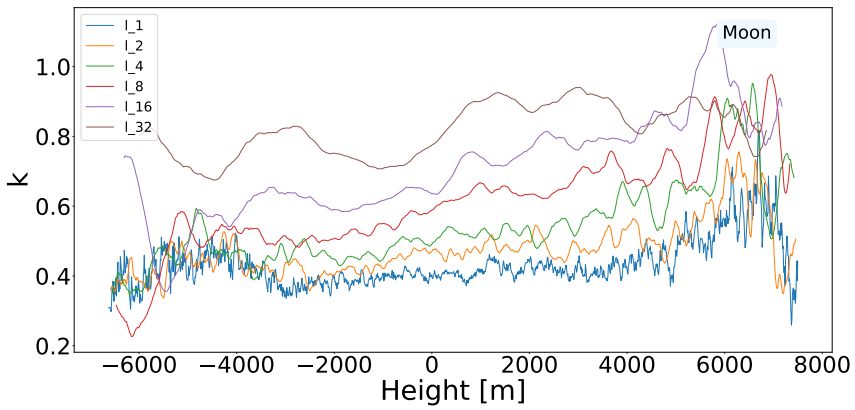

Figure 26. Parameter k for the Moon for different measuring lengths

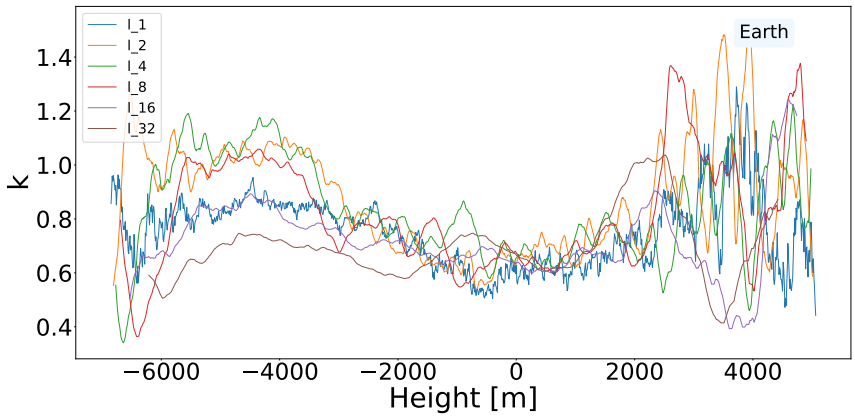

Figure 27. Parameter k for Earth for different measuring lengths

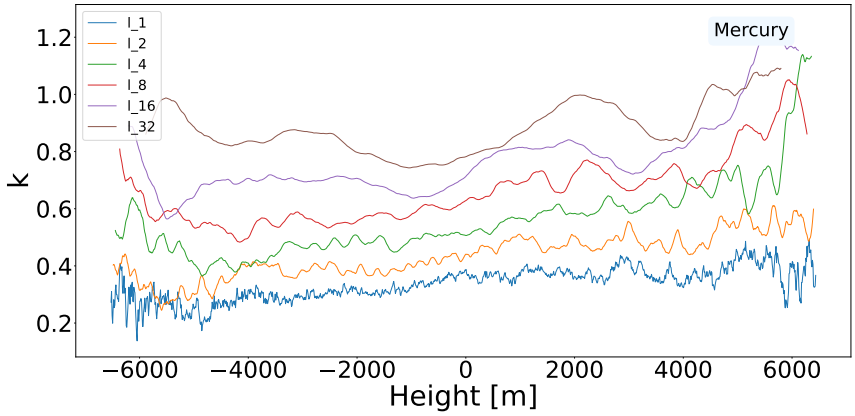

Figure 28. Parameter k for Mercury for different measuring lengths

For ΔD, i.e the measure for anomalies, the graphs can be seen in figures 29, 30, 31 and 32. We can see again the fingerprint of the water level in the graphs for Earth: all the curves have a minimum near the sea level, with values around zero (i.e. indicating absence of scaling anomaly). While both the graphs for D_k and D_1 showed a sharp jump for Earth at 0 meters, the same can't be said here. Such a behaviour can be explained by noticing that the expression for ΔD combines the values of D_k and D_1; the jumps of D_k and D_1 appear to compensate each other in this expression, and as a result, for ΔD the jump has been flattened.

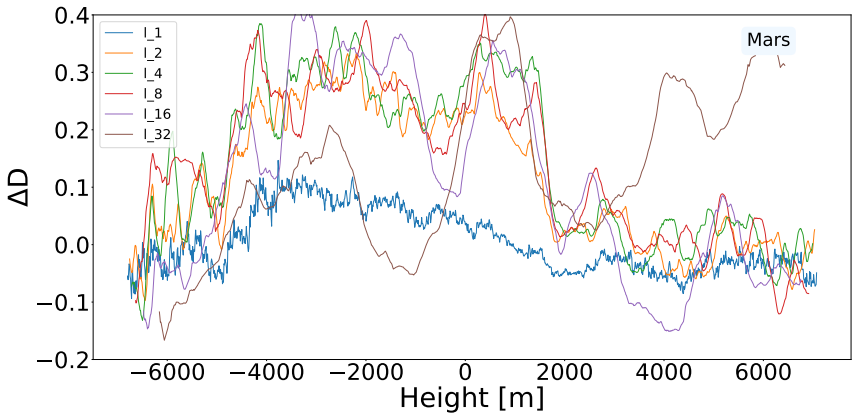

Figure 29. Parameter ΔD for Mars for different measuring lengths

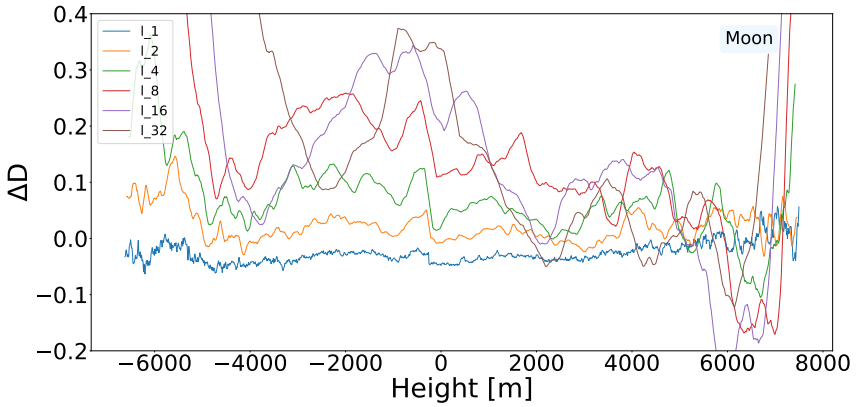

Figure 30. Parameter ΔD for the Moon for different measuring lengths

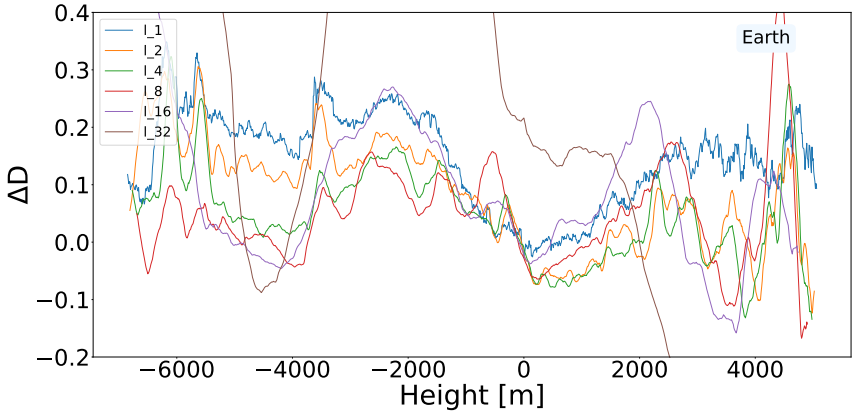

Figure 31. Parameter ΔD for Earth for different measuring lengths

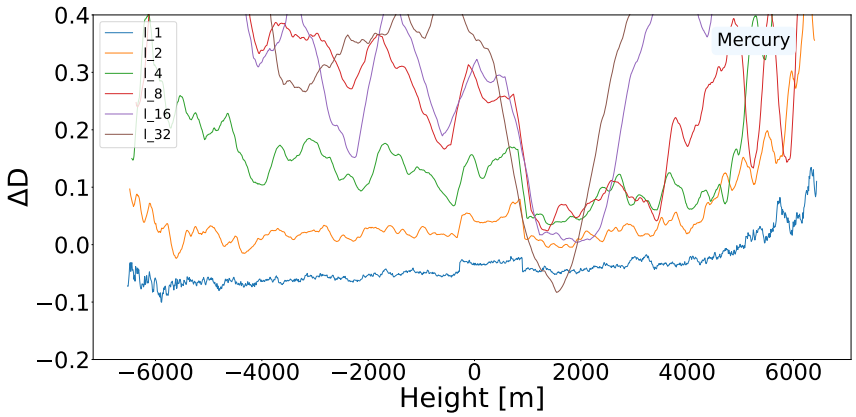

Figure 32. Parameter ΔD for Mercury for different measuring lengths

With the correction from [18] applied to ΔD (denoted as ΔD_c), the graphs change as can be seen in figures 33, 34, 35 and 36. The lines for the biggest scales (blue) are very close to 0 for both the Moon and Mercury, meaning they act as functions of Gaussian statistics, which means that the non-Gaussianity isn't relevant here.

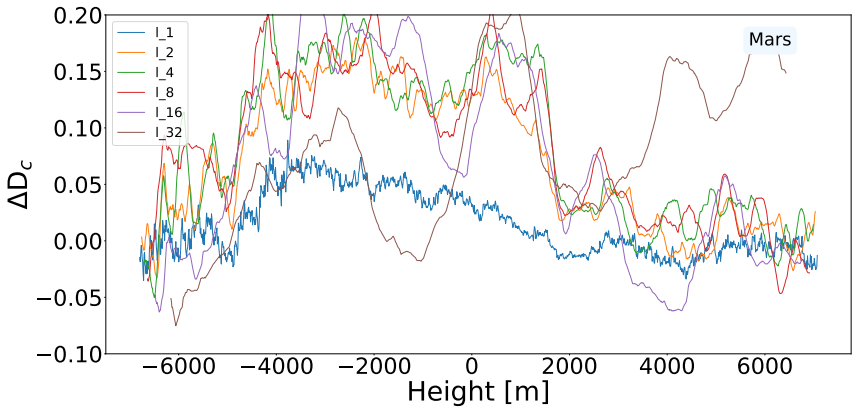

Figure 33. Parameter ΔD_c for Mars for different measuring lengths

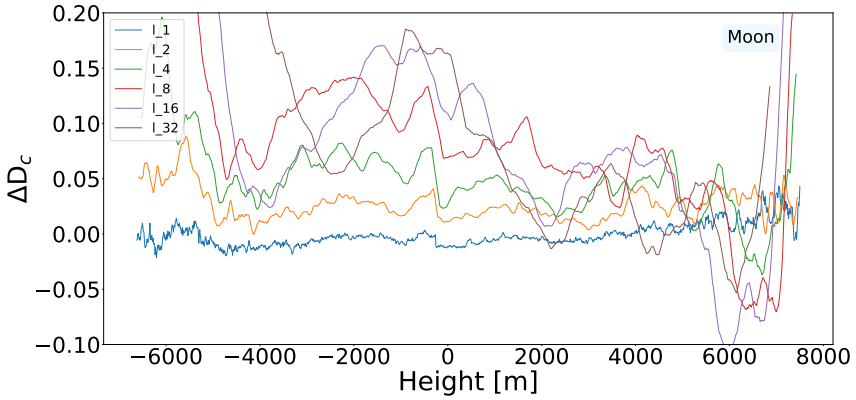

Figure 34. Parameter ΔD_c for the Moon for different measuring lengths

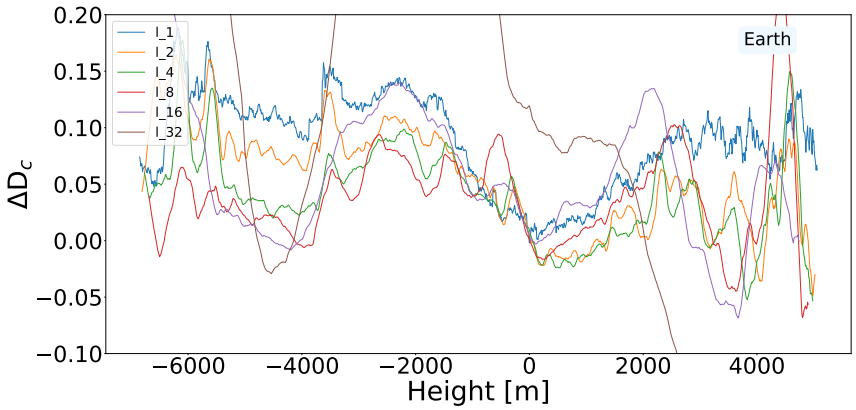

Figure 35. Parameter ΔD_c for Earth for different measuring lengths

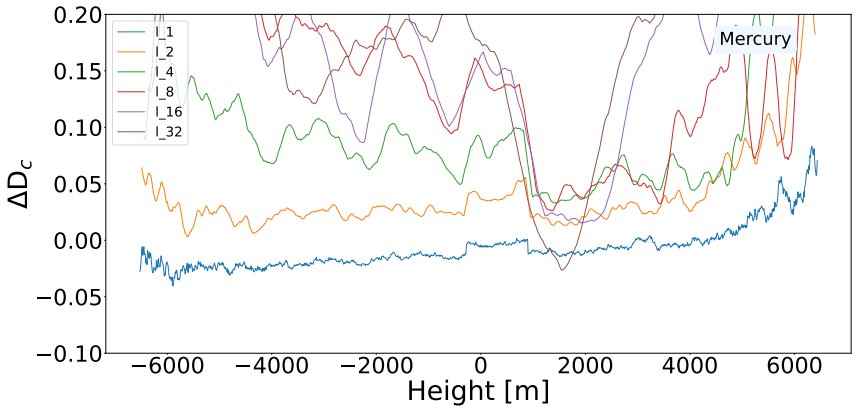

Figure 36. Parameter ΔD_c for Mercury for different measuring lengths

The parameter Δk for representing the anomalies for k can be seen in figures 37, 38, 39 and 40. Here, all the graphs except for Earth have most if not all the lines below 0, at around -0.2. For Earth, they are close to 0, which means that there are less small contours for Earth compared to the other celestial bodies; one can argue that this is due to the atmospheric and oceanic erosion.

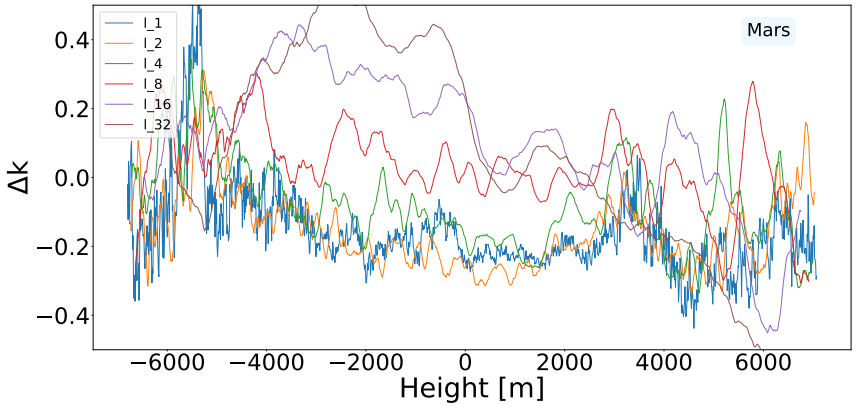

Figure 37. Parameter Δk for Mars for different measuring lengths

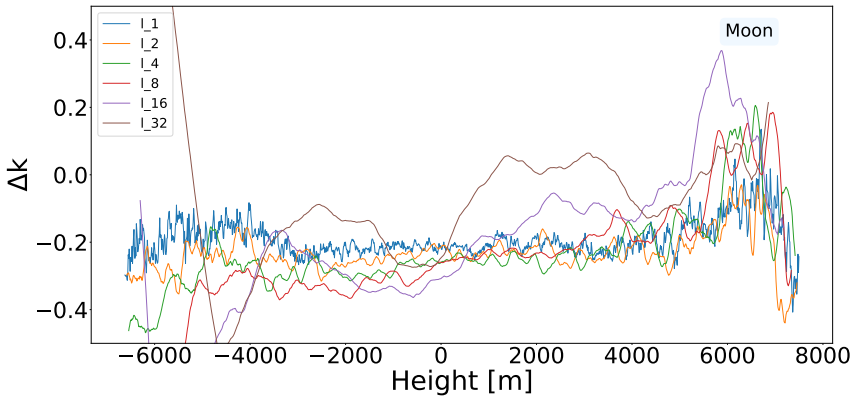

Figure 38. Parameter Δk for the Moon for different measuring lengths

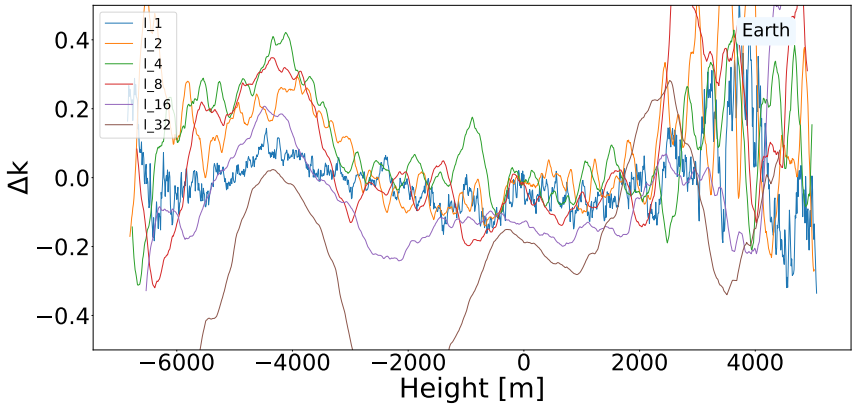

Figure 39. Parameter Δk for Earth for different measuring lengths

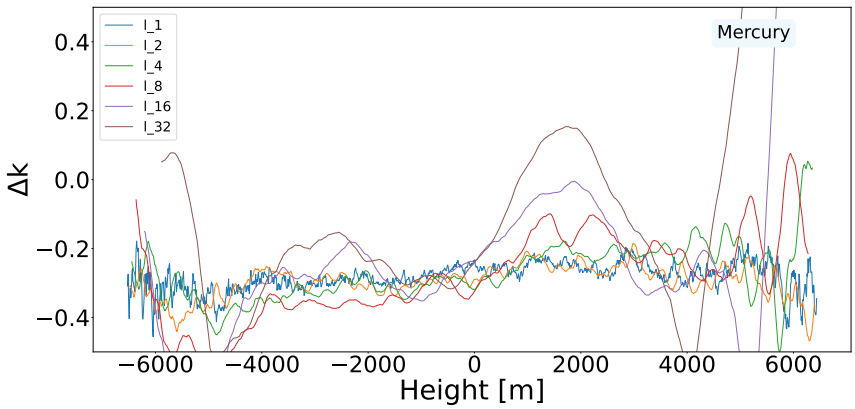

Figure 40. Parameter Δk for Mercury for different measuring lengths

In conclusion, the analysis of the scale-free measures of the statistical topography confirms the accuracy of the code by demonstrating consistency with analytical results for a test case of a fractional Brownian surface. Additionally, the presence of water on Earth is clearly visible in the D_k and D_1 graphs, confirming the hypothesis that statistical topography measures can be used to deduce the presence of water on a planet. This is further supported by the fact that the results for the Moon and Mercury show no statistically significant trends in the corresponding graphs, consistent with celestial bodies that never had free-flowing water on their surfaces. While the behavior of the graphs for Mars

differs slightly from that of Mercury and Moon, these differences can be attributed to the hemispheric dichotomy of Mars, rather than the effect of water erosion. While the absence of ancient water on Mars is not yet a definite conclusion, the results provide strong evidence that the hemispheric dichotomy of Mars is not caused by ancient oceans.

4.3 Possible improvements

There are a number of things that could be done to improve the results of this thesis. One of the biggest problems was the size of the datasets, which for this thesis were all converted to 720x1440 arrays (4 pixels per degree), which is a much lower resolution than what is actually available. This was done so that the code would run in a reasonable time. If the data sets had been larger, the results would have been more meaningful and concrete, with fewer anomalies. This could be achieved in a number of ways. Firstly, better code could be written for the marching squares algorithm, which took the longest to run. Another option is to use a different programming language altogether, one that is precompiled (like C++) rather than interpreted (like Python). Also, marching squares may not be the most efficient algorithm to use, and there may be other ways to collect the contour data.

Another possible modification is to modify the algorithm for determining the fractal dimensions. In this work, the length of the measuring stick was changed and the dimension was inferred based on how the measured length changed as a result. An alternative approach is to use maps of different resolutions — a series of downsampled maps. This would mean that one would only have to calculate the length of each contour and not have to deal with any measuring sticks. This method is expected to suffer less from the finite size effect.

5. Summary

The comprehensive analysis carried out in this thesis closely examined the topographic data of four different celestial bodies, namely Mars, the Moon, Earth and Mercury, as well as a simulated fractional Brownian surface (fBm). This data was carefully analysed and a series of graphs were produced showing various parameters derived from statistical topography, allowing a detailed comparative assessment between the subjects.

In the case of the fractional Brownian surface, the results demonstrated notable agreement with the analytical predictions, confirming the effectiveness of the methods used and the accuracy of the data. Earth's topography exhibited sudden and remarkable fluctuations in the fractal dimensions, D_k and D_1, at sea level. These observations supported the initial hypothesis that the discipline of statistical topography could serve as a viable tool for deducing the existence of water on a planet. This theory was further strengthened by the findings for the Moon and Mercury. Both bodies exhibited strikingly similar behaviour, with negligible deviations. The fractal dimensions remained virtually constant over the entire range of altitudes, underlining the consistency and predictability of their topographic nature. Mars, on the other hand, showed some changes at higher altitudes, particularly above 3000 metres. This observation is most likely due to its pronounced hemispheric dichotomy. However, there was no clear evidence of a water level, which leaves room for further investigation. While the absence of ancient water on Mars remains an open question, the results of this study provide strong evidence that the hemispheric dichotomy of Mars is not a consequence of ancient oceans.

This thesis lays the groundwork for future work in this area. The methods and results presented here could be further refined and extended by incorporating more advanced algorithms. Such improvements would not only increase efficiency, but also minimise potential finite-size effects, leading to even more robust and reliable results.

6. Acknowledgments

The author is grateful to his supervisor Jaan Kalda for guiding and helping him during the process. The author also aknowledges the help of ChatGPT for writing code used in the thesis. ChatGPT and DeepL Write were also used for spelling and language corrections.

Bibliography

[1] Kevin P Hand et al. “Astrobiology and the potential for life on Europa”. In: *Europa* (2009), pp. 589–629.

[2] John R Spencer and Francis Nimmo. “Enceladus: An active ice world in the Saturn system”. In: *Annual Review of Earth and Planetary Sciences* 41 (2013), pp. 693–717.

[3] M Lee Allison and Stephen M Clifford. “Ice-covered water volcanism on Ganymede”. In: *Journal of Geophysical Research: Solid Earth* 92.B8 (1987), pp. 7865–7876.

[4] James B Pollack et al. “The case for a wet, warm climate on early Mars”. In: *Icarus* 71.2 (1987), pp. 203–224.

[5] Alberto G Fairén. “A cold and wet Mars”. In: *Icarus* 208.1 (2010), pp. 165–175.

[6] John E Brandenburg. “The paleo-ocean of mars”. In: *Mars: Evolution of its Climate and Atmosphere*. Vol. 599. 1986, p. 6.

[7] Abbas Ali Saberi. “Evidence for an ancient sea level on Mars”. In: *The Astrophysical Journal Letters* 896.2 (2020), p. L25.

[8] Benoit B Mandelbrot and Benoit B Mandelbrot. *The fractal geometry of nature*. Vol. 1. WH freeman New York, 1982.

[9] Kenneth Falconer. *Fractal geometry: mathematical foundations and applications*. John Wiley & Sons, 2004.

[10] Michael B Isichenko. “Percolation, statistical topography, and transport in random media”. In: *Reviews of modern physics* 64.4 (1992), pp. 961–1043.

[11] Jens Feder. *Fractals*. Springer Science & Business Media, 2013.

[12] Michael B Isichenko and Jaan Kalda. “Statistical topography. I. Fractal dimension of coastlines and number-area rule for islands”. In: *Journal of Nonlinear Science* 1 (1991), pp. 255–277.

[13] Srinivasa B Ramisetti et al. “The autocorrelation function for island areas on self-affine surfaces”. In: *Journal of Physics: Condensed Matter* 23.21 (2011), p. 215004.

[14] Jaan Kalda. “Statistical topography of rough surfaces:“Oceanic coastlines” as generalizations of percolation clusters”. In: *Europhysics Letters* 84.4 (2008), p. 46003.

[15] Zeev Olami and Reuven Zeitak. “Scaling of island distributions, percolation, and criticality in contour cuts through wrinkled surfaces”. In: *Physical review letters* 76.2 (1996), pp. 247–250.

[16] MA Rajabpour and SM Vaez Allaei. “Scaling relations for contour lines of rough surfaces”. In: *Physical Review E* 80.1 (2009), p. 011115.

[17] Jané Kondev, Christopher L Henley, and David G Salinas. “Nonlinear measures for characterizing rough surface morphologies”. In: *Physical Review E* 61.1 (2000), pp. 104–125.

[18] Indrek Mandre and Jaan Kalda. “Monte-Carlo study of scaling exponents of rough surfaces and correlated percolation”. In: *The European Physical Journal B* 83 (2011), pp. 107–113.

[19] Jaan Kalda. “Gradient-limited surfaces: Formation of geological landscapes”. In: *Physical review letters* 90.11 (2003), p. 118501.

[20] Astrogeology Science Center. *Mars MGS MOLA Global Color Shaded Relief 463m v1*. USGS Astrogeology Science Center, Goddard Space Flight Center, NASA. Modified on 3 February 2020. Added to Astropedia on 10 February 2014. 2020.

[21] D. E. Smith et al. *Mars Global Surveyor Laser Altimeter Mission Experiment Gridded Data Record*. MGS-M-MOLA-5-MEGDR-L3-V1.0, NASA Planetary Data System. 2003. DOI: 10.17189/1519460.

[22] K. Becker, M. Robinson, and F. Pruesker. *MESSENGER MDIS DEM V1.0*. NASA Planetary Data System. 2015. DOI: 10.17189/1520282.

[23] NOAA National Geophysical Data Center. *ETOPO1 1 Arc-Minute Global Relief Model*. Accessed: 2023-04-14. 2009.

[24] C. Amante and B. W. Eakins. *ETOPO1 1 Arc-Minute Global Relief Model: Procedures, Data Sources and Analysis*. NOAA Technical Memorandum NESDIS NGDC-24. Accessed: 2023-04-14. National Geophysical Data Center, NOAA, 2009. DOI: 10.7289/V5C8276M.

[25] Japan Aerospace Exploration Agency (JAXA), National Institute of Advanced Industrial Science, and Technology (AIST). *KAGUYA (SELENE) Lunar Radar Altimeter (LALT) Derived Data*. 2009.

[26] Benoit B Mandelbrot and John W Van Ness. “Fractional Brownian motions, fractional noises and applications”. In: *SIAM review* 10.4 (1968), pp. 422–437.

[27] Ton Dieker. “Simulation of fractional Brownian motion”. PhD thesis. Masters Thesis, Department of Mathematical Sciences, University of Twente . . ., 2004.

[28] Peter F Craigmile. “Simulating a class of stationary Gaussian processes using the Davies–Harte algorithm, with application to long memory processes”. In: *Journal of Time Series Analysis* 24.5 (2003), pp. 505–511.

[29] Jaan Kalda. “Description of random Gaussian surfaces by a four-vertex model”. In: *Physical Review E* 64.2 (2001), p. 020101.

[30] Marina P Cipolletti et al. “Superresolution border segmentation and measurement in remote sensing images”. In: *Computers & Geosciences* 40 (2012), pp. 87–96.

[31] Bart Braden. “The surveyor’s area formula”. In: *The College Mathematics Journal* 17.4 (1986), pp. 326–337.

[32] Robert B Davies and DS Harte. “Tests for Hurst effect”. In: *Biometrika* 74.1 (1987), pp. 95–101.

Appendix 1 - Python code for gathering data

```
import os
import re
import json
import math
import numpy as np
from fbm import FBM
from scipy.ndimage import zoom
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
from multiprocessing.pool import ThreadPool

# Functions to read in the files

def read_img_file(metadata_file, img_file, v=0):
    with open(metadata_file, 'r') as f:
        metadata = f.read()
    lines = int(re.search(r'LINES\s+=\s+(\d+)',
    metadata, re.MULTILINE).group(1))
    line_samples = int(re.search(r'LINE_SAMPLES\s+=\s+(\d+)',
    metadata, re.MULTILINE).group(1))
    shape = (min(lines, line_samples), max(lines, line_samples))
    if v == 1:
        shape = (1025, 1024)
    data_type = re.search(r'SAMPLE_TYPE\s+=\s+(\w+)',
    metadata).group(1)
    if data_type == 'MSB_INTEGER':
        dtype = '>i2'
    elif data_type == 'MSB_UNSIGNED_INTEGER':
        dtype = '>u2'
    elif data_type == 'LSB_INTEGER':
        dtype = '<i2'
    elif data_type == 'LSB_UNSIGNED_INTEGER':
        dtype = '<u2'
    elif data_type == 'FLOAT':
        dtype = '<f4'
    elif data_type == 'PC_REAL':
        dtype = '<f4'
    else:
        raise ValueError(f'Unsupported data type: {data_type}')
    with open(img_file, 'rb') as f:
        data = np.fromfile(f, dtype=dtype)
    data = data.reshape(shape)
```

```
    return data


def resize_elevation_data(data, target_shape=(720, 1440)):
    resize_factor = (target_shape[0] / data.shape[0],
    target_shape[1] / data.shape[1])
    resized_data = zoom(data, resize_factor, order=0)

    return resized_data


def read_hdr(hdr_file):
    with open(hdr_file, 'r') as f:
        metadata = f.read()
    nrows = int(re.search(r'NROWS\s+(\d+)', metadata).group(1))
    ncols = int(re.search(r'NCOLS\s+(\d+)', metadata).group(1))

    return nrows, ncols


def read_etopo1_bin(hdr_file, bin_file):
    nrows, ncols = read_hdr(hdr_file)
    shape = (nrows, ncols)
    with open(bin_file, 'rb') as f:
        data = np.fromfile(f, dtype=np.int16)
    data = data.reshape(shape)

    return data

# Functions for gathering data

class Square():
    A = [0, 0]
    B = [0, 0]
    C = [0, 0]
    D = [0, 0]
    A_data = 0.0
    B_data = 0.0
    C_data = 0.0
    D_data = 0.0

    def GetCaseId(self, threshold):
        caseId = 0
        if (self.A_data >= threshold):
            caseId |= 1
        if (self.B_data >= threshold):
            caseId |= 2
```

```
        if (self.C_data >= threshold):
            caseId |= 4
        if (self.D_data >= threshold):
            caseId |= 8

        if caseId in (5, 10):
            if caseId == 10:
                if (int(self.A_data)+int(self.B_data)+int(self.C_data)
                +int(self.D_data))/4.0 >= threshold:
                    pass
                else:
                    caseId = 5
            elif caseId == 5:
                if (int(self.A_data)+int(self.B_data)+int(self.C_data)
                +int(self.D_data))/4.0 >= threshold:
                    pass
                else:
                    caseId = 10
        return caseId


    def GetLines(self, Threshold):
        lines = []
        caseId = self.GetCaseId(Threshold)

        if caseId in (0, 15):
            return []

        if caseId in (1, 14, 10):
            pX = (self.A[0] + self.B[0]) / 2
            pY = self.B[1]
            qX = self.D[0]
            qY = (self.A[1] + self.D[1]) / 2
            line = (pX, pY, qX, qY)
            lines.append(line)

        if caseId in (2, 13, 5):
            pX = (self.A[0] + self.B[0]) / 2
            pY = self.A[1]
            qX = self.C[0]
            qY = (self.A[1] + self.D[1]) / 2
            line = (pX, pY, qX, qY)
            lines.append(line)

        if caseId in (3, 12):
            pX = self.A[0]
            pY = (self.A[1] + self.D[1]) / 2
            qX = self.C[0]
```

```
            qY = (self.B[1] + self.C[1]) / 2
            line = (pX, pY, qX, qY)
            lines.append(line)

        if caseId in (4, 11, 10):
            pX = (self.C[0] + self.D[0]) / 2
            pY = self.D[1]
            qX = self.B[0]
            qY = (self.B[1] + self.C[1]) / 2
            line = (pX, pY, qX, qY)
            lines.append(line)

        elif caseId in (6, 9):
            pX = (self.A[0] + self.B[0]) / 2
            pY = self.A[1]
            qX = (self.C[0] + self.D[0]) / 2
            qY = self.C[1]
            line = (pX, pY, qX, qY)
            lines.append(line)

        elif caseId in (7, 8, 5):
            pX = (self.C[0] + self.D[0]) / 2
            pY = self.C[1]
            qX = self.A[0]
            qY = (self.A[1] + self.D[1]) / 2
            line = (pX, pY, qX, qY)
            lines.append(line)

        return lines


def process_square(args):
    j, i, Data, x, y, threshold = args

    a = Data[j + 1, i]
    b = Data[j + 1, i + 1]
    c = Data[j, i + 1]
    d = Data[j, i]
    A = [x[i], y[j + 1]]
    B = [x[i + 1], y[j + 1]]
    C = [x[i + 1], y[j]]
    D = [x[i], y[j]]

    square = Square()
    square.A_data = a
    square.B_data = b
    square.C_data = c
    square.D_data = d
```

```
    square.A = A
    square.B = B
    square.C = C
    square.D = D

    return square.GetLines(threshold)


def marching_square(Data, threshold):
    x = [i for i in range(len(Data[0]))]
    y = [i for i in range(len(Data))]

    Height = len(Data)
    Width = len(Data[1])

    squares = np.full((Height - 1, Width - 1), Square())
    sqHeight = squares.shape[0]
    sqWidth = squares.shape[1]

    args = [(j, i, Data, x, y, threshold)
    for j in range(sqHeight) for i in range(sqWidth)]
    with ThreadPool() as pool:
        results = pool.map(process_square, args)

    linesList = [line for lines in results for line in lines]

    return [linesList]


def group_contour_lines(lines):
    connections = {}
    for line in lines:
        p1, p2 = line[0:2], line[2:4]
        if p1 in connections:
            connections[p1].append(p2)
        else:
            connections[p1] = [p2]

        if p2 in connections:
            connections[p2].append(p1)
        else:
            connections[p2] = [p1]

    contours = []
    while connections:

        start_point = list(connections.keys())[0]
        contour = [start_point]
```

```
        current_point = start_point

        while True:
            if current_point not in connections:
                break

            next_points = connections[current_point]

            if len(next_points) != 2:
                break

            if len(contour) > 1:
                next_point = next_points[0]
                if next_points[0] != contour[-2] else next_points[1]
            else:
                next_point = next_points[0]

            if next_point == start_point:
                contours.append(contour)
                break

            contour.append(next_point)
            current_point = next_point

        for point in contour:
            connections.pop(point, None)

    return contours


def polygon_area(vertices):
    vertices_copy = vertices.copy()
    n = len(vertices_copy)
    area = 0.0
    for i in range(n):
        j = (i + 1) % n
        area += vertices_copy[i][0] * vertices_copy[j][1]
        - vertices_copy[j][0] * vertices_copy[i][1]

    return abs(area) / 2.0


def polygon_diameter(points):
    points = np.array(points)
    distances = cdist(points, points)
    i, j = np.unravel_index(distances.argmax(), distances.shape)
    return round(distances[i, j], 2)
```

```
def Kn (punktid):

    start_point = (punktid[0][0], punktid[0][1]) # (x, y)
    K_points = [2**i for i in range(7)]
    new_start = [start_point for k in range(len(K_points))]
    remainder = [0 for i in range(len(K_points))]
    kogu_pikkused = [0 for i in range(len(K_points))]

    for j, i in enumerate(punktid):
        for l, k in enumerate(K_points):
            dx = i[0] - new_start[l][0]
            dy = i[1] - new_start[l][1]
            K = math.sqrt(dx**2 + dy**2)
            if K>=k:
                new_start[l] = (i[0], i[1])
                kogu_pikkused[l] += K

    for n, m in enumerate(new_start):
        dx = m[0] - start_point[0]
        dy = m[1] - start_point[1]
        K = math.sqrt(dx**2 + dy**2)
        remainder[n] = K
        kogu_pikkused[n] += remainder[n]

    kogu_pikkused_dict = {'l_'+str(l):n for l,n in
    zip(K_points, kogu_pikkused)}

    return kogu_pikkused_dict


def process_data(grouped):
    kontuur = []
    kontuurid = []
    kogu_andmed = {}
    K_points = [2**i for i in range(7)]

    for j, i in enumerate(grouped):
        l_k = Kn(i)
        s = polygon_area(i)
        d = polygon_diameter(i)
        dict_i = {"l" : l_k, "s" : s, "d" : d}
        kontuur.append(dict_i)

    l_sums = {}
    for d in kontuur:
        for k, v in d['l'].items():
            if k not in l_sums:
```

```
                l_sums[k] = v
            else:
                l_sums[k] += v

    new_sums = {k.replace('l', 'L'): v for k, v in l_sums.items()}
    kogu_andmed = {"N" : len(grouped), "L_k" : new_sums}
    kontuurid.append(kogu_andmed)
    kontuurid.append(kontuur)

    return kontuurid

# Main function

def process_heights(maatriks, height_range,
progress_file='progress.json', output_file='output.json'):

    if os.path.exists(progress_file):
        with open(progress_file, 'r') as f:
            progress_data = json.load(f)
        current_height = progress_data.get('current_height',
        height_range[0])
        all_data = progress_data.get('all_data', {})
    else:
        current_height = height_range[0]
        all_data = {}

    heights = [i for i in range(current_height, *height_range[1:])]

    for i in heights:
        lines = marching_square(maatriks, i)[0]
        grouped = group_contour_lines(lines)
        kontuur= process_data(grouped)

        all_data[f"height_{i}"] = kontuur

        progress_data = {
            'current_height': i + 1,
            'all_data': all_data
        }
        with open(progress_file, 'w') as f:
            json.dump(progress_data, f)

    with open(output_file, 'w') as f:
        json.dump(all_data, f)

    if os.path.exists(progress_file):
        os.remove(progress_file)
```

```
    return all_data


def turn_into_file(data, file_name):
    json_str = json.dumps(data)
    with open(f'{file_name}_DATA.json', 'w') as f:
        f.write(json_str)

# Reading in the data

# Mars
data_Mars = read_img_file('Mars_metadata.txt', 'megt90n000cb.img')

# Moon
Moon_image_data = read_img_file("LALT_GGT_MAP_metadata.txt",
"LALT_GGT_MAP.IMG")
data_Moon_br = Moon_image_data * 1000
data_Moon = resize_elevation_data(data_Moon_br)
data_Moon = data_Moon.astype('int16')

# Mercury
data_Mercury_br = read_img_file('Mercury_metadata.txt',
'MSGR_DEM_USG_SC_I_V02.IMG')
data_Mercury = resize_elevation_data(data_Mercury_br)

# Earth
data_Earth_br = read_etopo1_bin('Earth_metadata.hdr', 'Earth_data.bin')
data_Earth = resize_elevation_data(data_Earth_br)

# Generating the fractional Brownian (fBm) surface

n = 2048
x = np.linspace(-0.5, 0.5, n)
y = np.linspace(-0.5, 0.5, n)
X, Y = np.meshgrid(x, y)
Z = np.zeros_like(X)

hurst_parameter = 0.5
length = 1
method = 'daviesharte'

number_of_wave_vectors = 8
angle_increment = np.pi / number_of_wave_vectors

for i in range(number_of_wave_vectors):
    angle = i * angle_increment
    wave_vector = np.array([np.cos(angle), np.sin(angle)])
```

```
    fbm = FBM(n=n-1, hurst=hurst_parameter, length=length,
    method=method).fbm()

    for j in range(n):
        for k in range(n):
            r = np.array([X[j, k], Y[j, k]])
            projection = np.dot(wave_vector, r)
            projection_normalized = (projection + 0.5) / np.sqrt(2)
            index = int(projection_normalized * (n - 1))
            index = np.clip(index, 0, n - 2)
            weight = projection_normalized * (n - 1) - index
            Z[j, k] += (1 - weight) * fbm[index]
            + weight * fbm[index + 1]

data_FBM = (Z*1000).astype(int)


# Generating results
Data = [data_FBM, data_Mars, data_Moon, data_Mercury, data_Earth]
names = ['FBM', 'Mars', 'Moon', 'Mercury', 'Earth']

for i, j in zip(Data, names):
    height_range = [int(MAATRIKS.min()), int(MAATRIKS.max()), 10]
    if j == 'FBM':
        height_range = [int(MAATRIKS.min()), int(MAATRIKS.max()), 100]
    results = process_heights(i, height_range)
    turn_into_file(results, j)
```

Appendix 2 - Python code for analysing data

```
import json
import math
import numpy as np
import matplotlib.pyplot as plt

# Function for reading in data

def read_data(file_name):
    with open(f'{file_name}_DATA.json', 'r') as f:
        json_str = f.read()
    data = json.loads(json_str)

    return data

# Reading in the data

FBM_data = read_data('FBM')
Mars_data = read_data('MARS')
Moon_data = read_data('MOON')
Earth_data = read_data('EARTH')
Mercury_data = read_data('MERCURY')

Data = [FBM_data, Mars_data, Moon_data, Earth_data, Mercury_data]
dataset_names = ['fBm','Mars', 'Moon', 'Earth', 'Mercury']

# Functions for getting all the different variables from the data

def Get_data(data):
    heights = [int(i[7:]) for i in data]
    N = [data[i][0]['N'] for i in data]
    L_k = [data[i][0]['L_k'] for i in data]

    S = [sum(j['s'] for j in data[i][1]) for i in data]
    D = [sum(j['d'] for j in data[i][1]) for i in data]

    keys = list(L_k[0].keys())
    L = {key: [j[key] for j in L_k] for key in keys}

    return N, heights, S, D, L


def calculate_dk(L):
```

```
    new_keys = [key.replace('L', 'l') for key in L.keys()]
    L_keys = list(L.keys())

    dd = [
        [(math.log(val1 / val2) / math.log(2)) if
        val2 != 0 else 0 for val1, val2 in
        zip(L[L_keys[i]], L[L_keys[i + 1]])]
        for i in range(len(L) - 1)
    ]

    dd_dict = dict(zip(new_keys, dd))
    return dd_dict


def calculate_d1(data):
    D_andmed2 = [
        {f'l_{2 ** (k - 1)}': math.log(sum([item
        ['l'][f'l_{2 ** (k - 1)}'] for item in dicts1 if
        item['l'][f'l_{2 ** k}'] != 0]) / sum([item['l'][f'l_{2 ** k}']
        for item in dicts1 if item['l'][f'l_{2 ** k}'] != 0])) /
        math.log(2)
        if sum([item['l'][f'l_{2 ** (k - 1)}'] for item in dicts1 if
        item['l'][f'l_{2 ** k}'] != 0]) > 0 else 0 for k in
        range(6, 0, -1)} for dicts1 in [data[i][1] for i in data]
    ]

    D_andmed = [{key: original_dict[key] for key in
    reversed(original_dict)} for original_dict in D_andmed2]

    keys = list(D_andmed[0].keys())
    l_new = {key: [d[key] for d in D_andmed] for key in keys}

    return l_new


def calculate_k(data, S0=32, s0=0, s1=7):
    heights = list(data.keys())
    actual_pindalad = [[j['s'] for j in
    data[height][1]] for height in heights]

    Sammud = [2 ** o for o in range(s0, s1)]

    N_A = [[len([x for x in pindalad if x > samm]) for samm in Sammud]
    for pindalad in actual_pindalad]

    results_K = [
        [0 if i[j+1] == 0 else math.log(i[j] / i[j+1]) / math.log(2)
        for j in range(len(i) - 1)]
```

```
        for i in N_A
    ]

    transposed_K = [[row[i] for row in results_K] for i in
    range(len(results_K[0]))]
    k_keys = [f'l_{2 ** k}' for k in range(0, 6, 1)]

    K_dict = dict(zip(k_keys, transposed_K))
    return K_dict

def calculate_Δd_Δk(d1, dk, k):
    Δd = {key: [(n-1) - 2 * (m-1) for m, n, _ in
    zip(d1[key], dk[key], k[key])] for key in d1.keys()}
    Δdp = {key: [-1 * ((m-1) - (n-1)/2 - 0.064 * (n-1) * (2 - n))
    for m, n, _ in zip(d1[key], dk[key], k[key])]
                for key in d1.keys()}
    Δk = {l - n/2 for _, n, l in zip(d1[key], dk[key], k[key])]
    for key in d1.keys()}

    return Δd, Δdp, Δk

# Getting the variables

def calculate_variables(data):
    N, h, S, D, L = Get_data(data)
    dk, d1, k = calculate_dk(L), calculate_d1(data), calculate_k(data)
    Δd, Δdp, Δk = calculate_Δd_Δk(d1, dk, k)

    return N, h, S, D, L, d1, dk, k, Δd, Δdp, Δk

results = {}
for name, data in zip(dataset_names, Data):
    variables = calculate_variables(data)
    variable_names = ['N', 'h', 'S', 'D', 'L', 'd1', 'dk',
    'k', 'Δd', 'Δdp', 'Δk']
    results[name] = {variable: value for variable, value in
    zip(variable_names, variables)}

# Plotting

def add_label_to_plot(label, ax=None, x=0.92, y=0.95):
# Adds label to plot at the top right
    if ax is None:
        ax = plt.gca()
    ax.annotate(
        label,
        xy=(x, y),
        xycoords='axes fraction',
```

```
        fontsize=40,
        ha='right',
        va='top',
        bbox=dict(boxstyle='round,pad=0.3',
        edgecolor='none', facecolor='aliceblue')
    )


def plot(name, y, ss, window_size = 2, p=0, tt=1):
    var_y = results[name][y]
    h = results[name]['h']
    start_index = h.index(ss[0])
    end_index = h.index(ss[1])
    h_subset = h[start_index:end_index + 1]
    plt.figure(figsize=(30, 14))
    t=0
    if isinstance(var_y, dict):
        if p == 1:
            key = next(iter(var_y))
            var_y_subset = var_y[key][start_index:end_index + 1]
            plt.plot(h_subset, var_y_subset, label=key)
        else:
            t+=1
            for key, values in var_y.items():
                var_y_subset = values[start_index:end_index + 1]
                if t > 1:
                    averaged_y_data = rolling_average(var_y_subset,
                    window_size*2**(t))
                    middle_offset = window_size*2**(t)//2
                    averaged_x_data = h_subset[middle_offset:
                    -middle_offset]
                    plt.plot(averaged_x_data, averaged_y_data,
                    label=key)
                else:
                    plt.plot(h_subset, var_y_subset, label=key)
                plt.legend(fontsize=30, loc='upper left')
    else:
        var_y_subset = var_y[start_index:end_index + 1]
        plt.plot(h_subset, var_y_subset)

    plt.xlabel('Height [m]', fontsize=60)
    if y == 'dk':
        plt.ylabel('D$_{k}$-1', fontsize=60)
    elif y == 'd1':
        plt.ylabel('D$_{1}$-1', fontsize=60)
    elif y == 'Δd':
        plt.ylabel('ΔD', fontsize=60)
    elif y == 'Δdp':
```

```
        plt.ylabel('ΔD$_{c}$', fontsize=60)
    else:
        plt.ylabel(y, fontsize=60)
    plt.xticks(fontsize=50)
    plt.yticks(fontsize=50)
    if y == 'dk':
        plt.ylim(1, 2    )
    if y == 'd1':
        plt.ylim(1, 1.4)
    if y == 'Δd':
        plt.ylim(-0.2, 0.4)
    if y == 'Δdp':
        plt.ylim(-0.1, 0.2)
    if y == 'Δk':
        plt.ylim(-0.5, 0.5)

    add_label_to_plot(name)
    output_filename = f"{name}_{y}.pdf"
    plt.savefig(output_filename, dpi=300, bbox_inches='tight')
    plt.close()


def cut(data, l=1000):
    First_index = None
    Last_index = None
    for i in data:
        if data[i][0]['L_k']['L_1'] > l:
            if First_index == None:
                First_index = int(i[7:])
            Last_index = int(i[7:])
    return (First_index, Last_index)


def rolling_average(data, window_size):
    averaged_data = []
    for i in range(window_size//2, len(data)-(window_size//2)):
        average = np.mean(data[i-(window_size//2):
        i+(window_size//2)+1])
        averaged_data.append(average)
    return averaged_data


def average(data, window_size):
    return [sum(data[i:i+window_size])/window_size
    for i in range(0, len(data) - window_size + 1, window_size)]

for i in variable_names:
    if i in ('h', 'D', 'S', 'L', 'N'):
```

```
        continue
    for j, k in zip(dataset_names, Data):
        dd = cut(k, l=1200)
        plot(j, i, dd)
```