Electronic Supplementary Material 1 to

A novel conceptual approach to objectively determine JRC using fractal dimension and asperity distribution of mapped fracture traces

Published in

Rock Mechanics and Rock Engineering

Author
Martin Stigsson[1, 2], Diego Mas Ivars[1, 3]

Affiliation and address
[1]SKB, Swedish Nuclear Fuel and Waste Management Co
PO Box 3091
SE-169 03 Solna
Sweden

[2]KTH, Royal Institute of Technology
Sustainable development, Environmental science and Engineering (SEED)
Teknikringen 10 B
SE-100 44 Stockholm
Sweden

[3]KTH, Royal Institute of Technology
Civil and Architectural Engineering
Brinellvägen 23
SE-10044 Stockholm
Sweden

Contact
Martin.stigsson@skb.se

Table of Contents

1 Introduction

The text in this electronic supplementary material is a collection of derivations, explanations and analyses that support the main paper. The information on the nature of fractal lines, together with their generation and evaluation (sections 2 to 5), is intended for the reader interested in understanding the methods. The flaw in using the divider method on self-affine fractals is demonstrated using an empirical example in section 6, a common erroneous implementation of the Box Count method is shown in section 7 and the problem with correlation of JRC to the Z_2 method is shown by an example in section 8. In section 9, a study on the number of realisations needed to get stable mean and variance of evaluated parameters is carried out, followed by a study in section 10 on how the evaluated fractal parameters are affected by the fractal dimension, length/resolution and asperity scaling on a set of synthetic traces in sec 10. The study in section 10 is the basis of inferring H and $\sigma\delta h(\Delta L)$ from real traces with as low uncertainty as possible. Finally, section 11 presents a short description of the procedures used when manually digitising traces.

2 Basics of fractals

Fractals can be either self-similar or self-affine. Fractals that are self-similar retain their properties and visual appearance through different levels of magnification, see Figure 2-1 left. A natural example of a self-similar fractal line may be the coastline of an island, where the coordinates are coupled to each other and need to have the same quantity along the two axes to make sense. Self-affine fractals, however, have decoupled measures on the two axes and hence need to be scaled differently in different directions to appear similar, see Figure 2-1 right. An obvious example of a self-affine fractal might be the data traffic as a function of time in a telecom network. It is not possible to define a square or an angle in time-bit, space since there is no geometric relationship between bits and seconds.

It may be easy to erroneously think, as an analogue to the fractal coastline above, that a height profile across the island would conform to a self-similar trace, since both the abscissa and ordinate may be expressed in the same units. However, this is not the case since the two axes may have different units on the axes, still resulting in the same fractal dimension. The intersecting line between a fractal surface and a plane will actually be self-similar if, and only if, the plane is parallel to the average of the fractal surface (Russ 1994). Hence, fractures should conform to self-affine fractals (Mandelbrot 1985; Den Outer et al. 1995; Russ 1994). Later research has shown that fractures can be described as mono-fractal self-affine surfaces over several orders of magnitude (e.g. Renard et al. 2006; Candela et al. 2009; Brodsky et al. 2011; Candela et al. 2012).

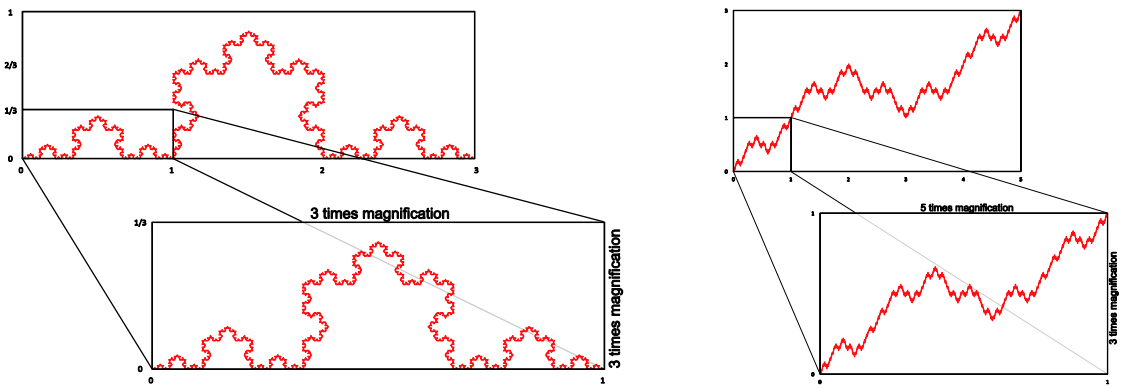

Figure 2-1. *Difference between a self-similar fractal, to the left, and a self-affine fractal, to the right.*

A self-similar fractal is constrained solely by its fractal dimension, D. A self-affine fractal, on the other hand, needs a scaling measure of the ordinate values in addition to the fractal dimension to be fully constrained. The dimension of a self-affine fractal steers the persistence of correlation between vertices at different distances, whilst the scaling measure steers the magnitude of the differences between vertices.

2.1 Fractal dimension

There is a neat relationship between an isotropic fractal surface and the trace of a transect crossing the surface:

$$D_{Line} = D_{Surface} - 1 \qquad \text{eq. 2-1}$$

where:
D_{Line} = the fractal dimension of the transect
$D_{Surface}$ = the fractal dimension of the fractal surface.

The fractal dimension of a self-affine fracture trace can span between the topological and Euclidian dimension. A fracture trace is topologically a line, a 1D object, defined in a Euclidean 2D space. Hence, the dimension of a fractal line is a real number between 1 and 2. Another measure to define

the fractal dimension is the Hurst exponent. The relationship between different fractal dimensions, D_x, and the Hurst exponent, H, is described by e.g. Russ (1994) as:

$$\begin{aligned} H &= 2 - D_{Line} \\ H &= 3 - D_{Surface} \\ H &= 4 - D_{Volume} \end{aligned} \qquad \text{eq. 2-2}$$

where:
H = the Hurst exponent
D_{Line} = the fractal dimension of a fractal line $1 < D_{Line} < 2$
$D_{Surface}$ = the fractal dimension of a fractal surface $2 < D_{Surface} < 3$
D_{Volume} = the fractal dimension of a fractal volume $3 < D_{Volume} < 4$.

This implies that H is restricted to be a real number between 0 and 1 to have a physical meaning. In theory, the Hurst exponent can be larger than 1, resulting in a line that is less than 1D, a Cantor dust, i.e. a line with voids. Theoretically, H can also be less than 0, resulting in a line that wiggles so much that it fills more than the 2D space. However, these fractal "lines" are of limited interest for fracture traces and are not discussed further.

Depending on the value of H, fracture traces can be divided into three groups. $H > 0.5$ reflects traces with long-range correlation; $H < 0.5$ traces with anti-correlation; and $H = 0.5$ traces following a random walk, i.e. the probability that the trace will continue or end the current trend is equal.

There are several methods to evaluate the Hurst exponent of a mono-fractal self-affine fracture trace. These include analysing the slope of power spectrum using Fast Fourier Transform, FFT (Russ 1994); the standard deviation of the correlation function, RMS-COR (Candela et al. 2009); the Korcak Plot of Zero Sets (Russ 1994); the Box Count approach (Malinverno 1990); and many more not covered in this work.

By evaluating a trace, the fractal dimension of the surface in the direction of the trace can be inferred. This means that traces in different directions are needed to characterise the fractal dimension in the different directions. This is particularly necessary when fractures have been sheared and, hence, are supposed to have different fractal dimension in different directions.

2.2 Magnitude measure

The magnitude parameter can be described in different ways. For example Brown (1987), Malinverno (1990) and Johansson and Stille (2014) use the constant $\kappa^{0.5}$ in eq. 2-3, whilst Renard et al. (2006), Candela et al. (2009) and Stigsson (2015) use $\sigma\delta h(\Delta x)$, i.e. the standard deviation of the height differences Δx apart:

$$\sigma\delta h(\Delta x) = \sqrt{\kappa} \cdot \Delta x^{H} \qquad \text{eq. 2-3}$$

where:
$\sigma\delta h(\Delta x)$ = standard deviation of height differences between locations Δx apart
κ = variance of height difference of points one unit apart
H = Hurst exponent.

The magnitude measure of a mono-fractal self-affine fracture trace can be evaluated using e.g. the intercept of power spectrum (Russ 1994) or RMS-COR (Candela et al. 2009).

2.3 Effect of scaling

Fracture traces are supposed to be self-affine, and hence scale differently in the length and height direction. This implies that a longer trace will appear smoother when downscaled, whilst shorter traces will appear rougher when upscaled. Figure 2-2 presents an example of a 1 m long trace with

H = 0.8 and $\sigma\delta h(0.1\ mm)$ = 0.025 mm linearly downscaled to 1 dm. In the same diagram a 1 cm piece of the 1 m trace is linearly upscaled to 1 dm. Despite the 1 cm trace being a part of the 1 m trace it appears to be much rougher than the long trace due to linear scaling.

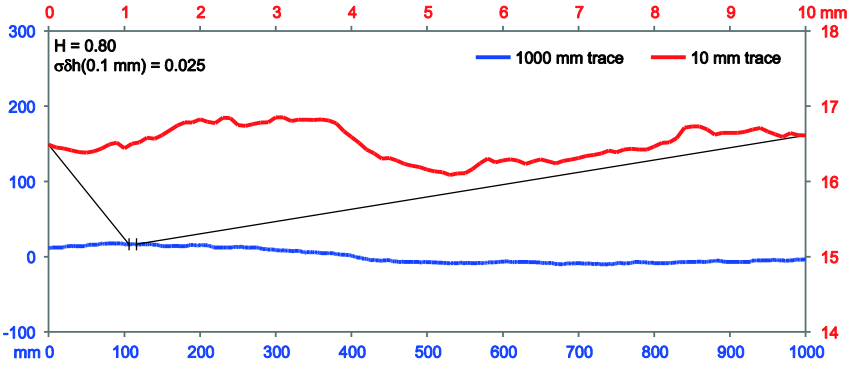

Figure 2-2. *Two traces with the same fractal parameters, H=0.8 and σδh(0.1 mm)=0.025 mm. Due to the different scaling, the blue downscaled trace appears to be smoother than the red upscaled trace.*

3 Fast Fourier Transform

According to the Fourier theorem (Fourier 1822), any complex motion can be broken down to a superimposed series of sine waves. Hence, any fractal line can be accurately reproduced by a series of sine waves. Fast Fourier Transform makes use of Fourier's theorem, but uses a fast algorithm attributed to Cooley and Tukey (1965). Their method is very similar to an unpublished method developed by Gauss, presumably in 1805, but published posthumously in Gauss (1866), see e.g. Goldstine (1977). The method is well described in e.g. Smith (1997).

Fast Fourier Transform, FFT, can only handle data that conform to 2^n entries. In its original implementation, the method transformed data between the time domain and the frequency domain. However, it can be applied to transformation between the spatial domain and the length frequency domain, i.e. between height values of a fracture trace and the power spectrum of the length frequencies. The workflow to transform data between the spatial and frequency domain is shown in Figure 3-1.

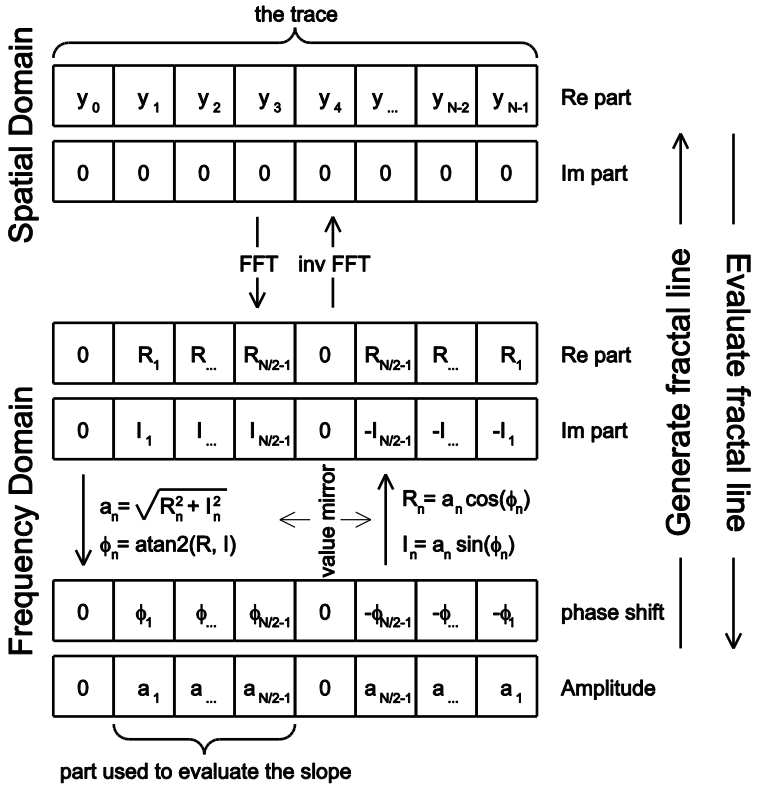

Figure 3-1. *The workflow for generating and evaluating fractal lines using Fast Fourier Transform (FFT).*

3.1 Theory of FFT

The FFT algorithm mainly consists of two parts; rearranging data according to bit reversal sorting, followed by butterfly calculations for each stage.

3.1.1 Bit reversal sorting

Bit reversal sorting or ordering is nothing more than arranging the entries according to the mirrored bit value, see Figure 3-2a.

An algorithm to construct the index order after bit reversal is shown in Figure 3-2b. For each stage, *s*, the algorithm multiplies the values in the former bit-reversed vector by two, and then appends the new bit-reversed vector with a copy of the first $2^s/2$ entries added one to each value. The procedure continues until the bit-reversed vector contains the desired number of entries. When the index order

vector is constructed, the entries in the real and imaginary vectors, Figure 3-1, are sorted accordingly.

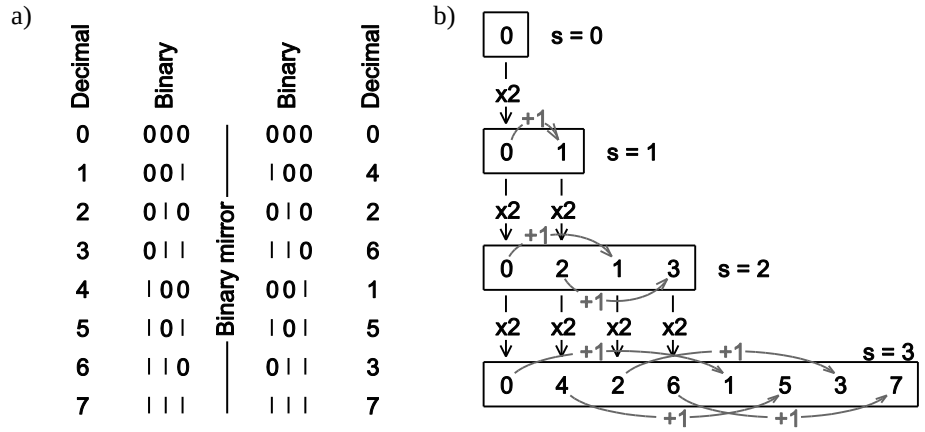

Figure 3-2. *a) Example of bit reversal of eight digits. b) An algorithm to perform bit reversal.*

3.1.2 Butterfly calculations

In butterfly calculation, two complex numbers are cross-added using multiplication of a twiddle factor. The method is called "butterfly" since the paths for the calculations resemble a butterfly, Figure 3-3. The output is two new complex numbers:

$$b_i = a_i + W_n^k \cdot a_j$$
$$b_j = a_i - W_n^k \cdot a_j$$

eq. 3-1

where:
a_x = the input complex number
b_x = the output complex number
W_n^k = the twiddle factor, eq. 3-2.

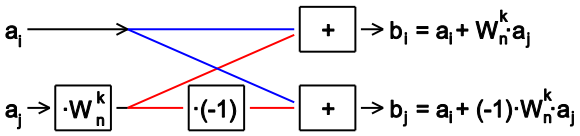

Figure 3-3. *A single butterfly calculation visualised.*

The complex value of the twiddle factor, W_n^k, depends on the number of entries in the current butterfly group, n, and the order, k, of the calculation in each butterfly group. The twiddle factor is calculated according to:

$$W_n^k = \cos\left(-2\pi \cdot \frac{k}{n}\right) + \sin\left(-2\pi \cdot \frac{k}{n}\right) i$$

eq. 3-2

where:
W_n^k = the twiddle factor
n = the number of entries in the current butterfly group
k = the order of the calculation in the current butterfly group.

The twiddle factor is hence a unit vector in the complex number space that repeats the result periodically. A visualisation of the three first stages of eq. 3-2 is provided in Figure 3-4.

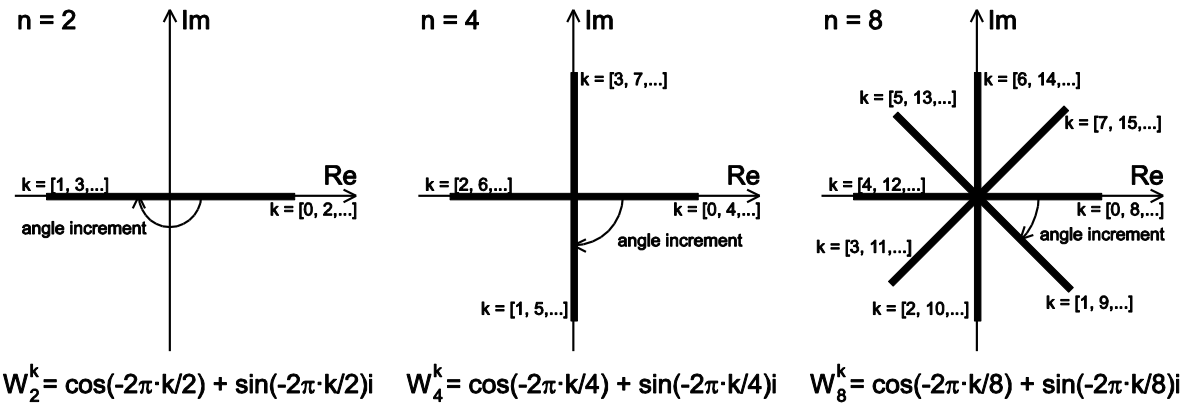

Figure 3-4. *Visualisation of calculation of the twiddle factors used in Figure 3-5.*

Recalling the rules of multiplying complex numbers, the calculation of the twiddle factor, W_n^k, times a_j is:

$$W_n^k \cdot a_j = \left(\mathrm{Re}\left[W_n^k\right] + \mathrm{Im}\left[W_n^k\right]i\right) \cdot \left(\mathrm{Re}\left[a_j\right] + \mathrm{Im}\left[a_j\right]i\right) = \\ \mathrm{Re}\left[W_n^k\right] \cdot \mathrm{Re}\left[a_j\right] - \mathrm{Im}\left[W_n^k\right] \cdot \mathrm{Im}\left[a_j\right] + \left(\mathrm{Re}\left[W_n^k\right] \cdot \mathrm{Im}\left[a_j\right] + \mathrm{Im}\left[W_n^k\right] \cdot \mathrm{Re}\left[a_j\right]\right)i \qquad \text{eq. 3-3}$$

The butterfly calculations are carried out for each stage where the distance between the *i* and *j* indices increases by a factor of two for each stage. Figure 3-5 shows all butterfly calculations for the three stages of an array of eight complex numbers.

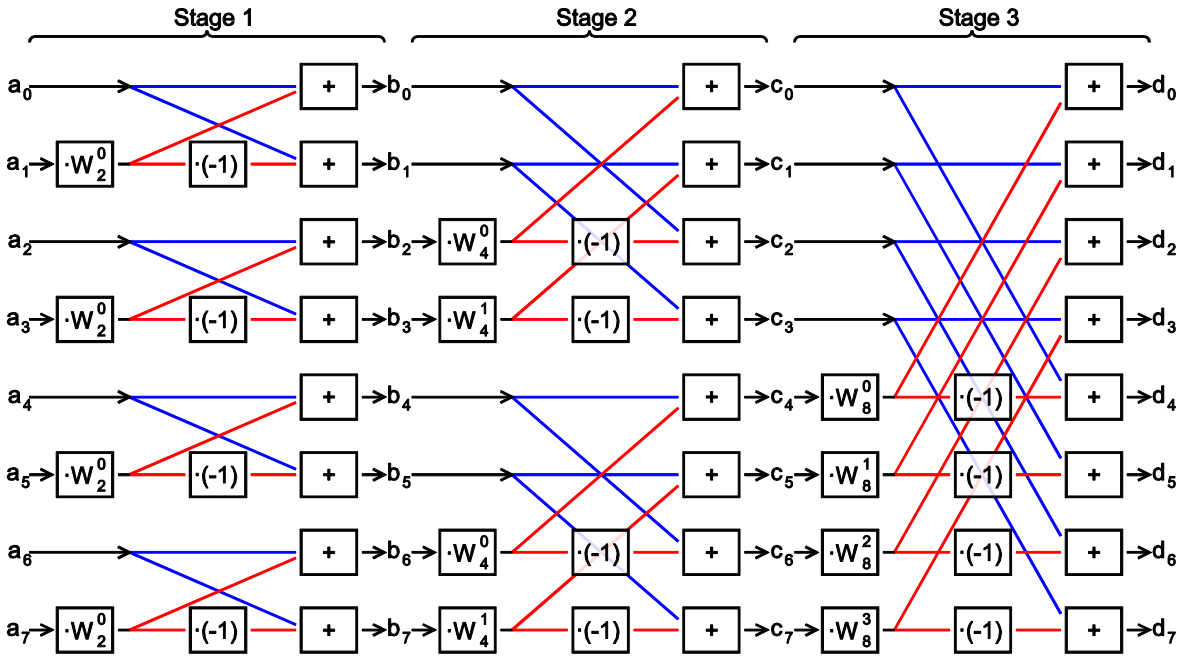

Figure 3-5. *Visualisation of all butterfly calculations for an array of eight complex numbers.*

3.2 Theory of Inverse FFT, IFT

There are different ways of doing Inverse FFT (IFT) using forward FFT, see e.g. Lyons (2015). One of the methods makes use of complex conjugation, i.e. multiplying the imaginary values by –1. The method starts with a complex conjugation, followed by forward FFT, another complex conjugation and a final division by the number of entries. However, for the generation of fractal lines the imaginary part of the spatial domain will be zero, and hence the second complex conjugation can be omitted for efficiency purposes. The workflow is visualised in Figure 3-6.

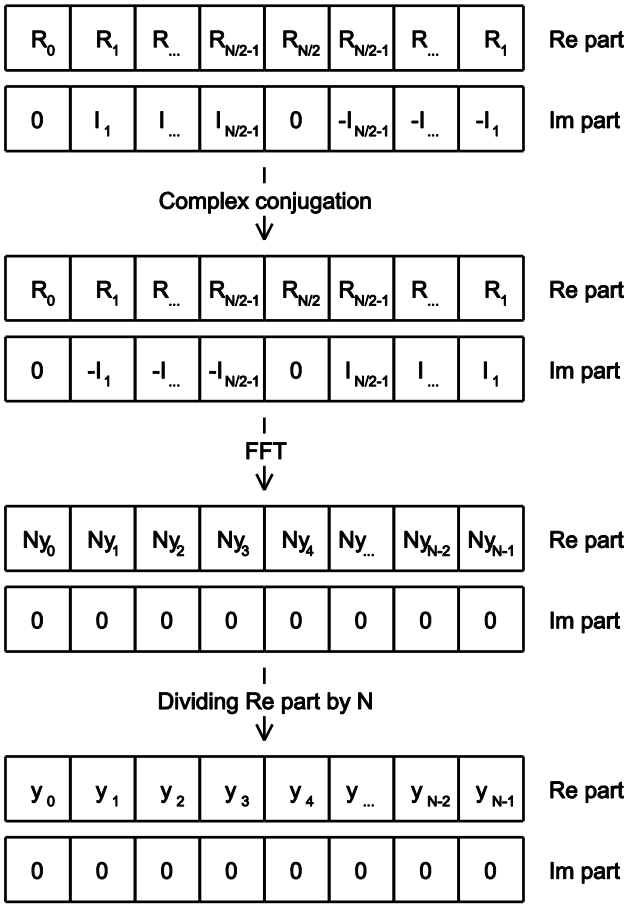

Figure 3-6. *The workflow for Inverse Fast Fourier Transform (IFT) by use of forward FFT.*

3.3 Codes

```
'===============================================================================
'===                                FFT                                      ===
'===============================================================================
'===  Routine that Calculate the FFT of an array of complex numbers          ===
'===                                                                         ===
'===  References:                                                            ===
'===  THE FAST FOURIER TRANSFORM                                             ===
'===  The Scientist and Engineer's Guide to Digital Signal Processing        ===
'===  By Steven W Smith                                                      ===
'===  ISBN: 978-0966017632                                                   ===
'===  copyright © 1997-1999 by California Technical Publishing,              ===
'===  San Diego California                                                   ===
'===  www.dspguide.com                                                       ===
'===                                                                         ===
'===  https://en.wikipedia.org/wiki/Bit-reversal_permutation                 ===
'===============================================================================
'====                                                                       ====
'===  Input:                                                                 ===
'===  Re(0 to N-1) = the y data vector in arithmetic space                   ===
'===  Im(0 to N-1) = the imaginary part of space vector, i.e. all 0          ===
'===  nofData      = the number of data in each data vector                  ===
'====                                                                       ====
'===  Output:                                                                ===
'===  Re(0 to N-1) = the real part of the frequency domain vector            ===
'===  Im(0 to N-1) = the imaginary part of the frequency domain vector       ===
'====                                                                       ====
'===============================================================================
'===  Written by Martin Stigsson 10 March 2017                               ===
'===  Please, report errors or improvements to: martin.stigsson@skb.se       ===
'===============================================================================
Sub FFT(ByRef Re() As Double, _
        ByRef Im() As Double, _
        ByRef nofData As Integer)
```

```
Dim pi As Double           'Pi, 3.14159265... You know
Dim i, j As Integer        'Counters/indices
Dim currBgrp As Integer    'Counter keeping current butterfly group
Dim locNdx As Integer      'Counter keeping local index in each butterfly
'                           group
Dim nofStages As Integer   'Number of stages in the FFT to be calculated
Dim currStage As Integer   'The current stage number
Dim nofBgroups As Integer  'number of Butterfly groups in current stage
Dim ndxDiff As Integer     'The difference between the indices in the
'                           current butterfly calculation
Dim nofEntries As Integer  'Number of entries in the butterfly arrays of
'                           the current stage
Dim ReW As Double          'Real part of Twiddle factor
Dim ImW As Double          'Imaginary part ofTwiddle factor
Dim ReWaj As Double        'Real part of the multiplication of Twiddle
'                           factor and aj, eq 3-2
Dim ImWaj As Double        'Imaginary  part of the multiplication of
'                           Twiddle factor And aj, eq 3-2
Dim dReW As Double         'Real part of the rotation increment of the
'                           twiddle factor
Dim dImW As Double         'Imaginary part of the rotation increment of
'                           the twiddle factor
Dim tmpRe As Double        'Temporary keep real part of a complex number
Dim tmpIm As Double        'Temporary keep imaginary part of a complex
'                           number

Dim bitRevOrder(nofData - 1) As Integer 'The order to sort the data
'                                        before performing the butterfly


'***  INITIATE PARAMETERS  ***
'Define the constant pi
pi = 4 * Atan(1)

'Calculate of Stages
nofStages = CInt(Log(nofData) / Log(2))


'***  CONSTRUCT THE BIT REVERSED ORDER ARRAY  ***
'Loop throgh all levels
For i = 0 To nofStages - 1

    'Multiply the former vector position with 2 and append it added 1,
    'Figure 3-2 b
    For j = 0 To CInt(2 ^ i) - 1
        bitRevOrder(j) = bitRevOrder(j) * 2
        bitRevOrder(j + 2 ^ i) = bitRevOrder(j) + 1
    Next
Next


'***  SORT THE DATA ACCORDING TO THE BIT REVERSED ORDER  ***
'Go throug the bit reversed index vector and swap entries accordingly
For i = 0 To nofData - 1
    'Only do the first swap, other wise the vector will look the same as
    'the input on return due to swap forth and back
    If i < bitRevOrder(i) Then

        'Temporary save real and imaginary values
        tmpRe = Re(bitRevOrder(i))
        tmpIm = Im(bitRevOrder(i))

        'Move values to corrrect index
        Re(bitRevOrder(i)) = Re(i)
        Im(bitRevOrder(i)) = Im(i)

        'Finalise the swap by coping the temporary saved data to correct
        'index()
        Re(i) = tmpRe
        Im(i) = tmpIm
    End If
Next
```

```
    '***   GO THROUGH ALL STAGES AND DO THE BUTTERFLY CALCULATIONS   ***
    'visualised in Figure 3-4
    'Loop for each stage
    For currStage = 1 To nofStages

        'Calculate number of butterfly groups
        nofBgroups = CInt(2 ^ (nofStages - currStage))

        'Calculate the difference between the indices in the current
        'butterfly calculation
        ndxDiff = CInt(2 ^ (currStage - 1))

        'Calculate the number of entries in the current butterfly
        nofEntries = ndxDiff * 2

        'Calculate Twiddle angle increment. Negative angles due to clockwise
        'rotation, (eq 3-3)
        dReW = Cos(-2 * pi / nofEntries)
        dImW = Sin(-2 * pi / nofEntries)

        'For each butterfly group
        For currBgrp = 0 To nofBgroups - 1

            'Initiate the real part, ReW, and imaginary part, ImW of the
            'twiddle(factor, W)
            ReW = 1
            ImW = 0

            'Do the calculation for each butterfly in the current sub array
            For locNdx = 0 To ndxDiff - 1

                'Calculate the indices to use in the butterfly
                i = locNdx + currBgrp * nofEntries
                j = i + ndxDiff

                'Calculation of the twiddle factor times the aj (eq 3-2)
                ReWaj = ReW * Re(j) - ImW * Im(j)
                ImWaj = ReW * Im(j) + ImW * Re(j)

                'Do the butterfly, (eq 3-1)
                Re(j) = Re(i) - ReWaj
                Im(j) = Im(i) - ImWaj
                Re(i) = Re(i) + ReWaj
                Im(i) = Im(i) + ImWaj

                'Update twiddle factors by rotating one step, by simple
                'rotation of vectors, i.e.
                ' x' = x·cos(a) - y·sin(a)
                ' y' = x·sin(a) + y·cos(a)
                tmpRe = ReW
                ReW = tmpRe * dReW - ImW * dImW
                ImW = tmpRe * dImW + ImW * dReW

            Next locNdx

        Next currBgrp

    Next currStage

End Sub
```

```
'================================================================================
'===                                invFFT                                    ===
'================================================================================
'===  Routine that Calculate the inverse FFT of an array of complex      ===
'===  numbers using complex conjugation and forward Fast Fourier         ===
'===  Transform                                                          ===
'===                                                                     ===
'===  References:                                                        ===
'===  THE FAST FOURIER TRANSFORM                                         ===
'===  The Scientist and Engineer's Guide to Digital Signal Processing    ===
'===  By Steven W Smith                                                  ===
'===  ISBN: 978-0966017632                                               ===
'===  copyright © 1997-1999 by California Technical Publishing,          ===
'===  San Diego California                                               ===
'===  www.dspguide.com                                                   ===
'===  section 12.3 and Table 12-5                                        ===
'===                                                                     ===
'===  https://www.dsprelated.com/showarticle/800.php                     ===
'===  see also Method 4, complex conjugate                               ===
'================================================================================
'====                                                                  ====
'===  Input:                                                             ===
'===  Re(0 to N-1) = the real part of the frequency array                ===
'===  Im(0 to N-1) = the imaginary part of the frequency array           ===
'===  nofData      = the number of data in each array                    ===
'====                                                                  ====
'===  Output:                                                            ===
'===  Re(0 to N-1) = the height values of the trace                      ===
'===  Im(0 to N-1) = the imaginary array, should be all zeros on return  ===
'====                                                                  ====
'================================================================================
'===  Written by Martin Stigsson 14 March 2017                           ===
'===  Please, report errors or improvements to: martin.stigsson@skb.se   ===
'================================================================================
Sub invFFT(ByRef Re() As Double,
           ByRef Im() As Double,
           ByRef nofData As Integer)

    Dim i As Integer

    '***  COMPLEX CONJUGATION  ***
    'Step 1 in Figure 3-6
    'Change the sign of Im, i.e. make the conjugate of the complex number
    For i = 0 To nofData - 1
        Im(i) = -Im(i)
    Next i

    '***  FAST FOURIER TRANSFORM  ***
    'Step 2 in Figure 3-6
    'run forward FFT
    Call FFT(Re, Im, nofData)

    '*** DIVIDING BY N ***
    'Step 3 in Figure 3-6
    'Divide the real part by the number of entries and the Re array will
    'contain the trace
    'The complex conjugation of the complex number is not necessary since
    'the imaginary part should be zero.
    For i = 0 To nofData - 1
        Re(i) = Re(i) / nofData
    Next i

End Sub
```

4 Generating fractal lines

Russ (1994) carried out a comprehensive study of different methods to generate fractal lines and different methods to evaluate the fractal properties. Russ used four methods to generate different fractal lines of 1024 points: 1) Iterative Midpoint Displacement, 2) Fractal Brownian Motion, 3) Mandelbrot-Weierstrass Function and 4) Inverse Fast Fourier Transform, IFT, of Power Spectrum. In this material only IFT of Power Spectrum is explained, since it seems to generate reasonable results over the physical interesting span of the Hurst exponent, i.e. 0<H<1.

The method uses a power spectrum of frequencies together with a random phase shift as input. These imaginary values are run through an IFT and the output is a fractal line. The fractal line is, hence, created of superimposed sine waves with random starting points. The method is visually explained in Figure 3-1 and the underlying equations are explained below.

4.1 Frequency Domain, Amplitude

For a fractal line, the power is proportional to the inverse of the frequency raised to β, i.e.:

$$p(f) \propto 1/f^{\beta} \qquad \text{eq. 4-1}$$

where:
p(f) = power of the wave with frequency *f*
f = frequency of the wave
β = slope of the power spectrum.

The power of the wave in the frequency domain has a direct relationship to the amplitude of the wave in the spatial domain according to:

$$p(f) = \left(a(f) \frac{N}{2} \right)^2 \qquad \text{eq. 4-2}$$

where:
p(f) = power of the wave with frequency *f*
a(f) = amplitude of the wave with frequency *f*
N = number of sampling points.

The relationship between β and the Hurst exponent, *H*, is:

$$\beta = 2H + 1 \qquad \text{eq. 4-3}$$

Combining equations eq. 4-1 to eq. 4-3 results in:

$$a(f) \propto \frac{2 \cdot f^{-(H+1/2)}}{N} \qquad \text{eq. 4-4}$$

The proportionality constant in eq. 4-4 is the square root of the intercept of the regression of the power spectrum. Hence, the amplitude *a(f)* is a function of the Hurst exponent, *H*, and frequency, *f*, expressed as:

$$a(f) = \sqrt{c_I} \cdot \frac{2}{N} \cdot f^{-(H+1/2)} \qquad \text{eq. 4-5}$$

where:
$a(f)$ = amplitude of the wave with frequency f
c_I = intercept of the regression of the power spectrum
f = frequency
H = the Hurst exponent.

The constant, c_I, in eq. 4-5 steers the amplitude measure $\sigma\delta h(1p)$, i.e. the standard deviation of the height differences of adjacent vertices of the trace. The standard deviation of height differences between adjacent vertices of one full single discretised sine wave can be expressed as:

$$\sigma_{1p} \approx \sqrt{2} \cdot A \cdot \sin\left(\pi \cdot f / N\right) \qquad \text{eq. 4-6}$$

where:
σ_{1p} = standard deviation of height differences of adjacent vertices over one full wave
A = amplitude
f = frequency
N = number of data points along one wave.

The derivation of eq. 4-6 is shown in appendix A. The value is not exact, since there is a negligible dependence of the result on the phase offset. The error is largest when a sine wave has a phase offset that is exactly in the middle of two adjacent vertices.

The standard deviation of the height differences of adjacent vertices for any of the frequencies in the power spectrum can hence be calculated by combining eq. 4-5 and eq. 4-6:

$$\sigma_{1p}\left(f\right) \approx \frac{2\sqrt{2}}{N} \cdot \sqrt{c} \cdot f^{-(H+1/2)} \cdot \sin\left(\pi \cdot f / N\right) \qquad \text{eq. 4-7}$$

The standard deviation of a series of independent distributions is calculated according to:

$$\sigma_{Total} = \sqrt{\sum_{i=1}^{I} \sigma_i^2} \qquad \text{eq. 4-8}$$

where:
σ_{Total} = standard deviation of all I independent distributions combined
σ_i = standard deviation of distribution i
I = number of independent distributions
i = distribution number.

Hence, the amplitude measure $\sigma\delta h(1p)$ for a series of sine waves is calculated according to:

$$\sigma\delta h\left(1p\right) \approx \frac{2\sqrt{2}}{N} \cdot \sqrt{c_I} \cdot \sqrt{\sum_{k=1}^{N/2-1} \left(k^{-(H+1/2)} \cdot \sin\left(\pi \cdot k / N\right)\right)^2} \qquad \text{eq. 4-9}$$

where:
$\sigma\delta h(1p)$ = standard deviation of height differences of adjacent vertices of the fractal line
c_I = intercept of the regression of the power spectrum
k = frequency, i.e. number of waves per trace length
N = number of vertices of the fractal line
H = the Hurst exponent.

Solving eq. 4-9 for c and inserting the results into eq. 4-5 yields:

$$a(f) \approx \frac{\frac{N}{2\sqrt{2}} \cdot \sigma\delta h(1p)}{\sqrt{\sum_{k=1}^{N/2-1} \left(k^{-(H+1/2)} \cdot \sin(\pi \cdot k/N)\right)^2}} \cdot f^{-(H+1/2)} \qquad \text{eq. 4-10}$$

where:

a(f)	=	amplitude of wave with frequency *f*
f	=	frequency
N	=	number of vertices to be generated
σδh(1p)	=	desired standard deviation of height differences of adjacent vertices
k	=	counter and frequency in summation
H	=	Hurst exponent.

The summation is only done over the frequencies 1 to $N/2 - 1$, since the wave numbers above the Nyquist frequency cannot be solved by the resolution at hand and will hence only add noise.

The amplitude vector is constructed in the following way. The first value in the amplitude vector and the value at index $N/2$ are assigned zeros, since these values only reflect the average vertical offset of the trace. The amplitude values at index 1 to $N/2 - 1$ follow eq. 4-10. The remaining amplitudes are mirrored around index N/2 according to:

$$a_{i+N/2} = a_{N/2-i} \quad 1 \le i \le N/2-1 \qquad \text{eq. 4-11}$$

4.2 Frequency Domain, Phase shift

The phase shift vector is needed to randomly spread the start point of the different sine waves. The phase shift vector is constructed in the following way. First, the phase shift vector is assigned a zero at index zero and index $N/2$. This is because the summation of these positions only reflects the average vertical offset of the trace. Thereafter, the $N/2$-1 positions in between are filled with random numbers following a uniform distribution between 0 and 2π. The remaining $N/2$-1 positions are assigned mirrored phase shift numbers multiplied by –1, i.e.:

$$\phi_{i+N/2} = -\phi_{N/2-i} \quad 1 \le i \le N/2-1 \qquad \text{eq. 4-12}$$

The mirroring of both the phase shift and amplitude equals a complex conjugate of the mirrored complex numbers.

4.3 Frequency Domain, Imaginary part

The imaginary part of the complex frequency is simply calculated as the amplitude, a_k, multiplied by the sine of the phase shift ϕ_k for all N entries, i.e.:

$$\mathrm{Im} = a_k \cdot \sin(\phi_k) \quad 0 \le k \le N-1 \qquad \text{eq. 4-13}$$

4.4 Frequency Domain, Real part

In the same way as the imaginary part is calculated, the real part of the complex frequency is calculated as the amplitude, a_k, multiplied by the cosine of the phase shift ϕ_k for all N entries, i.e.:

$$\mathrm{Re} = a_k \cdot \cos(\phi_k) \quad 0 \le k \le N-1 \qquad \text{eq. 4-14}$$

4.5 Spatial Domain

The complex numbers constructed by the vectors in eq. 4-13 and eq. 4-14 are run through the IFT explained in section 3.2. The result is a vector of real numbers describing the random fractal line.

4.6 Worked example

The generation of a fractal line using Inverse Fast Fourier Transform of Power Spectrum has the following steps

1. Choose desired *H*, *σδh(1p)* and number of vertices.
2. Construct the square root of power spectrum array.
3. Construct the random phase shifts array.
4. Calculate the imaginary numbers using the arrays.
5. Do a complex conjugation of the imaginary number array.
6. Sort the imaginary number array according to bit reversed order.
7. Do the butterfly calculations.
8. Divide the results by the number of vertices of the trace.

The example chosen here is to construct a fractal line with $H = 0.600$, $\sigma\delta h(1p) = 0.2$ and 16 vertices. From these data, the parts of the summation in the denominator of eq. 4-10 are shown in Table 4-1. The summation is used in eq. 4-10 to calculate the amplitude array *a(f)*. The phase shift array is constructed using a uniform distribution of values between 0 and 2π.

Table 4-1. Calculation of the seven unique values of the input arrays

k	$k^{-(H+1/2)} \cdot \sin(\pi \cdot k/N)$	f	a(f)	Phase shift
1	0.195	1	2.735	1.045
2	0.179	2	1.276	2.904
3	0.166	3	0.817	1.707
4	0.154	4	0.595	2.348
5	0.142	5	0.466	0.758
6	0.129	6	0.381	0.168
7	0.115	7	0.322	2.16

The values from Table 4-1 are used as input to eq. 4-11 and eq. 4-12 to construct the amplitude and phase shift arrays according to Figure 3-1, see Table 4-2. The real and imaginary parts are calculated according to eq. 4-13 and eq. 4-14 and shown in Table 4-2.

Table 4-2. The amplitude and random phase shift arrays constructed from the values in Table 4-1, together with the corresponding imaginary numbers array

a(f)	phase shift	Re part	Im part
0	0	0.000	0.000
2.735	1.045	1.373	2.366
1.276	2.904	-1.240	0.300
0.817	1.707	-0.111	0.809
0.595	2.348	-0.417	0.424
0.466	0.758	0.338	0.320
0.381	0.168	0.376	0.064
0.322	2.16	-0.179	0.267
0	0	0.000	0.000
0.322	-2.16	-0.179	-0.267
0.381	-0.168	0.376	-0.064
0.466	-0.758	0.338	-0.320
0.595	-2.348	-0.417	-0.424
0.817	-1.707	-0.111	-0.809
1.276	-2.904	-1.240	-0.300
2.735	-1.045	1.373	-2.366

The complex conjugation and the following bit reversal is carried out on the resulting imaginary array from Table 4-2, see Table 4-3.

Table 4-3. The complex conjugation and bit reversal ordering of the imaginary array in Table 4-2

Frequency domain			Complex conjugate		Bit reversed		
Entry	Re part	Im part	Re part	Im part	Entry	Re part	Im part
0	0.000	0.000	0.000	0.000	0	0.000	0.000
1	1.373	2.366	1.373	-2.366	8	0.000	0.000
2	-1.240	0.300	-1.240	-0.300	4	-0.417	-0.424
3	-0.111	0.809	-0.111	-0.809	12	-0.417	0.424
4	-0.417	0.424	-0.417	-0.424	2	-1.240	-0.300
5	0.338	0.320	0.338	-0.320	10	0.376	0.064
6	0.376	0.064	0.376	-0.064	6	0.376	-0.064
7	-0.179	0.267	-0.179	-0.267	14	-1.240	0.300
8	0.000	0.000	0.000	0.000	1	1.373	-2.366
9	-0.179	-0.267	-0.179	0.267	9	-0.179	0.267
10	0.376	-0.064	0.376	0.064	5	0.338	-0.320
11	0.338	-0.320	0.338	0.320	13	-0.111	0.809
12	-0.417	-0.424	-0.417	0.424	3	-0.111	-0.809
13	-0.111	-0.809	-0.111	0.809	11	0.338	0.320
14	-1.240	-0.300	-1.240	0.300	7	-0.179	-0.267
15	1.373	-2.366	1.373	2.366	15	1.373	2.366

The bit-reversed complex numbers are run through the four stages of butterfly calculations according to Table 4-4 to Table 4-7.

Table 4-4. The first stage of butterfly calculations

		Stage 1			
Input to calculation		Twiddle factor W_1^k		Result stage 1	
Re	Im	Re	Im	Re	Im
0.000	0.000			0.000	0.000
0.000	0.000	1	0	0.000	0.000
-0.417	-0.424			-0.835	0.000
-0.417	0.424	1	0	0.000	-0.849
-1.240	-0.300			-0.864	-0.237
0.376	0.064	1	0	-1.616	-0.364
0.376	-0.064			-0.864	0.237
-1.240	0.300	1	0	1.616	-0.364
1.373	-2.366			1.194	-2.098
-0.179	0.267	1	0	1.551	-2.633
0.338	-0.320			0.227	0.489
-0.111	0.809	1	0	0.449	-1.129
-0.111	-0.809			0.227	-0.489
0.338	0.320	1	0	-0.449	-1.129
-0.179	-0.267			1.194	2.098
1.373	2.366	1	0	-1.551	-2.633

Table 4-5. The second stage of butterfly calculations

		Stage 2			
Result stage 1		Twiddle factor W_2^k		Result stage 2	
Re	Im	Re	Im	Re	Im
0.000	0.000			-0.835	0.000
0.000	0.000			-0.849	0.000
-0.835	0.000	1	0	0.835	0.000
0.000	-0.849	0	-1	0.849	0.000
-0.864	-0.237			-1.729	0.000
-1.616	-0.364			-1.980	-1.980
-0.864	0.237	1	0	0.000	-0.473
1.616	-0.364	0	-1	-1.252	1.252
1.194	-2.098			1.421	-1.609
1.551	-2.633			0.422	-3.082
0.227	0.489	1	0	0.967	-2.587
0.449	-1.129	0	-1	2.681	-2.184
0.227	-0.489			1.421	1.609
-0.449	-1.129			-3.082	0.422
1.194	2.098	1	0	-0.967	-2.587
-1.551	-2.633	0	-1	2.184	-2.681

Table 4-6. The third stage of butterfly calculations

Result stage 2		Stage 3 Twiddle factor W_4^k		Result stage 3	
Re	Im	Re	Im	Re	Im
-0.835	0.000			-2.564	0.000
-0.849	0.000			-3.648	0.000
0.835	0.000			0.362	0.000
0.849	0.000			2.619	0.000
-1.729	0.000	1	0	0.894	0.000
-1.980	-1.980	0.707	-0.707	1.951	0.000
0.000	-0.473	0	-1	1.308	0.000
-1.252	1.252	-0.707	-0.707	-0.922	0.000
1.421	-1.609			2.842	0.000
0.422	-3.082			-1.459	-0.604
0.967	-2.587			-1.621	-1.621
2.681	-2.184			-0.759	-1.832
1.421	1.609	1	0	0.000	-3.218
-3.082	0.422	0.707	-0.707	2.303	-5.560
-0.967	-2.587	0	-1	3.554	-3.554
2.184	-2.681	-0.707	-0.707	6.121	-2.535

Table 4-7. The fourth stage of butterfly calculations and the resulting trace

Result stage 3		Stage 4 Twiddle factor W_8^k		Result stage 4		The trace	
Re	Im	Re	Im	Re	Im	vertex	Height
-2.564	0.000			0.279	0.000	0	0.017
-3.648	0.000			-5.228	0.000	1	-0.327
0.362	0.000			-1.930	0.000	2	-0.121
2.619	0.000			0.635	0.000	3	0.040
0.894	0.000			-2.324	0.000	4	-0.145
1.951	0.000			-4.067	0.000	5	-0.254
1.308	0.000			-3.718	0.000	6	-0.232
-0.922	0.000			-7.546	0.000	7	-0.472
2.842	0.000	1	0	-5.406	0.000	8	-0.338
-1.459	-0.604	0.924	-0.383	-2.069	0.000	9	-0.129
-1.621	-1.621	0.707	-0.707	2.654	0.000	10	0.166
-0.759	-1.832	0.383	-0.924	4.602	0.000	11	0.288
0.000	-3.218	0	-1	4.112	0.000	12	0.257
2.303	-5.560	-0.383	-0.924	7.969	0.000	13	0.498
3.554	-3.554	-0.707	-0.707	6.334	0.000	14	0.396
6.121	-2.535	-0.924	-0.383	5.703	0.000	15	0.356

As expected, the imaginary array only contains zeros after the final stage of butterfly calculations, and hence the complex conjugation does not have to be carried out. However, the real part needs to be divided by the number of entries, 16, to get the heights of the trace. The trace is shown in Figure 4-1.

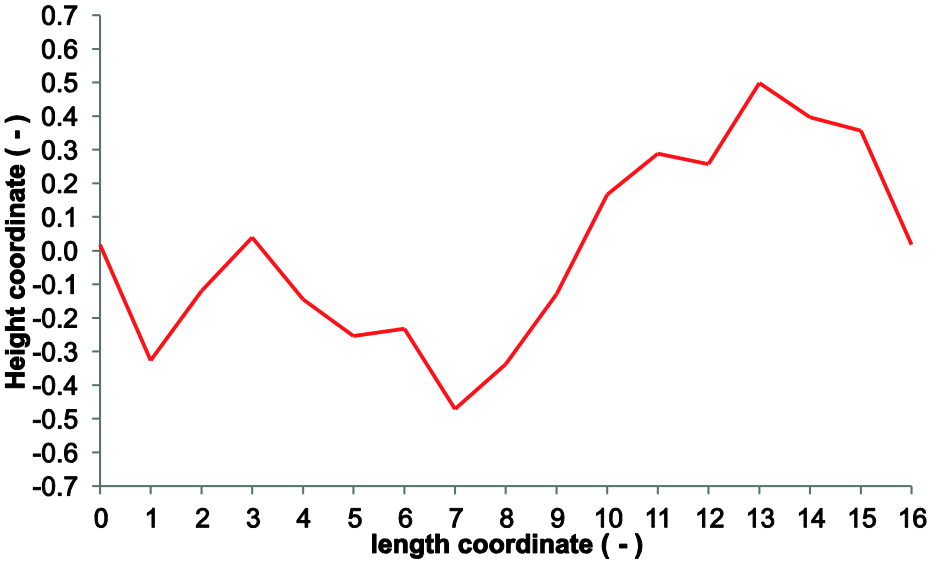

Figure 4-1. *Visualisation of the resulting fractal line from the power spectrum and phase shift listed in Table 4-1*

4.7 Code

```
'================================================================================
'===                          generate_FFT_line                                 ===
'================================================================================
'====                                                                          ====
'===  Routine that generates a fractal line using Inverse Fast Fourier         ===
'===  Transform                                                                ===
'====                                                                          ====
'================================================================================
'====                                                                          ====
'===  Input:                                                                   ===
'===  size  = Number of vertices along the line                                ===
'===  hurst = The Hurst coefficient defining the fractal dimension             ===
'===  sdh1p = Standard deviation of height difference between adjacent         ===
'===          vertices                                                         ===
'===  seed  = The seed for the random number generator                         ===
'====                                                                         ====
'===  Output:                                                                  ===
'===  dataVector   = The vector containing the height value describing         ===
'===                 the fractal line                                         ===
'===  errorMessage = If something goes wrong, save information in this         ===
'===                 string                                                   ===
'====                                                                         ====
'================================================================================
'===  Written by Martin Stigsson, March 14 2017                                ===
'===  Please, report errors or improvements to: martin.stigsson@skb.se         ===
'================================================================================
Sub generate FFT line(ByRef size As Integer,
                      ByRef hurst As Double, _
                      ByRef sdh1p As Double, _
                      ByRef seed As Double, _
                      ByRef dataVector() As Double, _
                      ByRef errorMessage As String)
    Dim i As Integer

    Dim pi As Double
    Dim slope As Double
    Dim amplConst As Double
    Dim amplVect(size / 2 - 1) As Double
    Dim phaseShift(size / 2 - 1) As Double
    Dim Re(size - 1) As Double
    Dim Im(size - 1) As Double

    'Define Pi
    pi = Atan(1) * 4

    'Initiate the seed
    Randomize(seed)
```

```
    'Calculate amplitude slope
    slope = -(hurst + 1 / 2)

    'Initiate datavector
    For i = 0 To size - 1
        dataVector(i) = 0
    Next

    'Calculate constant in eq. 4-10
    amplConst = calcAmplConst(sdh1p, size, hurst)

    'Construct the amplitude array (eq. 4-10) [ Step 2 ]
    For i = 1 To size / 2 - 1
        amplVect(i) = amplConst * i ^ slope
    Next

    'Generate the pseudo random phase shifts [ Step 3 ]
    For i = 1 To size / 2 - 1
        phaseShift(i) = Rnd() * 2 * pi
    Next

    'Set the first values to zero
    Re(0) = 0
    Im(0) = 0

    'Populate the first halves of the Real and imaginary arrays with values
    For i = 1 To size / 2 - 1

        'Divide the amplitude into the real and imaginary part according to
        'the phase shift (eq. 4-13 and 4-14) [ Step 4 ]
        Re(i) = amplVect(i) * Cos(phaseShift(i))
        Im(i) = amplVect(i) * Sin(phaseShift(i))
    Next

    'Make the Nyquist frequency be zero
    Re(size / 2) = 0
    Im(size / 2) = 0

    'Mirror the data with complex conjugate (eq. 4-11 and 4-12)
    For i = (size / 2) + 1 To size - 1
        Re(i) = Re(size - i)
        Im(i) = -Im(size - i)
    Next

    'Do the inverse FFT [ Step 5, 6, 7 and 8 ]
    Call invFFT(Re, Im, size)

    'Copy the data to return matrix
    For i = 0 To size - 1
        dataVector(i) = Re(i)
    Next
    dataVector(size) = Re(0)

End Sub
```

5 Evaluating fractal lines

There are many different methods to evaluate the fractal parameters. For example, Russ (1994) used five methods: 1) Minowski, 2) Korcak, 3) Hurst, 4) Root-mean-squares (RMS) differences, and 5) Power Spectrum using FFT. That evaluation revealed some substantial differences between the fractal dimension generated and the measured dimension. Candela et al. (2009) evaluated six methods: 1) Root-mean-squares Correlation, 2) Maximum-minimum Height Difference, 3) Height-height Correlation Function, 4) Standard Deviation of the Correlation Function, 5) Fourier Power Spectrum, and 6) Average Wavelet Coefficient Power Spectrum. Their analyses also revealed differences in the values obtained using different methods. Malinverno (1990) showed how the Box Count method can be applied to self-affine traces. This section explains four methods used in the main paper: 1) Analysing the Power Spectrum using Fast Fourier Transform 2) Standard Deviation of the Correlation Function, 3) Korcak Plot of Zero Sets and 4) Box Count.

The four methods are described in the following subsections. Each description begins with an introductory part where some advantages and disadvantages are discussed and then the theory is presented, followed by a worked example. Each method ends with a subsection where an example code of implementation in Visual Basic is provided.

All the worked examples use the same fractal line. The vertices are listed in Table 5-1 and the line is visualised in Figure 5-1.

Table 5-1. Vertices of the fractal line shown in Figure 5-1

X	Y
0.000	0.315
1.000	0.166
2.000	0.260
3.000	0.033
4.000	-0.241
5.000	-0.596
6.000	-0.450
7.000	-0.204
8.000	-0.281
9.000	0.125
10.000	0.105
11.000	-0.069
12.000	-0.041
13.000	0.171
14.000	0.333
15.000	0.443
16.000	0.315

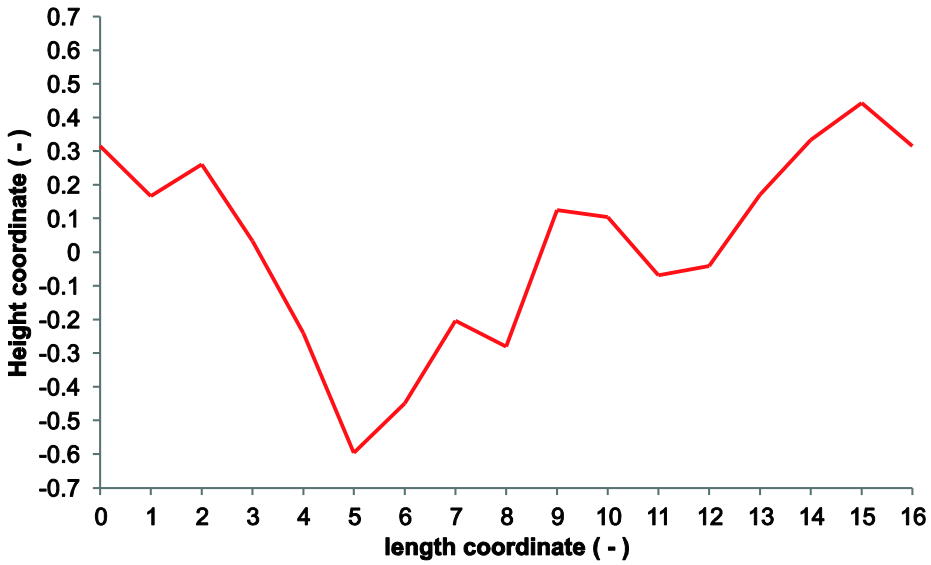

Figure 5-1. *Visualisation of the fractal line used in the worked examples.*

5.1 Analysing the Power Spectrum using Fast Fourier Transform, FFT

The advantage of analysing the power spectrum using the FFT method is that it is fast and can infer both the dimension and the asperity measure of the trace. The method is also good at spotting frequencies that are under-represented, i.e. the highest frequency that the digitalisation of the trace captures. A drawback is that only traces that conform to 2^n vertices can be analysed. If the trace does not conform to the 2^n vertices, one can use a shorter piece of the trace, fill the rest with zeros or use Discrete Fourier Transform, DFT (Smith 1997). However, DFT is a very slow process and not recommended. Another drawback with the FFT method is that the power spectrum is scattered at high frequencies and sensitive to any outlier at the low frequencies. These two shortcomings may make the regression uncertain.

5.1.1 Theory

The method is exactly the inverse of the generation method, IFT of Power Spectrum, see section 4. Hence, if a trace of 2^n vertices is extracted and run through FFT, the results from the transform will be the amplitude and phase shift vectors. The numbers in the amplitude vector are squared to get the power spectrum. The power is a function of the frequency, f, and is proportional to $1/f^{\beta}$. Hence, by plotting the power as a function of frequency in logarithmic space, a linear regression can be performed to evaluate β. The relationship between the slope of the regression line and H is described by:

$$H = \frac{(\beta - 1)}{2} \qquad \text{eq. 5-1}$$

where:
β = slope of the regression, between 1 and 3 for a fractal line
H = Hurst exponent.

The asperity measure $\sigma\delta h(1p)$, i.e. the standard deviation of height differences of adjacent vertices, is also inferred from the power spectrum according to:

$$\sigma\delta h(1p) \approx \frac{2\sqrt{2}}{N} \cdot \sqrt{c_I} \cdot \sqrt{\sum_{f=1}^{N/2-1} \left(f^{-(H+1/2)} \cdot \sin(\pi \cdot f / N) \right)^2} \qquad \text{eq. 5-2}$$

where:
$\sigma\delta h(1p)$ = standard deviation of height differences of adjacent vertices of the fractal line
c_I = intercept of the regression of the power spectrum
f = frequency, i.e. number of waves per trace length
N = number of vertices of the fractal line
H = Hurst exponent.

The derivation of eq. 5-2 can be found in section 4.1.

5.1.2 Worked example

The fractal line shown in Figure 5-1 is used in this example of how to infer the fractal parameters H and $\sigma\delta h(1p)$ using FFT and Power Spectrum. The method has the following steps:

1. Extract the height data coordinates.
2. Sort the data according to bit-reversed order.
3. Do the butterfly calculations.
4. Calculate the power of each length frequency.
5. Do a linear regression of logged powers and length frequencies.
6. Use the slope and intercept to calculate the fractal parameters.

The extracted coordinates are listed in Table 5-1. These data are sorted according to bit-reversed order, following the algorithm in Figure 3-2b, and showed in table Table 5-2.

Table 5-2. Reverse bit order of the 16 vertices and the bit reversal ordered data

S = 0	S = 1	S = 2	S = 3	S = 4	Ordered Data
0	0	0	0	0	0.315
	1	2	4	8	-0.281
		1	2	4	-0.241
		3	6	12	-0.041
			1	2	0.260
			5	10	0.105
			3	6	-0.450
			7	14	0.333
				1	0.166
				9	0.125
				5	-0.596
				13	0.171
				3	0.033
				11	-0.069
				7	-0.204
				15	0.443

The calculations of the twiddle factors for the butterfly calculations are shown in eq. 3-2. The input and output of the four stages of butterfly calculations can be followed in Table 5-3 to Table 5-6. The power (Table 5-6) is calculated as the sum of the squared real and imaginary numbers of the amplitude vector after stage 4.

Table 5-3. The first stage of butterfly calculations

Trace		Stage 1 Twiddle factor W_1^k		Result stage 1	
Re	Im	Re	Im	Re	Im
0.315	0			0.034	0
-0.281	0	1	0	0.596	0
-0.241	0			-0.282	0
-0.041	0	1	0	-0.201	0
0.260	0			0.365	0
0.105	0	1	0	0.156	0
-0.450	0			-0.116	0
0.333	0	1	0	-0.783	0
0.166	0			0.291	0
0.125	0	1	0	0.041	0
-0.596	0			-0.425	0
0.171	0	1	0	-0.767	0
0.033	0			-0.036	0
-0.069	0	1	0	0.102	0
-0.204	0			0.239	0
0.443	0	1	0	-0.647	0

Table 5-4. The second stage of butterfly calculations

Result stage 1		Stage 2 Twiddle factor W_2^k		Result stage 2	
Re	Im	Re	Im	Re	Im
0.034	0			-0.248	0.000
0.596	0			0.596	0.201
-0.282	0	1	0	0.317	0.000
-0.201	0	0	-1	0.596	-0.201
0.365	0			0.248	0.000
0.156	0			0.156	0.783
-0.116	0	1	0	0.481	0.000
-0.783	0	0	-1	0.156	-0.783
0.291	0			-0.135	0.000
0.041	0			0.041	0.767
-0.425	0	1	0	0.716	0.000
-0.767	0	0	-1	0.041	-0.767
-0.036	0			0.202	0.000
0.102	0			0.102	0.647
0.239	0	1	0	-0.275	0.000
-0.647	0	0	-1	0.102	-0.647

Table 5-5. The third stage of butterfly calculations

Result stage 2		Stage 3 Twiddle factor W_4^k		Result stage 3	
Re	Im	Re	Im	Re	Im
-0.248	0.000			0.000	0.000
0.596	0.201			1.260	0.644
0.317	0.000			0.317	-0.481
0.596	-0.201			-0.067	0.243
0.248	0.000	1	0	-0.496	0.000
0.156	0.783	0.707	-0.707	-0.067	-0.243
0.481	0.000	0	-1	0.317	0.481
0.156	-0.783	-0.707	-0.707	1.260	-0.644
-0.135	0.000			0.068	0.000
0.041	0.767			0.570	1.152
0.716	0.000			0.716	0.275
0.041	-0.767			-0.488	-0.381
0.202	0.000	1	0	-0.337	0.000
0.102	0.647	0.707	-0.707	-0.488	0.381
-0.275	0.000	0	-1	0.716	-0.275
0.102	-0.647	-0.707	-0.707	0.570	-1.152

Table 5-6. The fourth stage of butterfly calculations and the resulting power

Result stage 3		Stage 4 Twiddle factor W_8^k		Result stage 4		Values used for inference of H and σδh(1p)	
Re	Im	Re	Im	Re	Im	frequency	Power
0.000	0.000			0.068	0.000	-	*0.005*
1.260	0.644			2.227	1.490	1	7.180
0.317	-0.481			1.017	-0.793	2	1.663
-0.067	0.243			-0.606	0.548	3	0.668
-0.496	0.000			-0.496	0.337	4	0.360
-0.067	-0.243			0.472	0.063	5	0.227
0.317	0.481			-0.384	0.169	6	0.176
1.260	-0.644			0.293	0.203	7	0.127
0.068	0.000	1	0	-0.068	0.000	-	*0.005*
0.570	1.152	0.924	-0.383	0.293	-0.203	-	*0.127*
0.716	0.275	0.707	-0.707	-0.384	-0.169	-	*0.176*
-0.488	-0.381	0.383	-0.924	0.472	-0.063	-	*0.227*
-0.337	0.000	0	-1	-0.496	-0.337	-	*0.360*
-0.488	0.381	-0.383	-0.924	-0.606	-0.548	-	*0.668*
0.716	-0.275	-0.707	-0.707	1.017	0.793	-	*1.663*
0.570	-1.152	-0.924	-0.383	2.227	-1.490	-	*7.180*

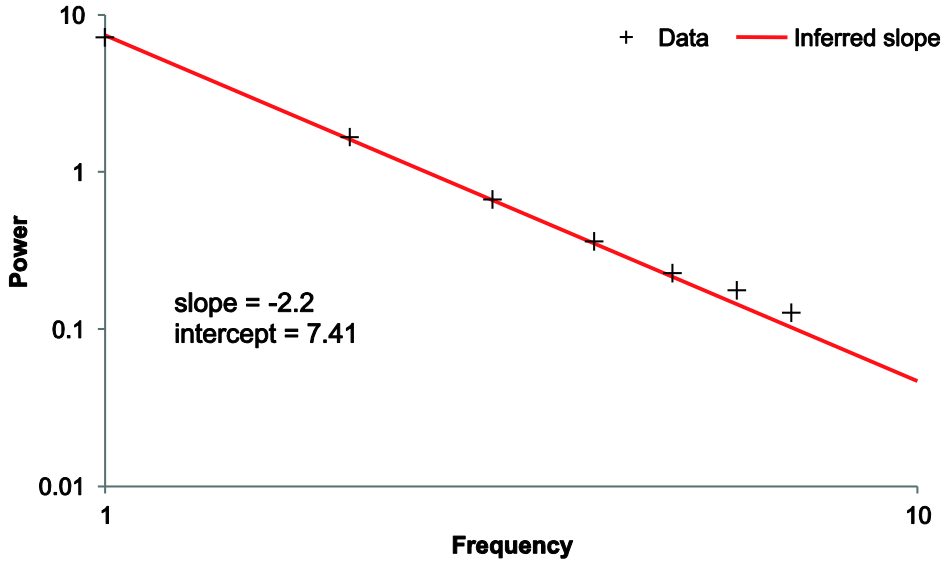

Figure 5-2. *Power as a function of length frequency.*

The power is plotted as a function of the frequency in logarithmic space, Figure 5-2. From this graph, the slope and intercept are inferred to –2.2 and 7.41, respectively. From the inferred β value, 2.2, the Hurst exponent, 0.6, is computed using eq. 5-1.

The asperity measure $\sigma\delta h(1p)$ is calculated using eq. 5-2 (with $N = 16$ and $\sqrt{c} = \sqrt{7.41} = 2.72$) and becomes 0.20.

5.1.3 Code

```
'=============================================================================
'===                          FFT evaluation                               ===
'=============================================================================
'===  Routine calculating the fractal dimension using Fast Fourier        ===
'===  Transform. The number of data has to conform to 2^n, and hence      ===
'===  the last value are omitted from the 2^n + 1 data points. The        ===
'===  value at index 0 is the same as the value at index nofData, since   ===
'===  the analysis is done on a simple detrending of the data             ===
'=============================================================================
'====                                                                     ====
'===  Input:                                                              ===
'===  yVect(0 to nofData) = the height data vector in arithmetic space    ===
'===  nofData             = the number of vertices of the trace           ===
'====                                                                     ====
'===  Output:                                                             ===
'===  H            = the Hurst exponent                                   ===
'===  sdh1p        = standard deviation of height differences between     ===
'===                 adjacent vertices, σδh(1p)                          ===
'===  errorMessage = string containing possible errorMessage              ===
'====                                                                     ====
'=============================================================================
'===  Written by Martin Stigsson, February 26 2016                        ===
'===  Please, report errors or improvements to: martin.stigsson@skb.se    ===
'=============================================================================
Sub FFT evaluation(ByRef yVect() As Double,
                   ByRef nofData As Integer, _
                   ByRef H As Double, _
                   ByRef sdh1p As Double, _
                   ByRef errorMessage As String)

    'Define variables
    Dim k As Integer         'counter
    Dim minBin As Integer 'index for the lowest bin to regard i.e 1
    Dim maxBin As Integer 'index for the highest bin to regard i.e. N/2-1

    Dim slope As Double        'Slope of the regression line
    Dim intersection As Double 'Intersection of the regression line
    Dim sum2 As Double         'Sum of squared values

    Dim ReVect(nofData) As Double          'real numbers
    Dim ImVect(nofData) As Double          'imaginary numbers
```

```
    Dim powVect(nofData / 2 - 1) As Double 'power in each bin
    Dim xVect(nofData / 2 - 1) As Double   'length frequencies


    'set up the Arrays to be used in the FFT-analysis [ Step 1 ]
    For i = 0 To nofData - 1
        ReVect(i) = yVect(i)
        ImVect(i) = 0
    Next

    'Do the Fast Fourier transform [ Step 2 and 3 ]
    Call FFT(ReVect, ImVect, nofData)

    'Make the power vector and frequency vector into log space [ Step 4 ]
    For i = 1 To nofData / 2 - 1
        powVect(i) = Log10((ReVect(i) ^ 2 + ImVect(i) ^ 2))
        xVect(i) = Log10(i)
    Next

    '----  Calculate the Hurst parameter  ----
    'The first bin is only dependent on the average value of the curve, and
    'hence arbitrary, if the average is 0, then the power will also be zero
    minBin = 1

    'N/2 is an arbitrary value dependent on the phase shift, and the N/2-1
    'following values are just mirrors of the N/2-1 proceeding values
    maxBin = nofData / 2 - 1

    'Do a linear regression in log space [ Step 5 ]
    Call regression analysis of line(xVect, powVect, minBin, maxBin,
                                     slope, intersection, errorMessage)

    'Calculate H using eq. 5-1 [ Step 6 ]
    H = (-slope - 1) / 2


    '----  Calculate σδh(1p)  ----
    'Initiate the sum varable
    sum2 = 0

    'Make the frequency vector into arithmetic space
    For i = 1 To nofData / 2 - 1
        xVect(i) = i
    Next

    'Make the intesection into arithmetic space
    intersection = 10 ^ intersection

    'Do the summation in the denominator in eq. 5-2
    For k = 1 To nofData / 2 - 1
        sum2 = sum2 +
               (xVect(k) ^ (-(H + 0.5)) * Sin(PI * xVect(k) / nofData)) ^ 2
    Next k

    'Calculate σδh(1p) according to eq. 5-2 [ Step 6 ]
    sdh1p = 2 * 2 ^ 0.5 / nofData * intersection ^ 0.5 * sum2 ^ 0.5

End Sub
```

5.2 Standard Deviation of the Correlation Function, RMS-COR

The Standard Deviation of the Correlation Function method is intuitive and easy to implement. The method is capable of estimating both the Hurst exponent and asperity measure and it can use an arbitrary amount of equally spaced vertices. However, a drawback is that it is sensitive to the finite length of the traces. The shorter the trace, the larger the effect.

5.2.1 Theory

The standard deviation of the correlation function makes use of the relationship between the standard deviation of height differences at different length intervals. A self-affine line needs to be scaled by different amounts in the two directions to appear similar, see Figure 2-1. Hence, if the abscissa is scaled by a factor λ, the ordinate needs to be scaled by λ^H. This implies that the standard deviation of height differences between vertices at different distances will scale in the same way, i.e.:

$$\sigma\delta h(\Delta v) = c_\sigma \cdot \Delta v^H \qquad \text{eq. 5-3}$$

where:
c_σ = standard deviation of height differences of adjacent vertices
Δv = distance between vertices
H = Hurst exponent.

The standard deviation can be calculated according to e.g. Johnson et al. (2011) as:

$$\sigma(x) = \sqrt{E(x^2) - E(x)^2} \qquad \text{eq. 5-4}$$

Hence, the standard deviation of height differences of vertices Δv apart can be calculated according to:

$$\sigma\delta h(\Delta v) = \sqrt{\frac{\sum_{v=0}^{N-\Delta v}\left(h(v+\Delta v) - h(v)\right)^2}{N+1-\Delta v} - \left(\frac{\sum_{v=0}^{N-\Delta v} h(v+\Delta v) - h(v)}{N+1-\Delta v}\right)^2} \qquad \text{eq. 5-5}$$

where:
$\sigma\delta h(\Delta v)$= standard deviation of height differences of vertices Δv apart
Δv = number of vertices between the height values
$h(v)$ = height value at vertex v
N = number of vertices of the trace.

For a self-affine fractal, plotting $\sigma\delta h(\Delta v)$ as a function of Δv will render a straight line in logarithmic space. The slope of the regression equals H and the intercept is $\sigma\delta h(1p)$. However, as Δv approaches N, $\sigma\delta h(\Delta v)$will not increase linearly in logarithmic space but rather decrease, due to the finite length effects. It is recommended to use only $\Delta v < 0.2 \cdot N$ (Malinverno 1990) to avoid these effects.

5.2.2 Worked example

The fractal line shown in Figure 5-1 is used in this example of how to infer the fractal parameters H and $\sigma\delta h(1p)$ using the standard deviation of the correlation function method. The method has the following steps:

1. Extract the height data coordinates.
2. Calculate the standard deviation of height differences using different length intervals.
3. Do a linear regression of logged height differences and length intervals.
4. Use the slope and intercept to calculate the fractal parameters.

The extracted coordinates are listed in Table 5-1 and recapitulated in Table 5-7. These data are used to calculate the height differences of adjacent vertices, two vertices apart, four vertices apart and eight vertices apart, see Figure 5-3 and Table 5-7. These numbers are used to calculate the population standard deviation according to eq. 5-3.

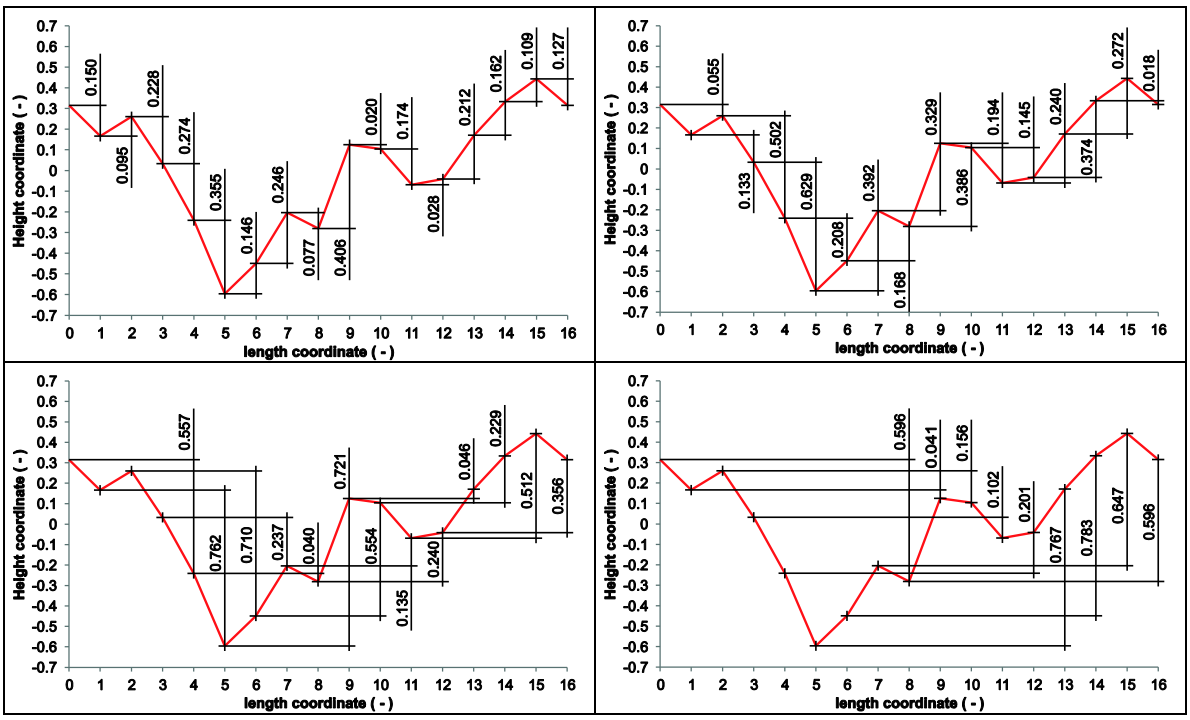

Figure 5-3. *Height differences using different length intervals.*

Table 5-7. The calculation of standard deviation of height differences of vertices different lengths apart, together with the population standard deviation

vertex	Height	Δv = 1	Δv = 2	Δv = 4	Δv = 8
0	0.315				
1	0.166	-0.150			
2	0.260	0.095	-0.055		
3	0.033	-0.228	-0.133		
4	-0.241	-0.274	-0.502	-0.557	
5	-0.596	-0.355	-0.629	-0.762	
6	-0.450	0.146	-0.208	-0.710	
7	-0.204	0.246	0.392	-0.237	
8	-0.281	-0.077	0.168	-0.040	-0.596
9	0.125	0.406	0.329	0.721	-0.041
10	0.105	-0.020	0.386	0.554	-0.156
11	-0.069	-0.174	-0.194	0.135	-0.102
12	-0.041	0.028	-0.145	0.240	0.201
13	0.171	0.212	0.240	0.046	0.767
14	0.333	0.162	0.374	0.229	0.783
15	0.443	0.109	0.272	0.512	0.647
16	0.315	-0.127	-0.018	0.356	0.596
Std dev		**0.204**	**0.314**	**0.462**	**0.461**

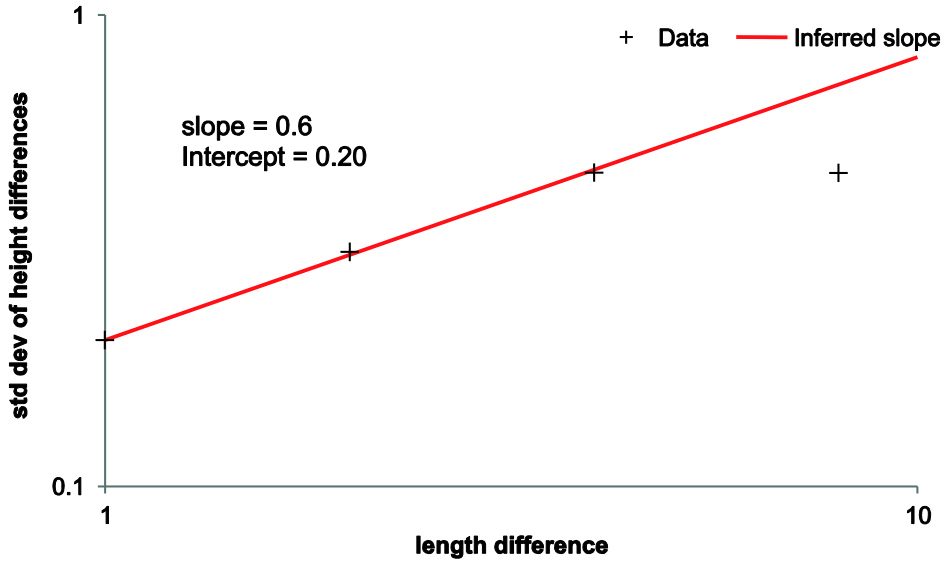

Figure 5-4. *Height differences as a function of length interval.*

The standard deviations of the height differences are plotted as a function of the length interval in logarithmic space, Figure 5-4. The standard deviation for the length interval equal to eight vertices is affected by the finite length and is, hence, not a part inferring the fractal parameters. The graph is used to infer the slope, 0.6, and the intercept, 0.20, which equals *H* and *σδh(1p)*, respectively.

5.2.3 Code

```
'=============================================================================
'===                        RMS_COR_evaluation                             ===
'=============================================================================
'===  Routine calculating the slope and amplitude using RMS COR           ===
'===  described in Candela et al., 2009, Characterisation of Fault        ===
'===  Roughness at Various Scales: Implications of Three-Dimensional      ===
'===  High Resolution Topography Measurements,                            ===
'===  Pure and Applied Geophysics, vol 166 page 1823                      ===
'=============================================================================
'====                                                                     ====
'===  Input:                                                              ===
'===  dataVector   = the vector containing the height data along the      ===
'===                 trace, starting at position 0, and end at position  ===
'===                 N                                                   ===
'===  nofData      = the number of data in dataVector                     ===
'===  minBin       = index for the lowest bin to regard                   ===
'===  maxBin       = index for the highest bin to regard                  ===
'====                                                                    ====
'===  Output:                                                             ===
'===  sdh1p        = standard deviation of height differences between     ===
'===                 adjacent vertices, σδh(1p)                          ===
'===  H            = the Hurst exponent                                   ===
'===  errorMessage = string containing possible errorMessage              ===
'====                                                                    ====
'=============================================================================
'===  Written by Martin Stigsson, February 12 2014                        ===
'===  Please, report errors or improvements to: martin.stigsson@skb.se    ===
'=============================================================================
Sub RMScor_evaluation(ByRef dataVector() As Double, _
                      ByVal nofData As Integer, _
                      ByVal minBin As Integer, _
                      ByVal maxBin As Integer, _
                      ByRef sdh1p As Double, _
                      ByRef H As Double, _
                      ByRef errorMessage As String)

    'Declaration of variables
    Dim pop As Boolean   'Flag for calculation of population std dev, TRUE,
    '                     or sample std dev, FALSE

    Dim i, j As Integer  'Counters

    Dim diff As Double   'difference between two numbers
    Dim xSum As Double   'sum of values
```

```
        Dim x2sum As Double  'sum of squared values
        Dim mean As Double   'mean value, not used but needed for the function
        '                     that calculates mean and standard deviation
        Dim stddev As Double 'standard deviation of height differences

        Dim steps(maxBin) As Double  'step sizes that will be used to calculate
        '                            the standard deviation between. It should
        '                            be integers, but need to be double to be
        '                            used in the regression analysis
        Dim Stdevs(maxBin) As Double 'vector containing the standard deviation
        '                            of height differences of vertices steps(x)
        '                            apart

        'initiate the flag for standard deviation calculations
        pop = True

        'generate Step sizes [1 2 4 8 16 32 64 128 ... etc]
        steps(1) = 1
        For i = 2 To maxBin
            steps(i) = steps(i - 1) * 2
        Next

        'Calculate the standard deviation of height differences for the
        'different step sizes, eq. 5-4  [ Step 2 ]
        For i = 1 To maxBin

            'initiate sums
            xSum = 0
            x2sum = 0

            'Calculate sum and squared sum to be used for calculation of
            'standard deviation
            For j = 0 To nofData - steps(i)
                diff = dataVector(j + steps(i)) - dataVector(j)
                xSum = xSum + diff
                x2sum = x2sum + diff * diff
            Next

            'Calculate standard deviation
            mean and stdev from sums(xSum, x2sum, j, pop,
                                     mean, stddev, errorMessage)

            'Save the current step size's standard deviation
            Stdevs(i) = stddev

        Next

        'transform data to log space
        For i = 1 To maxBin
            steps(i) = Log10(steps(i))
            Stdevs(i) = Log10(Stdevs(i))
        Next

        'Calculate the slope and intercept of the data in log space [ Step 3 ]
        Call regression_analysis_of_line(steps, Stdevs, minBin, maxBin, _
                                         H, sdh1p, errorMessage)

        'Convert intersection in log space back to arithmetic space [ Step 4 ]
        sdh1p = 10 ^ sdh1p

        'H = slope; hence, no transformations needed

errorTrap:

    End Sub
```

5.3 Korcak Plot of Zero Sets, Zero set/Korcak

The Korcak Plot of Zero Sets method is intuitive and easy to implement. The method can use an arbitrary amount of arbitrarily spaced vertices, which makes it very flexible. However, it can only estimate the Hurst exponent and not any asperity measure. Another drawback is that it is very sensitive to the length of the traces; the larger *H*, the more sensitive.

5.3.1 Theory

The Korcak Plot of Zero Sets method makes use of the fact that the crossings between a line parallel to the abscissa, the length axis, and a fractal line will produce a Cantor dust. The method is called the zero set of the fractal line since, in its original implementation, it only recorded the lengths between the points where the fractal line crossed the value zero. The complementary cumulative number of lengths larger than a specific length is known as the Korcak relationship and is expressed as (e.g. Russ 1994):

$$N\left(L \geq l\right) \propto l^{s} \qquad \text{eq. 5-6}$$

where:
$N(L \geq l)$ = number of lengths, L, exceeding the length l
l = studied length
s = slope of the regression line in logarithmic space.

The relationship between the slope and the Hurst exponent is:

$$H = s + 1 \qquad \text{eq. 5-7}$$

where:
H = Hurst exponent
s = slope of the line in logarithmic space.

A fracture trace is usually too short to give reliable results by only sampling intersections between the fractal line and the zero line. Instead, to get a larger number of length segments, it is possible to introduce multiple straight lines parallel with the abscissa and measure all lengths between the intersections of the fractal line and the multiple horizontal lines.

Due to the finite length of the trace, the method will constantly underestimate the number of long intervals. This implies that the cumulative number of lengths exceeding a specific length is always underestimated and will bias all number of lengths shorter than this specific length. Hence, the data will not plot as a straight line in logarithmic space, but drop towards a steeper slope at the right end. This can, in some sense, be overcome by ignoring the large length intervals and only using the smaller ones when evaluating the slope. However, even if the rightmost data points are disregarded, the Hurst exponent will always be underestimated.

5.3.2 Worked example

The fractal line shown in Figure 5-1 is used in this example of how to infer the Hurst exponent using the Korcak Plot of Zero Sets method. The method has the following steps:

1. Extract the height data coordinates.
2. Find the maximum and minimum height values.
3. Draw an arbitrary amount of lines parallel to the abscissa.
4. Calculate the lengths between the intersections of the horizontal line and the trace.
5. Sort the data and calculate the number of lengths longer than a specific length in logarithmic space.
6. Do a linear regression of number of lengths vs the specific length.
7. Use the slope to calculate the Hurst exponent.

The extracted coordinates are listed in Table 5-1 and recapitulated in Table 5-8. From the data, the maximum and minimum height is 0.443 and –0.596, respectively. Nine horizontal lines are drawn at heights listed in Table 5-8 and shown in Figure 5-5.

Table 5-8. The vertices, together with the data on the lines parallel to the abscissa

vertex	height
0	0.315
1	0.166
2	0.260
3	0.033
4	-0.241
5	-0.596
6	-0.450
7	-0.204
8	-0.281
9	0.125
10	0.105
11	-0.069
12	-0.041
13	0.171
14	0.333
15	0.443
16	0.315

Statistic	
Max	0.443
Min	-0.596
Diff	1.039
# lines	9
Δh	0.115

Zero lines	
Line #	Height
1	0.385
2	0.269
3	0.154
4	0.039
5	-0.077
6	-0.192
7	-0.308
8	-0.423
9	-0.538

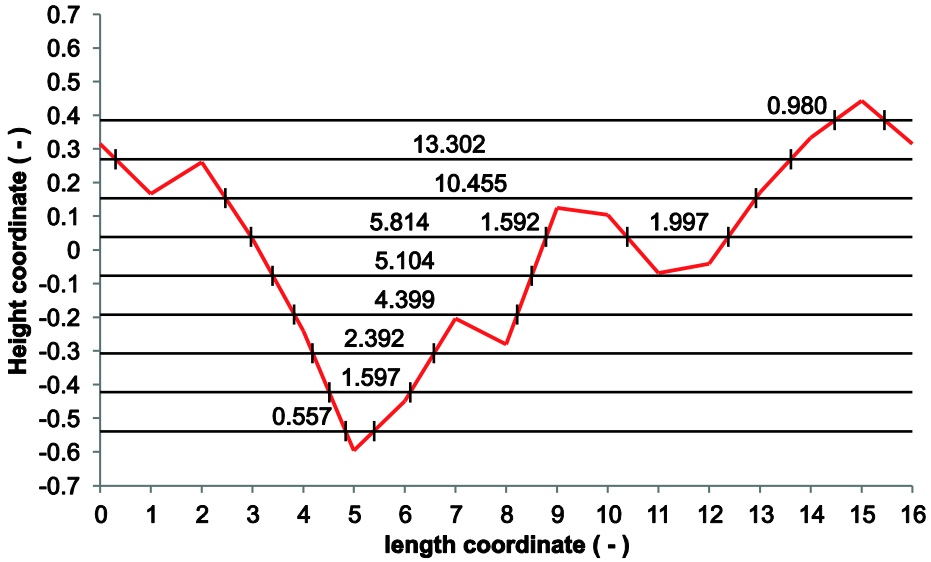

Figure 5-5. *Lengths of the zero sets.*

The lengths between the intersections of the nine horizontal lines and the fractal line are shown in Figure 5-5 and listed in Table 5-9. The lengths are sorted in ascending order and the number of lengths exceeding a given value is calculated, Table 5-9.

Table 5-9. The lengths between crossings, together with complementary cumulative number of lengths

Line #	Length
1	0.980
2	13.302
3	10.455
4	5.814
4	1.592
4	1.997
5	5.104
6	4.399
7	2.392
8	1.597
9	0.557

Sorted l
0.557
0.980
1.592
1.597
1.997
2.392
4.399
5.104
5.814
10.455
13.302

l	N(L≥l)
1	9
2	6
4	5
8	2

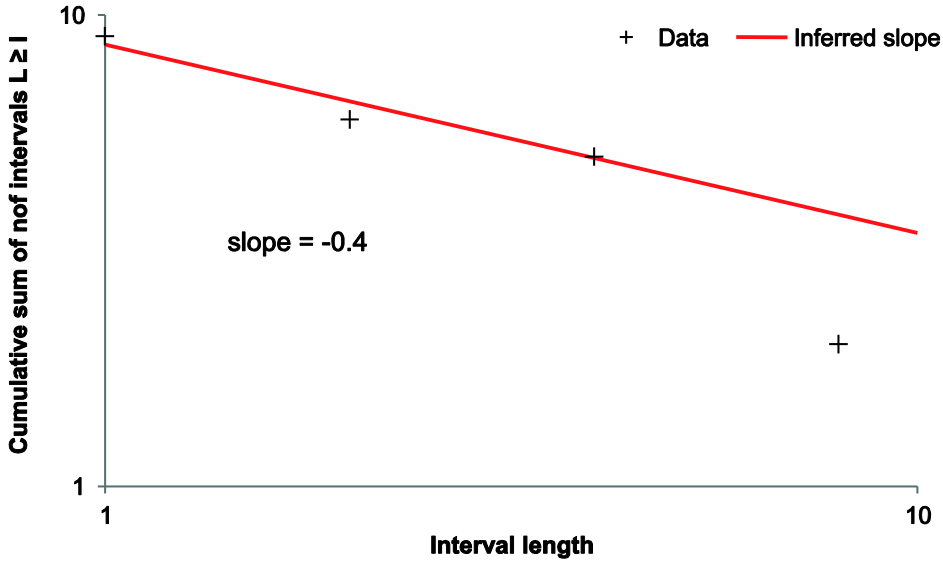

Figure 5-6. *Korcak plot of N(L≥l).*

The number of lengths exceeding a specific length is plotted in logarithmic space as a function of the specific length, Figure 5-6. The number of lengths >8 is affected by the finite length and is, hence, not a part inferring the slope. The slope is –0.4, which, according to eq. 5-7, results in a Hurst exponent of 0.6.

5.3.3 Code

```
'=============================================================================
'===                  zerosets_evaluation_variable_dx                      ===
'=============================================================================
'===  Routine calculating the slope using Korcak plot of zero sets as      ===
'===  described in "Fractal Surfaces" by John C. Russ, 1994,               ===
'===  ISBN 0-306-44702-9, pp 13-14.                                        ===
'===  The method may use arbitrary Δx btewwen vertices                     ===
'=============================================================================
'====                                                                      ====
'===  Input:                                                               ===
'===  lVect        = the length coordinates using indeces 0 to N-1         ===
'===  hVect        = the height coordinates using indeces 0 to N-1         ===
'===  nofData      = the number of data in hVect                           ===
'===  nofLines     = number of lines between max and min height that       ===
'===                 will be used for collecting lengths between          ===
'===                 intersections                                        ===
'===  minBin       = index of the lowest bin to regard                     ===
'===  maxBin       = index of the highest bin to regard                    ===
'====                                                                     ====
'===  Output:                                                              ===
'===  H            = the slope of the curve of standard deviations         ===
'===                 using all bins                                       ===
'===  errorMessage = string containing possible errorMessage               ===
'====                                                                     ====
'=============================================================================
'===  Written by Martin Stigsson, April 25 2016                            ===
'===  Please, report errors or improvements to: martin.stigsson@skb.se     ===
'=============================================================================
Sub zerosets_evaluation_variable_dx(ByRef lVect() As Double, _
                                    ByRef hVect() As Double, _
                                    ByVal nofData As Integer, _
                                    ByVal nofLines As Integer, _
                                    ByVal minBin As Integer, _
                                    ByVal maxBin As Integer, _
                                    ByRef H As Double, _
                                    ByRef errorMessage As String)

    Dim firstIsAlreadyHit As Boolean 'flag to keep track of the first
    '                                 intersection between the current zero
    '                                 line and the fractal line

    Dim i, j As Integer           'counters
    Dim lengthInterval As Integer 'truncated length to find correct bin
    Dim nofBinData As Integer     'number of elements in the array bindata
```

```
Dim maxZ As Double          'maximum value in hVect
Dim minZ As Double          'minimum value in hVect
Dim lineDist As Double      'vertical distance between the zero lines
Dim currLevel As Double     'height value for the current zero line
Dim currCoord As Double     'current x-coordinate where the zero line at
'                            currLevel intersects the fractal line
Dim lastCoord As Double     'last x-coordinate where an intersection
'                            between the current zero line and the
'                            fractral line occured
Dim length As Double        'length between intersections
Dim slope As Double         'slope of regression of logged values
Dim intersection As Double 'dummy; Needed for the regression sub

Dim bindata(CInt(lVect(nofData) - lVect(0)) + 1) As Integer
'                              Containing the number of lengths in each
'                              lengthe interval
Dim xData(maxBin) As Double  'Complementary cumulative number of lengths
Dim yData(maxBin) As Double  'lengths

'Calculate the number of possible bins of cumulative lengths
nofBinData = CInt(lVect(nofData) - lVect(0)) + 1

'initiate variables
minZ = 10000000000.0
maxZ = -10000000000.0
For i = 0 To nofBinData
    bindata(i) = 0
Next

'Find maximum an minimum values of the line in the hVect [ Step 2 ]
For i = 0 To nofData - 1
    If hVect(i) < minZ Then minZ = hVect(i)
    If hVect(i) > maxZ Then maxZ = hVect(i)
Next

'Calculate the distance between the investigated zero lines [ Step 3 ]
lineDist = (maxZ - minZ) / nofLines

'initiate the level of the investigated zero line, (half of the line
'distance above the maximum value
currLevel = maxZ + lineDist / 2

'run through all investigation lines. [ Step 4 ]
For i = 1 To nofLines

    'calculate the level of the current investigation line
    currLevel = currLevel - lineDist

    'initiate the boolean that flag the for the first time the fractal
    'line intersects the current zero line
    firstIsAlreadyHit = False

    'Go through all, but the first, points in the vector containing
    'the z values
    For j = 1 To nofData

        'check if the 2 investigated verices of the fractal line are on
        'different sides of the current investigated zero line. If so
        ' calculate the intersection coordinate
        If (hVect(j - 1) - currLevel) * (hVect(j) - currLevel) < 0 Then

            'if the first intesection is already hit then calculate the
            'next intersection and save the length, otherwise calculate
            'the coordinate of the first intersection
            If firstIsAlreadyHit Then

                'Interpolate the current intesection
                currCoord = (lVect(j) - lVect(j - 1)) / _
                            (hVect(j) - hVect(j - 1)) *
                            (currLevel - hVect(j - 1)) + lVect(j - 1)

                'Calculate the length between the 2 intersections
                length = currCoord - lastCoord

                'calculate which bin that should be added 1 hit
                'The output is supposed to be the Probability that the
                'length is larger than a specified length, hence lengths
                'in the interval 0 to 1 unit shall be stored in bin 0
                lengthInterval = Truncate(length)
```

```
                    'Add 1 hit to the correct bin
                    bindata(lengthInterval) = bindata(lengthInterval) + 1

                    'update the current coordinate to be the last known
                    'intersection coordinate
                    lastCoord = currCoord

                Else
                    'Change the boolean
                    firstIsAlreadyHit = True

                    'Interpolate the current intesection and save it as the
                    'last known intersection coordinate
                    lastCoord = (lVect(j) - lVect(j - 1)) /
                                (hVect(j) - hVect(j - 1)) * _
                                (currLevel - hVect(j - 1)) + lVect(j - 1)
                End If

            End If
        Next

    Next

    'Make bin data be cumulative [ Step 5 ]
    For i = nofBinData To 1 Step -1
        bindata(i - 1) = bindata(i) + bindata(i - 1)
    Next

    'generate Step sizes like [  1   2   4   8  16  32  64 ...],
    '                         [  2   4   8  16  32  64 128 ...],
    '                         [  4   8  16  32  64 128 256 ...],
    '                         [  8  16  32  64 128 256 512 ...] etc
    'The way to find the apropriate distance is to take the distance between
    '2 points of the trace, and divide it by the number of bins between
    'them. Therafter the value is converted into base 2 log-value, which is
    'rounded to nearest integer. Raising 2 to this number will give the
    'minimum distance in the 2^n form
    xData(1) = 2 ^ (CInt(Log((lVect(nofData) - lVect(0)) / _
                                              nofData) / Log(2)))

    'Create the other length values from the first
    For i = 2 To maxBin
        xData(i) = xData(i - 1) * 2
    Next

    'Save the cumulative values in log space that corresponds to the lengths
    'saved in xData, which should be used in the regression analysis
    For i = 1 To maxBin
        yData(i) = Log10(bindata(xData(i)))
    Next

    'Transform the length data to log space
    For i = 1 To maxBin
        xData(i) = Log10(xData(i))
    Next

    'Calculate the slope from linear regression analysis [ Step 6 ]
    Call regression analysis of line(xData, yData, minBin, maxBin, slope, _
                                     intersection, errorMessage)

    'Calculate the Hurst exponent according to eq. 5-6
    H = slope + 1

errorTrap:

End Sub
```

5.4 Box Count

The Box Count method is visually intuitive and easy to implement, though it is time- or memory-consuming depending on the implementation. The method can use an arbitrary amount of arbitrarily spaced vertices, which makes it very flexible. However, it can only estimate the Hurst exponent and not any asperity measure. It is also sensitive to the number of vertices of the traces.

5.4.1 Theory

The Box Count method uses the relationship between the number of boxes visited by the fractal line as a function of the number of divisions of the box. The method can be seen as a form of divider method, and will give incorrect results if erroneously implemented, as in e.g. Li and Huang (2015), and Chen et al. (2012) among others, see section 7. However, the implementation described in Malinverno (1990) will work on self-affine lines.

For a self-affine fractal line there is no such thing as a square or circle (Mandelbrot 1985). Instead the Box Count method has to find the minima and maxima of the trace length and asperities creating the "box". Using the maximum and minimum in each direction, the box is divided at the middle to create smaller sub-boxes. For each division, the number of boxes that contain any part of the trace is recorded. The relationship between the number of divisions and the number of boxes visited by the trace can be expressed as:

$$N(n) \propto n^{s} \qquad \text{eq. 5-8}$$

where:
N(n) = number of boxes visited by the fractal line
n = number of boxes along the axes
s = slope of the regression line in logarithmic space.

The relationship between the slope and the Hurst exponent is:

$$H = 2 - s \qquad \text{eq. 5-9}$$

where:
H = Hurst exponent
s = slope of the line in logarithmic space.

Plotting eq. 5-8 in logarithmic space will render a straight line where the slope equals 2 – *H*. The inference of the Hurst exponent is, hence, done using linear regression in logarithmic space.

The method suffers from edge effects in both the lower end and upper end. In the lower end, the first division will always result in three or four boxes being visited by the fractal line, which is a very low resolution, but might influence the inference of the Hurst exponent. In the upper end, the lack of resolution of the trace will render an underestimate of boxes visited by the trace and hence tend to overestimate the Hurst exponent. Hence, to avoid those edge effects the results from the first division should be omitted, together with the results from divisions that result in box sizes close to the resolution of the trace, especially for traces with low Hurst exponent.

5.4.2 Worked example

The fractal line shown in Figure 5-1 is used in this example of how to infer the Hurst exponent using the Box Count method. The method has the following steps:

1. Extract the height data coordinates.
2. Find the maximum and minimum length and height values to construct the "box".
3. Divide the box in the middle of each direction into four equally large sub-boxes and count the number of sub-boxes visited by the trace.
4. Continue to divide the box into smaller and smaller sub-boxes and count the number of sub-boxes visited by the trace.
5. Do a linear regression in logarithmic space of number of sub-boxes visited by the line as a function of number of sub-boxes along one side of the box.
6. Use the slope to calculate the Hurst exponent.

The extracted coordinates are listed in Table 5-1 and recapitulated in Table 5-10. From the data, the maximum and minimum vertices are 0 and 16, whilst the maximum and minimum height is 0.443 and –0.596, respectively. Hence the unity box is 16 width units wide and 1.039 height units high. The four divisions of the unity box are shown in Figure 5-7.

Table 5-10. The vertices together with the lateral and vertical extensions

vertex	height
0	0.315
1	0.166
2	0.260
3	0.033
4	-0.241
5	-0.596
6	-0.450
7	-0.204
8	-0.281
9	0.125
10	0.105
11	-0.069
12	-0.041
13	0.171
14	0.333
15	0.443
16	0.315

Extension	
Max vertex	16
Min vertex	0
Max height	0.443
Min height	-0.596

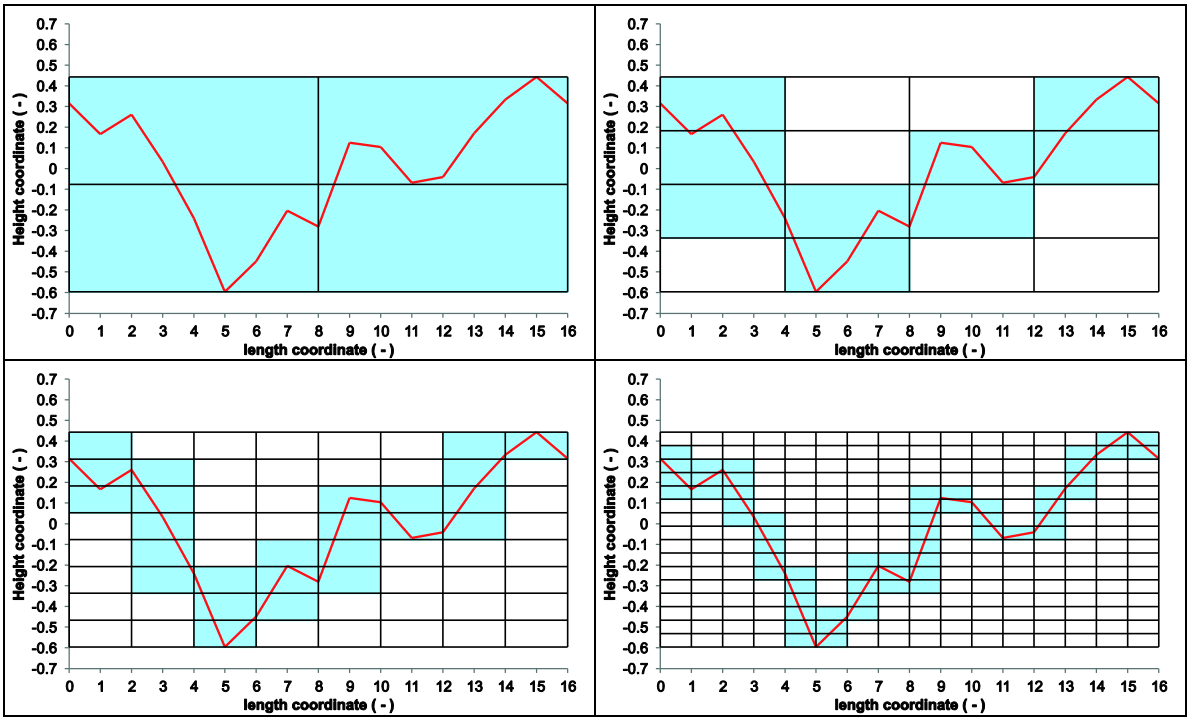

Figure 5-7. *Number of boxes visited by the trace (shaded) using different numbers of divisions. Upper left) One division, 4/4 boxes visited. Upper right) Two divisions, 9/16 boxes visited. Lower left) Three divisions, 25/64 boxes visited. Lower right) Four divisions, 60/256 boxes visited.*

The number of boxes visited for each stage of division is listed in Table 5-11 and plotted in Figure 5-8. From these data, the slope is inferred to be 1.4 and, hence, according to eq. 5-9, the Hurst exponent is 0.6.

Table 5-11. Number of divisions, d, number of boxes along the axes, n, and number of boxes visited by the fractal line, N(n)

d	n	N(n)
1	2	4
2	4	9
3	8	25
4	16	60

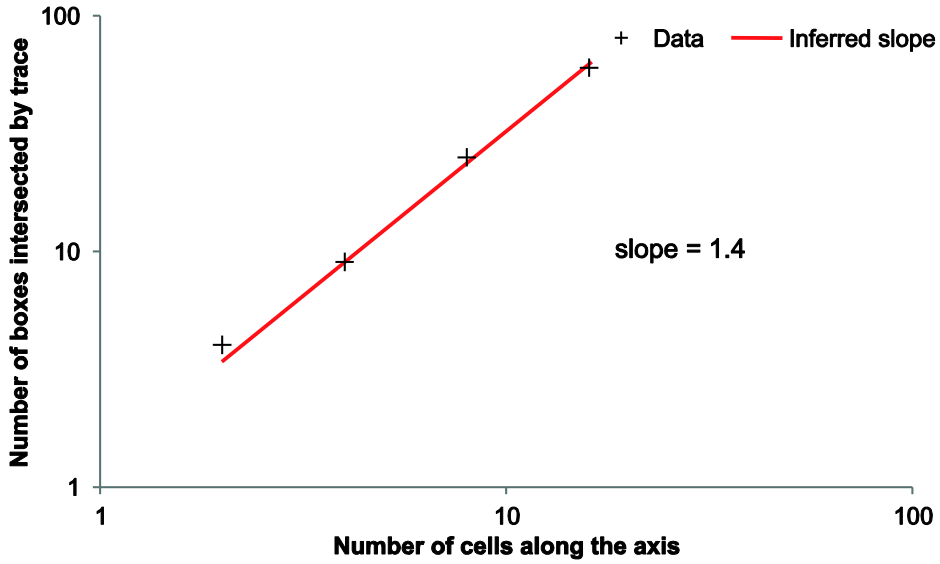

Figure 5-8. *Number of boxes visited by the trace using different numbers of boxes along each side of the main box.*

5.4.3 Code

```
'===============================================================================
'===                  BoxCounting evaluation variable dx                     ===
'===============================================================================
'===  Routine calculating the fractal dimension of a line using box          ===
'===  counting method described in:                                          ===
'===  Malinverno 1990, A simple method to estimate the fractal               ===
'===  dimension of a self-affine series. Geophys Res Lett 1990;              ===
'===  17:1953-6                                                              ===
'===============================================================================
'===                                                                         ===
'===     Point is considered as inside if it     +----+                      ===
'===     is on the left or bottom boundary       •----+  dot is inside     ===
'===                                                                         ===
'===     Point is considered as outside if       +---•                       ===
'===     it is on the right or top boundary      +----+  dot is outside    ===
'===                                                                         ===
'===                                                                         ===
'===     Division level 1; r = 0.5, 1/r = 2                                  ===
'===     +---------------+------•--------+                                   ===
'===     |               |      • • •     |                                   ===
'===     |               |  • •    • •    |                                   ===
'===     |               | • •         •  •                                   ===
'===     +---------------+•------------•-•+                                   ===
'===     |    ••         •              • |                                   ===
'===     • • •  •••      •|              |                                   ===
'===     |• •       •• •• |              |  3 squares intersected with       ===
'===     +------------•---+--------------+  lines between vertices           ===
'===                                                                         ===
'===                                                                         ===
'===     Division level 2; r = 0.25, 1/r = 4                                 ===
'===     +--------+-------+------•+-------+                                  ===
'===     |        |       |      • • •     |                                  ===
'===     +--------+-------+---•-•--+•-•-----|                                  ===
'===     |        |       | • •    |   •   •                                  ===
'===     +--------+-------+•-------+----•-•+                                  ===
'===     |     •• |       •        |      • |                                  ===
'===     •-•-•---•••------•+-------+--------|                                  ===
'===     |• •     | •• •• |        |        |  10 squares intersected with     ===
'===     +--------+---•---+--------+--------+  lines between vertices          ===
'===                                                                         ===
```

```
'===                                                                          ===
'===     Division level 3; r = 0.125, 1/r = 8                                 ===
'===     +---+---+---+---+---+--•+---+---+                                    ===
'===     +---+---|---+---|---+-•-•-•-+---|                                    ===
'===     +---+---+---+---+---+--•+--+•-•+---|                                ===
'===     |---+---|---+---|-•-•---|---•---•                                    ===
'===     +---+---+---+---+•--+---+---+•-•+                                    ===
'===     |---+••-|---+---•---+---|---+-•-|                                    ===
'===     •-•-•--•••------•+---+---+---+---|                                   ===
'===     |•--•+---|-••+••-|---+---|---+---|  20 squares intersected with      ===
'===     +---+---+---•---+---+---+---+---+  lines between vertices           ===
'===                                                                          ===
'===                                                                          ===
'===      N                                                                   ===
'===       |                                              y=1.07x^1.47        ===
'===    32 +                                          ,·'                     ===
'===       |                                      ,·'                         ===
'===    16 +                      (10)        ,·'  X          D = 1.47        ===
'===       |                        X   ,·'     (20)         H = 2 - D       ===
'===     8 +                       ,·'                       H = 0.53        ===
'===       |                  ,·'                                             ===
'===     4 +           (3) ,·'                                                ===
'===       |            ,X'                                                   ===
'===     2 +       ,·'                                                        ===
'===       |   ,·'                                                           ===
'===     1 X-----------+-----------+-----------+------> 1/r                   ===
'===       1           2           4           8                             ===
'===                                                                          ===
'===                                                                          ===
'===       • = Vertices                                                       ===
'===     -+---+---+---+---+---+---+---+-                                      ===
'===      |   |   •   |   |   • • |   |                                       ===
'===     -+---+---+---+---+-•-+---•-•-+-                                      ===
'===      |   |   |   |   |   |   |   |                                       ===
'===     -+---+---+---+---+---+---+---+-                                      ===
'===      | • |   | • •   •   |   |   •                                       ===
'===     -•---+-•-+---+-•-+---+---+---+-                                      ===
'===      |   •   |   |   |   |   |   |                                       ===
'===     -+---+---+---+---+---+---+---+-                                      ===
'===                                                                          ===
'===       O = Boxes that are hit with lines between vertices                 ===
'===     -+---+---+---+---+---+---+---+-                                      ===
'===      |   | O | O |   | O | O | O |                                       ===
'===     -+---+---+---+---+---+---+---+-                                      ===
'===      |   | O | O |   | O |   | O |                                       ===
'===     -+---+---+---+---+---+---+---+-                                      ===
'===      | O | O | O | O | O |   | O | C                                     ===
'===     -+---+---+---+---+---+---+---+-                                      ===
'===      | O | O |   |   |   |   |   |                                       ===
'===     -+---+---+---+---+---+---+---+-                                      ===
'===                                                                          ===
'==============================================================================
'====                                                                        ====
'===  Input:                                                                  ===
'===  xVect(0 to N-1) = vector containing x coordinates of vertices           ===
'===  yVect(0 to N-1) = vector containing y coordinates of vertices           ===
'===  N               = the number of data in the data vectors                ===
'===  minBin          = index for the lowest bin to regard                    ===
'===  maxBin          = index for the highest bin to regard                   ===
'====                                                                        ====
'===  Output:                                                                 ===
'===  H            = Hurst exponent                                           ===
'===  errorMessage = string containing possible errorMessage                  ===
'====                                                                        ====
'==============================================================================
'===  Written by Martin Stigsson, March 3 2016                                ===
'===  Please, report errors or improvements to: martin.stigsson@skb.se        ===
'==============================================================================
Sub BoxCounting evaluation variable dx(ByRef xVect() As Double,
                                       ByRef yVect() As Double, _
                                       ByRef N As Integer, _
                                       ByRef minBin As Integer, _
                                       ByRef maxBin As Integer, _
                                       ByRef H As Double,
                                       ByRef errorMessage As String)

    Dim i, j, k As Integer     'counters
    Dim nofBoxes As Integer    'number of boxes hit
    Dim currNofData As Integer 'number of boxes along one side of tyhe box
```

```
Dim x1Index As Integer     'x index for the start point of the current
'                           investigated line segment of the trace
Dim y1Index As Integer     'y index for the start point of the current
'                           investigated line segment of the trace
Dim x2Index As Integer     'x index for the end point of the curent
'                           investigated line segment of the trace
Dim y2Index As Integer     'y index for the end point of the curent
'                           investigated line segment of the trace
Dim xCIndex As Integer     'x index used in the yCIndex vector
Dim currXindex As Integer  'current x index when the invetsigated line
'                           segment strech over multiple x indices

Dim x1 As Double           'relative x coordinate for start point
Dim y1 As Double           'relative y coordinate for start point
Dim x2 As Double           'relative x coordinate for end point
Dim y2 As Double           'relative y coordinate for end point
Dim xMin As Double         'minimum x value in the data, i.e. xVect(0)
Dim yMin As Double         'minimum y value in the data
Dim yMax As Double         'maximum y value in the data
Dim xLength As Double      'extension in x direction of the data
Dim yLength As Double      'extension in y direction of the data
Dim intersection As Double 'dummy; needed when using regression sub
Dim slope As Double        'slope of the regression in log space

Dim hitMatrix(CInt(2 ^ maxBin), CInt(2 ^ maxBin)) As Boolean
'                           matrix of which cells that contain the line.
Dim yCIndex(N) As Integer            'y indices where the investigated
'                                     line is crossing the vertical
'                                     division lines
Dim nVect(Log(N) / Log(2)) As Double 'vector containing number of boxes
'                                     hit in each division level.
Dim dVect(maxBin) As Double          'Vector containing the 1/r values

'Get minimum value in x directions
xMin = xVect(0)

'Calculate length in x directions [ Step 2 ]
xLength = xVect(N - 1) - xMin

'Find the max and min values in y direction
yMin = 1.0E+20
yMax = -1.0E+20
For i = 0 To N - 1
    If yVect(i) < yMin Then
        yMin = yVect(i)
    End If
    If yVect(i) > yMax Then
        yMax = yVect(i)
    End If
Next

'Calculate length in y directions  [ Step 2 ]
yLength = yMax - yMin

'--- Find the boxes hit by the line on the finest grid ---

'Initiate the hitMatrix to be full of false and yCIndex to be -1
For j = 0 To 2 ^ maxBin
    yCIndex(j) = -1
    For k = 0 To 2 ^ maxBin
        hitMatrix(j, k) = False
    Next
Next

'Initiate the counter of used boxes
nofBoxes = 0

'Go through the vertices of the line
For j = 0 To N - 2  'I use minus two, because I look at the current
    '               point, and 1 forward

    'Calculate the indecies in the hitMatrix of the first point
    x1 = (xVect(j) - xMin) / xLength
    y1 = (yVect(j) - yMin) / yLength
    x1Index = Int(x1 * 2 ^ maxBin)
    y1Index = Int(y1 * 2 ^ maxBin)

    'Calculate the indecies in the hitMatrix of the second point
    x2 = (xVect(j + 1) - xMin) / xLength
```

```
y2 = (yVect(j + 1) - yMin) / yLength
x2Index = Int(x2 * 2 ^ maxBin)
y2Index = Int(y2 * 2 ^ maxBin)

'The uppermost point will be on the top border and resulting in a
'hit in a box outside the possible. Hence, Lower the index by one to
'force it being inside. This will happen exactly 1 time.
If y1Index = 2 ^ maxBin Then
    y1Index = y1Index - 1
End If
If y2Index = 2 ^ maxBin Then
    y2Index = y2Index - 1
End If
If x2Index = 2 ^ maxBin Then
    x2Index = x2Index - 1
End If

If x2Index > x1Index Then

    'interpolate the y value where the line segment between the
    'vertices crosses the vertical line of the adjacent boxes
    For xCIndex = x1Index + 1 To x2Index
        yCIndex(xCIndex) = Int(((y2 - y1) / (x2 - x1) * _
                           ((xCIndex / 2 ^ maxBin) - x1) + y1) * _
                           2 ^ maxBin)
    Next xCIndex

    'Depending on the order of the indecies the loop has to be done
    'in different orders.
    'If y index 2 is larger go from index1 to index2...
    If y2Index > y1Index Then

        currXindex = x1Index

        For k = y1Index To y2Index
            'Check if the box isn't marked
            If hitMatrix(currXindex, k) = False Then

                'Mark the box as used
                hitMatrix(currXindex, k) = True

                'Add one to the number of unique hit boxes
                nofBoxes = nofBoxes + 1
            End If

            'Is the investigation line crossing a vertical
            'division line?
            If k = yCIndex(currXindex + 1) Then

                'Add 1 to the x index and check the new cell
                currXindex = currXindex + 1

                'Check if the box isn't marked
                If hitMatrix(currXindex, k) = False Then

                    'Mark the box as used
                    hitMatrix(currXindex, k) = True

                    'Add one to the number of unique hit boxes
                    nofBoxes = nofBoxes + 1
                End If
            End If
        Next

    ElseIf y1Index > y2Index Then  '...else go from index 2 to 1

        currXindex = x1Index

        For k = y1Index To y2Index Step -1

            'Check if the box isn't marked
            If hitMatrix(currXindex, k) = False Then

                'Mark the box as used
                hitMatrix(currXindex, k) = True

                'Add one to the number of unique hit boxes
                nofBoxes = nofBoxes + 1
            End If
```

```
                'Is the investigation line crossing a vertical
                'division line?
                If k = yCIndex(currXindex + 1) Then

                    'Add 1 to the x index and check the new cell
                    currXindex = currXindex + 1

                    'Check if the box isn't marked
                    If hitMatrix(currXindex, k) = False Then

                        'Mark the box as used
                        hitMatrix(currXindex, k) = True

                        'Add one to the number of unique hit boxes
                        nofBoxes = nofBoxes + 1
                    End If
                End If
            Next

        Else 'y1index == y2index == yCindex

            For k = x1Index To x2Index

                'Check if the box isn't marked
                If hitMatrix(k, y1Index) = False Then

                    'Mark the box as used
                    hitMatrix(k, y1Index) = True

                    'Add one to the number of unique hit boxes
                    nofBoxes = nofBoxes + 1

                End If
            Next
        End If

    Else   'x2index == x1index

        'Depending on the order of the indecies the loop has to be done
        'in different orders.
        'If y index 2 is larger go from index1 to index2
        If y2Index > y1Index Then

            'go through all boxes between the lower and upper indices
            For k = y1Index To y2Index

                'Check if the box isn't marked
                If hitMatrix(x1Index, k) = False Then

                    'Mark the box as used
                    hitMatrix(x1Index, k) = True

                    'Add one to the number of unique hit boxes
                    nofBoxes = nofBoxes + 1

                End If
            Next

            'If y index 1 is larger go from index2 to index1
        ElseIf y1Index > y2Index Then

            'go through all boxes between the lower and upper indices
            For k = y2Index To y1Index

                'Check if the box isn't marked
                If hitMatrix(x1Index, k) = False Then

                    'Mark the box as used
                    hitMatrix(x1Index, k) = True

                    'Add one to the number of unique hit boxes
                    nofBoxes = nofBoxes + 1

                End If
            Next

            'If y indecies are equal
        Else
```

```
            'Check if the box isn't marked
            If hitMatrix(x1Index, y1Index) = False Then

                'Mark the box as used
                hitMatrix(x1Index, y1Index) = True

                'Add one to the number of unique hit boxes
                nofBoxes = nofBoxes + 1

            End If
        End If
    End If    'if x2index > x1index
Next j        'Go through the vertices of the line

'Save the number of boxes intersected by a line
nVect(maxBin) = nofBoxes

'Coarsen the grid and mark the cells that are intersected by checking if
'any of the four cells in the finer grid are intersected by the trace

' Hit cells in finest grid
'     0   1   2   3   4   5   6   7
' --+---+---+---+---+---+---+---+---+--
' 0 |   |   |   |   |   |   |   | X |
' --+---+---+---+---+---+---+---+---+--
' 1 |   |   |   |   |   |   | X | X |
' --+---+---+---+---+---+---+---+---+--
' 2 |   |   |   | X | X | X | X |   |
' --+---+---+---+---+---+---+---+---+--
' 3 |   |   |   | X |   | X | X |   |
' --+---+---+---+---+---+---+---+---+--
' 4 |   |   |   | X |   |   |   |   |
' --+---+---+---+---+---+---+---+---+--
' 5 |   | X | X | X |   |   |   |   |
' --+---+---+---+---+---+---+---+---+--
' 6 |   | X | X |   |   |   |   |   |
' --+---+---+---+---+---+---+---+---+--
' 7 | X | X |   |   |   |   |   |   |
' --+---+---+---+---+---+---+---+---+--
'
' Hit cells on the coarser grid
'       0       1       2       3
' --+-------+-------+-------+-------+--
'   |       |       |       |  \ /  |
' 0 |       |       |       |   X   |
'   |       |       |       |  / \  |
' --+-------+-------+-------+-------+--
'   |       |  \ /  |  \ /  |  \ /  |
' 1 |       |   X   |   X   |   X   |
'   |       |  / \  |  / \  |  / \  |
' --+-------+-------+-------+-------+--
'   |  \ /  |  \ /  |       |       |
' 2 |   X   |   X   |       |       |
'   |  / \  |  / \  |       |       |
' --+-------+-------+-------+-------+--
'   |  \ /  |  \ /  |       |       |
' 3 |   X   |   X   |       |       |
'   |  / \  |  / \  |       |       |
' --+-------+-------+-------+-------+--
'
' Hit cells on an even coarser grid
'            0               1
' --+---------------+---------------+--
'   |    \     /    |    \     /    |
'   |     \   /     |     \   /     |
'   |      \ /      |      \ /      |
' 0 |       X       |       X       |
'   |      / \      |      / \      |
'   |     /   \     |     /   \     |
'   |    /     \    |    /     \    |
' --+---------------+---------------+--
'   |    \     /    |               |
'   |     \   /     |               |
'   |      \ /      |               |
' 1 |       X       |               |
'   |      / \      |               |
'   |     /   \     |               |
'   |    /     \    |               |
' --+---------------+---------------+--
```

```
    'Go through the data from the finest to the coarsest grid
    For i = maxBin - 1 To minBin Step -1

        'Calculate the number of cells on each side of the lattice
        currNofData = 2 ^ i

        'Initiate the counter
        nofBoxes = 0

        'Go through the cells in the coarser grid...
        For j = 0 To currNofData - 1
            For k = 0 To currNofData - 1

                '...and see if they should be marked as intersected because
                'any of the four sub-cells of the finer grid is hit
                If hitMatrix(j * 2, k * 2) Or _
                   hitMatrix(j * 2 + 1, k * 2) Or _
                   hitMatrix(j * 2, k * 2 + 1) Or
                   hitMatrix(j * 2 + 1, k * 2 + 1) Then

                    'mark the cell in the coarser grid as hit
                    hitMatrix(j, k) = True

                    'Add one to the number of boxes hit
                    nofBoxes = nofBoxes + 1

                Else

                    'mark the cell in the coarser grid as no hit
                    hitMatrix(j, k) = False
                End If
            Next k
        Next j

        'Save the number of boxes intersected by the line
        nVect(i) = nofBoxes
    Next i

    'Construct vector containing the sidelengths of the cells
    For i = minBin To maxBin
        dVect(i) = 2 ^ i
    Next

    'Log the data
    For i = minBin To maxBin
        dVect(i) = Log10(dVect(i))
        nVect(i) = Log10(nVect(i))
    Next

    'Calculate the slope from all used Bins [ Step 5 ]
    Call regression analysis of line(dVect, nVect, minBin, maxBin, slope, _
                                     intersection, errorMessage)

    'Calculate the Hurst exponent eq. 5-8
    H = 2 - slope

End Sub
```

6 The flaw in the original divider method

The original divider method is very easy to implement and to evaluate. However, it is only applicable to self-similar fractals, and not to self-affine fractals (Den Outer et al. 1995; Odling 1994). The method can give reasonable results if the trace is scaled so that the difference between maximum and minimum height equals the horizontal length of the trace (Kulatilake et al. 2006). This transformation equals the creation of the unity "box" of the box counting method.

The results from the method are shown using a synthetic trace of 1024 vertices and resolution 0.1 mm with $H = 0.600$ and $\sigma\delta h(1\ mm) = 0.246$ mm. The vertices of the trace are provided in Online Resource 2 and visualised in Figure 6-1. Evaluating the trace with the method developed in the main paper renders JRC equal to 11.7.

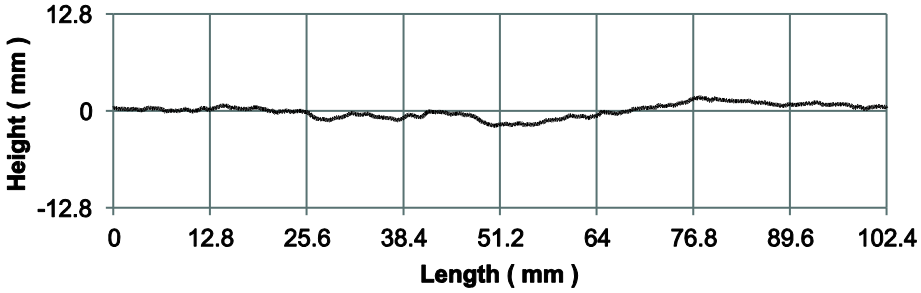

Figure 6-1. *Trace used for evaluation by the divider method.*

Following the original divider method, the length of the trace is measured using 0.1 mm steps in the length direction. Thereafter, the length of the trace is measured using steps of 0.2 mm, 0.4 mm, 0.8 mm etc. in the length direction. The trace length is then plotted as a function of the step length in logarithmic space and the slope is used to infer the Hurst exponent, Figure 6-2. The Hurst exponent relates to the slope as $H = 1 +$ slope. Using the example trace in Figure 6-1, the slope is –0.023 and hence $H = 0.977$. This number is within the range reported in many studies (e.g. Turk et al. 1987; Wakabayashi and Fukushige 1992; Bae et al. 2011) using the original divider method erroneously on self-affine traces.

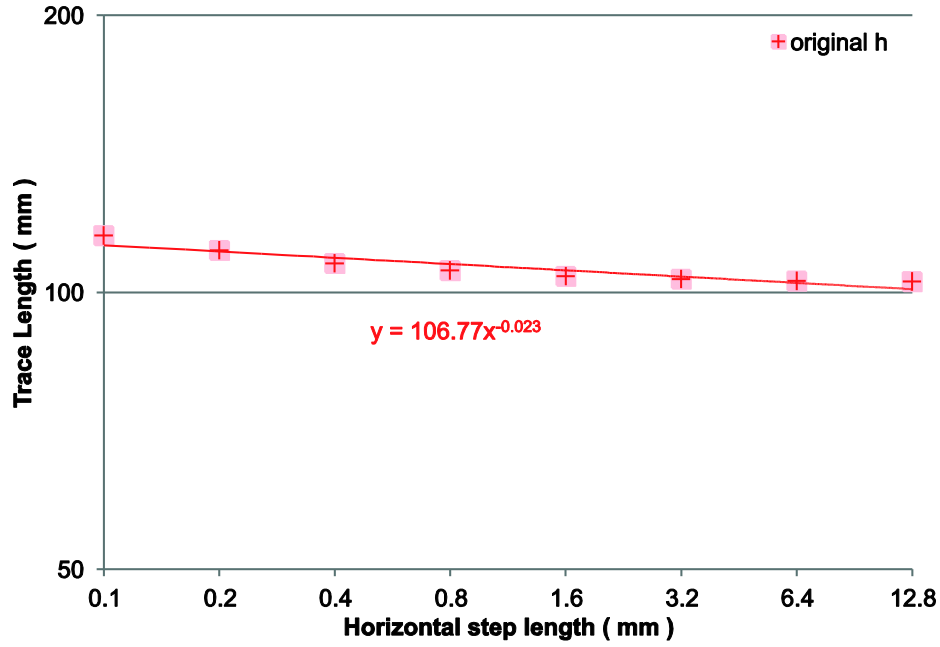

Figure 6-2. *Trace length as a function of horizontal step length using the original divider method.*

However, following the reasoning in Kulatilake et al. (2006), it is possible to achieve reasonable results using a modified version of the divider method. By exaggerating the height of the trace the

slope will increase and, hence, the Hurst exponent will decrease, Figure 6-3. The more exaggerated the heights, the lower *H*, although the decrease in *H* proceeds very slow after a certain exaggeration value, Figure 6-4. However, too large exaggeration will not be correct either. Instead, following the reasoning in Malinverno (1990), the exaggeration should be so large that the difference between the maximum and minimum height equals the horizontal length of the trace, i.e. so that a square is obtained. This corresponds to creating "the box" using the Box Count method. Using the trace in Figure 6-1, the exaggeration factor should be 27, Figure 6-5. Evaluating this trace using the divider method results in slope –0.371 and hence $H = 0.629$, which is close to the generated $H_{gen} = 0.6$.

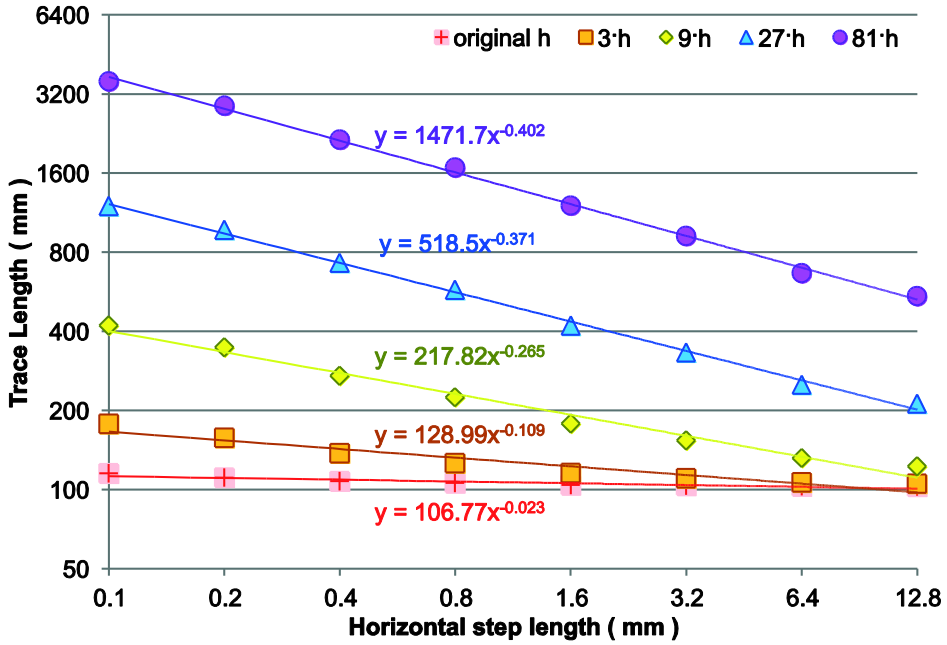

Figure 6-3. *Trace length as a function of horizontal step length using different exaggerations of the heights.*

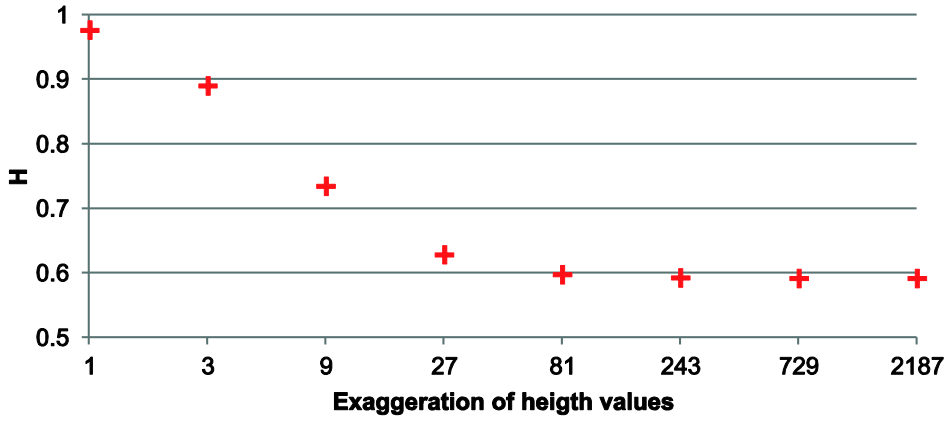

Figure 6-4. *Evaluated H as a function of exaggeration of height values.*

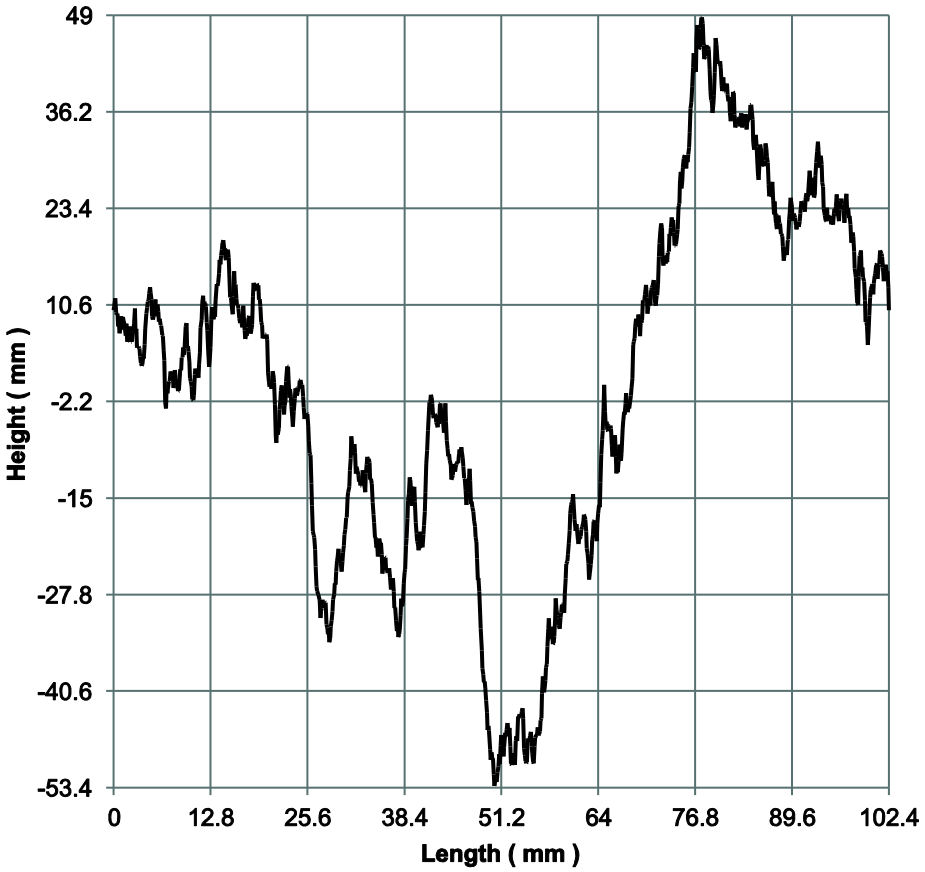

Figure 6-5. *The heights in Figure 6-1, multiplied by 27 to make the maximum height difference equal the horizontal length of the trace.*

7 Erroneous implementation of the Box Count method

The Box Count method is correctly implemented in section 5.4. However, there are many examples in the literature (e.g. Chen et al. 2012; Li and Huang 2015; among others) where the method has been erroneously implemented. The erroneous method is shown below. It should **not** be used for evaluating fractal lines.

The fractal line, with H_{gen} = 0.6, shown in Figure 6-1 is used in this example. Instead of the correct creation of the box, by finding the lowest and highest values in both the length and height directions, the box is created by making the height equal the length as shown in Figure 7-1 This means that only the boxes that are adjacent to the abscissa will be visited by the fractal line as long as the size of the boxes is larger than the maximum or minimum height. Hence, the number of visited boxes will approximately double for each new division into sub-boxes. This implies that H will be evaluated speciously as close to 0.

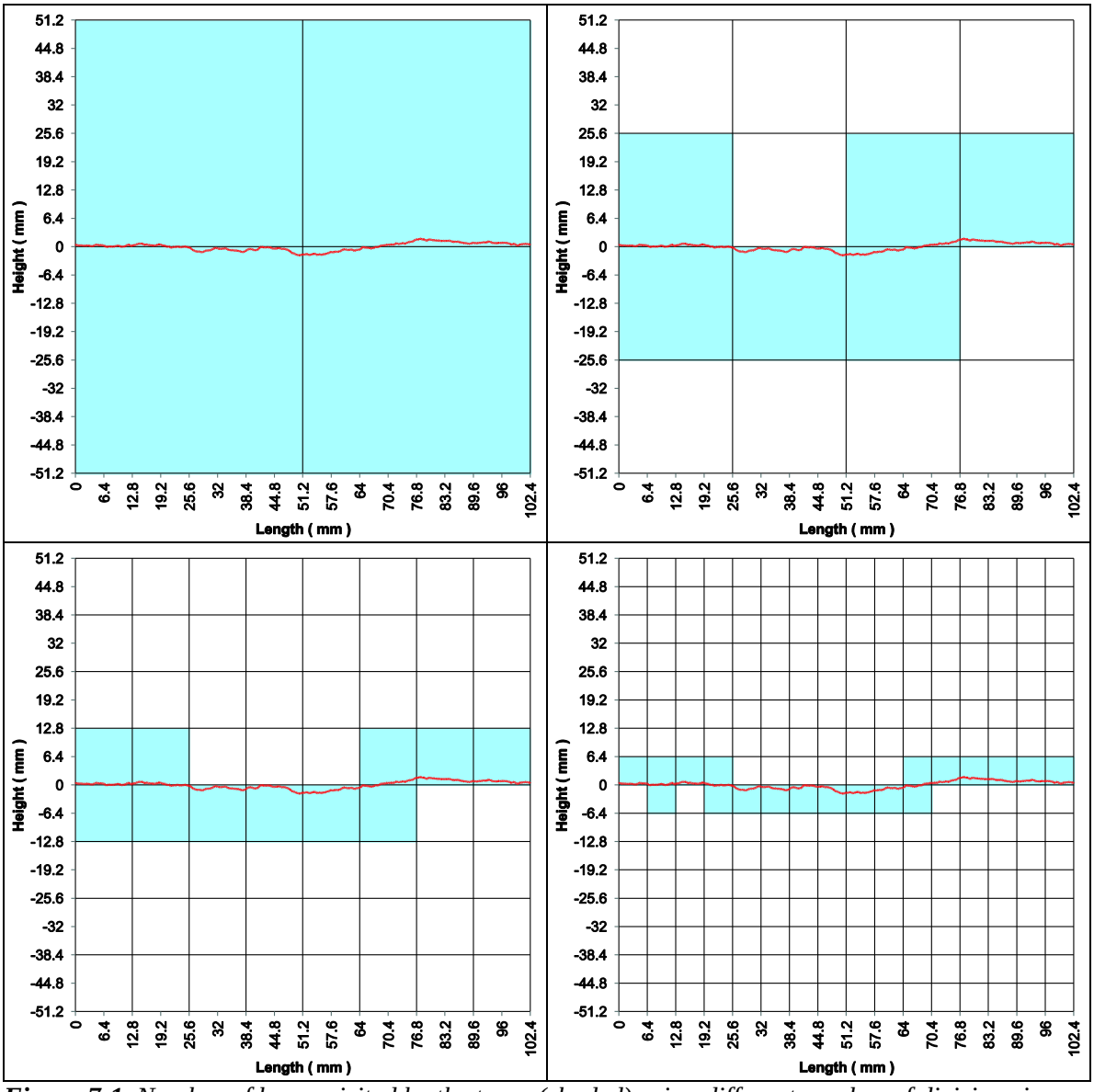

Figure 7-1. *Number of boxes visited by the trace (shaded) using different number of divisions in erroneous implementation of the Box Count method. Upper left) One division, 4/4 boxes visited. Upper right) Two divisions, 6/16 boxes visited. Lower left) Three divisions, 11/64 boxes visited. Lower right) Four divisions, 19/256 boxes visited.*

The boxes visited by the example fractal line are shown in Figure 7-1 for the first four divisions and the numbers of boxes visited for the first seven divisions are listed in Table 7-1. As expected, the number of boxes visited almost doubles for each division, resulting in a slope in log space of 0.9, see Figure 7-2. According to eq. 5-9 this corresponds to H = 1.1, which should be compared against the generated Hurst exponent, H_{gen}, = 0.6.

Table 7-1. Number of divisions, d, number of boxes along the axes, n, and number of boxes visited by the fractal line, N(n), in erroneous implementation of the Box Count method

d	n	N(n)
1	2	4
2	4	6
3	8	11
4	16	19
5	32	38
6	64	73
7	128	142

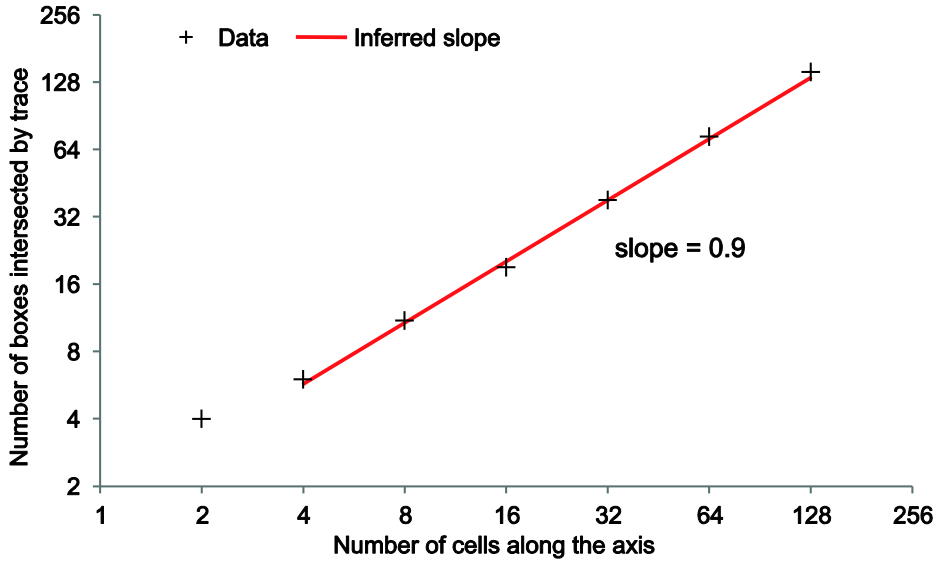

Figure 7-2. *Number of boxes visited by the trace using different number of boxes along each side of the main box in erroneous implementation of the Box Count method.*

Using this erroneous implementation, the speciously evaluated H will be more sensitive to the scale parameter $\sigma\delta h(\Delta x)$ than the Hurst exponent, since a larger $\sigma\delta h(\Delta x)$ will make the fractal line visit more boxes that are not adjacent to the abscissa. The method is also sensitive to the vertical location of the trace, giving the highest H if it is centred around the abscissa. By lowering the trace in the example above by 1.8 mm, only the boxes that are below the abscissa will be visited. Hence, the first four divisions will result in a doubling of the number of boxes visited for each division, i.e. $N(n)$ = 2, 4, 8, 16. This equals a slope of 1 and hence H = 1. The usage of this erroneous implementation will severely overestimate the Hurst exponent and hence underestimate the fractal dimension.

8 The problem with the Z_2 method

Another widely used method to evaluate JRC from traces is using a relationship to the Z_2 method defined by Myers (1962). The measure Z_2 is simply the root mean square of the first derivative of the traces. Its discrete form, according to Tse and Cruden (1979) is described by:

$$Z_2 = \sqrt{\frac{1}{N(\Delta x)^2}\sum_{i=1}^{N}(h_{i+1} - h_i)^2} \qquad \text{eq. 8-1}$$

where:

Z_2 = root mean square of the first derivative of the trace
N = number of intervals
Δx = step length
h_i = height value at node i.

Tse and Cruden (1979) developed a relationship between JRC and Z_2 for traces with Δx = 1.27 mm that is described by

$$JRC = 32.2 + 32.47 \cdot \log Z_2 \qquad \text{eq. 8-2}$$

To investigate the method, the trace in Figure 6-1 is used. The trace has 1024 vertices equally spaced in the horizontal direction, i.e. Δx = 0.1 mm and inferred JRC 11.7. To investigate the effect the resolution has on the resulting JRC, six different resolutions are used below (2 mm, 1.5 mm, 1.3 mm,1 mm, 0.5 mm and 0.1 mm), while a 10.2 mm piece of 0.1 mm resolution is used to evaluate the effect the length has on the resulting JRC. The results are shown in Table 8-1.

Table 8-1. Z_2 and Joint roughness coefficient (JRC) values of the trace in Figure 6-1 obtained using different resolutions and number of vertices

Δx (mm)	2	1.5	1.3	1	0.5	0.1	0.1
N	51	68	78	102	204	1024	102
Z_2	0.158	0.202	0.207	0.222	0.308	0.539	0.506
JRC	6.2	9.7	10.0	11.0	15.6	23.5	22.6

As can be seen, the Z_2 method gives an estimated JRC value in the range of the inferred one, I.e. 10.0 compared to 11.7, when the resolution is 1.3 mm, (i.e close to the distance, 1.27 mm, that Tse and Cruden (1979) used developing the relationship). When the resolution is 1 mm the estimated JRC is closer, 11.0 compared to 11.7. However, on doubling the resolution, i.e. Δx = 0.5 mm, Z_2 increases, resulting in a higher estimated JRC, 15.6. On further increasing the resolution, to Δx = 0.1 mm, the method results in even higher Z_2 and the estimated JRC = 23.5, Table 8-1. In the same way the interpreted JRC values will decrease if the resolution gets coarser, for example 1.5 or 2 mm. This is expected due to the nature of fractal lines, i.e. the higher the resolution, the longer the total trace length and the steeper the angled parts of the trace. These results are in accordance with the findings by Xu and Vayssade (1991) who invented relationships between Z_2 and JRC for sampling intervals of 1 mm, 0.5 mm and 0.25 mm.

The number of vertices used in the evaluation is of minor importance. Using a short trace, 10.2 mm, with high resolution, Δx = 0.1 mm, gives an estimated JRC value of 22.6. This is in the same range as for the long trace with same resolution, i.e. JRC = 23.5.

The conclusion is that the correlation between Z_2 and JRC developed by Tse and Cruden (1979) only is valid when the resolution is 1.27 mm and the error will be larger the larger the deviation is from the demanded resolution. This implies that the traces have to be sampled using 1.27 mm distance or at distances that are fractions of 1.27 mm. If the resolution is coarser, as may be the case for long traces, new empirical relationships have to be developed for each resolution.

To test the inference of JRC using the Z_2 method in a systematic way, the nine synthetic traces in Stigsson (2018) are analysed using a resolution of 1.3 mm. The JRC values of these traces were visually inferred by an ensemble of eleven geologists in Stigsson (2018) and are here compared with the JRC values calculated using Eq. 8-1 and 8-2. There is a good correlation between the visually interpreted JRC values and the JRC values inferred using the Z_2 method for values larger than about 5, see Figure 8-1. However, for the lower range, JRC less than about 5, there is an indication that the conversion from Z_2 to infer JRC using Eq. 8-2 will give unrealistic results.

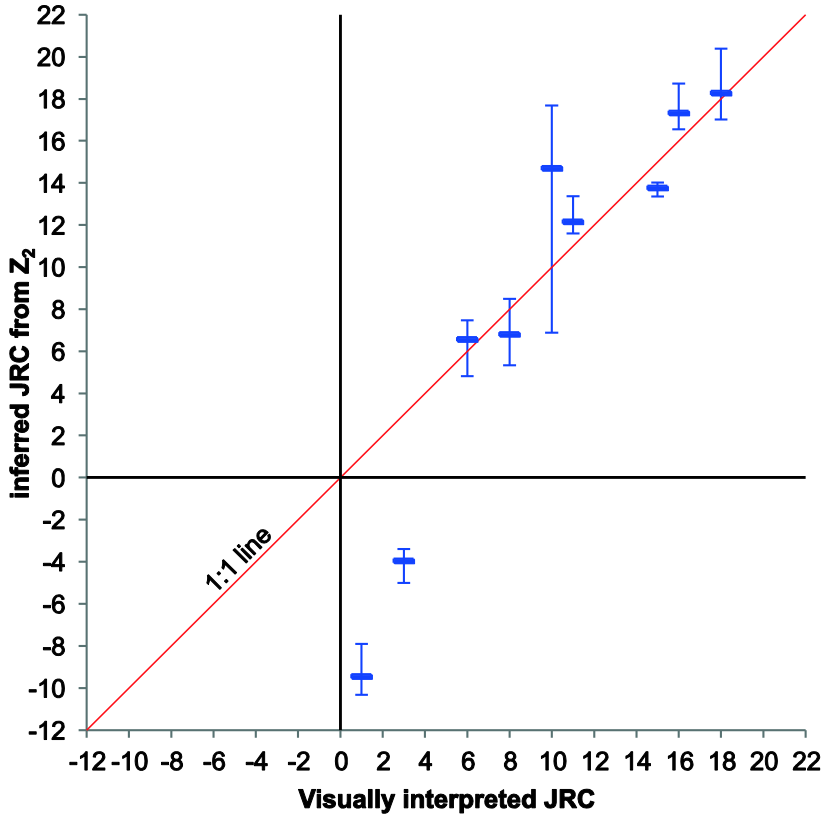

Figure 8-1. *Inferred JRC values calculated from Z_2 as a function of the visually interpreted JRC values from an ensemble of geologists. The middle bar shows median value while whiskers show the 25 and 75 percentile.*

9 Sensitivity study of number of realisations needed

A study to evaluate the number of generated traces needed to get stable measures, average and standard deviation, of H and $\sigma\delta h(1p)$, is carried out here for two cases, H = 0.975 and H = 0.600, both with $\sigma\delta h(1p)$ = 0.2. Results from H = 0.975 are shown in Figure 9-1 and results from H = 0.600 are shown in Figure 9-2.

In the study, 4096 traces of 65536 vertices are generated. From these traces, sub-traces with 8192 or 64 vertices are extracted and analysed. For each full trace (65536 vertices), there are 15 sub-traces of 8192 vertices and 2047 sub-traces of 64 vertices. The differences in arithmetic mean and standard deviation compared with the arithmetic mean and standard deviation of the 4096 traces are shown as a function of number of traces in Figure 9-1 and Figure 9-2. As can be seen, both the arithmetic mean and standard deviation values stabilise after 128 to 256 generated traces for all methods except for Zero set/Korcak, which needs 512 traces to be stable. Hence, it should be enough to generate 512 traces. However, 1024 realisations are carried out here to have some margin to the minimum required.

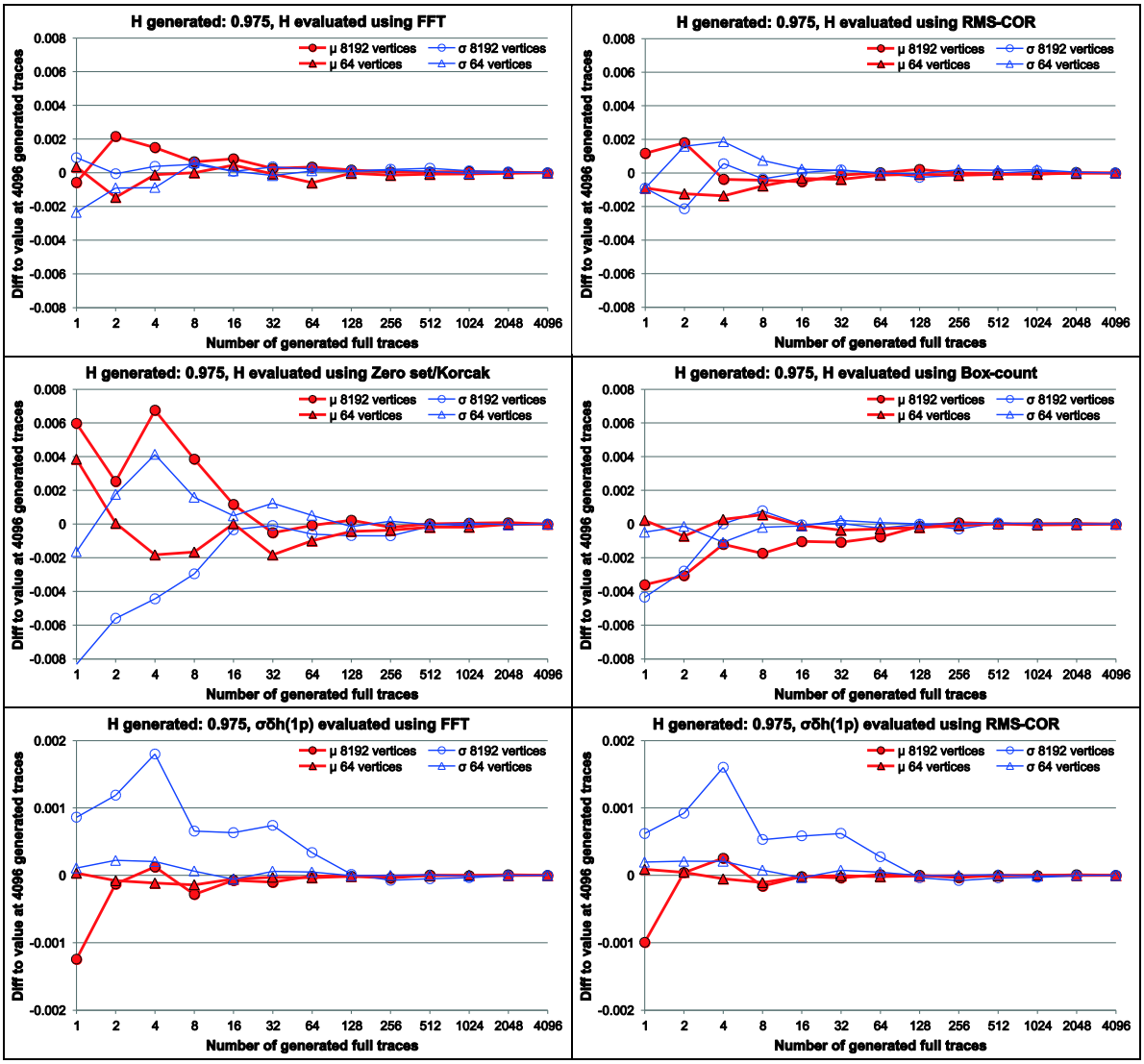

Figure 9-1. *Number of generations of full traces needed to get stable arithmetic mean and standard deviation of H and σδh(1p), using different evaluation methods, when H = 0.975.*

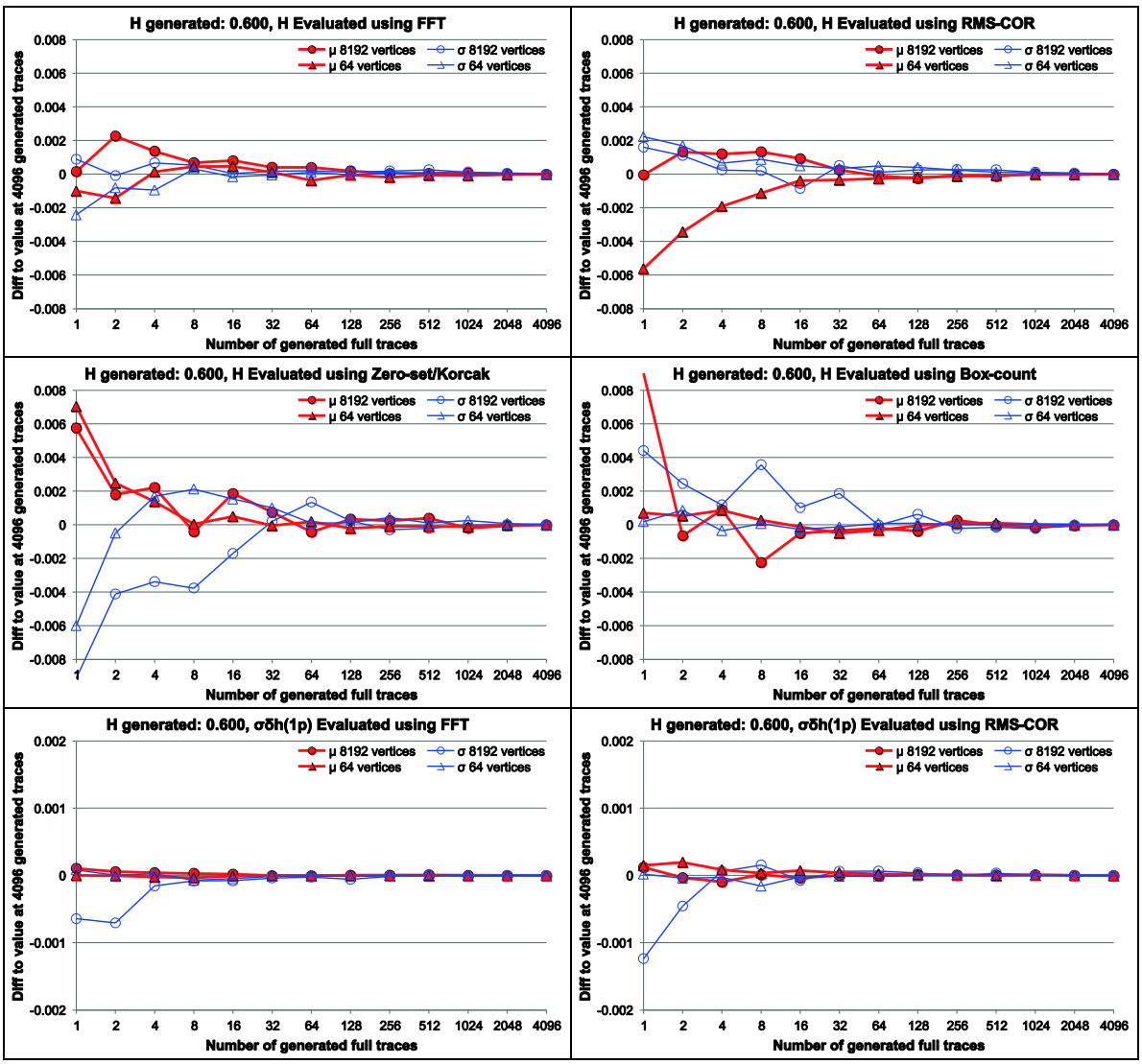

Figure 9-2. *Number of generations of full traces needed to get stable arithmetic mean and standard deviation of H and σδh(1p), using different evaluation methods, when H = 0.600.*

10 Parameters affecting evaluation of the fractal parameters H and σδh(ΔL)

10.1 Hurst exponent, H

The arithmetic mean and variance of the inferred Hurst exponent, *H*, may depend on the generated *H* itself and the number of vertices used during the evaluation, but not on the magnification of the asperities. The effect that the generated *H* will have on the evaluated *H* is analysed here by comparing equally long traces with different generated *H*. The sizes of the effects will differ due to the lengths of the evaluated traces and, hence, different lengths of traces are extracted. The full traces, 65536 vertices, are analysed together with two sub-samples of 1024 vertices and 64 vertices of the full trace. As a reference, another set of 1024 vertices traces are extracted, but from traces generated with 4096 vertices only. The results are presented in Figure 10-1.

The value of *H* obtained using the FFT method is right on the 1:1 slope, without any variance for the full trace, Figure 10-1a. This is expected, since that evaluation method is exactly the inverse of the generation method. However, if *H* is evaluated using a sub-trace, Figure 10-1b to d, the picture becomes different. As expected, the average of the FFT method is still on the 1:1 slope, but there is a variance around the mean. As the number of vertices decreases the variance increases, cf. Figure 10-1a to c. The variance is only dependent on the number of vertices used in the analysis, and not the ratio between generated and analysed number of vertices, cf. Figure 10-1b and d.

The value of *H* obtained by RMS-COR using the full 65536 vertices trace shows good agreement with the generated *H* up to 0.900. Above 0.900, the evaluated *H* deviates downward and at $H = 1.000$ the deviation is about 0.050, Figure 10-1a. However, the variance around the mean is insignificant for any *H* evaluating the full trace. As the evaluated trace gets shorter, the evaluated mean *H* obtained using the RMS-COR method deviates more from the 1:1 slope and the variance becomes larger, Figure 10-1b and c. The evaluated mean *H* and variance is not dependent on the number of vertices of the generated trace, but only the number of evaluated vertices, cf. Figure 10-1b and d.

The Zero set/Korcak evaluation method of *H* using the full 65536 vertices trace also shows good agreement with the generated *H* for the lower values of *H*, but for the higher values the method under-predicts the generated value by up to 0.1, Figure 10-1a. As the traces gets shorter, the method results in larger under-predictions, and more so for the higher Hurst exponents, Figure 10-1a to c. The variance is almost independent on *H*, but very dependent on the trace length, being 0.01 for the full trace and around 0.25 for the 64 vertices trace. The Zero set/Korcak method is not dependent on the ratio between the number of generated vertices and the evaluated number, but is dependent on the absolute number of vertices only, cf. Figure 10-1b and d.

Already for the full trace, 65536 vertices, the evaluated mean *H* using the Box Count method deviates from the 1:1 slope of *H*. The method over-predicts the generated *H* by approximately 0.05 for the lower generated *H* and under-predicts it by the same amount for the higher generated *H*, Figure 10-1a. As the number of evaluated vertices decreases, the deviation from the generated *H* increases, resulting in a flatter relationship between generated and evaluated *H*, Figure 10-1a to c. The variance increases with decreasing number of vertices, but is independent of *H*. As for the other three methods, the mean and also the variance are not dependent on the ratio between absolute number of vertices generated and evaluated, but only on the number of vertices evaluated, cf. Figure 10-1b and d.

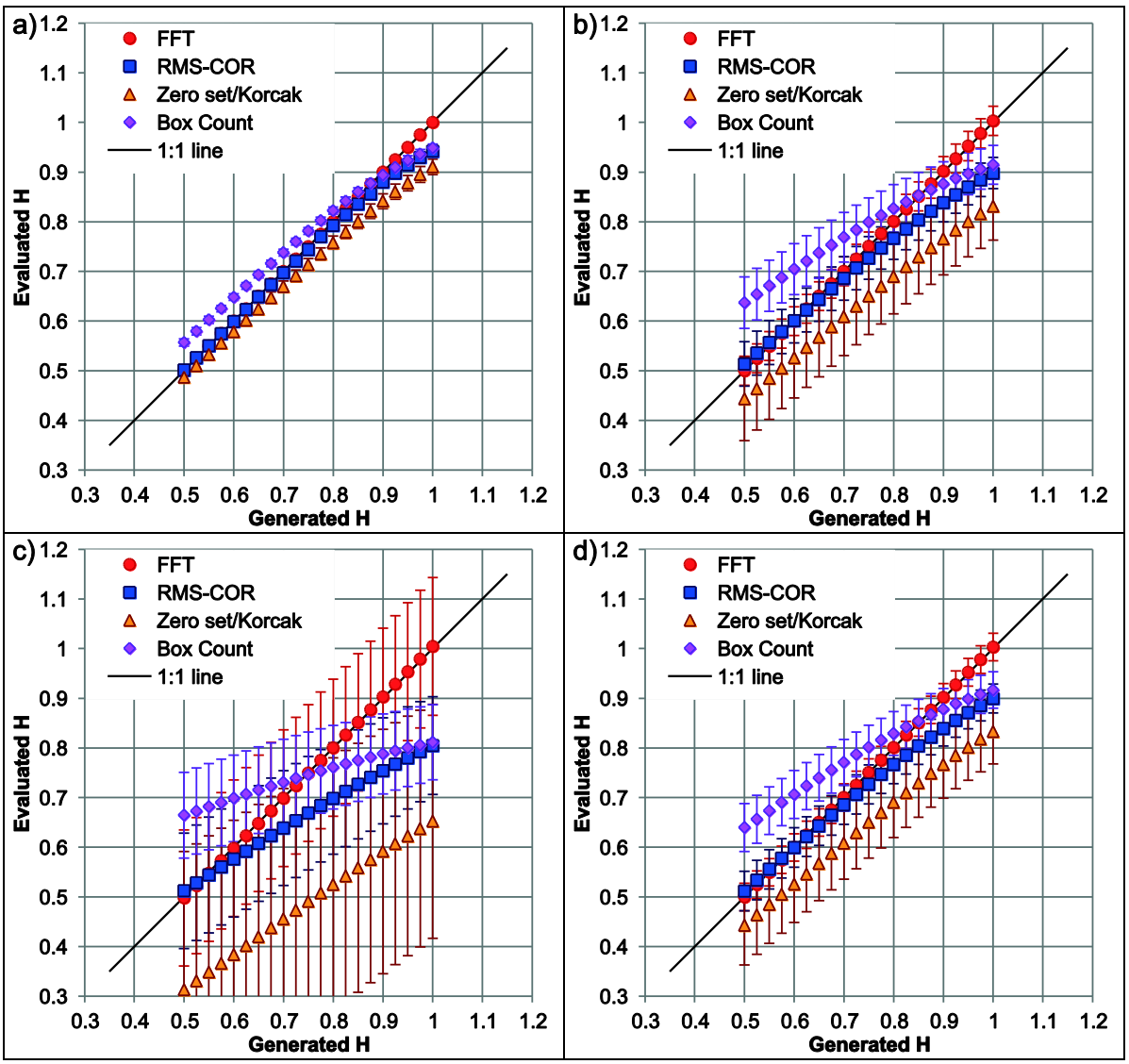

Figure 10-1. *The mean value of the Hurst exponent, as markers, and the standard deviation, as whiskers, of the evaluated H depending on the generated H. a) All 65536 vertices used, b) 1024 vertices used of 65536 available, c) 64 vertices used of 65536 available and d) 1024 vertices used of 4096 available.*

As indicated in Figure 10-1 and discussed above, the evaluated Hurst exponent is not only affected by the value of *H* itself, but also by the number of vertices used in the evaluation. The number of vertices can be reduced in two ways; a short piece with full resolution can be extracted for analysis, or the full length of the trace can be used, but only extracting every i:th vertex. Figure 10-2 shows the effect that decreasing the number of vertices will have on the evaluated *H*. The effect is shown using two different generated *H* values, 0.975 and 0.600. On reducing the number of vertices used in evaluation of the mean *H*, the deviation from the generated *H* will increase, independent of how the number of vertices is decreased. However, the deviation is slightly larger with larger *H*, i.e. the deviation is larger for the trace where the generated *H* is 0.975, compare Figure 10-2a with b and c with d. The variance does not seem to be affected by the absolute number of *H*, but by the method used to reduce the number of vertices, compare the standard deviations in Figure 10-2a with c and b with d. As expected, shorter traces with high resolution will have larger uncertainty than longer traces with low resolution given the same number of vertices.

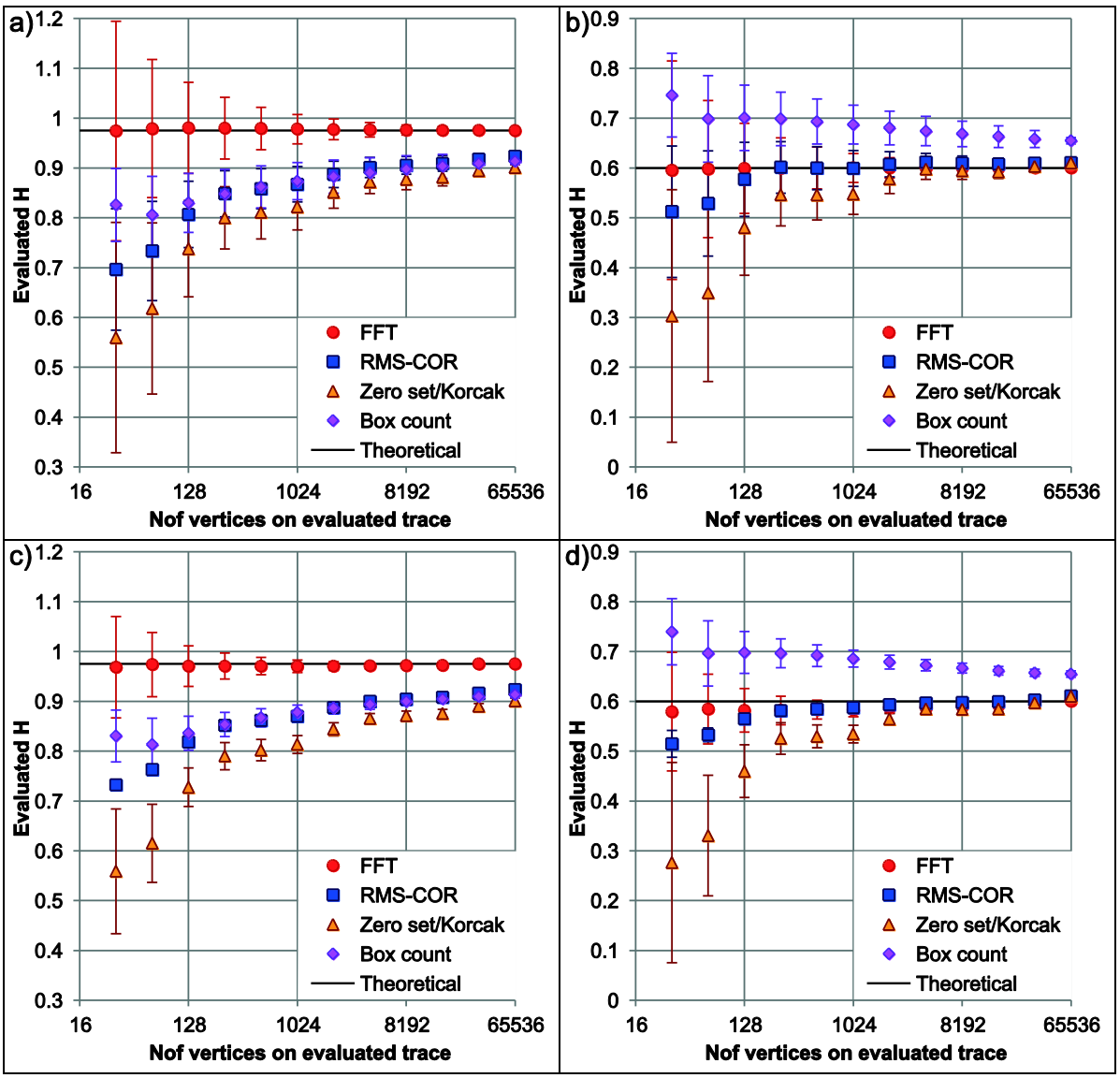

Figure 10-2. *The mean, as markers, and standard deviation, as whiskers, of the evaluated H as a function of the number of vertices analysed. In the upper row the number of vertices is changed by altering the length of the trace, whilst in the lower row the number of vertices is changed by skipping in-between vertices. In the left column, the generated H = 0.975; in the right column, H = 0.600.*

Since all four methods are suitable for self-affine traces, the evaluated *H* does not depend on the generated *σδh(1p)*, as expected, see Figure 10-3. However, when using e.g. the divider method the Hurst exponent would decrease as *σδh(1p)* increases, see section 6.

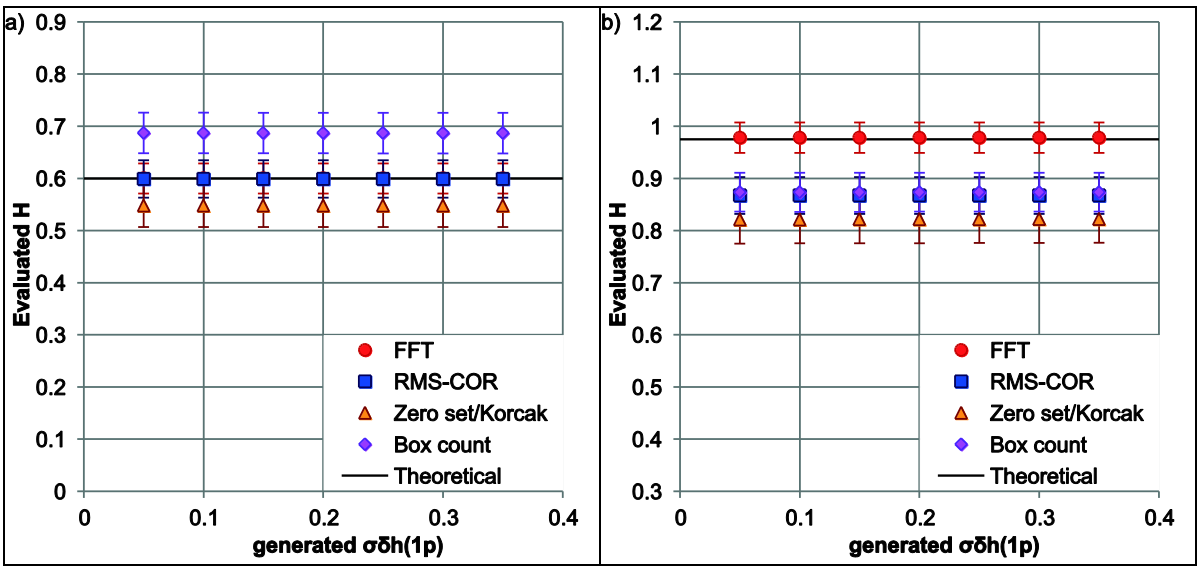

Figure 10-3. *The mean, as markers, and standard deviation, as whiskers, of the evaluated H as a function of σδh(1p) for a trace of 1024 vertices.*

10.2 Asperity measure, σδh(ΔL)

When measuring a fracture trace, there is always a trade-off between high resolution of a small piece or low resolution of the full trace. Measuring a full fracture trace with high resolution will always give the correct *σδh(ΔL)* for *ΔL* equal to the resolution and up. However, in the case where only a fraction of the fracture trace is measured, *σδh(1p)* will usually be underestimated compared with the true value due to the de-trending of the short evaluated trace. The larger the *H* of the investigated trace, or the fewer investigated vertices, the larger the underestimation of *σδh(1p)*, Figure 10-4a and b. However, the inferred *σδh(1p)* does not have to be underestimated by the investigation method used and regression performed, see e.g. Figure 10-4a where the RMS-COR method overestimates the generated *σδh(1p)* by ~0.01 mm down to traces that are 1/1024 of the original trace length. Still, the inferred values will decrease as *H* increases and the trace contains fewer vertices. The difference between the two methods may be explained by the linear regression using RMS-COR and by the fact that the calculation of *σδh(ΔL)* is not exact using eq. 5-2.

In the case where the full trace is measured, but at a low resolution, the evaluated *σδh(ΔL)* will scale as eq. 5-3, where ΔL will be the length between measured vertices, Figure 10-4c and d. Hence the *σδh(ΔL)* will increase as the resolution decreases, due to larger distance between the evaluated vertices. The theoretical and evaluated values of *σδh(ΔL)* will almost coincide when $H = 0.600$ for both the FFT and RMS-COR method, Figure 10-4c. However, the evaluated *σδh(ΔL)* values will be underestimated as the resolution becomes coarser, i.e. *ΔL* gets larger, for the case where $H = 0.975$, Figure 10-4d. The slope of the inferred values in Figure 10-4d is only 0.91, compared with the theoretical 0.975, indicating that the Inverse FFT method is not capable of generating correct traces as *H* approaches 1. As a consequence, the FFT method will overestimate *H* for traces where *H* is close to 1.

Estimating *σδh(ΔL)* for *ΔL* smaller than the distance between the measured points is delicate. Due to the need for extrapolation of a power function with an uncertain value of the exponent, *H*, the estimated *σδh(ΔL)* will be highly uncertain, Figure 10-4e and f. For a trace with $H = 0.600$, the error is less than 20% if using more than 256 vertices of the 65536 available, but then rapidly increases. Yet, for the case where $H = 0.975$, the error increases quite rapidly due to the trouble in correctly inferring *H* when close to 1.

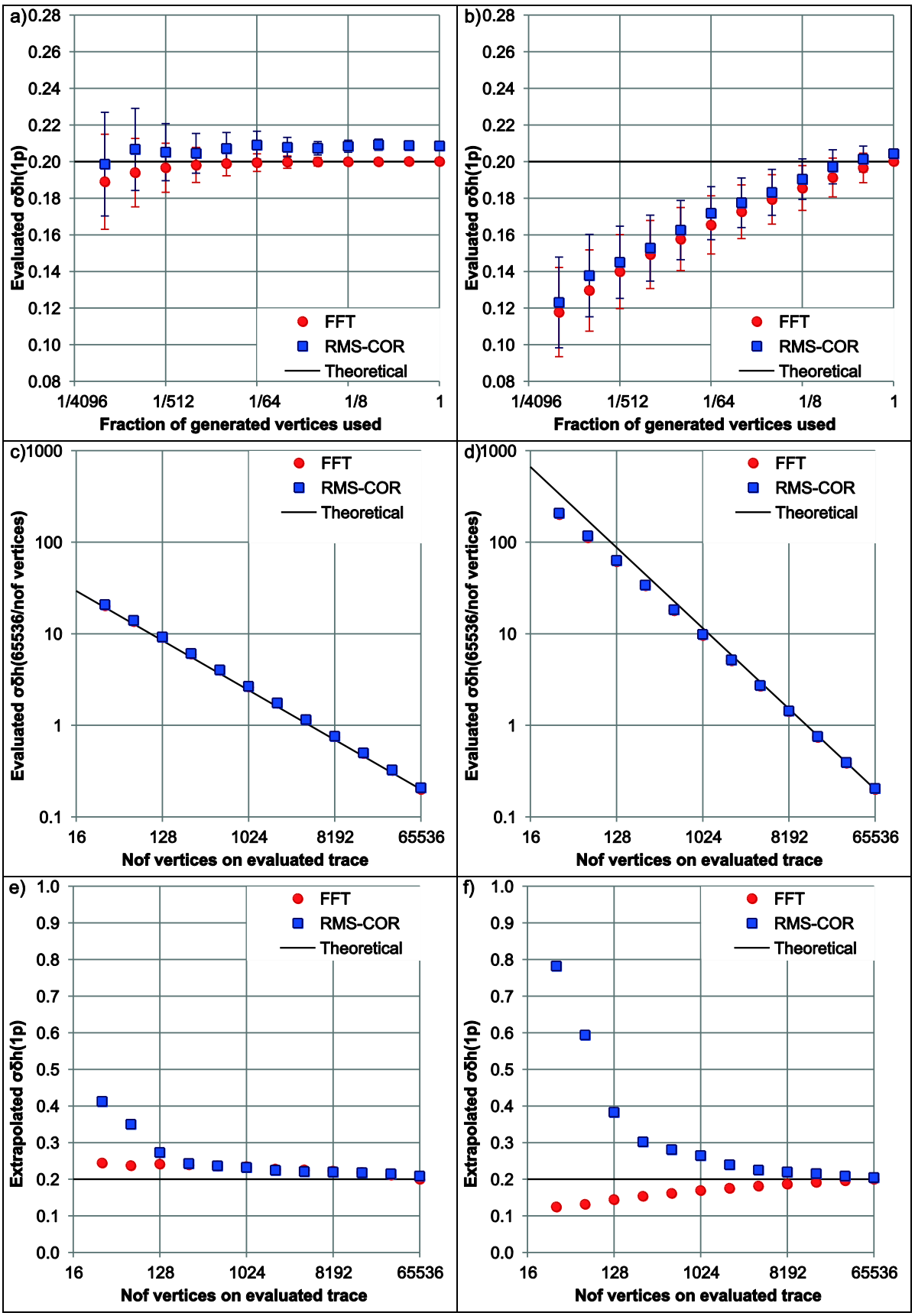

Figure 10-4. *Evaluated standard deviation of the asperity difference, σδh(Δx), using two different generated H, 0.600 in left column and 0.975 in right column. a) and b) The effect of using full resolution, 1p, but changing the length of the evaluated trace. c) and d) The effect of changing the resolution, i.e. the length between the evaluated vertices is changed, but full length of the trace is kept. e) and f) The effect of changing the resolution using full length trace, but extrapolating the result below the distance between the evaluated vertices.*

As indicated in Figure 10-4a and b, the Hurst parameter will affect the possibility to infer *σδh(1p)*. In Figure 10-5 *σδh(1p)* is plotted against the Hurst parameter for two traces of different lengths, one of 1024 vertices and one of 64 vertices. As the Hurst parameter increases the inferred *σδh(1p)* decreases, together with an increase in the variance; the shorter the trace, the larger the effect.

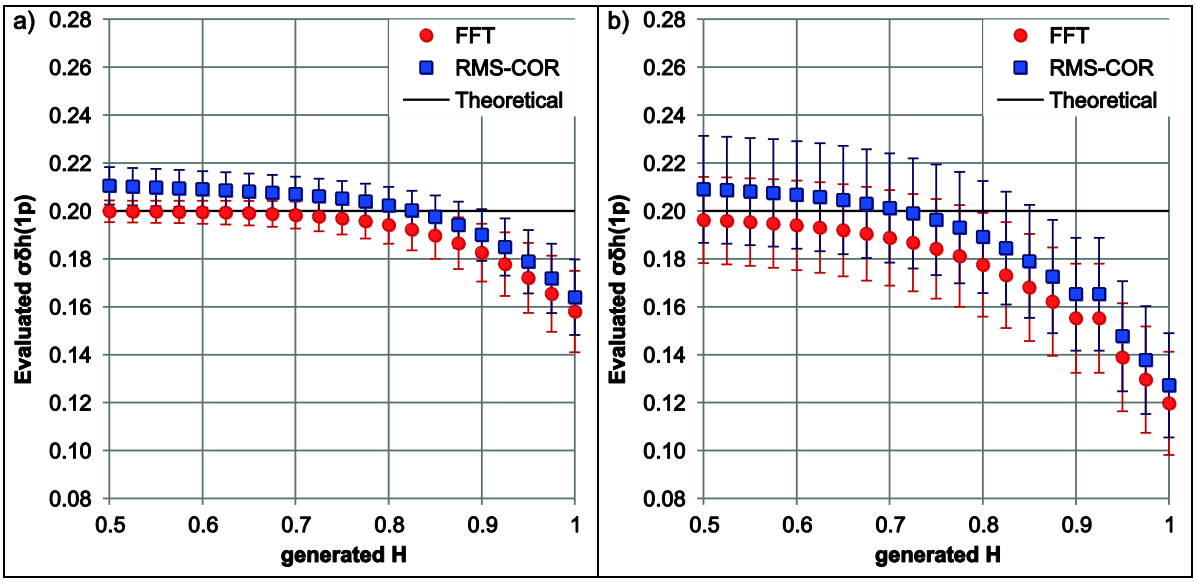

Figure 10-5. *Evaluated standard deviation of the asperity difference, σδh(1p) as a function of generated H, using two different lengths of the traces. a) 1024 vertices and b) 64 vertices.*

11 Manual digitalisation of traces

In order to evaluate the type traces described in Barton and Choubey (1977) and Bakhtar and Barton (1984), they need to be digitised. This can be done using algorithms, as e.g. Jang et al. (2014), or manually. An algorithm-based digitalisation is preferable due to its repeatability, but when such code is lacking manual digitalisation may be used. However, it is wise to execute multiple digitalisations of the traces in order to estimate the uncertainty in the results. For the study in the main paper the type traces in Barton and Choubey (1977) and Bakhtar and Barton (1984) were manually digitised twice, once from left to right and once from right to left by the authors. The difference between the two digitalisations was judged to be so small that no further digitalisation was needed.

The procedure to digitise the ten plus seven traces was as follows: Figure 8 in Barton and Choubey (1977) and Figure 5 in Bakhtar and Barton (1984) were imported into the software Surfer 12 by Golden software (Golden Software 2018). The judged breakpoints of the traces were digitised in paper space and exported as text files in the bln format. The scaling factor between real space and paper space was inferred by digitising the rulers at the bottom of each diagram. The traces were then scaled to real space and translated to begin at the origin of the coordinate system. The scaled and translated traces are shown in Online Resource 2, attached to the main paper. Two of the evaluation methods need the vertices to be at equal distances in the x direction, and hence the traces were sampled every 0.1 mm for the ~100 mm traces in Barton and Choubey (1977) and every 1 mm for the ~1000 mm traces in Bakhtar and Barton (1984).

During the analysis of the fractal parameters, the traces are automatically rectified using either Deming regression or simple de-trend to together with vertical adjustment to avoid artificial high power low frequency biases.

References

Bae D, Kim K, Koh Y, Kim J (2011) Characterization of joint roughness in granite by applying the scan circle technique to images from a borehole televiewer. Rock Mech Rock Eng 44:497–504

Bakhtar K, Barton N (1984) Large Scale Static and Dynamic Friction Experiments. In, Proc. 25th US Symposium on Rock Mechanics, Evanstone, pp. 457–466. https://doi.org/10.1016/0148-9062(86)91744-4

Barton N, Choubey V (1977) The shear strength of rock joints in theory and practice. Rock Mechanics 1/2:1-54. Vienna: Springer

Brodsky E, Gilchrist J, Sagy A, Collettini C (2011) Faults smooth gradually as a function of slip. Earth and Planetary Science Letters 302 :185–193

Brown SR (1987) A note on the description of surface roughness using fractal dimension. Geophys. Res. Lett. 14:1095-1098

Candela T, Renard F, Bouchon M, Marsan D, Schmittbuhl J, Voisin C (2009) Characterization of fault roughness at various scales: Implications of three-dimensional high resolution topography measurement, Pure Appl Geophys 166(10):1817–1851

Candela T, Renard F, Klinger Y, Mair K, Schmittbuhl J, Brodsky E (2012) Roughness of fault surfaces over nine decades of length scales. Journal of Geophysical Research, vol. 117, B08409, https://doi.org/10.1029/2011JB009041

Chen SJ, Zhu WC, Zhang MS, Yu QL (2012) Fractal description of rock joints based on digital image processing technique. Chin J Geotech Eng 34(11): 2087-92.

Cooley JW, Tukey JW (1965) An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation. 19 (90):297–301. https://doi.org/10.1090/S0025-5718-1965-0178586-1. ISSN 0025-5718

Den Outer A, Kaashoek J, Hack H (1995) Difficulties with using continuous fractal theory for discontinuity surfaces. Proceedings of International journal of rock mechanics and mining sciences & geomechanics abstracts, 1995. Elsevier

Fourier J (1822). Théorie analytique de la chaleur (in French). Paris: Firmin Didot Père et Fils. OCLC 268808

Gauss CF (1866) "Nachlass, Theoria Interpolationis Methodo Nova Tractata," in Carl Friedrich Gauss Werke, Band 3, K6niglichen Gesellschaft der Wissenschaften: G6ttingen, pp. 265-330, 1866.

Golden software (2018) http://www.goldensoftware.com/products/surfer

Goldstine HH (1977) A History of Numerical Analysis from the 16th Through the 19th Century. Berlin, Heidelberg, and New York: Springer-Verlag, 1977.

Jang HS, Kang SS, Jang BA (2014) Determination of Joint Roughness Coefficients Using Roughness Parameters. Rock Mech Rock Eng 47:2061–2073. https://doi.org/10.1007/s00603-013-0535-z

Johansson F, Stille H (2014) A conceptual model for the peak shear strength of fresh and unweathered rock joints. International Journal of Rock Mechanics and Mining Sciences 69:31-38

Johnson R, Freund J and Miller I (2011) Miller and Freund's Probability and Statistics for Engineers, 8th edition. Pearson Education, Boston. ISBN-13: 978-0-321-69498-0

Kulatilake PHSW, Balasingham P, Park J, Morgan R (2006) Natural rock joint roughness quantification through fractal technique. Geotech. Geol. Eng. 24:1181-1202

Li Y, Huang R (2015) Relationship between joint roughness coefficient and fractal dimension of rock fracture surfaces. International Journal of Rock Mechanics and Mining Sciences 75:15–22

Lyons R (2015) Four Ways to Compute an Inverse FFT Using the Forward FFT Algorithm, https://www.dsprelated.com/showarticle/800.php, Accessed 27 February 2017

Malinverno A (1990) A simple method to estimate the fractal dimension of a self affine series. Geophys Res Lett 17:1953–6

Mandelbrot BB (1985) Self-Affine Fractals and Fractal Dimension. Phys Scr 32:257-260

Myers NO (1962) Characteristics of surface roughness. Wear 5, 182-189.

Odling NE (1994) Natural Fracture Profiles, Fractal Dimension and Joint Roughness Coefficients. Rock Mech Rock Eng 27 (3):135-153

Renard F, Voisin C, Marsan D, Schmitbuhl J (2006) High resolution 3D laser scanner measurements of a strike-slip fault quantify its morphological anisotropy at all scales, Geophys Res Lett 33, L04305, https://doi.org/10.1029/2005GL025038.

Russ J (1994) Fractal Surfaces. Plenum Press, New York. ISBN 0-306-44702-9

Smith SW (1997) The Scientist and Engineer's Guide to Digital Signal Processing. San Diego, CA: California Technical Publishing 1997 ISBN-13: 978-0966017632

Stigsson M (2018) Finally, an objective way to infer JRC from digitized fracture traces. In: Proceedings of the 52nd US Rock Mechanics/Geomechanics Symposium, USA: Seattle; 2018. Paper ARMA 18-416. Available at: http://www.skb.com/article/2491457/Paper_ARMA_18-416.pdf

Stigsson M (2015) Parameterization of fractures - Methods and evaluation of fractal fracture surfaces. TR 15-27. POSIVA OY. Available at: http://www.posiva.fi/en/databank/workreports?xm_freetext=stiggson

Tse R, Cruden DM (1979) Estimating joint roughness coefficients. Int J Rock Mech Min Sci Geomech Abstr 16:303–7

Turk N, Greig MJ, Dearman WR, Amin FF (1987) Characterization of joint surfaces by fractal dimension. In: Proceedings of the 28th US rock mechanics symposium. Tucson; 1987. 1223–36

Wakabayashi N, Fukushige I (1992) Experimental study on the relation between fractal dimension and shear strength. In:Proceedings of the international symposium for fractured and joint rock masses. Berkeley: California 101–110

Yu X, Vayssade B (1991) Joint profiles and their roughness parameters. Int J Rock Mech Min Sci Geomech Abstr 1991; 28:333-336

Appendix A

Derivation of σδh(1p) for a single sine wave

A discretised single sine wave can be described as

$$y(x) = A \cdot \sin\left(2\pi \cdot f \frac{x}{N}\right) \qquad \text{eq. A-1}$$

Where
A = amplitude
f = frequency
N = number of vertices of the full wave
x = vertex number

The difference in height between 2 adjacent vertices is described by

$$\delta h(1p) = y(x+1) - y(x) \qquad \text{eq. A-2}$$

The standard deviation of all height differences between 2 adjacent vertices is

$$\sigma\delta h(1p) = \sqrt{E\left(\delta h(1p)^2\right) - E\left(\delta h(1p)\right)^2} \qquad \text{eq. A-3}$$

Where
$\sigma\delta h(1p)$ = standard deviation of height differences between 2 adjacent vertices
$E(\delta h(1p)^2)$ = Expected value of the squared differences
$E(\delta h(1p))$ = Expected value of the differences

The expected value of the squared differences is simply the mean of the squared differences, which can be expressed as the summation

$$E\left(\delta h(1p)^2\right) = \frac{\sum_{x=0}^{N-1}\left(y(x+1) - y(x)\right)^2}{N} \qquad \text{eq. A-4}$$

In the same way the expected value of the differences is the mean of the differences:

$$E\left(\delta h(1p)\right) = \frac{\sum_{x=0}^{N-1} y(x+1) - y(x)}{N} \qquad \text{eq. A-5}$$

Expanding the summation in eq. A-5 and recalling that $y(N) = y(0)$ gives

$$\frac{\sum_{x=0}^{N-1} y(x+1) - y(x)}{N} = \frac{\left(y(1) - y(0)\right) + \left(y(2) - y(1)\right) + \ldots + \left(y(N) - y(N-1)\right)}{N} = \frac{0}{N} \qquad \text{eq. A-6}$$

This is also intuitive since the summation is done over full waves. Hence eq. A-3 can be reduced to

$$\sigma\delta h(1p) = \sqrt{E\left(\delta h(1p)^2\right)} \qquad \text{eq. A-7}$$

Inserting eq. A-1 and eq. A-2 into eq. A-7 gives

$$\sigma\delta h(1p) = \sqrt{\frac{\sum_{x=0}^{N-1}\left(A\cdot\sin\left(2\pi\frac{f}{N}(x+1)\right) - A\cdot\sin\left(2\pi\frac{f}{N}x\right)\right)^2}{N}} \qquad \text{eq. A-8}$$

Squaring both sides of eq. A-8 and rearranging gives

$$\left(\sigma\delta h(1p)\right)^2\frac{N}{A^2} = \sum_{x=0}^{N-1}\left(\sin\left(2\pi\frac{f}{N}(x+1)\right) - \sin\left(2\pi\frac{f}{N}x\right)\right)^2 \qquad \text{eq. A-9}$$

Expanding the right side of eq. A-9 gives

$$\sum_{x=0}^{N-1}\left(\sin^2\left(2\pi\frac{f}{N}(x+1)\right) - 2\cdot\sin\left(2\pi\frac{f}{N}(x+1)\right)\sin\left(2\pi\frac{f}{N}x\right) + \sin^2\left(2\pi\frac{f}{N}x\right)\right) \qquad \text{eq. A-10}$$

Performing the summation over multiples of whole waves the equality

$$\sum_{x=0}^{N-1}\sin^2\left(2\pi\frac{f}{N}(x+1)\right) = \sum_{x=0}^{N-1}\sin^2\left(2\pi\frac{f}{N}x\right) \qquad \text{eq. A-11}$$

holds, and hence eq. A-10 can be written as

$$\sum_{x=0}^{N-1}2\cdot\sin^2\left(2\pi\frac{f}{N}x\right) - \sum_{x=0}^{N-1}2\cdot\sin\left(2\pi\frac{f}{N}(x+1)\right)\sin\left(2\pi\frac{f}{N}x\right) \qquad \text{eq. A-12}$$

The right summation in eq. A-12 can be expanded and developed according to eq. A-13 to eq. A-18

$$2\cdot\sin\left(2\pi\frac{f}{N}x + 2\pi\frac{f}{N}\right)\sin\left(2\pi\frac{f}{N}x\right) \qquad \text{eq. A-13}$$

Using the relationship

$$\sin(a+b) = \sin(a)\cos(b) + \cos(a)\sin(b) \qquad \text{eq. A-14}$$

eq. A-13 can be expanded to

$$2\cdot\left(\sin\left(2\pi\frac{f}{N}x\right)\cos\left(2\pi\frac{f}{N}\right) + \cos\left(2\pi\frac{f}{N}x\right)\sin\left(2\pi\frac{f}{N}\right)\right)\sin\left(2\pi\frac{f}{N}x\right) \qquad \text{eq. A-15}$$

And further expanded by moving the last sine term inside the parenthesis

$$2\cdot\sin^2\left(2\pi\frac{f}{N}x\right)\cos\left(2\pi\frac{f}{N}\right) + 2\cdot\cos\left(2\pi\frac{f}{N}x\right)\sin\left(2\pi\frac{f}{N}\right)\sin\left(2\pi\frac{f}{N}x\right) \qquad \text{eq. A-16}$$

Using the relationship

$$\cos(a)\sin(a)=\frac{1}{2}\sin(2a) \qquad \text{eq. A-17}$$

eq. A-16 can be written as

$$2\cdot\sin^2\left(2\pi\frac{f}{N}x\right)\cos\left(2\pi\frac{f}{N}\right)+\sin\left(4\pi\frac{f}{N}x\right)\sin\left(2\pi\frac{f}{N}\right) \qquad \text{eq. A-18}$$

Inserting eq. A-18 into eq. A-12 gives

$$\sum_{x=0}^{N-1}2\cdot\sin^2\left(2\pi\frac{f}{N}x\right)-\sum_{x=0}^{N-1}2\cdot\sin^2\left(2\pi\frac{f}{N}x\right)\cos\left(2\pi\frac{f}{N}\right)+ \\ +\sum_{x=0}^{N-1}\sin\left(4\pi\frac{f}{N}x\right)\sin\left(2\pi\frac{f}{N}\right) \qquad \text{eq. A-19}$$

Recalling that the last term in the last summation is a constant and the summation of values over a multiple of full waves is zero the last summation is zero. Hence eq. A-19 can be reduced to

$$\sum_{x=0}^{N-1}2\cdot\sin^2\left(2\pi\frac{f}{N}x\right)\cdot\left(1-\cos\left(2\pi\frac{f}{N}\right)\right) \qquad \text{eq. A-20}$$

Since the second term is constant and the first can be transformed using the relationship

$$2\sin^2(a)=1-\cos(2a) \qquad \text{eq. A-21}$$

eq. A-20 can be rewritten as

$$\left(1-\cos\left(2\pi\frac{f}{N}\right)\right)\cdot\sum_{x=0}^{N-1}1-\cos\left(4\pi\frac{f}{N}x\right) \qquad \text{eq. A-22}$$

Again the second term in the summation is zero for any summation over a multiple of full waves and the firs term sums to N, hence eq. A-22 simply becomes

$$\left(1-\cos\left(2\pi\frac{f}{N}\right)\right)\cdot N \qquad \text{eq. A-23}$$

Using the relationship in eq. A-21, eq. A-23 can be further reduced to

$$2\cdot\sin^2\left(\pi\frac{f}{N}\right)\cdot N \qquad \text{eq. A-24}$$

Substituting the right part in eq. A-9 with the results in eq. A-24

$$\left(\sigma\delta h(1p)\right)^2\frac{N}{A^2}=2\cdot\sin^2\left(\pi\frac{f}{N}\right)\cdot N \qquad \text{eq. A-25}$$

And hence

$$\sigma\delta h(1p) = \sqrt{2} \cdot A \cdot \sin\left(\pi \frac{f}{N}\right) \qquad \text{eq. A-26}$$

Appendix B

Extra codes

```
'================================================================================
'===                            calcAmplConst                                 ===
'================================================================================
'===  Function that Calculate the amplitude of the sine wave for the     ===
'===  length frequency 1 according to equation 4-10                      ===
'================================================================================
'====                                                                      ====
'===  Input:                                                              ===
'===  sdh1p = the desired standard deviation of the differences in        ===
'===          height between adjacent points on the fractal line          ===
'===  size  = the number of data points of the fractal line               ===
'===  hurst = the Hurst exponent of the fractal line                      ===
'====                                                                      ====
'================================================================================
'===  Written by Martin Stigsson 16 March 2017                            ===
'===  Please, report errors or improvements to: martin.stigsson@skb.se    ===
'================================================================================
Function calcAmplConst(ByVal sdh1p As Double,
                       ByVal size As Integer, _
                       ByVal hurst As Double)

    Dim k As Integer
    Dim sig2Sum As Double
    Dim pi As Double

    'initiate variables
    sig2Sum = 0
    pi = Atan(1) * 4

    'Do the summation of all squared strandard deviations
    For k = 1 To size / 2 - 1

        'Calculate the squared standard deviation and add to the existing
        'sum
        sig2Sum = sig2Sum + (k ^ (-(hurst + 0.5)) * Sin(pi * k / size)) ^ 2

    Next

    calcAmplConst = size / (2 * 2 ^ 0.5) * sdh1p / sig2Sum ^ 0.5

End Function
```

```
    '================================================================================
    '===                      mean and stdev from sums                            ===
    '================================================================================
    '===  Routine calculating the mean and standard deviation using the sum  ===
    '===  of the observed values and the sum of the squared observed         ===
    '===  values. The standard deviation of a population of values can be    ===
    '===  calculated according to                                            ===
    '===                                                                     ===
    '===  σ =(E|x^2|-E|x|^2)^0.5,                                            ===
    '===                                                                     ===
    '===  see e.g. https://en.wikipedia.org/wiki/Standard_deviation          ===
    '===  To transform a standard deviation of a population to a standard    ===
    '===  deviation of a sample simply multiply by (N / (N - 1))^0.5         ===
    '================================================================================
    '====                                                                    ====
    '===  Input:                                                             ===
    '===  xSum       = sum of the values                                     ===
    '===  x2Sum      = sum of the squared values                             ===
    '===  nofData    = number of values                                      ===
    '===  population = shall the standard deviation be calculated using an   ===
    '===               entire population, TRUE, or from a sample from the    ===
    '===               population, FALSE                                     ===
    '====                                                                    ====
    '===  Output:                                                            ===
    '===  mean         = mean of the data                                    ===
    '===  stdev        = standard deviation of the data                      ===
    '===  errorMessage = string containing possible errorMessage             ===
    '====                                                                    ====
    '================================================================================
    '===  Written by Martin Stigsson, March 31 2017                          ===
    '===  Please, report errors or improvements to: martin.stigsson@skb.se   ===
    '================================================================================
    Sub mean_and_stdev_from_sums(ByVal xSum As Double, _
                                 ByVal x2sum As Double, _
                                 ByVal nofData As Integer, _
                                 ByVal population As Boolean, _
                                 ByRef mean As Double, _
                                 ByRef stdev As Double, _
                                 ByRef errorMessage As String)

        Dim x2mean As Double      'the mean of the squared values
        Dim subtraction As Double 'the subtraction under the square root

        'Calculate mean
        mean = xSum / nofData

        'Calculate average square value
        x2mean = x2sum / nofData

        'Calculate the subtraction in the square root
        subtraction = x2mean - mean * mean

        'calculate the standard deviation of the population
        stdev = (subtraction) ^ 0.5

        'If sample population is desired correct the value accordingly
        If Not population Then

            'calculate the standard deviation of a sample
            stdev = stdev * (nofData / (nofData - 1)) ^ 0.5

        End If

errorTrap:

    End Sub
```

```
    '================================================================================
    '===                     regression analysis of line                          ===
    '================================================================================
    '===  Routine calculating the slope and intersection in arithmetic            ===
    '===  space using regression-analysis. The algorithm uses all of the         ===
    '===  accesible values to calculate the slope and intersection of the         ===
    '===  regression line. The R-value is also calculated for quality             ===
    '===  check. Observe that the data vectors should start at position 1         ===
    '===  and not 0                                                               ===
    '================================================================================
    '====                                                                        ====
    '===  Input:                                                                  ===
    '===  xData(1 to nofData) = the x data vector in arithmetic space             ===
    '===  yData(1 to nofData) = the y data vector in arithmetic space             ===
    '===  minNdx              = lowest vector index to regard                     ===
    '===  maxNdx              = highest vector index to regard                    ===
    '====                                                                        ====
    '===  Output:                                                                 ===
    '===  slope               = the slope of the curve in log space               ===
    '===  intersection        = the intersection of the line with the y           ===
    '===                        axis in log space                                 ===
    '===  errorMessage        = string containing possible errorMessage           ===
    '====                                                                        ====
    '================================================================================
    '===  Written by Martin Stigsson, February 26 2016                            ===
    '===  Please, report errors or improvements to: martin.stigsson@skb.se        ===
    '================================================================================
    Sub regression analysis of line(ByRef xData() As Double,
                                    ByRef yData() As Double,
                                    ByRef minNdx As Integer,
                                    ByRef maxNdx As Integer, _
                                    ByRef slope As Double, _
                                    ByRef intersection As Double, _
                                    ByRef errormessage As String)

        Dim i As Integer             'Counter
        Dim nofNdxsUsed As Integer   'number of indices that are used in the
        '                             regression analysis

        Dim Sx As Double             'Sum of x-values
        Dim Sy As Double             'Sum of y-values
        Dim Sxx As Double            'Sum of x^2 values
        Dim Syy As Double            'Sum of y^2-values
        Dim Sxy As Double            'Sum of x·y-values
        'Dim r As Double              'correlation coefficient

        'initiate the sums to be used in the regression analysis
        Sx = 0
        Sy = 0
        Sxx = 0
        Syy = 0
        Sxy = 0

        'Calculate sums
        For i = minNdx To maxNdx
            Sx = Sx + xData(i)
            Sy = Sy + yData(i)
            Sxx = Sxx + xData(i) ^ 2
            Syy = Syy + yData(i) ^ 2
            Sxy = Sxy + xData(i) * yData(i)
        Next

        'Calculate number of data that was used
        nofNdxsUsed = maxNdx - minNdx + 1

        'Calculate correlation coefficient
        'r = (nofBinsUsed * Sxy - Sx * Sy) / ((nofBinsUsed * Sxx - Sx ^ 2) * _
        '    (nofBinsUsed * Syy - Sy ^ 2)) ^ 0.5

        'Calculate slope by regresion, reference e.g.
        'http://en.wikipedia.org/wiki/Simple_linear_regression
        slope = (nofNdxsUsed * Sxy - Sx * Sy) / (nofNdxsUsed * Sxx - Sx ^ 2)

        'Calculate intercept
        intersection = Sy / nofNdxsUsed - slope * Sx / nofNdxsUsed

errorTrap:
    End Sub
```