UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

# Improvement of TestH: a C Library for Generating Self-Similar Series and for Estimating the Hurst Parameter

**Cristiano Duarte Gonçalves Ramos**

Dissertação para obtenção do Grau de Mestre em
**Engenharia Informática**
(2º ciclo de estudos)

Orientador: Prof. Doutor Pedro Ricardo Morais Inácio

**Covilhã, novembro de 2017**

*To the most important woman in my life,*
*my grandmother.*

# Acknowledgements

I would first like to thank my thesis supervisor Prof. Dr. Pedro R. M. Inácio for all the motivation he gave me in order to pursue this academic title. I would also like to thank him for always having his door open whenever it was needed.

I would like to thank my family, specially my mother, stepfather, grandmother and my girlfriend for all the support they have been giving me during this period (a kiss to each one of you for every cup of coffee you brewed for me).

To my friends, a big thank you for all the good stories we have and for giving a hand in shaping me into what I am today.

Finally, a special thanks to my beautiful blue-eyed white cat and to my rottweiler, Joana and Dóris, which give comfort and relief without ever questioning.

# Resumo

A descoberta da dependência consistente entre valores em certas séries de dados, abriu caminho para o desenvolvimento de algoritmos que permitissem, de alguma forma, classificar o grau de auto-semelhança entre valores e tecer considerações sobre o comportamento da série. A esta estatística dá-se o nome de Parâmetro de Hurst, que permite analisar e classificar o comportamento de uma série de dados como persistente, antipersistente ou puramente aleatória. Esta descoberta tem sido bastante relevante na área das redes de computadores, onde serve, p.ex., de ajuda às empresas para desenvolverem equipamentos e infraestruturas adequadas às suas necessidades. Para além do elevado interesse que a referida área apresentou por esta métrica, existem outros campos ciêntificos onde algoritmos para estimar o Parâmetro de Hurst de sequências de valores estão a ser aplicados, como por exemplo no estudo de fenómenos geológicos [KTC07], bem como em fenómenos ligados às ciências da saúde [VAJ08, HPS+12].

Existem vários algoritmos para estimar o Parâmetro de Hurst [Hur51, Hig88, RPGC06], tendo cada um deles as suas virtudes e fraquezas. A utilização destes algoritmos é por vezes difícil, motivando a criação de ferramentas e bibliotecas que os congregam e disponibilizam de uma forma mais amigável ao utilizador. Infelizmente, e apesar de ser uma área que está a ser alvo de estudos há décadas, as ferramentas existentes, para além de não implementarem a totalidade dos algoritmos mais relevantes, apresentam ainda algumas limitações. Desta forma, o trabalho apresentado nesta dissertação consiste, principalmente, na melhoria da `TestH`, uma biblioteca escrita em ANSI C para o estudo de séries temporais auto-semelhantes, inicialmente desenvolvida por Fernandes *et al*. [FNS+14]. Estas melhorias materializam-se sobretudo na adição de algoritmos para estimar o Parâmetro de Hurst e gerar séries de dados auto-semelhantes. Adicionalmente foram introduzidas funções auxiliares, foi efetuada a refactorização do código, documentação das *interfaces* de programação e ainda a criação de um sítio *web* para divulgação do projeto.

Esta dissertação dá enfase aos algoritmos de estimação do Parâmetro de Hurst e geração de séries auto-semelhantes. Relativamente à estimação, foram introduzidos na `TestH`, no âmbito deste trabalho, o Periodograma, o método de Higuchi, a estimação através da função de autocorrelação e o método de análise através da remoção das tendências. No que respeita à geração de séries, foi também introduzido o método de Davies e Hart. Com o objetivo de tornar a `TestH` robusta e credível, foram realizados vários testes, comparando os resultados destas implementações com os valores fornecidos por ferramentas semelhantes. Os resultados obtidos estão alinhados com o esperado e, inclusivamente, os algoritmos que se encontram implementados na `TestH` e restantes ferramentas analisadas (como por exemplo, o Periodograma), apresentaram valores bastante semelhantes entre si, corroborando a crença da correção da implementação dos vários métodos.

# Palavras-chave

TestH, Passeio Aleatório, movimento Browniano fracionário, ruído Gaussiano fracionário, Dependência de Longo Alcance , Auto-Semelhança, Geradores de Sequências Auto-Semelhantes, Estimadores do Parâmetro de Hurst.

# Resumo alargado

## Introdução

Escrita em língua Portuguesa, esta secção pretende expor o corpo da dissertação de uma forma mais detalhada que o resumo, estando a sua organização alinhada com a estrutura do corpo deste documento. Na primeira subsecção é feito um enquadramento da dissertação, seguida pelas subsecções que definem o problema abordado, os objetivos propostos a alcançar, a abordagem adotada para os realizar e uma breve discrição da materialização do trabalho feito ao longo da dissertação. De seguida é abordada uma secção que define alguns preliminares relevantes para o estudo desta área científica bem como uma análise a trabalhos similares. Nas últimas subsecções é feita uma apresentação dos algoritmos implementados bem como uma avaliação dos mesmos, terminando a secção com um resumo das conclusões e direções a seguir para o trabalho futuro.

## Motivação e Âmbito

Desde o inicio dos tempos o ser humano sempre teve a ambição de conseguir prever o futuro de certos eventos, sejam estes o estado do tempo nos dias seguintes ou o comportamento de uma ação no mercado bolsista. Embora em certas situações possa aparentar, estas previsões não são ingénuas e resultam de uma reflexão sobre os valores no passado. Por outras palavras, esta reflexão indica que os valores futuros têm um certo grau de dependência dos valores passados, sendo este fenómeno conhecido como dependência de longo alcance (*Long Range Dependence*) e que, ocorre quando a dependência entre valores espaçados no tempo decresce mais suavemente que uma decadência exponencial. Esta auto-semelhança entre os valores de uma série de dados pode ser medida através da estatística conhecida como Parâmetro de Hurst, o qual é descrito em maior detalhe no capítulo 2. De uma forma bastante resumida o Parâmetro de Hurst permite determinar com um certo grau de certeza se uma série apresenta um comportamento persistente (tendência nos valores), anti-persistente (inversão da tendência) ou se é simplesmente imprevisível.

Esta área cientifica foi particularmente impulsionada devido à descoberta de propriedades de auto-semelhança no tráfego das redes de computadores por Leland *et al*. [LTWW93]. Estes avanços têm ajudado as companhias ligadas à área de redes a planificarem melhor a sua infraestrutura e a desenvolverem melhores dispositivos, poupando assim recursos e melhorando a experiencia de utilização dos clientes. Existem outras áreas onde o estudo da auto semelhança tem sido amplamente aplicado, como o estudo de fenómenos naturais [KTC07] ou ainda estudos na área das ciências da saúde [VAJ08, HPS+12]. Na literatura é possível encontrar diversos algoritmos [Hur51, Hig88, RPGC06] para estimar o Parâmetro de Hurst, cada um deles tendo as suas virtudes e fraquezas. Para além disso, parte destes algoritmos são parametrizáveis, deixando espaço para diferentes implementações. Infelizmente as ferramentas para a análise de séries

de dados auto-semelhantes que implementam estes algoritmos são de certa forma limitadas, levando a que os investigadores por vezes optem por fazer as suas próprias implementações. Para além dos possíveis erros que podem resultar da implementação os investigadores ainda têm que dispor do seu templo para implementar algo que não está diretamente relacionado com a seu trabalho.

Com o cenário apresentado, o trabalho referente a esta dissertação consiste na introdução de melhorias na TestH, uma biblioteca escrita em ANSI C para o estudo de séries auto-semelhantes, inicialmente desenvolvida por Fernandes *et al*. [FNS+14]. Estas melhorias passam pela adição à biblioteca de geradores de séries auto-semelhantes e de estimadores do Parâmetro de Hurst, adição de funções de apoio, refactorização do código, documentação do mesmo e ainda a criação de um sítio *web* para divulgação do projeto.

## Problema e Objetivos

O problema identificado é a falta de uma ferramenta para o estudo de séries auto-semelhantes que tenha sido amplamente testada, seja possível a sua adaptação e ainda que o seu código seja disponibilizado de forma livre. Como foi identificado anteriormente, a falta de uma ferramenta com estas característica leva a que os investigadores tenham que implementar as suas versões dos algoritmos, fazendo com que conhecimentos de programação sejam quase um requisito para esta área. Para os investigadores que não possuem estes conhecidos existem ferramentas com *interfaces* gráficas, contudo, estas apresentam limitações (discutidas no capítulo 3). Parte destas ferramentas também não providenciam nenhuma informação relativa à maneira de implementação dos algoritmos, o que pode criar uma certa desconfiança relativamente aos resultados por elas apresentados.

A `TestH` é uma biblioteca pensada para investigadores que ambiciona ser robusta, amplamente testada, facilmente adaptável e que tenha uma maneira de utilização bastante simples. A escolha da criação de uma biblioteca em vez de uma aplicação com *interface* gráfica foi pensada para permitir aos investigadores que possuam conhecimentos de programação uma maior adaptação às suas necessidades. Adicionalmente, a *interface* de programação da biblioteca encontra-se devidamente documentada, possibilitando o seu uso por investigadores que possuam conhecimentos reduzidos de C. Com o problema e os objetivos identificados, as tarefas propostas para este trabalho são:

1. implementação de estimadores do Parâmetro de Hurst;
2. implementação de geradores de sequências auto-semelhantes;
3. refactorização do código;
4. implementação de um sistema de geração de documentação da *interface* de programação;

5. construção de um sítio *web* para divulgação do projeto e hospedagem da documentação;
6. validação dos algoritmos implementados através de testes e comparação com as ferramentas similares disponíveis.

## Abordagem para Resolver o Problema

Para resolver o presente problema e para atingir os objetivos propostos acima mencionados, a estratégia definida na seguinte lista foi seguida:

1. etapa 1 – a familiarização com os conceitos inerentes a esta dissertação (como por exemplo, a auto-semelhança) foi o primeiro paço tomado para a realização da mesma. Esta tarefa incluiu também um estudo profundo do estado da arte, seguido pela recolha da informação referente aos algoritmos a implementar;
2. etapa 2 – com uma grande extensão de código já implementado e sem documentação, a segunda etapa consistiu na compreensão da arquitetura da biblioteca. Ao longo deste estudo foram identificadas oportunidades de melhorias. Foi também efetuado uma comparação das ferramentas disponíveis com a `TestH`, visando a extração de ideias passíveis de implementação;
3. etapa 3 – neste paço foi efetuada a implementação dos algoritmos. Por motivos de rapidez de execução e desenvolvimento, estes foram implementados à parte da biblioteca, tendo sido posteriormente integrados na `TestH`. Em termos técnicos, a abordagem seguida para a implementação dos mesmos foi a de desenvolvimento orientado por testes. Ou seja, foram desenhados testes previamente, com a implementação a ser considerada incompleta enquanto algum deles não fosse satisfeito positivamente;
4. etapa 4 – a biblioteca já beneficiou de várias modificações feitas por diversos contribuidores. Contudo, estas nem sempre tiveram em atenção a uniformização do código. De encontro a este problema, neste paço foi efetuada uma refactorização do código;
5. etapa 5 – por fim seguiu-se a fase de validação dos algoritmos, onde cada um deles foi submetido a vários testes de forma a averiguar a sua correta implementação.

## Principais Contribuições

A principal contribuição resultante deste trabalho é a melhoria de uma ferramenta de cariz científico que pode vir a servir como um meio para o progresso neste campo. Segue-se uma breve discrição das principais contribuições:

1. a primeira contribuição é a implementação de cinco novos algoritmos, quatro relativos a estimadores do Parâmetro de Hurst e um relativo à geração de sequências auto-semelhantes;

2. para além dos algoritmos previamente referidos, outras funções foram adicionadas, como por exemplo a transformada rápida de Fourier ou as somas cumulativas;

3. o código foi refactorizado para tornar mais harmoniosa a sua adaptação e melhorar a sua legibilidade;

4. de uma perspetiva funcional, a *interface* de programação da biblioteca foi documentada com recurso à Clang based DOCumentation generator (CLDOC), uma ferramenta de código aberto de geração de documentação;

5. por último, um sítio *web* para divulgação da biblioteca e consulta da documentação foi contruído, o qual pode ser acedido através da seguinte ligação: `https://cdgramos.github.io/testh/`.

## Preliminares

É sempre importante relembrar os eventos históricos que levaram a descoberta do Parâmetro de Hurst. Durante os estudos para a construção de uma barragem no rio Nilo que pudesse guardar eficientemente a água durante os períodos de seca ou de cheia, Harold Hurst percebeu que era comum anos de cheia serem precedidos por anos de cheias, e que o mesmo se aplicava para as secas. Este tipo de semelhança indicou que as observações presentes têm um certo grau de dependência das observações efetuadas no passado, o que o levou a criar um método que a permitisse medir esta dependência, dando assim origem à unidade estatística que hoje carrega o seu nome, o Parâmetro de Hurst. Esta estatística $H$ assume valores entre $]0, 1[$, e dependendo do intervalo em que se encontre pode indicar:

- para $H \in ]0, 0.5[$, o processo é considerado como anti persistente (também denominado por negativamente correlacionado), o que significa que uma tendência verificada irá inverter no futuro próximo;

- para $H = 0.5$, o processo é classificado como sem memória [Sam06], ou seja, impossível de prever o seu comportamento (p.ex. um passeio aleatório);

- para $H \in ]0.5, 1.0[$ o processo é considerado como persistente (também denominado como correlacionado) e indica que a tendência verificada na série irá continuar a verificar-se.

## Trabalho Relacionado e TestH

A `TestH` viu o seu desenvolvimento começar em 2013 mas está longe de ser a primeira ferramenta do género. A SELF-similarity analysIS (SELFIS), Self-similarity and Quality of Service (SELQoS) e Herramienta de Análisis de Tráfico Telemático Autosemejante (HATTA) surgiram com o objetivo de estudar e simular dados de tráfego de redes informáticas, enfatizando novamente a relevância deste campo científico. Estas ferramentas são as mais referenciadas na literatura,

tendo sido por isso escolhidas para serem objeto de análise nesta dissertação (capítulo 3). No entanto, existem mais ferramentas como por exemplo a Local Analysis of Self-Similarity (LASS) [STP+06]. A partir da análise da literatura e do estudo das ferramentas, é possível notar que os estimadores mais comuns, como o Rescaled Range Statistics (R/S), estão implementados em todas as ferramentas acima mencionadas. No entanto, não é garantido que os resultados de um algoritmo específico, aplicado às mesmas séries temporais, sejam iguais entre eles. Isto deve-se à variedade de possíveis implementações de um algoritmo (p. ex., tamanhos de bloco definidos de forma estática ou dinâmica) ou até mesmo a possíveis erros que possam ter acontecido aquando da sua implementação.

A ideia chave da `TestH` é a de fornecer aos investigadores uma ferramenta robusta, mas de simples uso, para o estudo da auto-semelhança em séries temporais. Esta fácil utilização é alcançada com a camada de abstração fornecida pela *interface* de programação, que permite a utilização da biblioteca sem conhecimentos profundos de programação. Até ao momento, a biblioteca apresenta uma coleção de oito estimadores de Parâmetro de Hurst, seis geradores de sequências auto-semelhantes, baterias de teste e capacidade de usar dados de geradores externos à biblioteca.

## Geradores e Estimadores Implementados

Ao todo foram introduzidos cinco algoritmos na `TestH` durante esta dissertação, nomeadamente o Periodograma, o método de Higuchi, a estimação através da função de auto correlação e a análise através da remoção das tendências como algoritmos de estimação, e o método de Davies e Hart como algoritmo de geração de séries auto-semelhantes. Abaixo encontra-se uma breve descrição sobre cada um deles.

O Periodograma foi proposto por Geweke e Porter-Hudak [GPH83]. O algoritmo apresentado para estudar processos com memória de longo alcance baseia-se na análise do espectro $f(\lambda)$ e propõe-se a aproximá-lo por $I(\lambda) = \frac{1}{N} \sum_{j=-(N-1)}^{N-1} \hat{\gamma}(j) \times e^{ij\lambda}$, onde $N$ é o número de amostras, $\lambda$ a frequência e $\hat{\gamma}(j)$ uma estatística semelhante à função de autocorrelação. Este cálculo deve ser executado para várias frequências $k$. O valor de $H$ é obtido através do declive da linha ajustada dos mínimos quadrados ajustada aos logaritmos de $I(\lambda_k)$ e os de $\lambda_k$ , de acordo com a fórmula $H = (1 - \beta)/2$ em que $\beta$ é o declive.

Em 1988, Higuchi propôs um algoritmo para estimar o Parâmetro de Hurst que viria a carregar o seu nome [Hig88]. Semelhante a outros algoritmos, este também tira partido do uso de diferentes escalas de janela $m_k$ para aplicar a função definida pelo autor. Para extrair o Parâmetro de Hurst uma linha dos mínimos quadrados deve ser ajustada aos logaritmos dos valores dados pela função de Higuchi $L(m_k)$ e $m_k$ para diferentes $m_k \in \mathbb{N}$. O Parâmetro de Hurst é obtido adicionando $2$ ao declive da linha encontrada.

A estimação através da função de auto correlação é dos métodos mais recentes descritos na literatura para a estimação do parâmetro Hurst. Este método foi introduzido por Rezaul *et al*. [RPGC06] seguindo o trabalho de Kettani *et al*. [KG02] sobre a auto-semelhança do tráfego em redes informáticas. O algoritmo depende da função de auto correlação $\gamma(k)$. Por exemplo, se $k = 1$ for a distância da vizinhança a considerar, deve-se calcular a auto correlação entre $Y(t+1)$ e $Y(t)$. A estimativa de do Parâmetro de Hurst é obtida a partir de $H = \frac{1}{2}log_2(\gamma(1) + 1) + \frac{1}{2}$.

Ao estudar sequências de ADN, Peng textit et al. [PBH+94] propôs o método de análise através da remoção das tendências (Detrended Fluctuation Analysis (DFA)). Este método requer a definição de uma família de escalas $m_k$, para as quais a série é integrada e os seus valores ajustados à linha dos mínimos quadrados para a remoção do efeito de tendência. Esta remoção é feita para cada escala, calculando posteriormente a variância dos valores. O Parâmetro Hurst é obtido através do declive de uma linha dos mínimos quadrados ajustada aos logaritmos das escalas e aos logaritmos das médias da variância para cada escala.

À semelhança de outros algoritmos este método também carrega o nome dos seus criadores, Davies e Hart [DH87]. Este método é considerado como sendo exato, ou seja, para um dado Parâmetro de Hurst gera uma sequência de valores com a auto-semelhança pretendida. Ele assenta na definição de uma matriz circulante contendo a matriz de covariância. A matriz circulante é então fatorizada usando a transformada de Fourier. É importante realçar que este algoritmo é o mais rápido entre os três métodos exatos conhecidos, sendo o método de Hosking e o de Cholesky os outros dois.

## Avaliação

Os esforços feitos no desenvolvimento da biblioteca são justificados se a `TestH` encontrar o seu lugar entre a comunidade científica e, para conseguir isso, é necessário que os investigadores confiem nos resultados que ela produz, tornando o processo de validação crucial. Vários testes foram realizados recorrendo às ferramentas HATTA e SELFIS para obter uma confirmação da precisão dos geradores e estimadores implementados, uma vez que constituem uma boa linha de base para comparação. O primeiro tipo de algoritmos a serem testados foram os geradores devido à inexistência de ferramentas amplamente conhecidas que, para um determinado valor de $H$, produzam uma sequência auto-similar. Tendo sido estes métodos posteriormente usados para testar os estimadores. Estes resultados irão ser brevemente discutidos aqui, estando a sua apresentação e discussão presentes no capítulo 5.

Foi observado que, para o ruído anti-persistente próximo de zero, o Periodograma tem um desempenho fraco, enquanto que, por outro lado, pode ser o mais útil para estudar séries temporais persistentes. Também foi demonstrado que o tempo de processamento cresce exponencialmente para o método Higuchi, tornando-o o pior candidato para aplicações que exigem resultados em tempo real. De forma geral, os estimadores apresentaram resultados alinhados

com as espectativas e em concordância com as ferramentas similares à `TestH`.

Os geradores tiveram um comportamento notavelmente bom nos testes efetuados, reforçando a ideia de que estão corretamente implementados. Devido à elevada precisão dos métodos não há nenhuma recomendação em particular sobre qual gerador deve ser usado para séries de tamanho igual ou superior a $2^{20}$ valores. No entanto, foi verificado que para séries mais pequenas, de $2^{12}$ valores, a precisão dos algoritmos aproximados é bastante menor. É também importante notar que se o tempo for uma restrição, o método de Hosking deve ser evitado devido à sua elevada complexidade computacional.

## Conclusões e Trabalho Futuro

A literatura estudada durante este trabalho não deixa espaço para duvidas relativamente à importância do papel do Parâmetro de Hurst em diversas áreas, tais como a de modelação e simulação de tráfego de rede. Com a conclusão desta dissertação, a `TestH` passou a incorporar mais geradores e estimadores do que as outras ferramentas aqui estudadas. No entanto, há oportunidades de melhoria (como por exemplo a introdução de optimizações para processamento multi-núcleo) a serem consideradas para tornarem a biblioteca cada vez mais robusta.

Como trabalho futuro é proposta a adição dos restantes algoritmos de estimação, bem como a implementação do método de Cholesky (único gerador exato em falta). Uma grande vantagem das ferramentas analisadas é a produção de gráficos, algo impossível de realizar em ANSI C, porém deve-se considerar a integração da biblioteca com a *GNU Plot*, uma ferramenta de código livre para o efeito. O crescimento do código também o torna mais suscetível a erros, por isso como contribuição futura deve ser criado um novo módulo dedicado a testes unitários.

Por último, mas não menos importante, o objetivo final deste trabalho só é cumprido se a ferramenta for efetivamente usada, por isso o seu uso deve ser encorajado junto da comunidade científica, que validará a implementação e fará possíveis contribuições para o código-fonte.

# Abstract

The discovery of consistent dependencies between values in certain data series paved the way for the development of algorithms that could, somehow, classify the degree of self-similarity between values and derive considerations about the behavior of these series. This self-similarity metric is typically known as the Hurst Parameter, and allows the classification of the behavior of a data series as persistent, anti-persistent, or purely random. This discovery was highly relevant in the field of computer networks, inclusively helping companies to develop equipment and infrastructure that suit their needs more efficiently. The Hurst Parameter is relevant in many other fields, and it has been for exemple applied in the study of geologic phenomena [KTC07] or even on areas related with health sciencies[VAJ08, HPS+12].

There are several algorithms for estimating the Hurst Parameter [Hur51, Hig88, RPGC06], and each one of them has its strengths and weaknesses. The usage of these algorithms is sometimes difficult, motivating the creation of tools or libraries that provide them in a more user-friendly manner. Unfortunately, and despite of being an area that has been studied for decades, the tools available have limitations and do not implement all algorithms available in the literature. The work presented in this dissertation consists on the improvement of `TestH`, a library written in ANSI C for the study of self-similarity in time series, which was initially developed by Fernandes *et al*. [FNS+14]. These improvements are materialized as the addition of algorithms to estimate the Hurst Parameter and to generate self-similar sequences. Additionally, auxiliary functions were implemented, along with code refactoring, documentation of the application programming interface and the creation of a website for the project.

This dissertation is mostly focused on the algorithms that were introduced in `TestH`, namely the Periodogram, the Higuchi method, the Hurst Exponent by Autocorrelation Function and the Detrended Fluctuation Analysis estimators, and the Davies and Hart method for generating self-similar sequences. In order to turn `TestH` into a robust and trustable library, several tests were performed comparing the results of these implementations with the values provided by similar tools. The overall results obtained in these tests are in line with expectations and the algorithms that are simultaneously implemented in `TestH` and in the other tools analyzed (for example, the Periodogram) returned very similar results, corroborating the belief that the methods were well implemented.

# Keywords

TestH, Random Walk, fractional Brownian motion, fractional Gaussian noise, Long Range Dependence, Self-Similarity, Self-Similar Sequences Generator, Hurst Parameter Estimators.

# Contents

# List of Figures

# List of Tables

# Acronyms

| | |
|---|---|
| **4SG** | Simple Self-Similar Sequences Generator |
| **ACM** | Association for Computer Machinery |
| **AMT** | Absolut Moments Time |
| **API** | Application Programming Interface |
| **AV** | Abry-Veitch |
| **BSD** | Berkeley Software Distribution |
| **CCS** | Computing Classification System |
| **CDF** | Cumulative Distribution Function |
| **CLDOC** | Clang based DOCumentation generator |
| **DFA** | Detrended Fluctuation Analysis |
| **DNA** | Deoxyribonucleic Acid |
| **EBP** | Embedded Branching Process |
| **FARIMA** | Fractional Autoregressive Integrated Moving Average |
| **fBm-SGA** | Fractional Brownian motion Sequential Generation Algorithm |
| **fBm** | Fractional Brownian Motion |
| **FFT** | Fast Fourier Transform |
| **fGn** | Fractional Gaussian Noise |
| **GUI** | Graphical User Interface |
| **HATTA** | Herramienta de Análisis de Tráfico Telemático Autosemejante |
| **HEAF** | Hurst Exponent by Autocorrelation Function |
| **HTML** | HyperText Markup Language |
| **HIG** | Higuchi |
| **LAN** | Local Area Network |
| **LASS** | Local Analysis of Self-Similarity |
| **LRD** | Long Range Dependence |
| **LMS** | Least Means Square |
| **LU** | Lower Upper |
| **MLE** | Maximum Likelihood Estimation |
| **PDF** | Probability Density Function |
| **PER** | Periodogram |
| **Ph.D.** | Doctor of Philosophy |
| **PRNG** | Pseudo Random Number Generator |
| **RAM** | Random-access Memory |
| **R/S** | Rescaled Range Statistics |

**SELFIS** SELF-similarity analysIS

**SELQoS** Self-similarity and Quality of Service

**SRD** Short Range Dependence

**TDD** Test Driven Development

**UBI** Universidade da Beira Interior

**URL** Uniform Resource Locator

**VT** Variance Time

# Chapter 1

# Introduction

This dissertation addresses problems related with the estimation of the self-similarity on time series. In section 1.1 of this chapter, the motivation behind this work along with a brief contextual explanation regarding its importance is discussed. The problems addressed in this work, the objectives and the adopted solution to solve them will also be subject of discussion in section 1.2 and 1.3. Close to the end o this chapter, a summary of the main contributions to the specific field is included, in section 1.4, followed by a description of the dissertation organization, in section 1.5.

## 1.1 Motivation and Scope

From knowing the next day weather to predicting the behavior of the stock market, humans have always had a natural desire to predict the future. The meaning of prediction however can be vague and prone to errors. Additionally, one does not simply predict the future accurately without taking into account the past observations, unless of course, it is a lucky shot. This action of looking to the past in a time series in order to understand the future implies that future observation values possess a certain degree of dependence from the values previously observed. This kind of phenomenon is typically referred to as Long Range Dependence (LRD) [R+02], also known as long memory process. This self-similarity between values can be measured in a way that it is possible to tell if a series will continue with the same behavior as observed before, if it will invert its behavior or if it is simply unpredictable. The Hurst parameter [Cle06] constitutes a metric for measuring such property and it will be described with more detail in chapter 2.

The discovery of self-similar properties in the networks traffic by Leland *et al*. [LTWW93] has been pushing this scientific field forward during the past decades. Doing a prior analysis and modeling of the network traffic, companies are better informed to develop network devices or plan their infrastructure, thus saving money, improving the end user experience and avoiding possible network bottlenecks. Even though the computer networks in one of the main fields where this studies have been applied, it will be shown ahead that other fields of expertise have been making progress in this area as well, such as the fields of seismology [KTC07] and health-care [VAJ08, HPS+12].

There are several algorithms described in the literature to estimate the Hurst Parameter value, with each one having its own weaknesses and strengths [Hur51, Hig88, RPGC06]. Besides that,

there is a huge range of different possibilities to implement each of them. This number can increase if one takes into account that some of them have input values where no proper study was conducted to decide how they should be optimally adjusted. Unfortunately, there are only a few tools available implementing these algorithms [PRV08], and researchers are sometimes forced to implement them. With this scenario, the researchers are forced to implement those methods themselves, which can lead to biased values since no proper testing have been made.

Having in mind all these contigencies, the scope of the present work is to improve the library `TestH`, which has been initially developed by Fernandes *et al*. [FNS+14]. These improvements aim to provide more algorithms to estimate the Hurst Parameter, the addition of new self-similar sequences generators, and to refactor and document the code that is already in place. It is also under the scope of this dissertation a proper validation of the implemented algorithms to this date. This work involves the fields of computer science and statistical mathematics and according to the Computing Classification System (Computing Classification System (CCS)) created by the Association for Computer Machinery (Association for Computer Machinery (ACM)), this work could be categorized as the following:

- **Mathematical software**∼**Statistical software**;
- *Mathematical software*∼*Time series analysis*;
- Information systems∼Similarity measures.

## 1.2 Problem Statement and Objectives

The problem identified in this work is the lack of a trustworthy, adaptable and open source tool to be easily used by those working with or studying the self-similarity in time series. It is often observed in the literature that researchers have to implement themselves the algorithms as mentioned in the previous section, which may have a negative impact on their work as there may be biases due to implementation faults. Even if the implemented algorithm is heavily tested and reviewed this may produce a loss of focus of what is the real work. This also implies that the researcher needs to possess a considerable set of programming skills to implement the algorithms. Those who do not know how to program are forced to use third party tools, which may not provide a description on how the algorithms were implemented nor the source code of the algorithm. Once again, even if the researcher uses an available tool the algorithms may output biased results and has to be limited to the capabilities of the tool, such as the length of the time series.

`TestH` is a library with its objectives well defined, and as it is the scope of this work to improve it, its objectives were inherited. The main objective is to provide the researchers with a robust, tested, adaptable and user friendly tool in order to ease their work, therefore al-

lowing a better focus on their core objective. The choice of building a library instead of a program with a Graphical User Interface (Graphical User Interface (GUI)) was also carefully considered. This allows the users that possess programming knowledge to manipulate the code to fit their needs, such as the batch processing of a set of time series. Additionally, the library has a well-documented user friendly application programming interface (Application Programming Interface (API)) which simplifies its usage.

With the problem and the core objective identified, the tasks set up for this work were:

1. Implementation of missing Hurst Parameter estimators;
2. Implementation of missing self-similar sequence generators;
3. Refactoring the existing code;
4. Implement a system to generate the API documentation;
5. Building a website with the documented API;
6. Validating the implemented algorithms against the other tools available.

## 1.3 Adopted Approach for Solving the Problem

In order to solve the identified problems and to successfully achieve the above-mentioned objectives, a strategy based on the following steps was pursued during this project:

1. Step 1 – the familiarization with the concepts such as the self-similarity, long range dependence, amongst others not so trivial at a first glance. This step included a deeper study of the state of the art, followed by the gathering of information regarding the algorithms to be implemented;
2. Step 2 – with code already in place, the next step was to get into `TestH` code in order to understand its architecture, implementation techniques and possible points of improvements. A comparison with the other tools available was also made at this stage;
3. Step 3 – this step consisted in the implementation of the algorithms, both for the Hurst Parameter estimation and for generating self-similar sequences, always favoring the description given by their original author. For simplicity and debugging purposes, the algorithms were first implemented separately, and then integrated on `TestH`. The Test Driven Development (Test Driven Development (TDD)) technique was used as the driving force of this part of the work. It was proposed by American Software Engineer Kent Beck [Bec03], which states that the tests should be made ahead of the implementation, and that the implementation is only considered completed after all tests are positive;

4. Step 4 – the refactoring and documentation of the code was the fourth step. To publish the documented API, a website where the user could search for a specific function was also build;

5. Step 5 – the last step, but perhaps the most important was the validation of the algorithms implemented on `TestH` up to the date in which this dissertation was written. This validation included running several tests on `TestH` and its peers, and also performing a comparative analysis of the results of those tests.

## 1.4 Main Contributions

The main contribution resulting from this work is the improvement of a scientific tool which may lead to additional progress in this field. The work can be summarized as follows:

1. The first contribution, directly related with the library, is the implementation of five algorithms, four of them to estimate the Hurst Parameter and another one to generate self-similar sequences;

2. Besides the core `TestH` algorithms, other general functions were added to the `TestH` API, such as the cumulative sums and the Fast Fourier Transform (FFT), which embody a valuable addition to the API as they are commonly used;

3. Another addition directly related with `TestH` concerns the refactorization of the code. As there have been additions over time to `TestH`, each of them with its own coding style, a uniformization of the code was necessary for the sake of legibility;

4. From a functional perspective the API was documented using the *clang* based documentation generator (Clang based DOCumentation generator (CLDOC)), an open source documentation generator for C and C++;

5. Finally, a website where a user can get code samples, consult the API documentation and search for a specific function was prepared and published online, which can be accessed using the following Uniform Resource Locator (URL): `https://cdgramos.github.io/testh/`. The objective of this site is also to give more visibility to the project.

The work described in this dissertation is the main subject of a journal paper under preparation. Its submission is expected to happen in the months following the discussion of this work.

## 1.5 Dissertation Overview

In order to reflect the work performed along this project, this document is decomposed in 6 chapters, identified and briefly described as follows:

1. The first chapter – **Introduction** – presents the motivation behind this work, along with its scope. It includes the identified problems and the method used to approach them. A brief summary of the contributions of this project is also included, followed by an overview of the organization of the document;

2. The second chapter – **Preliminaries** – introduces some of the concepts this work relies on, such as the self-similarity and fractal dimension. An historic overview of the Hurst parameter, its applications and limitations is also made on this chapter. This chapter is concluded with a brief description of the Hurst parameters estimators and self-similar sequence generators;

3. The third chapter – **Related Work and TestH** – contains a brief analysis conducted on tools similar or related with `TestH` followed by a description of the `TestH` directory structure and architecture. The remaining of this chapter is dedicated to a deeper analysis between some of this tools and `TestH` where, weaknesses and strengths are pointed out;

4. The fourth chapter – **Contributions** – gives a formal description of the implemented Hurst Parameter estimators and self-similar sequence generators. On this chapter, the process of documenting the code and the procedure one may follow to simply generate a new set of documentation files is also described. The chapter ends with the analysis of a small, yet executable piece of code;

5. The fifth chapter – **Evaluation** – contains the testing apparatus used for evaluating both the implemented generators and estimators. This chapter also discusses the results obtained from the tests;

6. The sixth chapter – **Conclusions and Future Work** – contains a discussion of the the main achievements and results of this work. This chapter concludes with a set of suggestions, pointing out the next steps in the landscape of the `TestH` development;

7. Appendix A – **Full Set of Results Per Estimator** – presents a table with the average results for several estimators present in TestH and in other similar tools.

# Chapter 2

# Preliminaries

## 2.1 Introduction

The main objective of this chapter is to provide an overview of some of the concepts related with self-similarity and that are required to understand the subsequent chapters. An historical perspective over the subject of self-similarity and Hurst Parameter is given in section 2.2, along with the presentation of relevant concepts such as self-similarity and random walks. This chapter concludes with a description of the most famous estimators and generators in sections 2.3 and 2.4, respectively.

## 2.2 Self-similarity and Hurst Parameter

This section contains the description of several concepts related with the self-similarity and Hurst Parameter. These concepts are deemed important for a better comprehension of this work.

### 2.2.1 Historical Overview and the Hurst Parameter

In order to better understand the concept of self-similarity, one should first consider peeking into the fractal theory, which states that an object will be perceived as the same no matter the distance it is observed from. The father of the so called fractals or fractal geometry is Benoit Mandeldrot, which introduced the concept of fractal and fractal dimension in 1967 [Man67]. Benoit believed that certain elements from nature and also some human made artifacts could be described using equations that conveyed the meaning of an form that repeats itself over different scales. After his first work on this field, he studied the roughness of the coast of Britain by describing its fractal dimension $d$.

The Hurst Parameter (also known as Hurst Exponent) inherited his name from its creator, the hydrologist Harold Hurst, and was discovered during his research on the Nile river. Hurst was seeking a way to build a perfect water reservoir that could withstand the drought periods and avoid floods. While analyzing the records from the previous years he noticed a pattern: a year with a high flood was most likely to be followed by another year with a high flood; and the same occurred for drought periods. This indicated that the results observed in a certain year had a certain degree of dependence from the ones observed in previous years. With this in mind, he came up with the Hurst Parameter, which is the unit that measures the self-similarity of a

time series, and it is defined in a way that it can probabilistic predict the future behavior of a time series given its past. Finding the Hurst Parameter $H$ can be very challenging, as some estimation methods converge slowly as the size of the observation increases. There are other factors which can compromise the results and should be taken into account, such as trends in the signal or episodic events.

Both the fractal dimension $d$ and the Hurst Parameter $H$ are related for some stochastic processes via $d = 2 - H$, where $d \in ]1, 2[$ and $H \in ]0, 1[$. The value of $H$ for a certain process can have three different meanings, which are:

- for $H \in ]0, 0.5[$, the process is said to be anti-persistent / negatively correlated, which means that the tendency verified before will the inverted in the near future;
- for $H = 0.5$, the process is said to be memory less [Sam06], thus not possible to predict its behavior (e.g. random walk);
- and for $H \in ]0.5, 1.0[$ the process is considered persistent / correlated, meaning that the values will keep following their previous behavior.

### 2.2.2 Self-similarity

A continuous-time stochastic process is considered self-similar if its statistical behavior remains closely the same upon scaling. Formally, let $X = \{X(t)\}_{t \in \mathbb{N}}$ be a continuous stochastic process. $X$ is said self-similar if

$$X(at) \overset{d}{=} a^H X(t), a \in \mathbb{R}_{>0}, \quad \textbf{(2.1)}$$

where $\overset{d}{=}$ denotes the equality of finite-dimensional distributions, $a$ is the scaling factor (a scalar) and $H$ the Hurst Parameter.

While the most common scenarios might be the ones involving continuous-time stochastic processes, for finite discrete stochastic processes with stationary increments, the first order differences of the main process can also be considered for the purpose of defining self-similarity. Let $X = \{X(t)\}_{t \in \mathbb{N}}$ be a finite discrete stochastic process with stationary increments and as its first order differences consider $Y = \{Y(t)\}_{t \in \mathbb{N}} = X(t+1) - X(t)$. Notice also the definition of the *aggregation processes* of $Y$ given by:

$$Y^m(i) = \frac{Y(mi) + Y(mi+1) + ...Y((m+1)i-1)}{m}, m \in \mathbb{N}. \quad \textbf{(2.2)}$$

Given these definitions, $X$ is said self-similar if:

$$Y^d \overset{d}{=} m^{1-H} Y^m(i), m \in \mathbb{N} \quad \textbf{(2.3)}$$

where $m$ denotes the aggregation scale (non-overlapping blocks) and $H$ is the Hurst Parameter. Fractional Autoregressive Integrated Moving Average (FARIMA), Fractional Brownian Motion (fBm) and Fractional Gaussian Noise (fGn) are examples of processes that exhibit long range dependence (long memory). fGn is the first order differences of fBm, and it is self-similar in the sense given by equation 2.3.

### 2.2.3 Random Walk

A random walk [Spi13] is a process which describes a path in a $nth$ dimension composed of a set of independent and equally distributed random steps and has been used to model phenomena in a wide range of fields, including physics [Kem03] and biology [CPB08]. The sum of the values obtained by assigning 1 or -1 to heads or tails during an experiment consisting of systematically tossing a coin into the air is a practical example of a single dimension process that resembles a random walk. Even though this process cannot be considered as a random walk (most of the coins are not balanced, the stamina of the thrower decreases over time and the air temperature and wind can affect the air resistance), it still holds as a good enough example to explain the concept.

More formally (and generic), let $R(t), t \in \mathbb{N}$ be a discrete-time stochastic process and $S(t)$ a random discrete variable defined in $t$. $R(t)$ is considered a Random Walk if equation 2.4 holds true:

$$R(t) = \begin{cases} 0, & \text{if } t = 0 \\ R(t-1) + S(t), & \text{otherwise} \end{cases} \quad (2.4)$$

If $S(t)$ is a random variable, the resulting process $R(t)$ has a Hurst Parameter value of 0.5, *i.e.* a memory less process.

This kind of process is important because it can be used to generate time series that exhibit persistent or anti-persistent behavior [Enr04]. It will also allow an easier understanding of the concepts to be introduced next.

### 2.2.4 Fractional Brownian Motion

The fractional Brownian motion (fBm), also known as fractal Brownian motion or the random walk process, is a generalization of the Brownian motion and it is named after Robert Brown, a botanist that did remarkable work in this field while studying biological phenomena [NNNN67]. It is widely used to model LRD and Short Range Dependence (SRD) processes in fields associated with analysis of network traffic [THJ07] or stock markets [Øks03]. To formalize this concept, let $B_h(t), t \in \mathbb{N}$ be a discrete-time process, $H \in ]0, 1[$ the Hurst Parameter, and $\Gamma$ the Gamma

function $\Gamma(\alpha) = \int_0^\infty e^{-x}dx$. $B_H(t)$ can be modeled as a fBm according to Equation 2.5 [MVN68]:

$$B_H(t) = \frac{1}{\Gamma(H+1/2)}\left(\int_{-\infty}^{0}(t-s)^{H-1/2} - (-s)^{H-1/2}dB(s) + \int_0^t [(t-s)^{H-1/2}]dB(s)\right). \quad (2.5)$$

### 2.2.5 Fractional Gaussian Noise

Formalized by Mandelbrot in 1965 [Man65], fGn is another way to model persistent/anti-persistent processes and relies on the first order differences of a fBm, as shown in equation 2.6, where $G_h(t), h \in ]0,1[$ represents the fGn:

$$G_h(t) = B_H(t+1) - B_H(t), t \in \mathbb{N}. \quad (2.6)$$

Algorithms to generate self-similar sequences, such as the Paxson method [Pax97], use the fGn as the base model, making it highly relevant under the scope of this work.

## 2.3 Estimators for the Hurst Parameter

This section contains brief description of the most well-known Hurst Parameter estimation algorithms, with a more detailed explanation of the algorithms implemented in the scope of this work being then given in section 4.2. The complete description of all algorithms would take longer and falls out of scope of this dissertation. Nonetheless, references included in the several sections point out to where the reader may obtain more information on each subject or estimator. These algorithms can be classified in three major groups: (i) the ones that take advantage of the time domain properties, such as Rescaled Range Statistics (R/S), Variance Time (VT), Absolut Moments Time (AMT), Embedded Branching Process (EBP), Higuchi (HIG), Detrended Fluctuation Analysis (DFA) and the Hurst Exponent by Autocorrelation Function (HEAF); (ii) the ones that take advantage of the frequency properties such as Periodogram and the Whittle estimator; (iii) and the ones based on wavelets such as the Abry-Veitch estimator.

DFA, Periodogram, HEAF and the Higuchi methods were integrated on `TestH` as part of this project and will be better described ahead. This section presents the most important estimators, but an honorable mention is due to the FFT [CSZ07] and the Visibity Graph [LLLN09] methods, presented in 2007 and 2009, respectively. These most recent methods, such as the HEAF, the FFT and the Visibility Graph demonstrate that this scientific field is still very active.

### 2.3.1 Rescaled Range Statistics

The first algorithm to be included in `TestH` was R/S, mainly because of its historical value, since it was the one presented by Harold Hurst [Hur51] himself and later formalized my Mandelbrot and Wallis [MW69]. Nonetheless, this algorithm is amongst the most biased ones. This method works by splitting the values of the time series under analysis in $K$ non-intersecting blocks, with

size $m_k$. For each block size, the partial sums $S_k^i(j)_{j\in\mathbb{N}}$ of the deviations from its mean are then calculated, as shown in equation 2.7:

$$S_k^i = \sum_{l=im_k}^{im_k+j} (Y(l) - Y^{m_k}(i)). \quad \textbf{(2.7)}$$

Note that in the previous equation $Y^{m_k}(i)$ denotes the particular value at index $i$ for an aggregation block. This process is repeated for different block sizes $m_k$, and for each block size a maximum and a minimum are calculated as suggested in equations 2.8 and 2.9, respectively:

$$Max_k(i) = max(0, S_k^i(1), ..., S_k^i(m_k - 1)), \quad \textbf{(2.8)}$$

$$Min_k(i) = min(0, S_k^i(1), ..., S_k^i(m_k - 1)). \quad \textbf{(2.9)}$$

The rescaled range is then found by dividing the differences between the maximum and the minimum by the standard deviation, as can be observed on equation 2.10:

$$RS_k(i) = \frac{Max_k(i) - Min_k(i)}{\sigma(i)}. \quad \textbf{(2.10)}$$

The values of the rescaled range series and of the non-overlapping blocks $m_k$ are then plotted using a logarithmic scale for both the $x$ (block sizes) and $y$ axis ($RS_k(i)$), in a chart also known as the *pox plot*. The $H$ parameter estimation can be obtained by calculating the slope of a line fitted to the *pox plot* graph using the Least Means Square (LMS). Due to the short range dependence of values at a lower scale and the lag between higher scales, the line should be fitted ignoring the points on the edges.

### 2.3.2 Variance Time

VT is available on `TestH` since its initial iterations. This method is closely related with the R/S with its main difference being the computation done at each block size which is, and as the name suggests, the variance instead of the rescaled sums, denoted by $\mathbb{V}$. The method is based on the relation between variances of different block sizes formalized in equation 2.11:

$$\mathbb{V}(Y) = \mathbb{V}(m^{1-H}Y^m). \quad \textbf{(2.11)}$$

Due to the variance properties, and introducing the logarithm into both sides of the previous

equation, it is possible to obtain equation 2.12:

$$log(\mathbb{V}(Y^{(m_k)}) = log(\mathbb{V}(Y)) + \beta \times log(m_k), \text{where } \beta = 2H - 2. \quad (2.12)$$

The Hurst exponent is given by the formula $H = 1 - \frac{\beta}{2}$, where $\beta$ is the slope of a line fitted to logarithms of $m_k$ and $\mathbb{V}(Y^{m_k})$ after particularizing to different aggregation blocks $m_k \in \mathbb{N}$. Both VT and R/S are amongst the most biased estimators, however VT is widely used in network related researches [KRD11];

### 2.3.3 Absolut Moments Time

Also available in `TestH` since the beginning, this method is a generalization of the VT method. In this case, the estimator is based on the computation of absolute moments instead of the variance or rescaled statistics. To perform a given estimation, an order for the absolute moments must be selected a priori. In fact, VT is as a particular case of the AMT for order 2, as it will be pointed by the end of this subsection. Let $M_n(Y)$ denote the order $n$ absolute moment of process $Y = \{Y(t)\}_{t \in \mathbb{N}}$. AMT is based in the relation depicted in equation 2.13:

$$M_n(Y) = m^{n(1-H)} M_n(Y^m)). \quad (2.13)$$

Analogously to what was shown to VT, it is possible to obtain equation 2.14 via the application of logarithms to both sides or the previous relation::

$$log(M_n(Y)) = n(1 - H) \times log(m) + log(M_n(Y^m)), n \in \mathbb{N}. \quad (2.14)$$

Finally, one can solve the equation in order to $H$, and an estimation can be obtained from the calculations of the absolute moments for different aggregation blocks $m\_k$. This should be achieved, once again, by fitting a line to the logarithms of the coordinates $(m_k, M_n(Y^{m_k}))$, as equation 2.15 suggests:

$$H = 1 + \frac{\beta}{n}, \beta = \frac{log(M_n(Y)) + log(M_n(Y^m))}{log(m)} \quad (2.15)$$

It is easy to see that the expression behind the VT estimator is retrieved by replacing $n$ with 2 in equation 2.15, making it a particular case of the AMT.

### 2.3.4 Embedded Branching Process

EBP was also already available in `TestH` at the beginning of this project. This method was presented by Jones and Shen in [JS04] and was initially used to study network packet traces, revealing once again the importance of these methods in the field of computer networks. This method requires the definition of a family of multiple horizontal lines. The points of the (self-similar) process that cross the line are then considered unless they pass once again trough the

same line they have passed in the last recorded moment. The Hurst Parameter can be estimated by considering the time it took the process to increase its statically properties in a power of two logic, as suggested by the relation in equation 2.16 ($\mu_k$ is defined afterwards):

$$log(2^k) = H \times log(\mu_k). \quad \textbf{(2.16)}$$

The requirement here is the computation of enough $\mu_k$ in order to obtain statistical significance. This can be achieved using the equation 2.17, where $N_k$ is the cardinality of a set of crossings of the process under analysis with lines with different magnitudes:

$$\mu_k = \frac{N_0 \times \mu_1 + ... + N_{k-1} \times \mu_k}{N_1 + ... + N_k}. \quad \textbf{(2.17)}$$

The Hurst Parameter can be retrieved by applying the LMS to the graph with logarithmic scales for $\mu_k$ and $log(2^k)$. The estimation for the Hurst Parameter coincides with the slope of the fitted line.

### 2.3.5 Detrended Fluctuation Analysis

While studying Deoxyribonucleic Acid (DNA) sequences [PBH+94], and later other kinds of phenomena such as the Heartbeats [PHSG95], Peng *et al*. proposed the DFA method. Similar to the previous methods presented, DFA also requires the definition of a family of scales $m_k$ but, in this case, the time series is integrated and then detrended against the best fitted line to mitigate the effects of trends. For each scale, ones calculate the local trend by finding the line that best fits the local random walk points within each block. The detrended walk is then obtained by subtracting the local trend to its original trend, which is then used to compute the variance. The Hurst Parameter is retrieved once again from the slope of a line fitted to a plot with logarithmic scales, containing the average variance at each scale against the aggregation scales. This algorithm was implemented under the scope of this work and a deeper explanation will be given ahead.

### 2.3.6 Periodogram

This method was proposed by Geweke and Porter-Hudak in [GPH83]. The presented algorithm to study the long memory processes is based on the analysis of the spectrum $f(\lambda)$ and proposes to approximate it by $I(\lambda) = \frac{1}{N} \sum_{j=-(N-1)}^{N-1} \hat{\gamma}(j) \times e^{ij\lambda}$, where $N$ is the number of samples, $\lambda$ the frequency and $\hat{\gamma}(j)$ a statistic resembling the autocorrelation function. This calculation should be executed for several frequencies $k$ since the Hurst Parameter is obtained by computing the slope of the fitted line between the logarithms of $I(\lambda_k)$ against the ones of $\lambda_k$, according to the formula $H = (1 - \beta)/2$ where $\beta$ is the slope. As the previous algorithm, this one was also implemented under the scope of this project and will be the subject of lengthier explanation in chapter 4.

### 2.3.7 Whittle Estimator

Based on the Maximum Likelihood Estimation (Maximum Likelihood Estimation (MLE)), the Whittle Method [Die04, TT97] modus operandi is based on the minimization of a goodness of fit function. Popular variations of this method include the Aggregated Whittle and the Local Whittle. The main idea behind the Aggregated Whittle is that for a long enough time series, its data can be aggregated in order to produce a smaller time series that will possess the same properties of a fGn process.

On the other hand, the Local Whittle offers a semi-parametric estimator to be used when the spectral density is assumed for frequencies near zero. Unlike the majority of its peers, this method does not produce a graphical output. More formally, this method is focused on discovering the parameters that minimise a goodness of fit function $Q(H)$. This function is tipically defined as in equation 2.18, where $f(\lambda)$ is the spectrum function obtained from the autocorrelation function and $f_H(\lambda)$ the spectral density:

$$Q(H) = \int_{-\pi}^{\pi} \frac{f(\lambda)}{f_H(\lambda)} d\lambda. \quad \textbf{(2.18)}$$

If the spectral density is only assumed for values near zero, the method is considered to be the local Whittle method.

### 2.3.8 Wavelets-Based Estimator

Darryl Veitch and Patrice Abry [AV98] proposed a wavelets based method for the estimation of the Hurst Parameter, which is nowadays better known as the Abry-Veitch method. This method explores the recursive decomposition of a time series in a set of details $d_j(k), kin\mathbb{N}$. Considering the family of wavelets defined by $\psi_{j,k}(t))2^{-j/2}\psi_0(2^{-j}t - k$, the Hurst Parameter can be retrieved by fitting a line to the set of points $j$ and $log_2(\mu_j)$ using LMS, with $\mu$ denoting variance and $n_j$ denoting the number of details coefficients for each $j$:

$$\mu_j = \frac{1}{n_j} \sum_{k=1}^{n_j} d_j^2(k). \quad \textbf{(2.19)}$$

### 2.3.9 Higuchi Method

Back in 1988, Higuchi proposed an algorithm to estimate the Hurst exponent that now bears his name [Hig88]. This method has resemblances with the AMT, whereas a sliding window is used instead of non-intersecting blocks, thus making its computational complexity considerably higher. Similarly to other methods previously presented, a line should be fitted to the logarithms of the values given by Higuchi's function $L(m_k)$ and $m_k$ for different $m_k \in \mathbb{N}$. The Hurst exponent is then obtained by adding $2$ to the slope of the fitted line. This estimator will be discussed in chapter 4, as it was implemented as part of the work described in this dissertation.

### 2.3.10 Hurst Exponent by Autocorrelation Function

HEAF is the most recent method for the estimation of the Hurst Parameter. It was introduced by Rezaul *et al*. [RPGC06] following the work of Kettani *et al*. [KG02] on the self-similarity of network traffic. The algorithm relies on the autocorrelation function $\gamma(k)$ of the self-similar processes. For example, if $k = 1$ is the lag being considered to obtain an estimate, one should compute the autocorrelation between $Y(t + 1)$ and $Y(t)$, and the Hurst Parameter estimation can be obtained from $H = \frac{1}{2}log_2(\gamma(1) + 1) + \frac{1}{2}$. This algorithm is part of the set of estimators integrated on `TestH` during this project, and will it be discussed with more detail on chapter 4.

## 2.4 Self-similar Sequences Generators

`TestH` aims to provide methods for handling self-similarity, estimating the Hurst Parameter, and generating series of points with a predefined $H$ value. This section is thus devoted to the brief presentation of self-similar sequences generators.

The self-similar sequence generators can be divided in two families, the exact generators (namely the Hosking, Cholesky and Paxson) and the approximate generators. Their complete definition requires typically several pages, and the description below contains pointers where more information can be found. The discussion of the approximate methods is only confined to the ones available in `TestH` up to the date this dissertation was written, since there is a large number of generators available nowadays. The Davies and Hart generator was integrated in `TestH` as part of the work accomplished in this project and will be better described in chapter 4.

### 2.4.1 Hosking

The Hosking method [Hos81, Hos84], which was named after its creator, defines a procedure to simulate an exact realization of a fGn. It is based on the covariance matrix of points of the process. One of the features is that it does require the definition of a fixed number of points a priori, making it a good candidate for on-demand point generation applications. Nonetheless, it is a computationally intensive procedure, with a computationally complexity growing with the square of the number of points.

### 2.4.2 Cholesky

The main difference between this method and the aforementioned one is the different decomposition applied to the covariance matrix. This method uses the Lower Upper (LU) decomposition, producing two triangular matrices [CP16]. For this decomposition process, known as the Cholesky decomposition, a larger number of operations has to be made, thus taking longer than the Hosking method to generate the points of the process.

### 2.4.3 Davies and Hart

Also named after its creators, the Davies and Hart method [DH87] is similar to the Hosking method, with differences in the way the covariance matrix is computed. In this case, the method elaborates on the possibility to define a circulant matrix containing the covariance matrix. The circulant matrix can then be factorized using the Fast Fourier Transform. The key feature of this algorithm is that its computational complexity is lower than the other exact methods, *i.e.* the fastest one amongst them (on the order of $n\log(n)$). This method will be explained on the following chapter, as its integration on `TestH` was part of the work done during this project.

### 2.4.4 Aggregation of Processes

Formalized by Willinger et al. while studying the Ethernet Local Area Network (LAN) traffic [WTSW97], the aggregation of processes method was initially used to model the bit rate per minute of a network aggregation point. To understand the logic behind it, one can imagine a set of independent computers $s$, connected between via a central network. Each one will be in the transmitting state (`ON` period, if and only if $W^s(t) = 1$) or in the `OFF` state otherwise, with $W$ denoting a stationary binary process. When rescaling time by a factor $T$ , the cumulative sum of all transmitting computers can be defined according to equation 2.20:

$$W_s(Tt) = \int_0^{Tt} (\sum_0^s (u))du. \quad (2.20)$$

It can be shown that $W_s(Tt)$ converges to an fBm, as formalized in equation 2.21, with $B_H$ denoting an fBm with Hurst Parameter $H$:

$$\sigma B_H(t) = \lim_{T\to\infty} \lim_{S\to\infty} T^{-H} S^{-1/2} \left( W_S(Tt) - \frac{TSt}{2} \right). \quad (2.21)$$

### 2.4.5 Paxson

Paxson proposed a method to simulate instances of fGns, and consequently fBms, by generating a process in the frequency domain and then transforming it into the time domain [Pax97]. This method is relatively similar to the Davies and Hart method, and tends to be exact as the number of samples grow to infinite. It is also faster than Davies and Hart due to the less amount of FFT computations required.

### 2.4.6 Fractional Brownian Motion Sequential Generation Algorithm

Proposed by Inácio during his Doctor of Philosophy (Ph.D.) work [Ina09], the Fractional Brownian motion Sequential Generation Algorithm (fBm-SGA) elaborates a set of random walks with a persistent behavior in order to formulate the persistence probabilities. It is important to mention that this method relies on the dependence of every point in the sequence towards the first one,

but also that for larger scales the process acts as a random walk.

### 2.4.7 Simple Self-Similar Sequences Generator

As the previous algorithm, this generator was also developed by Inácio et al. during his Ph.D. work [Ina09]. The Simple Self-Similar Sequences Generator (4SG), unlike the previous one, is capable of producing series with both anti-persistent and persistent behavior, i.e., with $H \in ]0,1[$ (the fBm-SGA is only defined for $H \in ]0.5,1[$). In this algorithm, a self-similar process is approximated by having a set of several constant parts (with the size being always a power of 2) that contribute from time to time to the generation of new values.

## 2.5 Conclusion

It is now possible to see that there is a wide range of algorithms to estimate the Hurst Parameter, and there is not a single answer for which one is the better, as their performance might depend on features of the processes to be analyzed. The panoply of available estimators is also an indication that the values obtained for the Hurst Parameter are truly estimations, since they do not arrive all necessarily to the same value. It also corroborates the need to have a tool with a wide range of implemented algorithms from this area of knowledge. Many estimators elaborate on statistical measures of the processes for different aggregation scales and on fitting a line to plots with logarithmic scales. This fact eases the understanding of the different and sometimes difficult procedures and, consequently, it eases their implementation.

There are also several methods to simulate series of values from self-similar processes. The exact methods have typically higher computational complexity. It is worth mentioning that, for generators, the baseline in which they elaborate is typically more diverse than for estimators.

# Chapter 3

# Related Work and TestH

## 3.1 Introduction

`TestH` has been in development since 2013 but it is far from being the first tool of its kind. This chapter presents several works (section 3.2) focused on the creation of a tool to study the self-similarity of time series, which basically constitute the state of the art on this subject. Some of these works were in fact a source of inspiration for the development of this library and they will be compared against `TestH` right after the presentation of its philosophy, directory structure and architecture overview in section 3.3.

## 3.2 Related Works

The main purpose of this section is to provide a brief description of the most well-known tools to measure self-similarity available to date, with a comparative analysis being given on section 3.4. Please note that some of these tools were not available to download (they might have been in the past), which means that the analysis had to be performed with basis on information included in several publications mentioning them.

### 3.2.1 SELFIS

Introduced in 2002, the SELF-similarity analysIS (SELF-similarity analysIS (SELFIS)) is a tool with a GUI published by Karagiannis *et al.* and written in Java [KF02]. SELFIS includes features such as the implementation of the most common Hurst parameter estimation algorithms (Absolut Moments, Abry-Veitch, Periodogram, Rescaled Statistics, Variance of Residuals and Whittle). This tool has already been used in many studies but there are also authors who have reported issues related with the processing of long series, which results in long delays and infinite loops for lengths greater than $2^{18}$, and also issues related with the high biases of many algorithms [PMK14]. Figure 3.1 contains a screenshot of the main window of SELFIS as an example of its GUI after loading a time series and executing the Periodogram method.

### 3.2.2 SELQoS

Self-similarity and Quality of Service (SelQoS) is a tool developed in C++ and it also possesses a GUI. The main features of this tool include a wide range of different estimators (Absolut Moments, Abry-Veitch, Modified Allan Variance, Periodogram, Rescaled Statistics, Variance of Residuals and Whittle) and the Paxson's algorithm as a self-similar sequences generator. Statis-

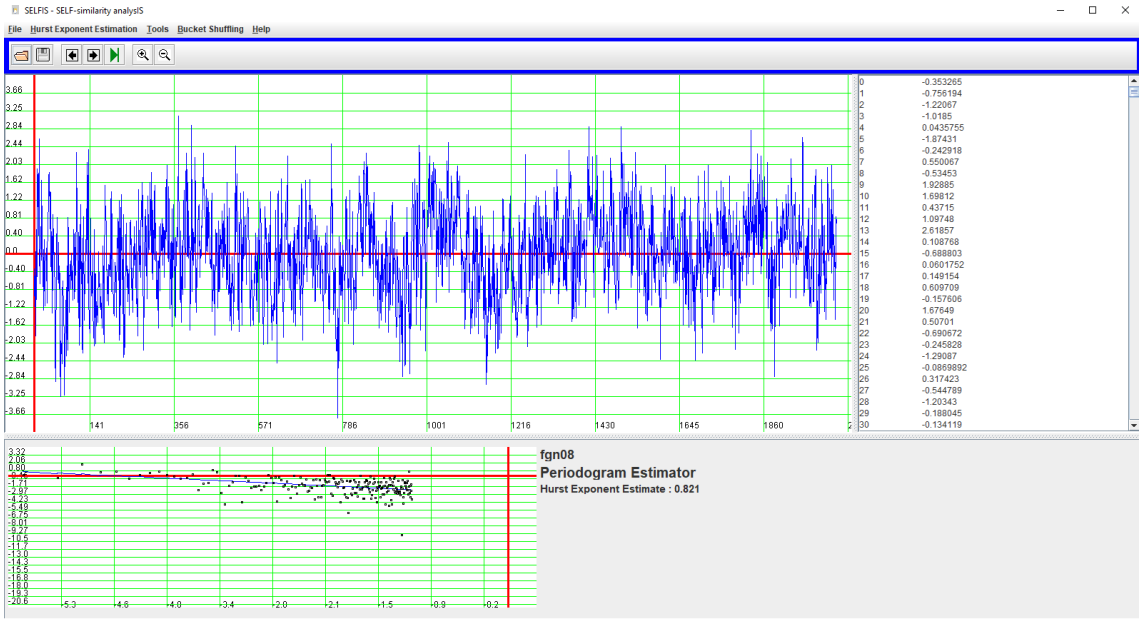


Figure 3.1: Screenshot showing the SELFIS Graphical User Interface.

tical methods ranging from the most basic (*e.g.*, average and standard deviation) to the most advanced (e.g., power spectral density function and histogram estimation) are also provided along with the capabilities to export the results (even in the form of charts) and adapt the paramenters. Reported disadvantages of this tool include the biases of some algorithms and the long processing time when the modified Allan algorithm is applied to long sequences [PRV08]. No source code or executable was found for this application. The screenshot of this tool shown in Figure 3.2 was taken from [RT05].

### 3.2.3 HATTA

Another known tool is HATTA, created by Bravo *et al.* and with its name coming from the Spanish words that describes it (Self-Similar Telematics Traffic Analysis Tool). HATTA has a simple GUI and is also written in C++ but isn't as complete as its peers. Nonetheless, it still has a large variety of estimators implemented (Differenced Variance, Periodogram, Rescaled Statistics, Variance of Residuals and Whittle). Some of the reported issues related with this tool that were reported include the high biases in some algorithms and the limited estimations that can be achieved in each session, forcing it to be reinitialized every 10 iterations [PRV08]. Presented in figure 3.3 we can see HATTA interface after loading a time series and computing the Periodogram.

### 3.2.4 Statistical Tools

R is a powerfull and well-known statistical tool. It is a solution initially developed by Ross Ihaka and Robert Gentleman [rLa] at the University of Auckland. R has a wide variety of packages available, but the most interesting package in the field of the self-similarity is fARMA, supported by Wuertz et al. [fAR]. This package includes a wide range of estimators (namely, Absolut Mo-

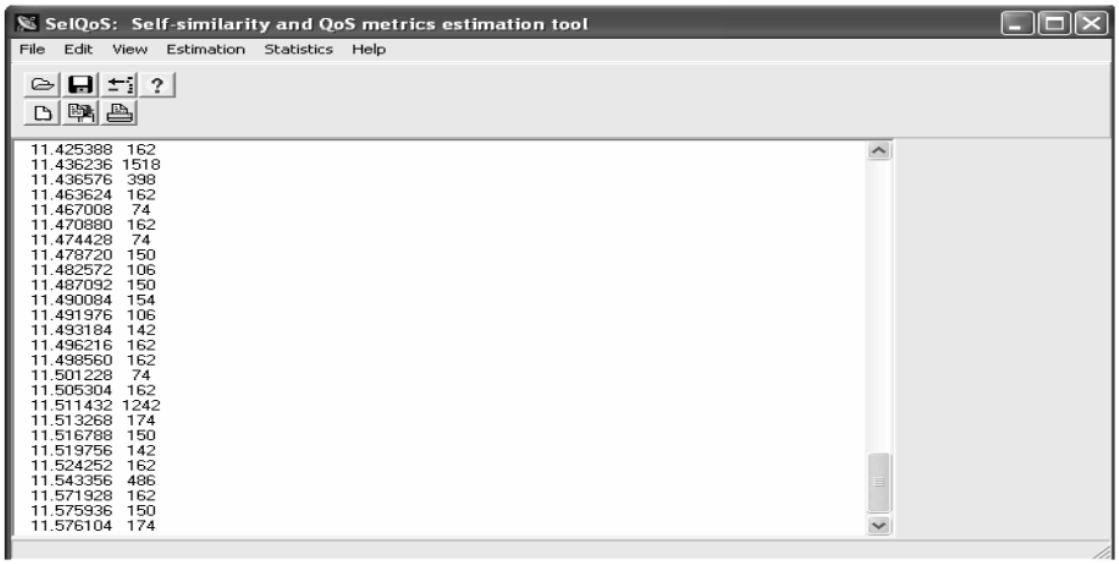


Figure 3.2: Screenshot showing the SELQoS Graphical User Interface (adapted from [RT05]).

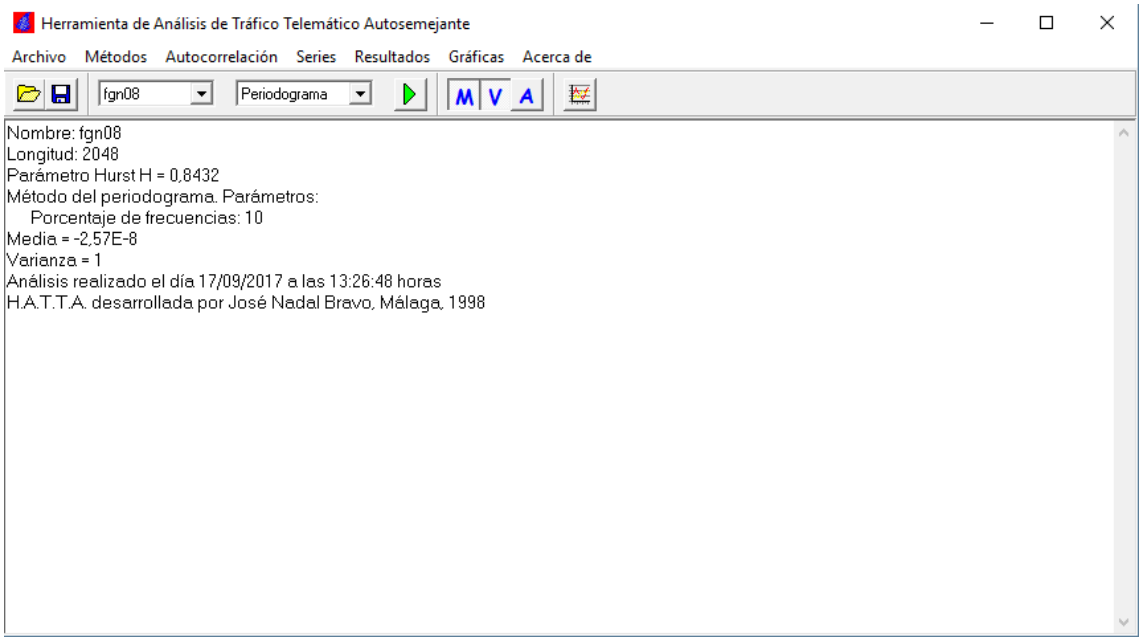


Figure 3.3: Screenshot showing the HATTA Graphical User Interface.

ments, Abry-Veitch, Differenced Variance, Higuchi, Periodogram, Rescaled Statistics, Variance of Residuals and Whittle) and generators for both fractional Gaussian noise (fGn) and fractional Autoregressive Moving Average (FARIMA) series. A later check on fARMA official web page shows that the package have been archived due to problems that were not corrected.

MATLAB is also a widely used statistical tool but, unlike R, it does not have a dedicated package related to this particular subject. Instead, there are many isolated implementations around the Internet that can be used for the purposes at hand, but none of them have been scrutinized as much as the solutions aforementioned.

### 3.2.5 State of the Art Wrap-up

SELFIS, Self-similarity and Quality of Service (SELQoS) and Herramienta de Análisis de Tráfico Telemático Autosemejante (HATTA) emerged with the main purpose of studying and simulating network traffic data, emphasizing once again that the exhaustive studies in this field were influential for the development of such tools, as mentioned in chapter 1. These are the most widely used tools for the analysis of self-similarity described in the literature, being chosen for analysis in this dissertation because of that. Nonetheless, other non-popular tools such as Local Analysis of Self-Similarity (LASS) [STP+06] can also be found.

From the analysis of the literature, one can notice that the most common estimators, such as R/S, are implemented in all of the aforementioned tools. However, it is not guaranteed that the results of a specific algorithm, applied to the same time series, are equal across them. This is due to the variety of possible implementations of an algorithm (*e.g.*, static or dynamic block sizes) or even misunderstandings during the implementation process.

## 3.3 TestH

This section starts by providing an overview of the `TestH` library, followed by its folder structure and architecture, where each module purpose is described. This chapter ends with a brief discussion of the additional features that are available out of the box on TestH.

### 3.3.1 Library Overview

The tools described in previous sections were the main source of inspiration for the development of TestH, but it is also important to mention another source on which this work is based: `TestU01`, developed by L'ecuyer *et al.* et al., is a statistical tool with the purpose of studying the quality of random number generators, also written in ANSI C. The `TestH` library is thus being implemented in the widely used ANSI C programming language and released under the Berkeley Software Distribution (BSD) 3-clause license. The core idea behind its design and implementation is to empower researchers with a robust, yet simple to use, tool to study the self-similarity. This user-friendly usage is achieved with the layer of abstraction provided by the meaningful

API, which aims to remove the burden related with the self-similarity estimation, thus enabling a better focus on the real problem the users are facing.

So far, the features included in the library are a collection of eight Hurst parameter estimators, six self-similar sequence generators, testing batteries to measure the performance of different algorithms when exposed to the same data and the possibility to use external generators or external data.

### 3.3.2 Directory Structure

The addition of documentation related features on `TestH` during this project increased the complexity of the folder structure of the library, which is illustrated in Figure 3.4. In this figure, the rectangles represent a directory and the sheet represents a file or set of files. The green highlighted archives are actually not part of the project, as the contents of their folders is generated at each compilation. However, the folders should be created in order to run the test files. The documentation related archives are emphasized in blue. Even though the files under the `documentation` directory are generated with the `make file`, `TestH` is shipped with them generated as they are part of the site documentation and could be used to consult the API offline (note that, in order to generate these files, one needs to install CLDOC locally). The files under `docs` are also part of the API documentation, but they are not intended to be modified, as they only define the description of each module and of the library itself. The core files of the library are displayed with the yellow color. The directory `include` contains the header files which define the API, while the `src` directory has the `.c` files implementing the library code. This directory also contains a test file named `test.c` and a make file, which upon invocation compiles the project and executes the executable generated from the `test.c`.

### 3.3.3 Library Architecture

For a better organization of the code, the library is divided in twelve modules, each of them with a well-defined purpose:

- `proc.h` – this module provides the interface to manipulate processes, prototyping the most relevant data type on `TestH`, the so-called `proc_Process`, which holds the information regarding a specific process. This structure is an extension of other two, explained next. `proc_Points` carries the data points, and `proc_Scales` carries the scales to be considered (when applicable). Besides holding the points, `proc_Points` can also contain other relevant data, such as the process average value and standard deviation. The `proc_Scales` structure contains a `proc_ScalesConfig`, which has information regarding the scales configuration, such as the their maximum and minimum values. This last two kinds of structures are not intended to be manipulated directly, as the API contains specific functions to configure them such as `proc_CreateScalesConfig();`

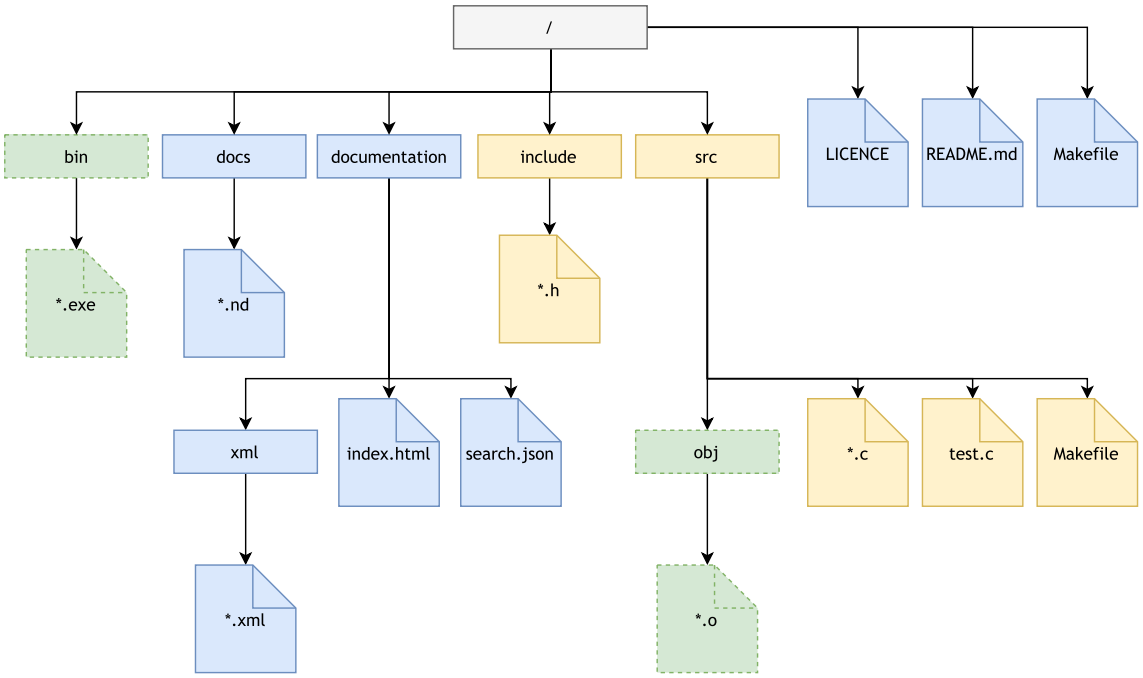


Figure 3.4: Diagram showing TestH directory structure.

- `gen.h` - the self-similar sequence generators are defined in this module, but its functionalities are not limited to that specific objetive. It contains also the functions to import self-similar sequences from external files trough functions like `gen_ReadFile()`. These functions can be called in a static manner via their name, or dynamically via the `gen_GenProc()` function, where the first argument is the generator to be used, as defined in the enumeration `TestHGenerator`. This enumeration has a practical organization and allows the addition of more generators without modifying its logic (the Gaussian generator is at position zero, the external generators have positions below zero and the TestH ones have positions above zero), thus making the software retro compatible. One should notice that most of the functions present on this module output a `proc_Process` structure;

- `est.h` - the purpose of this part of the library is to define the Hurst Parameter estimators. It also includes the enumeration `TestHEstimator`, containing an entry for each estimator, and it is used to output printing purposes. Every estimator prints their estimation related values at the end, which may vary for the different methods, as it depends on the mathematical principles each one relies on. They also return the Hurst Parameter as a double precision C value, which can be useful for subsequent computations. Regarding their input parameters, every single estimator requires a `proc_Process` structure, with the AMT and EBP requiring additional parameters, namely the number of absolut moments and crossing lines;

- `batt.h` - this module contains the function `batt_Generator`, which concatenates a battery of tests enabling the quick testing of self-similar sequences and Hurst Parameter estimators. This function takes as input a self-similar sequence generator, as defined in

`TestHGenerator`, two Hurst Parameter estimators, defined in `TestHEstimator`, the lower and upper boundaries for the tests and the step increment. For each step, this method computes the self-similar sequence with the desired Hurst value, feeds the sequence to the estimators and then prints the results;

- `dist.h` – focused on probability functions and their manipulation, this module offers several implementations of algorithms to simulate variables with the Gaussian (e.g., the Box-Muller method), Pareto, and uniform distributions, with most of them having both the Probability Density Function (PDF) and Cumulative Distribution Function (CDF) associated methods implemented;
- `reg.g` – functions related with regression analysis are defined in this module. Being one of the smallest, with only three functions, it defines the coefficient of determination and the LMS functions. It also defines a variant of the LMS function which allows the computation to be done over a specific interval. Both the LMS and the LMS variant return a specific structure called `reg_Linear`, also defined therein, which stores the values related with the line fitted by regression (slope and y-axis interception value);
- `rng.g` – with two functions, this module provides a high quality Pseudo Random Number Generator (PRNG). Both functions are interrelated, as one is used to define the seed for the generator and the other one to generate the values. The PRNG algorithm in place is the standard Mersenne Twister [MN98];
- `stat.h` – this module implements both basic statistical functions (such as the average value, standard deviation, auto-covariance and auto-correlation) and more complex functions and methods, such as the goodness of fit Chi-Square and the goodness of fit Kolmogorov-Smirnov statistical tests;
- `func.h` – this module offers a set of mathematical functions that did not fit under the scope of the four mathematical modules described above. It includes functions related with FFT and cumulative sums, a function to compute the complement of the incomplete gamma function and, finally, a function to perform a numerical integration over a closed interval using the Simpson's rule. This module also adds a data type to manage complex numbers, called `func_complexNumber`, for usage with the FFT;
- `sim.h` – containing just one function, this module can be used to define simulators that resort to self-similar processes in their inner-workings. The only function there is currently used to simulate network traffic. It takes as input a self-similar sequence generator or a path to a file, and returns a process `proc_Process` to be further analyzed;
- `io.h` – in order to avoid code duplication, this module offers a set of input and output functions, *e.g.*, for printing errors that may have occurred during the program execution or managing files. A set of boolean alike functions are also available, such as the validation of a Hurst Parameter value (between zero and one) and if a value is a power of two;

- `util.h` – this module was added with the intention of avoiding code duplication also, as it is responsible for the memory management, time and sorting functions. The memory management related functions are encapsulations of the ones defined in C (`calloc`, `malloc`, `realloc` and `free`), with the particularity of having a successful memory allocation validation embedded. The time related functions are used with `TestH` generators and estimators and are intended to measure the computation time of an algorithm. The time is printed at the end of each algorithm and is expressed in seconds. Besides these methods, there are some system functions, *e.g.*, for printing the user name or the hostname and an implementation of the bubble sort algorithm.

The output of the programs can be configured to some extent through a set of variables. For example, one can enable the printing of the memory used by a certain algorithm. Is is important to note that the aforementioned functions represent the API of `TestH` and are the ones documented and supposed to be used by the end user. There are several functions that were not included in the API as they are used by a specific algorithm and are not seen as relevant from a user perspective.

The name of the functions, enumerations and structures present in the API follow a certain logic: contain the name of the module, followed by an underscore, and the name of the method written following the *Upper Camel Case rule* (also known as the *Pascal Case*) [cap].

### 3.3.4 Additional Features

The conception of the library took into account several aspects that may ease the end user work. One of them is the possibility to not only adapt the code easily to receive new estimators and/or generators, but also to read and write from external sources. Another feature that is worth mentioning is the battery method, which allows several tests to run sequentially without interrupting the flow of the program. This last feature could be particularly useful when deciding which estimator should be used to measure a certain kind of signal.

### 3.3.5 Code Management

`TestH` is an open-source project with several contribution from different developers. The system chosen to manage the code is GitHub, a popular web hosting service that implements the GIT version control system (originally developed by Linus Torvalds) [git]. This assures the availability of the code to everyone at any time. Another advantage is the seamless integration of contributions, which can be done following the steps below:

1. fork the project present on the URL `https://github.com/cdgramos/testh`. This creates a branch called `master` by default;
2. create one or more branches, each one dedicated to a particular issue;

3. commit changes made on that branch;
4. merge the created branch with the local `master` branch;
5. open a pull request from the forked version (`master` branch) to the original version (`master` branch).

If a submitted pull request concerns code following the `TestH` philosophy, the branches can be merged with a single click. Troubles may arise when another contributer changes the same files, resulting in conflicts which should be solved in the last pull request sent.

GitHub features an issue tracking system where a user can report bugs, suggestions and requests. Besides this, GitHub allows the creation of project websites through the creation of a specific branch named `gh-pages`, which `TestH` is using for presenting a beautiful form of the documentation and can be found at the following URL: `https://cdgramos.github.io/testh/`.

## 3.4 Tool Comparison

From all the tools mentioned in section 3.2, only SELFIS and HATTA were found to be available to download on the Internet, which is the main reason the following analysis is made with those two tools. On table 3.1 we can see a comparative analysis between them and `TestH`. All possess advantages and disadvantages. `TestH` strengths are the language of the documentation and API (note that HATTA is written in Spanish, which may be a barrier to some researchers), the guaranteed platform independence offered by the ANSI C language, the number of algorithms implemented and the availability of the code on-line. As weakness one can point out that `TestH` lacks a GUI and the possibility to directly plot charts. Note that this analysis already takes into account the improvements made under the scope of this work.

## 3.5 Conclusion

This chapter presented the works that are more closely related with the project described herein. Tools for estimating the Hurst parameter or to generate self-similar sequences were discussed with more detail, with their advantages and disadvantages identified and compared to the ones of `TestH`. The implementation philosophy of `TestH` was also presented, namely the directory structure over which it elaborates on and the way the code is organized in modules. The comparison with other tools took into account the improvements made during this project. Nonetheless, it is the next chapter that better discusses the work performed during this project.

Table 3.1: Comparison between TestH, SELFIS and HATTA.

| | TestH | SELFIS | HATTA |
|---|---|---|---|
| Software Language | English | English | Spanish |
| Price | Free | Free | Free |
| GUI | No | Yes | Yes |
| Graphical Plotting | No | Yes | Yes |
| Programming Language | ANSI C | Java | C++ |
| Open Source | Yes (BSD 3-clause) | Yes | Not verified |
| Time Series Maximum Length | Not verified, supported $2^{20}$ | $2^{18}$ | Not verified, supported $2^{20}$ |
| Number of Hurst Estimators | 8 | 7 | 5 |
| Number of Self-Similar Sequence Generators | 6 | 0 | 0 |

# Chapter 4

# Implemented Estimators and Generators

## 4.1 Introduction

This chapter of the dissertation emphasizes and is focused on the description of the direct contributions given to `TestH`. Section 4.2 describes more accurately the estimators added under the scope of this work, namely the DFA, Higuchi, Periodogram and HEAF. The following section 4.3 describes the self-similar sequence generator integrated on `TestH`, called Davies and Hart. Lastly, section 4.4 presents additional API contributions, the approach followed to document the code and a brief code sample.

## 4.2 Implemented Estimators of the Hurst Parameter

This section elaborates on the mathematical formalization and implementation details of the algorithms behind the estimators that were integrated on `TestH` as part of the work of this dissertation.

### 4.2.1 Detrended Fluctuation Analysis

DFA should be seen as an improvement of R/S. The main differences lies in the way the rescaled sums and the line to compute the slope is obtained. In DFA, given a time series with $N$ entries, the first step consists in calculating its cumulative sums (also referred to as integration), as in R/S. A time integrated series, which is herein denoted by $y(t)$, is divided into blocks of length $m$. For each block, a line is then fitted to the values using LMS, thus giving the trend for that set of values. Consider that, inside a given block with size $m$, the fitted line is denoted by $y_m(i)$. The detrended signal is then achieved by measuring, for each scale, the values of the integrated series against the ones in $y_m(i)$, according to equation 4.1:

$$F(m) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}[y(i) - y_m(i)]^2}. \quad (4.1)$$

This operation is then repeated for each block size, with the resulting values being computed in a log-log plot against $k$ different aggregation scales $m_k$.

The implementation of this algorithm is straightforward, having the following particular details in `TestH`:

1. a set of block sizes is automatically setup for a given time series of length $N$. The minimum and maximum block sizes are set to 4 and $N/4$, respectively. This ensures that the largest block size has enough samples for the calculation of $F(m)$;

2. the cumulative sums $y$ of the time series are computed;

3. the computation of the detrended signal is shown in Listing 4.1, which works by iterating each window size (line 1) and by determining the LMS for each non-overlapping block (lines 3 and 4). The detrended sum is cumulatively obtained from all LMS values calculated for each particular window size, which is stored in $yyAxis$ at the end and for each block size (line 9);

4. the values from the $yyAxis$ can then used to produce a log-log plot against the values of $blockSize$;

5. finally, a LMS line is fitted to the logarithms of the average value of detrended sums and block sizes, and an Hurst Parameter estimation is obtained directly from the slope of the fitted line.

```
1  for (i = 0; i < totBlocks; i++) {
2      sum = 0.0;
3      for (j = 0; j <= N - blockSize[i]; j += blockSize[i]) {
4          r = reg_LeastSquareMeansLimited(pr->points, y, j, j + blockSize[i], OFF
       );
5          for(z = j; z <= j + blockSize[i]; z++) {
6              sum += pow( y[z] - r.m - (r.b * z), 2);
7          }
8      }
9       yyAxis[i] = sum / (N / blockSize[i] * blockSize[i]);
10 }
```

Listing 4.1: Excerpt of code showing how the detrended sums are calculated in the scope of DFA.

### 4.2.2 Higuchi

The Higuchi estimator works by obtaining a new series $y_k^m(i)$ from a time series $y(t)$ of length $N$ as per Equation 4.2, where $m$ represents the initial interval and $k$ the width:

$$y_k^m : y(m), y(m+k), y(m+2k), ..., y(m + \lfloor \frac{N-m}{k} \rfloor k); k, m \in \mathbb{R}_{>0} \quad (4.2)$$

Note that, on the equation above, $\lfloor \rfloor$ represents the floor function, also known as the truncation function.

The function $L_m(k)$ is then computed as can be seen on Equation 4.3:

$$L_m(k) = \left\{ \frac{N-1}{\left[\frac{N-m}{k}\right] k} \left( \sum_{i=1}^{\left[\frac{N-m}{k}\right]} |y(m+ik) - y(m+(i-1)k)| \right) \right\} \Big/ k. \quad (4.3)$$

In practice, this estimator works by defining once again a fixed set of block sizes $m_j \in \mathbb{N}$ (e.g., 2, 4, 8, ...) and calculating $L_{m_j}(k)$ for each one of those block sizes. The next step consists on building a log-log plot with the logarithms of $L_{m_j}(k)$ and $m_j$, and then fit a line to those values. The Hurst parameter is finally obtained by using the formula $H = \beta + 2$, where $\beta$ is the slope of the fitted line. The computational complexity of this method is high and is by far the slowest method present on `TestH` to date.

The implementation of the Higuchi estimator in `TestH` can be described as follows:

1. given the time series of length $N$, a set of logarithmically spaced blocks are computed between $log_{10}(1)$ and $log_{10}(N/5)$. $N/5$ and $bins = 50$ were chosen by the authors of `TestH`, since the algorithm does not define this in practice. The blocks are thus computed such that the first one is zero and the others are the addition between the previous value found and the increment $log_{10}(N/5)/bins$;

2. with the blocks computed on the previous step, a new array containing the block sizes $blockSize$ is created and initialized with 10 to the power of each previously computed block;

3. Listing 4.2 depicts the core of the algorithm, which computes and stores the sum of the local curve length for each block size and for each overlapping block. Note that the computation that is made for the overlapping blocks highly increases the complexity of this method, since processing is performed for all points of the sequence several times (instead of being performed for non-overlapping blocks, as in many other methods);

4. finally, a line is fitted to the logarithms of the $curveLength$ and $blockSize$ using LMS, from which one can obtain an estimation of the Hurst parameter via the formula $H = \beta + 2$, where $\beta$ is the slope of the fitted line. In the implementation, only the first half of the values are considered to fit the LMS, decision that results from the fine tunning process to maximize the algorithm accuracy.

```
for(h = 1; h <= n; h++){
    aux = 0;
    k = floor((N - blockSize[h]) / blockSize[h]);

    for(i = 1; i <= blockSize; i++){
        aux1 = 0;
        for(j = 1; j <= k; j++){
            temp_lenght[i][j] = abs(pr->points[i + j * blockSize[h]] - sequence
    [i + (j - 1) * blockSize[h]]);
            aux1 += temp_lenght[i][j];
        }

        aux1 /= k;
        aux += aux1;
    }

    curveLength[h] = aux * ((N - 1) / pow(blockSize[h], 3));
}
```

Listing 4.2: Excerpt of code showing some of the core operations of the Higuchi method.

### 4.2.3 Periodogram

For a given process $y(t)$, let the Periodogram $P(\lambda)$ be computed according to the equation 4.4, where $N$ denotes the length of the time series and $\lambda$ denotes a frequency. Consider also that this calculation is repeated for several frequencies $\lambda_k$, with $k > 0$:

$$P(\lambda) = \frac{1}{N} \left| \sum_{i=1}^{N-1} y(i) \times e^{ij\lambda} \right|^2 . \quad (4.4)$$

Analogously to many other previously described estimators, an estimation of the Hurst parameter is taken from the slope of the line that best fits the logarithms of $P(\lambda_k)$ and the logarithms of $\lambda_k$ using $H = (1-\beta)/2$, where $\beta$ is the slope of the fitted line. It is common to find sources in the literature [TTW95] where the authors only consider a certain percentage of the values close to the origin to fit the line. However, there is no exhaustive study that highlights the advantages or disadvantages of choosing such values. As such, it should be considered as a parameter of the method.

Programmatically, the Periodogram estimator can be written in a few lines of code because it does not require calculations over overlapping blocks. Its implementation in `TestH` can be

described as follows:

1. the first step consists in computing the FFT on the given process of length $N$, resulting in an array of complex numbers $complexVec$;

2. the data is then ready for the main Periodogram computations, which are shown in Listing 4.3. The variables $x$ and $y$ represent two arrays of length $N$. Notice that the transformation into the frequency domain is performed resorting to FFT, which eases this description;

3. in the end, a graph can be plotted using a log-log scale and the LMS can be used to fit the coordinates. Once again, the Hurst Parameter estimation is given be $H = 1 - \beta/2$, with $\beta$ denoting the slope of the fitted line. In this implementation, following the suggestion in the original description of the estimator, only the first $2\%$ of the values were considered to fit the line. This empirical value was chosen as the one that minimized the verified error close to the origin.

```
for(i = 0; i < N; i++){
    complexVec[i].real =  (pow(complexVec[i].real, 2) + pow(complexVec[i].imag
    ,2)) / (PI2 * N);
}

for(i = 0; i < N; i++){
    x[i] = log((PI / N) * (i + 1));
}

for(i = 0; i < N; i++){
    y[i] = log(complexVec[i + 1].real);
}
```

Listing 4.3: Excerpt of code representing some of the core computations of the Periodogram estimation method.

### 4.2.4 HEAF

The HEAF method, as highlighted in section 2.3, is relatively new compared to its peers. This method elaborates on the autocorrelation function of the self-similar process to obtain an estimation of the Hurst parameter. This can be done by calculating one or more autocorrelation values, for one or more lag values, though it was initially defined for the lag of 1. Let $y(t)$ be a time series of length $n$, $\rho_n(1)$ (the autocorrelation for lag 1) is then computed according to

equation 4.5, where $\gamma$ denotes the covariance and $\sigma$ the variance:

$$\rho_n(1) = \frac{\gamma_n(1)}{\sigma_n^1}. \quad (4.5)$$

The next step is to compute $d_n$ as per equation 4.6:

$$d_n = \frac{\rho_n(1)}{1 + \rho_n(1)}. \quad (4.6)$$

At this point, the Hurst parameter can be obtained with equation 4.7:

$$H = \frac{1}{2}\left(\frac{1}{2log(2)}\right) \times log\left(1 + \frac{\rho_n(1)}{d_n}\right). \quad (4.7)$$

The implementation of this algorithm is remarkably simple and its computational complexity is low (the algorithm is extremely fast). Notwithstanding, as pointed out in [Ina09], this method does not fully capture the self-similar features of a process since the lag used is always one, thus only comparing neighbor values. Its implementation in `TestH` is illustrated as follows:

1. the average value $mu$ of the sequence of values under analysis is first calculated;
2. the rest of the main operations required by this method can be observed in Listing 4.4, where one can see how the Hurst Parameter value is obtained at the end of the procedure (line 11). `N` denotes the number of values of the sequence under analysis.

Note that the computations are only performed for the lag equal to one (line 8), as pointed out before. Unlike the previous implementations, as is, this algorithm produces no output for a graphical representation.

```
1  for(i = 0; i < N; i++){
2      gamma_0 += (pr->points[i] - mu) * (pr->points[i] - mu);
3  }
4  gamma_0 = gamma_0 / N;
5
6  k = 1;
7  for(i = 0; i< (N - k); i++){
8      gamma_1 += (pr->points[i] - mu) * (pr->points[i + k] - mu);
9  }
10 gamma_1 = gamma_1 / N;
11 H = 0.5 + (1 / (2 * log(2))) * log(1 + (gamma_1 / gamma_0));
```

Listing 4.4: Excerpt of code showing important operations of the HEAF method.

## 4.3 Implemented Generators of Self-Similar Sequences

This section presents a more detailed description of the Davies and Hart method. This algorithm was specifically chosen to be implemented in `TestH` during the development of this project because it is the fastest generator amongst the exact methods.

### 4.3.1 Davies and Hart

For a given input $H$ (the Hurst parameter) and $n$ (the desired length) of the sequence to be generated, the Davies and Hart method starts by computing the so-called row components $R(i)$, which are auto covariances, according to the formula in equation 4.8. The several $R(i)$ are then bind together with its inverse $\hat{R}(i)$, thus resulting in an array $D$ of length $2n$.

$$R(i) = \frac{1}{2} \times \left(i^{2H} - 2 \times (i+1)^{2H} + (i+2)^{2H}\right). \quad \text{(4.8)}$$

This method thus works by defining the covariances of the values to be generated prior to their generation. Covariances depend on the expected Hurst parameter. It then depends on how to make sure that the generated values will have those covariances. This is achieved resorting to FFT, which is applied to the aforementioned array. After that transformation, additional iterations are performed to the obtained values, according to the following conditions:

- For i = 0: $\hat{D}(i) = \sqrt{D(i)/(2n)} \times ND1(i)$;
- For i < n: $\hat{D}(i) = \sqrt{D(i)/(4n)} \times (ND1(i) + 1 \times ND2(i))$;
- For i = n: $\hat{D}(i) = \sqrt{D(i)/(2n)} \times ND2(i)$;
- For i < 2n: $\hat{D}(i) = \sqrt{D(i)/(4n)} \times (ND1(2 \times n - i) - 1 \times ND2(2 \times n - i))$.

$ND1$ and $ND2$ denote two random distributions, whose values are simulated prior to the application of the expressions. FFT is applied once again to this newly obtained array $\hat{D}(i)$. The self-similar series is then achieved by multiplying the first $n$ elements of $\hat{D}(i)$ by the scaling factor $1/n$.

## 4.4 Other Contributions

This section elaborates on the crucial step taken towards the development of `TestH`, which was the documentation of the API, thus allowing a faster comprehension of the library. A usage example of the library is also presented and discussed at the end of this section.

### 4.4.1 API Additions

In sections 4.2 and 4.3 estimator and generator algorithms were presented. Each of these algorithms result in one functional addition to the API. Nonetheless, other minor or secondary

additions were made to the library under the scope of this work. The full set of functions added to the API of `TestH` can be briefly described as follows:

- `func_complexNumber` – defines a data structure intended to represent complex algorithms. This structure represents a commodity when using the FFT function as ANSI C does not have director support for such numbers;
- `func_FastFourierTransform` – implements the FFT algorithm. This function is recursive and a temporary structure (see above) that should be passed as parameter;
- `func_CumulativeSums` – this function computes the cumulative sums without overwriting the given process;
- `reg_LeastSquareMeansLimited` – this function constitutes an adaptation of the LMS function already available on `TestH`. This version of the function enables the specification of the boundaries for fitting the line through its parameters;
- `est_Higuchi` – this function embodies the interface for the Higuchi method for estimation of the Hurst parameter;
- `est_Periodogram` – this function is defined in the `est` module and implements the Periodogram method for the estimation of the Hurst parameter;
- `est_DetrendedFluctuationAnalysis` – this function embodies the interface for the DFA estimator;
- `est_AutocorrelationFunction` – similarly to the 3 functions mentioned above, this function is also part of the `est` module and implements the HEAF method for the estimation of the Hurst parameter;
- `gen_DaviesHart` – defined in the `gen` module, this function constitutes the interface for a self-similar sequence generator and it accepts the expected Hurst parameter as input. It implements the algorithm proposed by Davies and Hart.

These additions to the API follow the meaningful naming principle and should be easily perceived by the end users. The aforementioned additions were mainly performed to the `func`, `est` and `gen` modules of the library, respectively. Regarding the FFT there was already an internal implementation on the code ($Paxson\_FastFourierTransform$) but it was implemented specifically for the purpose of being used with the Paxson method, justifying the need to write a generic function.

### 4.4.2 Documentation and Website

Even though C is a widely used programming language, there is no truly standard code documentation tool nor one that is seen as a reference in the community. Because of this, several tools

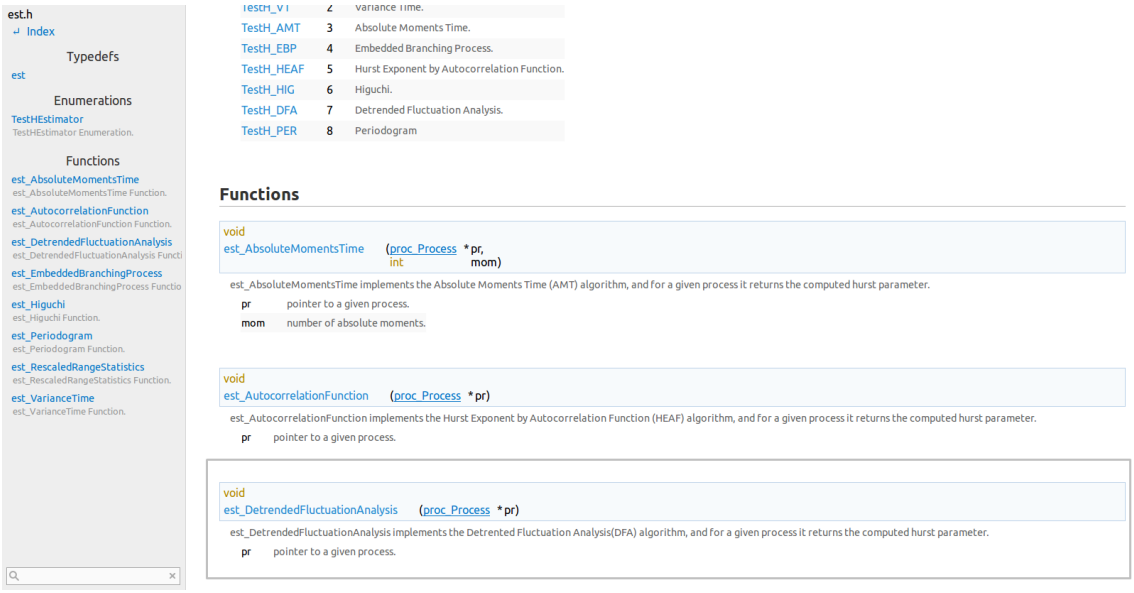


Figure 4.1: Sample of a documentation page for `TestH` generated by CLDOC with the DFA example highlighted.

were considered for building up the documentation, with the choice falling over to CLDOC in the end. The main features that were taken into account to choose CLDOC were its open-source nature, with its entire code available on GitHub, no requirement of configurations, the simplicity of the code that needed to be added to `TestH`, the client side search filter and the seamless integration of the generated documentation on a website. Listing 4.5 shows the documentation of the DFA method with its respective generated documentation displayed in gigure 4.1. The structure of the documentation should be made with precision by including a brief description of the function, the input values and the expected output, in this order. Without respecting the order or indentations, the outputted HyperText Markup Language (HTML) might contain faults and thus cause rendering problems.In order to ease the work of those who wish to contribute or development of `TestH`, a *makefile* dedicated to the generation of these HTML files was developed.

```
1 /* est_DetrendedFluctuationAnalysis Function.
2  * @pr pointer to a given process.
3  *
4  * est_DetrendedFluctuationAnalysis implements the Detrented Fluctuation
     Analysis (DFA) algorithm, and for a given process it returns the computed
     Hurst Parameter.
5  * @return the hurst parameter for a given process.
6 */
7 double est_DetrendedFluctuationAnalysis (proc_Process *pr);
```

Listing 4.5: Documentation of the DFA function using CLDOC.

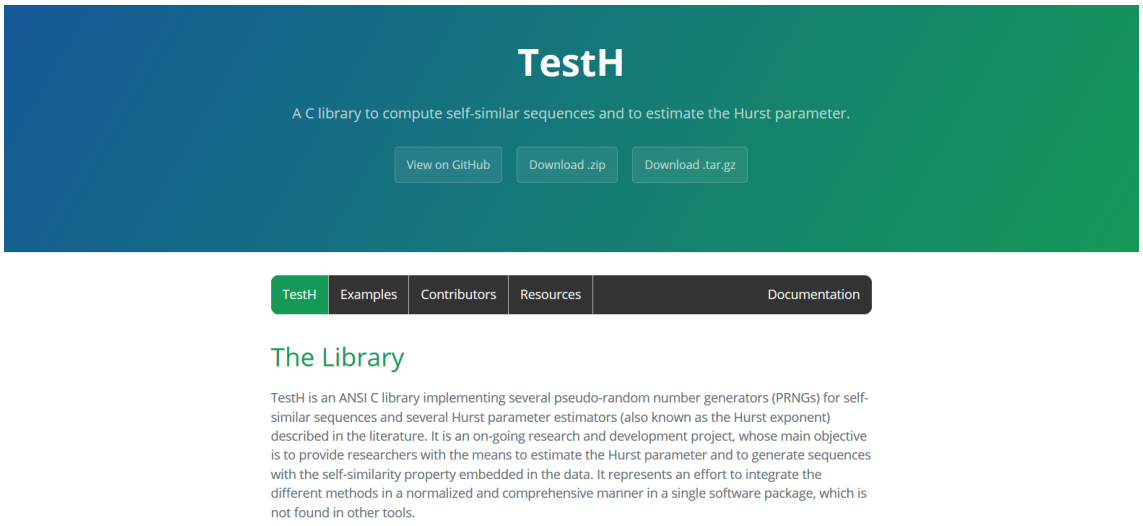


Figure 4.2: The `TestH` website.

The main goal of this documentation was for it to be accessible online, assuring that a researcher could check the API and see if it fits his needs in a short time. In order to do that, a GitHub page website was built, not only to host the API documentation, but also to promote the library. Its structure consists of five main sections/pages, briefly described as follows:

- `TestH` - this separator (and landing page) gives the client a brief introduction to the library, making him or her aware of the `TestH` features;
- `Examples` - short examples are given in this separator, with the purpose of showing the simplicity of the library;
- `Contributors` - this page pays tribute to the `TestH` contributors;
- `Resources` - articles, thesis and any other relevant materials related with the library can be found on this separator;
- `Documentation` - this separator links to the API documentation files generated by CLDOC.

Figure 4.2 shows how the site looks like, particularly in terms of its appealing appearance and functionality that allows to download code directly from it. Download buttons fetch the last version of the code from the GitHub repository automatically.

### 4.4.3 Usage Example

Listing 4.6 presents a sample program that uses the `TestH` library, illustrating a minimally working snippet that can be compiled and executed with success:

```c
/* TestH and ANSI C includes */

#define N   pow(2,15)
#define H   (double) 0.25

int main (int argc, char *argv[]) {

    proc_Process *pr = gen_Hosking (N, H, TestH_fBm);

    proc_ScalesConfig *conf = proc_CreateScalesConfig (TestH_POW, 7, 11, 2);

    proc_CreateScales (pr, conf);

    est_RescaledRangeStatistics (pr);

    proc_DeleteProcess (pr);

    return 1;
}
```

Listing 4.6: Sample C code using the `TestH` library.

The code can be adapted to each own style and objectives as it is intended to be only for demonstration purposes. The first instruction in the main body instantiates a process generator for the Hosking method, initializing it with a length of $N$ and an expected Hurst value of $H$, with the additional specification of the output to be a fBm. The next two instructions are required in order to create the scales and to assign them to the process. The process is then passed as an argument in the fourth instruction, where the Hurst parameter is finally estimated using the R/S method. It is important to mention that every estimator in `TestH` returns the Hurst Parameter as a double-type value. This can be particularly useful to compute the mean of a set of estimations, for example.

## 4.5 Conclusion

This chapter dived into the algorithms that were implemented under the scope of this work, offering an additional formalization of them. Regarding the estimators, with four additions to the library, `TestH` now has a total of eight readily avaialble methods, with the Abry-Veitch and Whittle being the ones missing from the set of ten estimators described in chapter 2. A step was also taken towards the implementation of the generators. Davies and Hart is the fastest method amongst the exact ones, making it particularly good to generate long time series with

many data points. A challenge faced during the implementation of these algorithms was the lack of information for some of them, notable with respect to their parameters.

As it is possible to observe at the end of this chapter, the documentation of the code plays an important role in this work, not only because it helps keeping the code organized but also because it improves the learning curve of those who are not familiarized with the library. Documentation of the library is performed using CLDOC, which enables the developer to easily embed documentation in a non intrusive manner, and also to transform the documentation into HTML pages, already published in a website, fulfilling another objective of this work.

# Chapter 5

# Evaluation

## 5.1 Introduction

Efforts placed on the development of the library are only worth if `TestH` finds its place amongst the scientific community and, in order to achieve that, it is required that researchers trust on the results it outputs, making the validation process a crucial part of this work. Section 5.4 of this chapter presents the method used to test both estimators and generators. In section 5.3 and section 5.4, the self-similar sequence generators and Hurst Parameter estimators will have their test results presented, respectively. Results will be divided in three main parts: one dedicated to computational resources, namely computational time, another one dedicated to the memory consumption, and the last part dedicated to their accuracy. This chapter ends with a brief discussion of the obtained results.

## 5.2 Testing Apparatus

Several tests were performed resorting to the HATTA and SELFIS tools to obtain a confirmation of the precision of the implemented generators and estimators, since they constitute a good baseline for comparison. The first type of algorithms to be tested were the generators due to the fact that there are no widely known tools that, for a given $H$ value, produce a self-similar sequence. This basically implied that the `TestH` generators had to be used to study the `TestH` estimators. Another challenge faced during the tests was related with length of the time series supported by SELFIS, which only supports the analysis of series with $2^{18}$ values maximum.

In order to test the accuracy of the generators, two sets of 20 time series each were generated. One set had time series with $2^{18}$ values, to be used on SELFIS. The other set included series with $2^{20}$ values, to be used on HATTA and `TestH`. These series were then fed to the tools, which were asked to produce estimations of the Hurst parameter. The average of the resulting values is presented and discussed on section 5.3. Following the same procedure, the estimator tests were done with the series produced for the tests of the generators. The scales considered for the windowed methods followed a power of 2 order, with the minimum number of different scale sizes being 7 and the maximum 11.

The testing procedures were automated using scripts and the the time that the computer took during the tests was measured, so as to obtain an indication of the efficiency of the implemen-

tations. This evaluation was not made to conclude about the computational complexity of the methods, since this is known from the literature and tests would have to be designed differently. Tests were performed in a computer with an quad-core 2.4GHz processor, 4GB of DDR3 Random-access Memory (RAM) and Ubuntu 14.04 Operating System.

## 5.3 Evaluation of Generators

This section presents the results of the tests regarding the self-similar sequence generators implemented in `TestH` at the time of writing of this dissertation. The justification of the choice of the generator that was used to test the estimators later on is also included in here.

### 5.3.1 Generators Computational Time

Table 5.1 shows the average time required to generate a series of values against the lengths of $2^{16}$, $2^{18}$ and $2^{20}$. Note that the lengths of the series increase by a factor of 4 from column to column. The times to generate the series should also quadruplicate for linear complexity algorithms. It can be concluded that this happens for all the generators, except for Hosking. Due to its exact nature, the Hosking method processing time rapidly increases as the length of the time series grows.

### 5.3.2 Generators Memory Consumption

The development of the generators in `TestH` is focused on the computational efficiency rather than on memory consumption. The library uses arrays with data structures instead of lists, i.e., the points will be allocated as a whole chunk in the heap during the program execution. Nonetheless, the API provides memory related functions to properly tackle this kind of issues (such as the lack of memory) and the auxiliary memory blocks are freed once they are no longer needed. Table 5.2 presents the approximated maximum memory consumption for each implemented generator, where $N$ denotes the desired time series length. Note that this analysis is approximated, i.e., non static memory allocations which do not depend on the length of the time series were ignored.

By observing these results, one may notice that there are relevant differences between the several generators, with the Davies and Hart being the one that requires more memory. As expected, the exact methods are the ones that require more memory, due to their higher dependency of previous values to generate the next ones. Even though the Paxson method fall in the category of the approximated methods, it is possible to notice that it roughly requires four times more memory than the other approximated methods. This is attributed to the memory necessary to compute the FFT.

### 5.3.3 Generators Accuracy

The results included in table 5.3 show the average error of each generator. These results were achieved by computing the average value of the (absolute value of) differences between expected and effectively obtained Hurst parameter values for each one of the analyzed tools (note that lower values indicate a better performance). Every generator had a good performance, with the two exact methods (Hosking and the Davies and Hart) along with the Paxson (recall that it tends to be exact) achieving the best results.

One of the generators was then selected for the evaluation of all implemented estimators, with the most logical choice being the fastest exact estimator: Davies and Hart. Nonetheless, as this estimator was implemented under the scope of this work, it was decided to do the tests using the Hosking method to favor impartiality and avoid bias.

## 5.4 Evaluation of Estimators

On this section, the results of the Hurst Parameter estimators are presented and compared against the ones obtained via the closely related tools peers. These tests were performed resorting to sequences of values generated using the Hosking method, due to the reasons mentioned in the previous section.

### 5.4.1 Estimators Computational Time

Similarly to the analysis reported on subsection 5.3.1, the experiments concerning the computational efficiency of the estimators outputted results aligned with the expectations. As it is highlighted on this work, the computational complexity of Higuchi is high, and one can obtain confirmation of this fact by comparing the respective value with the ones of others on table 5.4.

It is important to mention that even for sequences of length $2^{20}$, HEAF took less than a tenth of a second to execute, making it a good candidate when the processing time is critical.

### 5.4.2 Estimators Memory Consumption

As discussed in subsection 5.3.2, the estimators were also implemented with an emphasis on the computational efficiency rather than on optimum memory management. To obtain an idea of the memory usage of the implemented estimators, they were all fed with the same series and the peak of memory used during their processing was recorded. Table 5.5 provides an idea of the behavior of the estimators, by showing (i) that the memory is linearly dependent from the length of the series (denoted by $N$) and (ii) the difference between estimators as a factor of $N$, for comparison purposes. Note that these are approximated values and that the real maximum will depend on the scales defined for each windowed estimator. Variables whose size do not rely on the length of the time series were also ignored. One can see that the Periodogram method

requires far more memory that the other estimators. This is attributed to the use of the FFT, which requires a $4N$ memory allocation during execution.

### 5.4.3 Estimators Accuracy

Since the generators tests provided a certain degree of confidence that both the generator and the estimators are well implemented, it was possible to proceed with the accuracy tests for the estimators implemented on `TestH`. The tests followed the same structure as the ones reported on previous sections. In other words, 20 series with $2^{18}$ or $2^{20}$ values were generated with the Hosking method for each one of the expected Hurst parameter values chosen for evaluation. In other words, the generator was initialized with Hurst parameter values of $\{0, 0.05, 0.10, 0.15, ..., 0.85, 0.90, 0.95\}$ and, for each one of those, 20 series with $2^{18}$ and $2^{20}$ values were generated. Series with $2^{18}$ values were fed to SELFIS, while series with $2^{20}$ values were fed to HATTA and `TestH`. Each estimator was then asked to output estimations and the average value of the 20 samples was calculated and plotted against the expected value. The resulting charts are included in the figures of this section.

R/S was the first estimator tested due to its historic relevance. Figure 5.1 depicts the average values for the estimations of the Hurst parameters against the expected values for this specific method. It is possible to observe that the values for each different tool overlap most of the times, which may indicate that the implementation of the algorithm is similar for all tools. It is also important to notice that, as expected, this is the estimator that displays the worst results in terms of bias, specially for small values of $H$.

Figure 5.2 depicts the results obtained for the VT estimator. It is notorious that every tool produces noticeable deviations from the ideal Hurst parameter values (represented by a line), with more accentuated differences at the edges.

Figure 5.3 makes it clear that the Periodogram is not the best option for estimating Hurst parameter values close to 0. This behavior was verified across the three tools, with the estimated values being very good for expected values of $H \geq 0.25$. One can also notice that for values in the interval $[0.25, 0.95]$, most of the results overlap.

Figure 5.4 congregates the results of all remaining estimators considered in the scope of this work. These results were obtained using `TestH` only. They were not compared with the ones returned by SELFIS or HATTA, either because the estimators are not implemented in those tools, or because the tests were too biased due to a miss configuration of the input parameters that could not be solved. It can be seen that the AMT is the estimator with the poorest performance, followed by the Higuchi method. Nevertheless, all the estimators mentioned in the figure provide estimations close to the expected values, which make them a reliable choice to estimate the Hurst parameter. The full set of results obtained during these tests are presented

on Appendix A, for the sake of completeness.

For the sake of completeness, it was decided to test the estimators implemented in the scope of this work with the full set of generators available on `TestH`. This test included the generation of 20 sequences of values using all generators and for all Hurst parameter values between $[0.05, 0.95]$ with increments of 0.05 (except for fBm-SGA, where the Hurst parameter values were in $[0.50, 0.95]$). A total number of 1710 sequences were generated (20 sequences per each one of the 19 Hurst parameter values for each one of the 4 generators plus the sequences generated with the fBm-SGA). The average value of the absolute differences between the estimated Hurst parameter values and the expected ones was then calculated for each one of the new estimators. The results are summarized in table 5.6. Note that values close to 0 indicate a better overall precision.

It is possible to see in the previous table that the average values are all really close to each other and are typically low (recall that lower values indicate a better precision), with some estimators having in average a three decimal point precision. Something worth mentioning during the analysis of the table is that the poorest and the best performance were registered for the same estimator: HEAF. This algorithm results follow the ones of other estimators closely, but it performs remarkably bad for the fBm-SGA generator, with an average error of 0.04. This can be attributed to its predefined setup of analyzing a time series with a lag of 1. It was also the one that performed better, this time against 4SG, followed closely by DFA.

The lowest mean error was expected for the exact estimators, but this did not happen. As highlighted in this work, some of the approximated methods to generate self-similar sequences tend to be exact as their length tends to infinity. This might explain these results partially, as the length of the sequences was very large ($2^{20}$). However, to confirm the suspicion that approximate methods could even provide better results for lengthier sequences, a new set of analogous tests was performed, this time using sequences with length of $2^{12}$, producing the results in table 5.7.

As it is possible to see in the previous table, the worst performance was again registered for HEAF estimator regarding the series generated by the fBm-SGA algorithm, possible due to the same aforementioned reasons. As expected, for such small series, the exact methods have a better behavior than the approximate ones. Nonetheless, this might be an indication that the algorithms are properly implemented. It is worth mentioning that the 4SG algorithm keeps performing astoundingly well when compared with the other approximated methods, with even better results than the exact methods for the DFA estimator.

## 5.5 Discussion of the Results

While it is true that there is not a direct answer on which one is the best Hurst Parameter estimator, as it will also depend on the nature of the process, this section provided a useful benchmark for future implementations. As an example, it was observed that for anti-persistent noise close to zero, the Periodogram has a poor performance, while in the other hand, it may be the most useful one to study persistent time series. It was also demonstrated that the processing time grows exponentially for the Higuchi method, thus making it the worse candidate for applications that demand near real time results.

The generators performed remarkably well in the conducted tests, providing assurance of their correct implementation. There is no particular recommendation concerning which generator is better in terms of accuracy, except for an honorable mention to 4SG. However, if time is a constraint, the Hosking method should be avoided due to its computational complexity in the order of $n^2$.

One important conclusion that could be inferred from these tests is that, for large enough series, the implemented approximate generators produce sequences with self-similarity embedded as accurately as the ones produced by the exact methods (behavior verified for length of $2^{20}$), at least for the aggregation block with which the estimators were initialized. The same does not apply as clearly for short lengths, as there was a huge gap of precision between some of the the exact and approximate methods.

## 5.6 Conclusion

This chapter presented the tests made to `TestH` to evaluate the quality of the already available and newly implemented algorithms. It can be said that the tests were very successful. Almost every self-similar sequence generator and Hurst Parameter estimator outputted results near to the expected values, and even for the estimators that turned out to perform worse, such as R/S or the Periodogram for close to zero values, one can safely conclude that the behavior is more likely to be a result of the nature of the algorithm rather than an implementation issue, since the other tools, HATTA and SELFIS, produced similar behaviors.

Tests also allow to draw conclusions regarding the precision of the approximate methods. Some tests corroborate the idea that their output might be better as the lenght of the generated sequences increases. This was verified by comparing estimated Hurst Parameter for sequences with length $2^{12}$ and $2^{20}$.

Table 5.1: Time taken to generate series with $2^{16}$, $2^{18}$ and $2^{20}$ values (average values of 20 samples) measured in seconds.

| | $2^{16}$ | $2^{18}$ | $2^{20}$ |
|---|---|---|---|
| Hosking | 28,31819 | 484,34890 | 5227,24821 |
| Davies and Hart | 0,261582 | 1,17025 | 4.53401 |
| Paxson | 0,00279 | 0,27323 | 1,31897 |
| Aggregation of Renewals | 53,889433 | 216,50727 | 884,089 |
| fBmSGA | 0,00797 | 0,03982 | 0,13288 |
| 4SG | 0,01268 | 0,05036 | 0,17209 |

Table 5.2: Approximated maximum memory consumption for `TestH` generators.

| Generator | Hosking | Davies and Hart | Aggregation of Renewals | fBm-SGA | 4SG | Paxson |
|---|---|---|---|---|---|---|
| Memory | 4N | 6N | N | N | N | 4.5N |

Table 5.3: Accuracy results for the generators according to estimators in `TestH`, SELFIS and HATTA.

| Hosking | Davies and Hart | Paxson | 4SG | fBm-SGA |
|---|---|---|---|---|
| 0,012064 | 0,00930 | 0,01117 | 0,01513 | 0,02129 |

Table 5.4: Average processing time measured in seconds of 20 series of a given length generated with the Hosking method

| | $2^{16}$ | $2^{18}$ | $2^{20}$ |
|---|---|---|---|
| R/S | 0,82656 | 3,15176 | 12,48033 |
| VT | 0,84554 | 3,12868 | 11,99754 |
| AMT | 0,85564 | 3,85445 | 12,59458 |
| EBP | 0,02591 | 1,22448 | 4,84564 |
| HEAF | 0,001147 | 0,00460 | 0,07216 |
| HIG | 7,02145 | 113,8947 | 1810,95895 |
| DFA | 0,23715 | 1,04020 | 5,11899 |
| PER | 0,05262 | 0,22560 | 2,90498 |

Table 5.5: Approximated maximum memory consumption for `TestH` estimators as a function of the length of the series ($N$) under analysis.

| Estimator | R/S | VT | AMT | EBP | HIG | DFA | PER | HEAF |
|---|---|---|---|---|---|---|---|---|
| Memory | N | N | N | N | 3N | 3N | 6N | N |

Table 5.6: Average error for the other estimators implemented under the scope of this dissertation along with the AMT. Values presented for each combination estimator/generator. Sequences with length $2^{20}$.

| | HEAF | HIG | PER | DFA |
|---|---|---|---|---|
| Hosking | 0,00096 | 0,01813 | 0,0118 | 0,00884 |
| Davies and Hart | 0,00322 | 0,00987 | 0,01688 | 0,01279 |
| Paxson | 0,00254 | 0,00128 | 0,00677 | 0,01173 |
| fBm-SGA | 0,042295 | 0,00218 | 0,00111 | 0,01782 |
| 4SG | 0,000289 | 0,01494 | 0,0019 | 0,000593 |

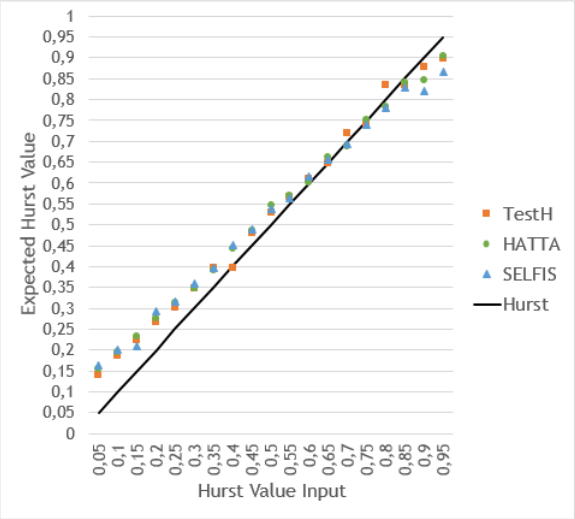


Figure 5.1: Accuracy related results obtained for the R/S estimator.

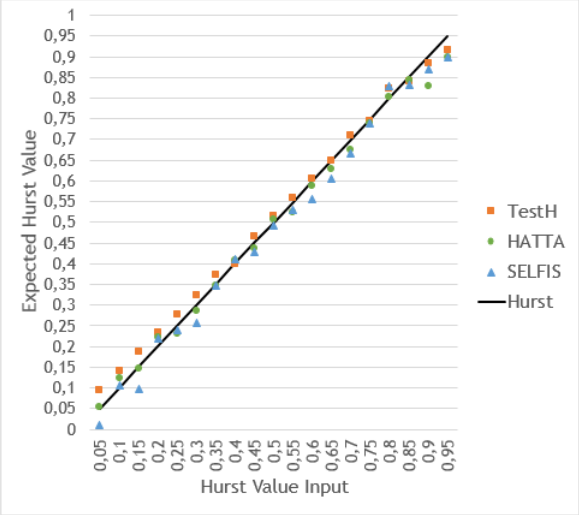


Figure 5.2: Accuracy related results obtained for the VT estimator.

Table 5.7: Average error for the estimators implemented under the scope of this dissertation. Values presented for each combination estimator/generator. Sequences with length ($2^{12}$).

| | HEAF | HIG | DFA | PER |
|---|---|---|---|---|
| Hosking | 0,144954 | 0,033922 | 0,06135 | 0,004715 |
| Davies and Hart | 0,08651 | 0,054692 | 0,064713 | 0,038236 |
| Paxson | 0,097462 | 0,123507 | 0,066947 | 0,174994 |
| fBm-SGA | 0,560249 | 0,204133 | 0,069827 | 0,25385 |
| 4SG | 0,108385 | 0,097792 | 0,037164 | 0,157843 |

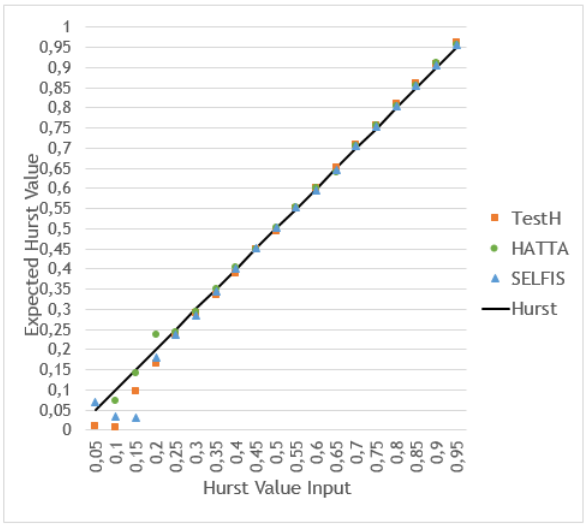


Figure 5.3: Accuracy related results obtained for the Periodogram estimator.

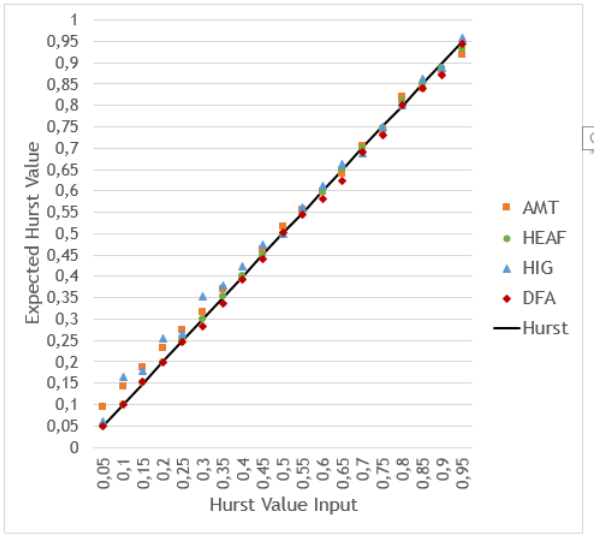


Figure 5.4: Accuracy related results obtained for the estimators implemented under the scope of this dissertation.

# Chapter 6

# Conclusions and Future Work

The main conclusions drawn from the work described in this dissertation are presented in section 6.1, and guidelines to keep pushing the work forward are discussed in section 6.2.

## 6.1 Main Conclusions

The literature studied during this work leaves no space for doubting the important role that the self-similarity plays in fields such as the modelling and simulation of network traffic, stock market analysis and even the study of biological phenomena. The analysis of the literature revealed that the existing solutions are not perfect, lacking several algorithms, have not been updated for quite some time, lack proper documentation, do not reveal the implementation nuances of the algorithms, and have constraints regarding the data length, amongst others.

The work presented in this dissertation is the continuation of an effort to create a tool to study and simulated self-similar processes, which embraces these three main principles: (i) reliability – the users need to have a certain degree of confidence regarding the results the tool outputs; (ii) dynamic – the tool should be easily adapted to the needs of users; and (iii) *made for humans* – meaning that a researcher should be able to quickly start using the Library with little or no skills regarding C programming.

The work evolved to the identification of the self-similar self-similar sequence generators and the Hurst Parameter estimators to be implemented. The chosen generator was the fastest exact self-similar sequence generator known to date, and its choice was mostly due to those two characteristics. Regarding the estimators, the only concern taken into account was if they were included on the list of the most important estimators, discussed in chapter 2.Prior to the implementation, the development apparatus was set up. In other words, in order to develop a new estimator, a set of sequences with known Hurst values were generated from the generators already available on `TestH`. This allowed the TDD technic to be applied as it was possible to validate the results right away. With the apparatus defined, the development (and most difficult phase) began. The documentation regarding the algorithms is scarce and leaves room for doubts. The usage of ANSI C also added an extra layer of complexity to this work, as it gives no support for encapsulating complex mathematical principles (e.g. manipulating complex numbers). In this last case, as the development of the library keeps moving forward, it becomes easier to implement new algorithms, as most of the mathematical functions in `TestH` are exposed as an

API.

Several tests on different tools (SELFIS, and HATTA) were carried out, mainly on the algorithms integrated in TestH under the scope of this dissertation, but not limited to them. This was performed to fulfill the reliability principle mentioned above.The most interesting test is perhaps the one that compares the Periodogram on the TestH and its peers, where one can clearly see that the biased behavior close to the origin is verified in every tool analyzed. This common behavior is also verified on R/S, another method available on every tool. This may indicate that the underlying implementation of this algorithm does not vary much from solution to solution.

From the main estimator algorithms set, TestH is still missing two important ones: the Abry-Veitch (AV) and Whittle, whose implementation had to be sacrificed in order to properly document the current API. Nonetheless, the objectives for this work were achieved and a significant step towards the library usability was performed with its website on-line and the API documented. This does not mean that there is little room for improvement, as there are new algorithms coming out from time to time, new features could be implemented and the algorithms could always be tested more. There are also minor bugs that need to be fixed in the following algorithms: AMT, VT, EBP and the Paxson method. This bugs are related the length of the time series or the configurable parameters, which can be fixed by validating such inputs.

As it was discussed in this dissertation, the description of some algorithms is not explicit enough on which values should be used at certain points. Taking this lack of information as an advantage, an empirical fine-tuning effort was made to minimize errors of the estimators, resulting in the setting up of *magical numbers* that should be replaced once proper studies are made regarding the best values to use.

## 6.2 Future Work

TestH incorporates more generators and estimators than other closely related tools. However, it does not provide a GUI to interact with the program. To counterbalance this disadvantage, TestH needs to be the most reliable tool available, such that the users see an added value in using it on their experiments. In order for this to happen, the missing algorithms should be added to the library. Besides these algorithms, there are some other interesting functions that related tools possess, such as the possibility to apply an estimator only to a certain interval of points, that might be interesting to integrate also. This is something that can be done programmatically on TestH but the development should be done in view of those who have little programming skills.

One feature that draws a lot of attention on SELFIS and HATTA is the possibility to plot graphs. This kind feature on TestH is not possible to reproduce as it will require the usage of non ANSI C libraries. Nonetheless an integration with the GNU Plot could be made by generating specific

GNU Plot files and explain its usage in the `TestH` documentation.

As `TestH` is an open source project (everyone is invited to collaborate), a way to guarantee that a future contribution will not jeopardize the already produced code is by having a new module (say `utest.h` with a set of unitary tests defined. This could be executed before accepting any merge request on the master branch to assure that the new additions do not have a negative impact on the production code. The function result `passed` or `reproved` could be simply modulated as an integer data type. In addition to this suggestion, a detailed report could be printed with the test results.

Creating more support material such as a tutorial to get the software up and running, documentation with respect to the integration of more algorithms and more examples should also be considered.

Last but not least, the final objective of this work (and of `TestH`) is only fulfilled if the tool is effectively used. The usage of this tool should be encouraged as a means of obtaining the approval from the scientific community, which will validate the implementation and make potential contributions to the source code. This will be made by publishing this work on high quality international venues in the future.

# Bibliography


[AV98] Patrice Abry and Darryl Veitch. Wavelet analysis of long-range-dependent traffic. *IEEE transactions on information theory*, 44(1):2--15, 1998. 14

[Bec03] Kent Beck. *Test-driven development: by example*. Addison-Wesley Professional, 2003. 3

[cap] Capitalization conventions. `tps://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/capitalization-conventions`. Accessed: 2017-05-18. 26

[Cle06] Richard G Clegg. A practical guide to measuring the hurst parameter. *arXiv preprint math/0610756*, 2006. 1

[CP16] Jean-Francois Coeurjolly and Emilio Porcu. Fast and exact simulation of complex-valued stationary gaussian processes through embedding circulant matrix. *arXiv preprint arXiv:1604.00362*, 2016. 15

[CPB08] Edward A Codling, Michael J Plank, and Simon Benhamou. Random walk models in biology. *Journal of the Royal Society Interface*, 5(25):813--834, 2008. 9

[CSZ07] YangQuan Chen, Rongtao Sun, and Anhong Zhou. An improved hurst parameter estimator based on fractional fourier transform. In *ASME 2007 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 1223--1233. American Society of Mechanical Engineers, 2007. 10

[DH87] Robert B Davies and DS Harte. Tests for hurst effect. *Biometrika*, 74(1):95--101, 1987. xiv, 16

[Die04] Ton Dieker. Simulation of fractional brownian motion. *MSc theses, University of Twente, Amsterdam, The Netherlands*, 2004. 14

[Enr04] Nathanaël Enriquez. A simple construction of the fractional brownian motion. *Stochastic Processes and their Applications*, 109(2):203--223, 2004. 9

[fAR] fARMA cran - package. `https://cran.r-project.org/web/packages/fArma/index.html`. Accessed: 2017-02-14. 20

[FNS+14] Diogo A. B. Fernandes, Miguel Neto, Liliana F. B. Soares, Mário M. Freire, and Pedro R. M. Inácio. A tool for estimating the hurst parameter and for generating self-similar sequences. In *Proceedings of the 2014 Summer Simulation Multiconference*, SummerSim '14, pages 40:1--40:8, San Diego, CA, USA, 2014. Society for Computer Simulation International. Available from: `http://dl.acm.org/citation.cfm?id=2685617.2685657`. vii, x, xvii, 2

[git] Git a short story. `https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git`. Accessed: 2017-05-23. 26

[GPH83] John Geweke and Susan Porter-Hudak. The estimation and application of long memory time series models. *Journal of time series analysis*, 4(4):221--238, 1983. xiii, 13

[Hig88] Tomoyuki Higuchi. Approach to an irregular time series on the basis of the fractal theory. *Physica D: Nonlinear Phenomena*, 31(2):277--283, 1988. vii, ix, xiii, xvii, 1, 14

[Hos81] Jonathan RM Hosking. Fractional differencing. *Biometrika*, 68(1):165--176, 1981. 15

[Hos84] Jonathan RM Hosking. Modeling persistence in hydrological time series using fractional differencing. *Water resources research*, 20(12):1898--1908, 1984. 15

[HPS+12] Richard Hardstone, Simon-Shlomo Poil, Giuseppina Schiavone, Rick Jansen, Vadim V Nikulin, Huibert D Mansvelder, and Klaus Linkenkaer-Hansen. Detrended fluctuation analysis: a scale-free view on neuronal oscillations. *Scale-free Dynamics and Critical Phenomena in Cortical Activity*, page 75, 2012. vii, ix, xvii, 1

[Hur51] Harold Edwin Hurst. Long-term storage capacity of reservoirs. *Trans. Amer. Soc. Civil Eng.*, 116:770--808, 1951. vii, ix, xvii, 1, 10

[Ina09] Pedro RM Inacio. Study of the impact of intensive attacks in the self-similarity degree of the network traffic in intra-domain aggregation points, 2009. 16, 17, 34

[JS04] Owen Dafydd Jones and Yuan Shen. Estimating the hurst index of a self-similar process via the crossing tree. *IEEE Signal Processing Letters*, 11(4):416--419, 2004. 12

[Kem03] Julia Kempe. Quantum random walks: an introductory overview. *Contemporary Physics*, 44(4):307--327, 2003. 9

[KF02] Thomas Karagiannis and Michalis Faloutsos. Selfis: a tool for self-similarity and long-range dependence analysis. In *1st Workshop on Fractals and Self-Similarity in Data Mining: Issues and Approaches (in KDD), Edmonton, Canada*, 2002. 19

[KG02] Houssain Kettani and John A Gubner. A novel approach to the estimation of the hurst parameter in self-similar traffic. In *Local Computer Networks, 2002. Proceedings. LCN 2002. 27th Annual IEEE Conference on*, pages 160--165. IEEE, 2002. xiv, 15

[KRD11] Ludmila Kirichenko, Tamara Radivilova, and Zhanna Deineko. Comparative analysis for estimating of the hurst exponent for stationary and nonstationary time series. *Information Technologies & Knowledge*, 5(1):371--388, 2011. 12

[KTC07] V Kuksenko, N Tomilin, and A Chmel. The rock fracture experiment with a drive control: A spatial aspect. *Tectonophysics*, 431(1):123--129, 2007. vii, ix, xvii, 1

[LLLN09] Lucas Lacasa, Bartolo Luque, Jordi Luque, and Juan Carlos Nuno. The visibility graph: A new method for estimating the hurst exponent of fractional brownian motion. *EPL (Europhysics Letters)*, 86(3):30001, 2009. 10

[LTWW93] Will E Leland, Murad S Taqqu, Walter Willinger, and Daniel V Wilson. On the self-similar nature of ethernet traffic. In *ACM SIGCOMM Computer Communication Review*, volume 23, pages 183--193. ACM, 1993. ix, 1

[Man65] Benoit Mandelbrot. Une classe de processus stochastiques homothetiques a soi-application a la loi climatologique de he hurst. *COMPTES RENDUS HEBDOMADAIRES DES SEANCES DE L ACADEMIE DES SCIENCES*, 260(12):3274--+, 1965. 10

[Man67] Benoit B Mandelbrot. How long is the coast of britain. *science*, 156(3775):636--638, 1967. 7

[MN98] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3--30, 1998. 25

[MVN68] Benoit B Mandelbrot and John W Van Ness. Fractional brownian motions, fractional noises and applications. *SIAM review*, 10(4):422--437, 1968. 10

[MW69] Benoit B Mandelbrot and James R Wallis. Robustness of the rescaled range r/s in the measurement of noncyclic long run statistical dependence. *Water Resources Research*, 5(5):967--988, 1969. 10

[NNNN67] Edward Nelson, Edward Nelson, Edward Nelson, and Edward Nelson. *Dynamical theories of Brownian motion*, volume 3. Princeton university press Princeton, 1967. 9

[Øks03] Bernt Øksendal. Fractional brownian motion in finance. *Preprint series. Pure mathematics http://urn. nb. no/URN: NBN: no-8076*, 2003. 9

[Pax97] Vern Paxson. Fast, approximate synthesis of fractional gaussian noise for generating self-similar network traffic. *ACM SIGCOMM Computer Communication Review*, 27(5):5--18, 1997. 10, 16

[PBH+94] C-K Peng, Sergey V Buldyrev, Shlomo Havlin, Michael Simons, H Eugene Stanley, and Ary L Goldberger. Mosaic organization of dna nucleotides. *Physical review e*, 49(2):1685, 1994. xiv, 13

[PHSG95] C-K Peng, Shlomo Havlin, H Eugene Stanley, and Ary L Goldberger. Quantification of scaling exponents and crossover phenomena in nonstationary heartbeat time series. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 5(1):82--87, 1995. 13

[PMK14] Al-Sakib Khan Pathan, Muhammad Mostafa Monowar, and Shafiullah Khan. *Simulation Technologies in Networking and Communications: Selecting the Best Tool for the Test*. CRC Press, 2014. 19

[PRV08] Julio César Ramırez Pachecoa, Deni Torres Romána, and Leopoldo Estrada Vargasa. Software tools for fractal analysis. 2008. 2, 20

[R+02] Peter Robinson et al. Long-range dependence. *Encyclopedia of Environmetrics*, 2002. 1

[rLa] R what is r? `https://www.r-project.org/about.html`. Accessed: 2017-02-18. 20

[RPGC06] Karim Mohammed Rezaul, Algirdas Pakstas, Robert Gilchrist, and Thomas M Chen. Heaf: a novel estimator for long-range dependent self-similar network traffic. In *NEW2AN*, pages 34--45. Springer, 2006. vii, ix, xiv, xvii, 1, 15

[RT05] PJC Ramirez and RD Torres. A tool for analysis of internet metrics. In *Electrical and Electronics Engineering, 2005 2nd International Conference on*, pages 60--63. IEEE, 2005. xxiii, 20, 21

[Sam06] Gennady Samorodnitsky. Long memory and self-similar processes. In *ANNALES-FACULTE DES SCIENCES TOULOUSE MATHEMATIQUES*, volume 15, page 107. Université Paul Sabatier, 2006. xii, 8

[Spi13] Frank Spitzer. *Principles of random walk*, volume 34. Springer Science & Business Media, 2013. 9

[STP+06] Stilian Stoev, Murad S Taqqu, Cheolwoo Park, George Michailidis, and JS Marron. Lass: a tool for the local analysis of self-similarity. *Computational statistics & data analysis*, 50(9):2447--2471, 2006. xiii, 22

[THJ07] Xianhai Tan, Yuanhui Huang, and Weidong Jin. Modeling and performance analysis of self-similar traffic based on fbm. In *Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on*, pages 543--548. IEEE, 2007. 9

[TT97] Murad S Taqqu and Vadim Teverovsky. Robustness of whittle-type estimators for time series with long-range dependence. *Communications in statistics. Stochastic models*, 13(4):723--757, 1997. 14

[TTW95] Murad S Taqqu, Vadim Teverovsky, and Walter Willinger. Estimators for long-range dependence: an empirical study. *Fractals*, 3(04):785--798, 1995. 32

[VAJ08] Ghazaleh Vaziri, Farshad Almasganj, and Mohammad Sadegh Jenabi. On the fractal self-similarity of laryngeal pathologies detection: the estimation of hurst parameter. In *Information Technology and Applications in Biomedicine, 2008. ITAB 2008. International Conference on*, pages 383--386. IEEE, 2008. vii, ix, xvii, 1

[WTSW97] Walter Willinger, Murad S Taqqu, Robert Sherman, and Daniel V Wilson. Self-similarity through high-variability: statistical analysis of ethernet lan traffic at the source level. *IEEE/ACM Transactions on Networking (ToN)*, 5(1):71--86, 1997. 16

# Appendix A

# Full Set of Results Per Estimator and Generator

This appendix contains tables A.1, A.2, A.3, A.4, which shows the results obtained for TestH estimators when exposed to the `TestH` Davies and Hart, Paxson, 4SG and fBm-SGA algorithms. It is also present table A.5, which shows a full set of results obtained for each estimator using the Hosking method as chapter 5 contains only the average values of the ones presented herein.

Table A.1: Average estimations for the algorithms present in TestH resulting from 20 time series of length $2^{20}$ produced with the Davies and Hart method.

| Expected | R/S | VT | AMT | HEAF | HIG | PER | DFA |
|---|---|---|---|---|---|---|---|
| 0,05 | 0,14272 | 0,09694 | 0,09612 | 0,05115 | 0,04388 | 0,01969 | 0,0487 |
| 0,10 | 0,19198 | 0,14593 | 0,14538 | 0,09989 | 0,08196 | 0,01355 | 0,09823 |
| 0,15 | 0,22571 | 0,18848 | 0,18534 | 0,15124 | 0,1337 | 0,08241 | 0,14181 |
| 0,2 | 0,25988 | 0,23017 | 0,22924 | 0,20046 | 0,19081 | 0,15781 | 0,19767 |
| 0,25 | 0,30361 | 0,27725 | 0,27549 | 0,25088 | 0,24372 | 0,22562 | 0,24562 |
| 0,3 | 0,34281 | 0,32317 | 0,31788 | 0,30353 | 0,2939 | 0,28716 | 0,28764 |
| 0,35 | 0,38376 | 0,36737 | 0,36209 | 0,35098 | 0,34203 | 0,33791 | 0,33515 |
| 0,4 | 0,44036 | 0,4194 | 0,41555 | 0,39844 | 0,39389 | 0,39874 | 0,3869 |
| 0,45 | 0,46533 | 0,45643 | 0,45148 | 0,44754 | 0,43877 | 0,43341 | 0,43267 |
| 0,5 | 0,50665 | 0,50272 | 0,49709 | 0,4988 | 0,49124 | 0,49478 | 0,4819 |
| 0,55 | 0,57582 | 0,56248 | 0,55626 | 0,54914 | 0,54172 | 0,54933 | 0,53049 |
| 0,6 | 0,59748 | 0,59783 | 0,59239 | 0,59818 | 0,59481 | 0,61392 | 0,58188 |
| 0,65 | 0,66458 | 0,6581 | 0,65688 | 0,65162 | 0,65161 | 0,6589 | 0,64796 |
| 0,7 | 0,70086 | 0,7002 | 0,69393 | 0,69953 | 0,69326 | 0,70582 | 0,68071 |
| 0,75 | 0,76488 | 0,75741 | 0,75153 | 0,74994 | 0,74403 | 0,75567 | 0,73229 |
| 0,8 | 0,79767 | 0,79809 | 0,79474 | 0,79851 | 0,79665 | 0,8093 | 0,78846 |
| 0,85 | 0,83909 | 0,84266 | 0,83766 | 0,84623 | 0,84119 | 0,85105 | 0,83123 |
| 0,9 | 0,8793 | 0,88444 | 0,88102 | 0,88959 | 0,88718 | 0,89806 | 0,87933 |
| 0,95 | 0,88742 | 0,89532 | 0,90366 | 0,90321 | 0,90982 | 0,88618 | 0,92825 |

Table A.2: Average estimations for the algorithms present in TestH resulting from 20 time series of length $2^{20}$ produced with the Paxson method.

| Expected | R/S | VT | AMT | HEAF | HIG | PER | DFA |
|---|---|---|---|---|---|---|---|
| 0,05 | 0,14381 | 0,09557 | 0,09644 | 0,04734 | 0,04161 | 0,01544 | 0,04995 |
| 0,10 | 0,18406 | 0,14165 | 0,14066 | 0,09923 | 0,09689 | 0,06544 | 0,09628 |
| 0,15 | 0,22661 | 0,18855 | 0,18806 | 0,15049 | 0,15607 | 0,09799 | 0,14902 |
| 0,2 | 0,26035 | 0,11685 | 0,16962 | 0,19787 | 0,19035 | 0,16708 | 0,19904 |
| 0,25 | 0,29686 | 0,18993 | 0,21433 | 0,25172 | 0,24804 | 0,23004 | 0,24069 |
| 0,3 | 0,34942 | 0,25697 | 0,24172 | 0,2986 | 0,30272 | 0,28467 | 0,27464 |
| 0,35 | 0,39127 | 0,36695 | 0,3783 | 0,34975 | 0,35501 | 0,34452 | 0,34446 |
| 0,4 | 0,42396 | 0,37602 | 0,36092 | 0,39903 | 0,39555 | 0,38809 | 0,39361 |
| 0,45 | 0,48189 | 0,44228 | 0,44432 | 0,44892 | 0,45459 | 0,45737 | 0,44088 |
| 0,5 | 0,51024 | 0,5185 | 0,51351 | 0,499 | 0,51395 | 0,50149 | 0,48513 |
| 0,55 | 0,56231 | 0,57054 | 0,57989 | 0,54944 | 0,57652 | 0,55577 | 0,53071 |
| 0,6 | 0,60714 | 0,60011 | 0,611 | 0,59919 | 0,59479 | 0,61174 | 0,58142 |
| 0,65 | 0,65783 | 0,6412 | 0,63534 | 0,64891 | 0,65164 | 0,65694 | 0,63735 |
| 0,7 | 0,69659 | 0,71584 | 0,71546 | 0,69794 | 0,71759 | 0,70804 | 0,68767 |
| 0,75 | 0,72709 | 0,76591 | 0,77571 | 0,74963 | 0,77165 | 0,75976 | 0,73799 |
| 0,8 | 0,77098 | 0,80343 | 0,79732 | 0,79809 | 0,81497 | 0,79571 | 0,78005 |
| 0,85 | 0,83563 | 0,8413 | 0,83515 | 0,84697 | 0,82352 | 0,86258 | 0,82854 |
| 0,9 | 0,86347 | 0,87952 | 0,88129 | 0,89557 | 0,90525 | 0,90952 | 0,90087 |
| 0,95 | 0,89254 | 0,90834 | 0,90657 | 0,92414 | 0,86499 | 0,95928 | 0,91884 |

Table A.3: Average estimations for the algorithms present in TestH resulting from 20 time series of length $2^{20}$ produced with the 4SG method.

| Expected | R/S | VT | AMT | HEAF | HIG | PER | DFA |
|---|---|---|---|---|---|---|---|
| 0,05 | 0,20966 | 0,13069 | 0,15683 | 0,05172 | 0,12517 | 0,09757 | 0,13013 |
| 0,10 | 0,23433 | 0,16683 | 0,18501 | 0,09934 | 0,14169 | 0,14179 | 0,15388 |
| 0,15 | 0,25654 | 0,2034 | 0,21476 | 0,15027 | 0,17119 | 0,18114 | 0,18433 |
| 0,2 | 0,28624 | 0,24415 | 0,25293 | 0,20206 | 0,23036 | 0,21491 | 0,22839 |
| 0,25 | 0,32311 | 0,28726 | 0,29339 | 0,2514 | 0,29146 | 0,276 | 0,26981 |
| 0,3 | 0,35204 | 0,32633 | 0,32543 | 0,30062 | 0,34013 | 0,29717 | 0,29794 |
| 0,35 | 0,39687 | 0,37237 | 0,37314 | 0,34786 | 0,3352 | 0,35953 | 0,35018 |
| 0,4 | 0,42789 | 0,41232 | 0,41376 | 0,39675 | 0,4144 | 0,39259 | 0,40109 |
| 0,45 | 0,47286 | 0,46126 | 0,45165 | 0,44966 | 0,42715 | 0,44329 | 0,42083 |
| 0,5 | 0,51434 | 0,50814 | 0,49718 | 0,50195 | 0,4539 | 0,49355 | 0,46907 |
| 0,55 | 0,57374 | 0,56266 | 0,55221 | 0,55157 | 0,5632 | 0,54916 | 0,52023 |
| 0,6 | 0,59857 | 0,59833 | 0,58807 | 0,59808 | 0,58258 | 0,58717 | 0,56731 |
| 0,65 | 0,64954 | 0,65027 | 0,64231 | 0,651 | 0,67066 | 0,63831 | 0,6271 |
| 0,7 | 0,69829 | 0,69974 | 0,6896 | 0,70118 | 0,69301 | 0,7017 | 0,67077 |
| 0,75 | 0,73172 | 0,74024 | 0,73369 | 0,74877 | 0,77675 | 0,73862 | 0,72912 |
| 0,8 | 0,76926 | 0,78507 | 0,78047 | 0,80088 | 0,81746 | 0,7873 | 0,78708 |
| 0,85 | 0,80447 | 0,83009 | 0,82416 | 0,8557 | 0,88037 | 0,81772 | 0,83792 |
| 0,9 | 0,88563 | 0,89176 | 0,89114 | 0,89788 | 0,87768 | 0,86283 | 0,89602 |
| 0,95 | 0,87048 | 0,90415 | 0,928 | 0,93782 | 0,99155 | 0,88367 | 0,94755 |

Table A.4: Average estimations for the algorithms present in TestH resulting from 20 time series of length $2^{20}$ produced with the fBm-SGA method.

| Expected | R/S | VT | AMT | HEAF | HIG | PER | DFA |
|---|---|---|---|---|---|---|---|
| 0,5 | 0,53003 | 0,51402 | 0,50941 | 0,49801 | 0,4919 | 0,50163 | 0,48417 |
| 0,55 | 0,55749 | 0,54127 | 0,54169 | 0,52504 | 0,55642 | 0,54804 | 0,52632 |
| 0,6 | 0,60404 | 0,58147 | 0,58626 | 0,5589 | 0,59557 | 0,59651 | 0,57329 |
| 0,65 | 0,66233 | 0,62929 | 0,64471 | 0,59625 | 0,69378 | 0,65054 | 0,64252 |
| 0,7 | 0,70121 | 0,67075 | 0,68067 | 0,64029 | 0,74299 | 0,69919 | 0,67004 |
| 0,75 | 0,75451 | 0,72143 | 0,73518 | 0,68836 | 0,74236 | 0,76042 | 0,7296 |
| 0,8 | 0,80722 | 0,77412 | 0,78737 | 0,74102 | 0,77594 | 0,80491 | 0,78077 |
| 0,85 | 0,81385 | 0,80204 | 0,80877 | 0,79023 | 0,83425 | 0,84898 | 0,81041 |
| 0,9 | 0,86853 | 0,86111 | 0,87492 | 0,85369 | 0,8767 | 0,89668 | 0,89511 |
| 0,95 | 0,89526 | 0,9261 | 0,93205 | 0,93528 | 0,96191 | 0,95418 | 0,9596 |

Table A.5: Average estimations for the algorithms present in TestH, Herramienta de Análisis de Tráfico Telemático Autosemejante and SELF-similarity analysIS tools resulting from 20 time series of length $2^{20}$ ($2^{18}$ for SELF-similarity analysIS) for each expected Hurst value, produced with the Hosking method.

| | TestH | | | | | | | HATTA | | | SELFIS | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Expected** | **R/S** | **VT** | **AMT** | **HEAF** | **HIG** | **PER** | **DFA** | **R/S** | **VT** | **PER** | **R/S** | **VT** | **PER** |
| 0,05 | 0,13986 | 0,09503 | 0,09521 | 0,0502 | 0,0617 | 0,01023 | 0,05073 | 0,1555 | 0,05438 | -0,0284 | 0,162 | 0,011 | 0,069 |
| 0,10 | 0,1852 | 0,14244 | 0,14218 | 0,09968 | 0,16565 | 0,00629 | 0,09891 | 0,1935 | 0,1238 | 0,0715 | 0,202 | 0,106 | 0,033 |
| 0,15 | 0,22363 | 0,18685 | 0,18781 | 0,15007 | 0,1781 | 0,09611 | 0,15294 | 0,2318 | 0,1472 | 0,14 | 0,209 | 0,098 | 0,03 |
| 0,2 | 0,26692 | 0,23337 | 0,23329 | 0,19983 | 0,25505 | 0,16586 | 0,19958 | 0,2758 | 0,2235 | 0,2379 | 0,292 | 0,218 | 0,179 |
| 0,25 | 0,30184 | 0,27645 | 0,27512 | 0,25106 | 0,26513 | 0,2358 | 0,24708 | 0,3141 | 0,2298 | 0,244 | 0,317 | 0,239 | 0,237 |
| 0,3 | 0,34636 | 0,32336 | 0,3178 | 0,30036 | 0,35379 | 0,29115 | 0,28368 | 0,347 | 0,2857 | 0,2924 | 0,359 | 0,256 | 0,286 |
| 0,35 | 0,39668 | 0,37457 | 0,36946 | 0,35246 | 0,37941 | 0,33387 | 0,33713 | 0,3901 | 0,3458 | 0,349 | 0,398 | 0,348 | 0,345 |
| 0,4 | 0,39688 | 0,39858 | 0,39649 | 0,40027 | 0,42375 | 0,38823 | 0,39403 | 0,4425 | 0,4089 | 0,4045 | 0,451 | 0,411 | 0,401 |
| 0,45 | 0,48058 | 0,46507 | 0,46201 | 0,44957 | 0,47432 | 0,44989 | 0,4404 | 0,4851 | 0,4376 | 0,4502 | 0,49 | 0,428 | 0,453 |
| 0,5 | 0,52935 | 0,51481 | 0,51574 | 0,50028 | 0,49873 | 0,49408 | 0,50306 | 0,5461 | 0,5065 | 0,502 | 0,537 | 0,492 | 0,503 |
| 0,55 | 0,56792 | 0,55944 | 0,55695 | 0,55096 | 0,56277 | 0,54908 | 0,54348 | 0,5698 | 0,5237 | 0,5532 | 0,565 | 0,529 | 0,552 |
| 0,6 | 0,61019 | 0,60483 | 0,59845 | 0,59946 | 0,61281 | 0,60087 | 0,58032 | 0,6015 | 0,5878 | 0,6001 | 0,615 | 0,556 | 0,596 |
| 0,65 | 0,64613 | 0,64818 | 0,63932 | 0,65023 | 0,6637 | 0,65112 | 0,62365 | 0,663 | 0,6296 | 0,6404 | 0,655 | 0,605 | 0,647 |
| 0,7 | 0,71884 | 0,70871 | 0,70581 | 0,69858 | 0,68924 | 0,70774 | 0,68989 | 0,6885 | 0,6739 | 0,7043 | 0,693 | 0,667 | 0,706 |
| 0,75 | 0,74073 | 0,74465 | 0,73885 | 0,74857 | 0,75138 | 0,75547 | 0,73117 | 0,7507 | 0,7383 | 0,7553 | 0,739 | 0,739 | 0,753 |
| 0,8 | 0,8343 | 0,82175 | 0,81902 | 0,80919 | 0,80149 | 0,80956 | 0,80101 | 0,7833 | 0,8031 | 0,8052 | 0,779 | 0,829 | 0,804 |
| 0,85 | 0,83479 | 0,84177 | 0,83959 | 0,84875 | 0,86135 | 0,85957 | 0,84221 | 0,8396 | 0,8421 | 0,8553 | 0,828 | 0,833 | 0,856 |
| 0,9 | 0,87952 | 0,88374 | 0,87792 | 0,88796 | 0,8885 | 0,90811 | 0,87051 | 0,8461 | 0,8291 | 0,9105 | 0,821 | 0,871 | 0,906 |
| 0,95 | 0,89724 | 0,91555 | 0,91874 | 0,93386 | 0,95737 | 0,96275 | 0,94344 | 0,9047 | 0,8987 | 0,9567 | 0,866 | 0,898 | 0,956 |