Computationally efficient Bayesian approximation of fractional Gaussian noise using AR1 processes

Eirik Myrvoll-Nilsen

Master of Science in Physics and Mathematics
Submission date: September 2016
Supervisor: Håvard Rue, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

The goal of this thesis is to explore a way of performing efficient Bayesian inference of fractional Gaussian noise series using the R-INLA framework. Finding the MLE of the Hurst exponent and the innovation variance of an FGN can easily be implemented for INLA using a latent Gaussian model. However, since the variables of an FGN process are conditionally dependent, the INLA program will run so slow that it is not deemed viable as a method for performing Bayesian inference. To combat this another approach is considered, namely to approximate the FGN as a weighted sum of AR1 models with parameters that are determined by numerical optimization techniques. The AR1 models and the weighted sum forms another latent field for the LGM, one that has more variables, but less conditional dependence. This approximation is revealed to be much faster, but also more inaccurate than the previous model, as the accuracy of the estimation was found to be biased and connected to the true value of the Hurst exponent. This could be improved if the process of finding the parameters for the AR1 models was repeated with more precision.

Sammendrag

I denne oppgaven utforskes en ny metode for å oppnå Bayesiansk inferens av fraksjonelt Gaussisk støy ved å bruke R-INLA rammeverket. Å finne sannsynlighetsmaksimeringsestimatoren av Hurst eksponenten og innovasjonsvariansen til en FGN kan enkelt bli implementert for INLA ved å bruke en latent Gaussisk modell. Siden variablene i en FGN prosess er betinget avhengige vil INLA kjøre så sakte at denne metoden dermed ikke vil være nyttig i praksis. I håp om å øke hastigheten blir FGN prosessen approksimert som en vektet sum av et endelig antall AR1 modeller som blir funnet fra numeriske metoder. AR1 tidsrekkene og den vektede summen utgjør et nytt underliggende felt i den latente Gaussiske modellen. Den har flere variabler, men disse er i større grad betinget uavhengige. Denne tilnærmingen blir målt til å være langt raskere, men ikke like nøyaktig som den ordinære modellen. Unøyaktigheten i modllen er særlig knyttet til den sanne Hurst eksponenten som skal bli målt. Dette kan i teorien bli forbedret om optimeringsprosedyren hvor vektene og parametrene for AR1 modellene blir funnet blir gjentatt med større presisjon.

Preface

The master thesis presented consists of 30 ECTS credits and completes my Master of science degree in physics and mathematics at The Norwegian University of Science and Technology (NTNU). It was written at the Department of Mathematical Sciences under the supervision of Håvard Rue.

I would especially like to thank my advisor Håvard Rue for all the help, guidance and inspiration he provided throughout the course of this work, which was greatly appreciated.

I would also like to thank Sigrunn Holbek Sørbye at the University of Tromsø for valuable feedback and for introducing me to examples regarding climate dynamics.

Contents

Chapter 1

Introduction

Fractional Gaussian noise (FGN) is a stationary time series process that exhibit the *long-memory* property. There are different definitions of this property, but models displaying it typically exhibit a slower than exponential decay of dependence between two points as the distance between them increases. FGN processes are therefore well suited to model processes of long interdependence between points in time and are frequently used in fields such as finance, hydrology and climate dynamics.

The decay of dependence of a fractional Gaussian noise process is explained only by a single parameter, the *Hurst exponent*. Methods of estimating this parameter has been thoroughly discussed in many studies, but yet there is not an efficient methodology for performing Bayesian inference of long FGN series, nor is there a computational Bayesian framework that properly support FGN terms in a linear predictor explaining the data.

Bayesian inference about the Hurst exponent can be performed by restating the likelihood function of the FGN as a latent field in a Bayesian hierarchical model, more specifically a *latent Gaussian model* (LGM). Inference from these models can be obtained by the *integrated nested Laplace approximation* (INLA) method, which is supported by a computational Bayesian framework called R-INLA that is compatible with the programming environment R. FGN models are easily implemented for INLA as LGMs, but since the temporal points of an FGN processes are by nature very dependent, the latent field will have a dense precision matrix which restricts the efficiency of INLA.

It is known that the sum of short term processes will exhibit long-memory when the parameters are drawn randomly from a distribution. The aim of this thesis is to investigate the method of approximating an FGN as a sum of AR1 processes, such that the associated latent field of the LGM will have a sparse precision matrix that allows for faster computations with INLA.

Chapter 2 introduces theory essential to the understanding of fractional Gaussian noise and other long-memory processes. Different models of long-memory is presented and the associated properties are discussed. Methods of simulating fractional Gaussian noise processes will also be established.

Chapter 3 concerns the problem of estimating the Hurst exponent. Different methods is presented and compared by accuracy and efficiency. This chapter also discusses Bayesian inference and latent Gaussian modeling and how it can be applied to fractional Gaussian noise. The maximum likelihood estimator for the Hurst exponent is obtained by implementing a suitable LGM for INLA. The results are compared to non-Bayesian methods.

Chapter 4 introduces the concept of *aggregation*, or the theory that the sum of short term memory processes whose parameter are drawn randomly from a distribution will exhibit long-memory properties. Attempts are then made to approximate a long-memory process as a sum of weighted AR1 processes in the hopes of constructing an LGM that is more compatible with INLA. This is done by finding the optimal weights and coefficients of the AR1 process with numerical optimization methods. The latent model associated with the approximation is then implemented for INLA.

Chapter 5 performs analysis of both the speed and accuracy of the approximated latent field. The runtime and deviation from the exact MLE is measured against the number of AR1 components used, the length of the observed series and the Hurst exponent to be estimated. The model is then applied to real-life examples regarding the famous Nile water level data and the HadCRUT4 dataset for annual global average temperature anomalies.

Chapter 2

Time series and long-memory

2.1 Time series analysis

To understand fractional Gaussian noise and long-memory processes it is necessary to introduce fundamental theory about time series and stochastic processes. The first concept to be introduced in this thesis is the *random variable.* This is a variable whose value is subject to variation due to chance or randomness. The random variable X can take any value or *state* x from a state space Ω each with an associated probability, denoted

$$P(X = x). \tag{2.1}$$

A series of random variables can be collected into a *stochastic process* (26)

$$\{X(t) \mid t \in \mathbb{T}\}. \tag{2.2}$$

Each random variable $X(t)$ represents the state at point t in a set of indexes $\mathbb{T}$. If the indexes are interpreted as time the process is known as a *time series process.* If $\mathbb{T}$ is countable the process is a *discrete-time* series process. If on the other hand, $\mathbb{T}$ is a continuous sub-interval of $\mathbb{R}$ it is a *continuous-time* series process. A *realization* of a time series process, $\{x(t), t \in \mathbb{T}\}$ is a deterministic outcome of the process, often referred to simply as a *time series.* The thesis will focus on time series in discrete time, where $\{x_t\}$ will denote $\{X(t) \mid t \in \mathbb{T}\}$ and $\mathbb{T} = \{0, \pm 1, \pm 2, ...\}$.

The mean value of time series processes is a function of time, and defined as

$$\mu_t = \mathrm{E}[X_t]. \tag{2.3}$$

Similarly, the variance function and covariance function between the points X_{t_1} and X_{t_2} are defined as follows in 2.4 and 2.5

$$\sigma_t^2 = \mathrm{E}\left[(X_t - \mu_t)^2\right] \tag{2.4}$$

$$\gamma(t_1, t_2) = \mathrm{E}\left[(X_{t_1} - \mu_{t_1})(X_{t_2} - \mu_{t_2})\right]. \tag{2.5}$$

The correlation function between X_{t_1} and X_{t_2} is found from scaling the covariance function 2.5 by the product of the standard deviations at time t_1 and t_2,

$$\rho(t_1, t_2) = \frac{\gamma(t_1, t_2)}{\sqrt{\sigma_{t_1}^2 \sigma_{t_2}^2}}. \tag{2.6}$$

When the properties of a time series do not change with time, the series is said to be *stationary.* More specifically, a time series is an nth order *weakly stationary* time series if it has time-invariant joint moments up to order n. A 2nd-order weakly stationary time series will have constant mean $\mathrm{E}[X_t] = \mu < \infty$, variance $\mathrm{Var}(X_t) = \sigma^2 < \infty$ and covariance $\mathrm{Cov}(X_t, X_{t+k}) = \gamma(k)$. If the joint distribution is time-invariant it is said to be *strongly stationary.* The definitions for strong and weak stationarity are stated more clearly in Def: 2.1.1 and Def: 2.1.2.

Definition 2.1.1 (Strong stationarity)**.** *A time series process $\{X(t) \mid t \in \mathbb{T}\}$ is strongly stationary if the joint distribution is time-invariant, i.e. for any $t_1, ..., t_n, k, t_1 + k, ..., t_n + k$ the equation*

$$F_{X_{t_1},...,X_{t_n}}(x_1, ..., x_n) = F_{X_{t_1+k},...,X_{t_n+k}}(x_1, ..., x_n) \tag{2.7}$$

holds for all $n = 1, 2, ...$

Definition 2.1.2 (Weak stationarity)**.** *A time series process $\{X(t) \mid t \in \mathbb{T}\}$ is weakly stationary of order n if all its joint moments up to order n exist and are time invariant.*

For a stationary time series process $\{x_t\}$, the covariance and correlation between two points X_t and X_s depend only on the time difference, or *lag* $k = |t - s|$ between them and not where the points are located in the series. They are then called the *autocovariance-* and *autocorrelation functions* (ACF), defined in Equations 2.8 and 2.9:

$$\gamma_k = \mathrm{Cov}(X_t, X_{t+k}) \tag{2.8}$$

$$\rho_k = \frac{\gamma_k}{\gamma_0}. \tag{2.9}$$

2.1.3 Basic time series processes

This thesis will feature different types of time series processes. The most basic ones include *white noise* and *random walk*, which can also be considered to be the simplest case of the more general *moving average-* and *autoregressive* classes of processes, respectively.

White Noise

White noise is a zero-mean stochastic process $\{\epsilon_t\}$ with constant variance $\sigma_\epsilon^2 < \infty$ and a probability density function (pdf) that does not depend on t, implying that $\mathrm{Cov}(\epsilon_t, \epsilon_s) = 0, t \neq s$. The most common pdf used to model white noise is the normal distribution

$$\epsilon_t \sim \mathcal{N}\left(0, \sigma_\epsilon^2\right). \tag{2.10}$$

Random walk

A *random walk* process $\{x_t\}$ is a time series where each step x_t can be determined from its previous step,

$$x_t = \mu + x_{t-1} + \epsilon_t. \tag{2.11}$$

The stochastic variable x_t is found by adding the drift term μ to the previous step along with a white noise term $\epsilon_t \sim \mathcal{N}(0, \sigma_\epsilon^2)$. If there is no drift present in the model, i.e. $\mu = 0$, the random walk process has mean zero $\mathrm{E}[x_t] = 0$. An important random walk process is the *Brownian motion*, defined in Def: 2.1.4.

Definition 2.1.4 (Brownian motion). *The stochastic process $B(t)$ is a Brownian motion if it is almost surely continuous, $B(0) = 0$, its increments are independent and*

$$B(t) - B(s) \sim \mathcal{N}(0, \sigma^2|t-s|). \tag{2.12}$$

Autoregressive (AR) processes

An *autoregressive* process of order p, $\{x_t\}$ is a time series where the state at time t depends on the states at the p earlier steps in time,

$$x_t = \phi_1 x_{t-1} + \cdots + \phi_p x_{t-p} + \epsilon_t, \tag{2.13}$$

where $\{\epsilon_t\}$ denotes a white noise process with fixed variance σ_ϵ^2,

$$\epsilon_t \sim \mathcal{N}(0, \sigma_\epsilon^2) \tag{2.14}$$

and $\{\phi_i\}_i^p$ are non-zero coefficients of $\mathbb{R}$. By introducing the backshift operator B, defined such that $Bx_t = x_{t-1}$, Eq. 2.13 can be restated as

$$(1 - \phi_1 B - \cdots - \phi_p B^p)x_t = \epsilon_t. \tag{2.15}$$

The polynomial $(1 - \phi_1 B - \cdots - \phi_p B^p)$, denoted by the function $\phi_p(B)$, is called a *linear filter.* For the AR(p) process to be stationary, the roots of $\phi_p(B) = 0$ must lie outside of the unit circle. x_t is known to be *conditionally independent* of x_s, i.e.

$$\pi(x_t, x_s|\mathbf{x}_{-ts}) = \pi(x_t|\mathbf{x}_{-ts})\pi(x_s|\mathbf{x}_{-ts}), \tag{2.16}$$

for any pair of s and t such that $|t-s| > p$, where $\mathbf{x}_{-ts}$ denotes the vector of all elements in the series, except x_t and x_s. This is an important property for this thesis as it is shown in e.g. (27) that this property will result in a sparse precision matrix with bandwith p for the joint pdf which is well suited for efficient matrix operations. This thesis puts emphasis on AR processes of the first order only,

$$(1 - \phi_1 B)x_t = \epsilon_t, \tag{2.17}$$

which is stationary if $|\phi_1| < 1$ with autocorrelation function

$$\rho_k = \phi_1^k. \tag{2.18}$$

Moving average (MA) processes

A time series is a *moving average* process of order q if it can be represented by the difference equation

$$x_t = \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q} = \theta_q(B)\epsilon_t, \tag{2.19}$$

where $\{\epsilon_t\}$ is white noise with fixed variance and $\{\theta_i\}_{i=1}^{q}$ are non-zero coefficients of $\mathbb{R}$. Since $\mathrm{Var}(x_t) = 1 + \theta_1^2 + \cdots + \theta_q^2 < \infty$, a moving average process is always stationary, and it is also invertible if the roots of $\theta_q(B) = 0$ lie outside the unit circle.

Autoregressive moving average (ARMA) processes

A process represented by the linear difference equation

$$\phi_p(B)x_t = \theta_q(B)\epsilon_t, \tag{2.20}$$

with white noise $\{\epsilon_t\}$ and linear filters $\phi_p(B)$ and $\theta_q(B)$ is called an *autoregressive moving average* process of order p and q. The process has properties of both AR and MA processes and is stationary if the roots of $\phi_p(B) = 0$ lie outside the unit circle and also invertible if the roots of $\theta_q(B) = 0$ lie outside the unit circle. It is also assumed that $\phi_p(B) = 0$ and $\theta_q(B) = 0$ share no common roots.

2.1.5 Integrated series

Sometimes a non-stationary times series x_t can achieve stationarity by differencing,

$$z_t = (1 - B)x_t = x_t - x_{t-1}. \tag{2.21}$$

This is done to account for trends such as drift or seasonal variation in a model. A time series $\{x_t\}$ that has to be differenced d times to become stationary

$$z_t = (1 - B)^d x_t \tag{2.22}$$

is called an *integrated series* of order d, denoted $x_t \sim I(d)$. A popular model that incorporate integrated series is the autoregressive *integrated* moving average (ARIMA) model, which for parameters p, d and q is described by the difference equation

$$\phi_p(B)(1 - B)^d x_t = \theta_q(B). \tag{2.23}$$

Some simple, but useful properties of integrated series will be stated, see e.g. (8) for context. If y_t is defined such that if differenced D times it forms the process $x_t \sim I(d)$,

$$(1 - B)^D y_t = x_t, \tag{2.24}$$

then y_t is an integrated series of order $d + D$

$$y_t \sim I(d + D). \tag{2.25}$$

If $z_t \sim I(d')$ and is independent of x_t, then

$$x_t + z_t \sim I\left(\max(d, d')\right). \tag{2.26}$$

These properties apply for any $d \in \mathbb{R}$, and not just positive integers, which is the concept for what is called *fractional integrated series*, a class of models that can be shown to exhibit long-memory properties, and is revisited in section 2.6.

2.2 Spectral analysis

For now, every time series discussed have only been represented in the *time-domain*, i.e. x_t as a function of time. It is often beneficial to consider the alternative frequency- or *spectral domain* as well, as it is particularly suited for analysis of data with high frequencies.

2.2.1 Fourier analysis

From Fourier analysis it is known that a function can be written as a unique linear combination of trigonometric functions, meaning that a realization of a stochastic process consists of trigonometric functions with a particular frequency. These trigonometric functions are found using the *Fourier transform* (Def: 2.2.2) and transform the time series from the time domain to the spectral domain. Conversely, one can use the *inverse Fourier transform* (Def: 2.2.3) to transform the time serise from the spectral domain to the time domain.

Definition 2.2.2 (Fourier transform). *The Fourier transform of a time series $\{x_t\}$ is a function $\mathcal{F} : \mathcal{R} \to \mathcal{C}$ defined as*

$$\mathcal{F}(x)(\lambda) = \sum_{t=-\infty}^{\infty} e^{-i\lambda t} x_t. \tag{2.27}$$

This operation transforms the $\{x_t\}$ from a real-valued function of t to a complex function of λ.

Definition 2.2.3 (Inverse Fourier transform). *The inverse Fourier transform returns the Fourier transformation back to the time domain,*

$$x_t = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{i\lambda t} \mathcal{F}(x)(\lambda) d\lambda. \tag{2.28}$$

It can be shown (see for example (23) for proof) that there will be no loss of information when performing either the Fourier transform or the inverse Fourier transform.

2.2.4 Spectral densities

The *spectral density* $f(\lambda)$ of a time series x_t is defined as the Fourier transform of the autocovariance function,

$$f(\lambda) = \sum_{t=-\infty}^{\infty} \gamma(\lambda) e^{it\lambda} \tag{2.29}$$

and yields information about which frequencies of x_t that contribute the most to the variance. The spectral density exists if

$$\sum_{t=-\infty}^{\infty} |\gamma(t)| < \infty. \tag{2.30}$$

For an AR1 process,

$$x_t = \phi_1 x_{t-1} + \epsilon_t \tag{2.31}$$

where $\epsilon_t \sim \mathcal{N}(0, \sigma_\epsilon^2)$ the spectral density is found from 2.29 to be

$$f(\lambda) = \frac{1}{2\pi} \frac{\sigma_\epsilon^2}{1 - 2\phi_1 \cos(\lambda) + \phi_1^2} \tag{2.32}$$

and is displayed in Fig: 2.1. It is evident that for $\phi_1 > 0$ most of the mass of the spectral density is distributed along the lower frequencies and for $\phi_1 < 0$ the higher frequencies are more prominent.

It can be shown (see section 12.2.1 of (31)) that the spectrum of ARMA(p, q) models is

$$f(\lambda) = \frac{\sigma_\epsilon^2}{2\pi} \left| \frac{\theta_q\left(e^{-i\lambda}\right)}{\phi_p\left(e^{-i\lambda}\right)} \right|. \tag{2.33}$$

The frequencies λ for which the numerator of 2.33 is zero are called *zeros*, while the frequencies of which the denominator is zero are called *poles*.

2.2.5 Periodogram

Periodogram analysis is used to search for periodicities in a spectrum and can be used to estimate the spectral density. Given a time series of n observations, the periodogram $I(\lambda)$ is defined as the modulus-squared of the discrete Fourier transform,

$$I(\lambda_j) = \frac{1}{2\pi} \left| \sum_{t=1}^{n} (x_t - \bar{x}_n) e^{it\lambda_j} \right|^2, \tag{2.34}$$

where $\lambda_j = 2\pi j/n$ for $j = 1, 2, ..., (n-1)/2$ are the Fourier frequencies and $\bar{x}$ is the sample mean. It is shown in (25) that 2.34 is equivalent to

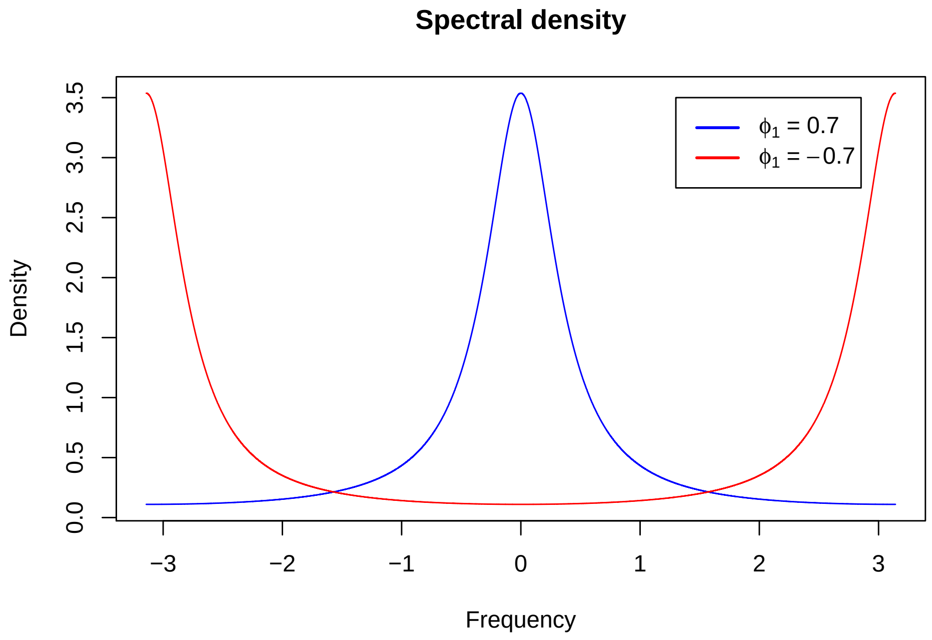

Figure 2.1: The spectral density of an AR1 series with positive and negative coefficients respectively.

$$I(\lambda) = \frac{1}{2\pi} \sum_{k=-(n-1)}^{n-1} \tilde{\gamma}(k) e^{ik\lambda_j} \tag{2.35}$$

with sample covariances

$$\hat{\gamma}(k) = \frac{1}{n} \sum_{t=1}^{n-|k|} (x_t - \bar{x})(x_{t+|k|} - \bar{x}). \tag{2.36}$$

(14) shows that the periodogram is an asymptotically unbiased estimator of the spectral density, i.e.

$$\lim_{n\to\infty} \mathrm{E}\big[I(\lambda)\big] = f(\lambda). \tag{2.37}$$

2.3 Long-memory processes

In 1951 the British hydrologist Harold Edwin Hurst published (13) a study of the behaviour of the *rescaled range* for the yearly variation of the water level of the Nile. The rescaled range, or R/S statistic, is a statistical measure of variability defined as

$$(R/S)_t = \frac{R_t}{S_t}, \tag{2.38}$$

where R_t is the range of the cumulative mean adjusted series $z_t = \sum_{i=1}^{t}(x_t - \bar{x})$,

$$R_t = \max(z_1, z_2, ..., z_t) - \min(z_1, z_2, ..., z_t) \tag{2.39}$$

and S_t is the standard deviation series of $\{x_t\}$

$$S_t = \sqrt{\frac{1}{t}\sum_{i=1}^{t}\left(x_k - \bar{x}_k\right)^2} \tag{2.40}$$

with

$$\bar{x}_k = \frac{1}{k}\sum_{i=1}^{k} x_i. \tag{2.41}$$

Assuming the yearly variations to be a Brownian process it was expected that the R/S statistic would show an asymptotic growth of order $n^{1/2}$, but the study revealed instead an asymptotic growth of $\approx n^{0.82}$. This was not the first time observations of this nature occurred (Kolmogorov observed this when studying turbulence in 1941 (18)), but the phenomenon remained unexplained until a series of papers made by Benoit Mandelbrot and his colleagues in the 1960s.

The Hurst exponent

Sparked by the findings of Hurst (13), Mandelbrot and van Ness introduced in 1968 (21) what is known as *long-memory* processes (or *long range dependence*). These processes are described only by a single parameter, known as both the *Hurst exponent* or the *Hurst coefficient*, named after Hurst for his observations of this phenomenon. This parameter is defined from the R/S statistic as shown in Eq: 2.42 and explains the behaviour of the ACF, or *memory* of the process. If $0.5 < H < 1$, the process has a persistent ACF and long-memory properties, and if $0 < H < 0.5$ the process has anti-persistent behaviour. A Hurst exponent of $H = 0.5$ has no memory and show no correlation between its points. Some of the definitions used for long-memory processes will be stated and discussed in section 2.3.3.

$$\mathrm{E}\left[\frac{R(n)}{S(n)}\right] \propto n^H \quad \text{as } n \to \infty, \tag{2.42}$$

Self similarity

Self-similarity, or *self-affinity* is a property associated with long-memory processes and was introduced by Kolmogorov in 1941 (18). The theory was applied to long-memory processes in 1968 by Mandelbrot and van Ness (21). Let $\{x_t\}_{t=0}^{n}$ be a stationary stochastic process in discrete time, divided in groups of $m > 1$. Let $x_t^{(m)}$ denote the average of the t^{th} group,

$$x_t^{(m)} = \frac{1}{m}\sum_{k=(t-1)m+1}^{tm} x_k, \qquad t = 1, 2, ..., n/m. \tag{2.43}$$

Self-similarity can then be defined if each variable x_t of the time series is connected to the corresponding group average $x_t^{(m)}$, see Def: 2.3.1.

Definition 2.3.1 (Self-similarity). *A stationary process $\{x_t\}_{t=0}^{n}$ is self-similar with Hurst exponent H if x_t and $m^{1-H}x_t^{(m)}$ are equal in distribution for all $t = 1, ..., n/m$ and $m > 1$.*

This can also be extended to apply for continuous stochastic processes, see Def: 2.3.2.

Definition 2.3.2 (Continuous self-similarity). *A stationary continuous process $Y = \{Y(t) : 0 \leq t < \infty\}$ is self-similar with Hurst exponent H if $Y(at)$ and $a^{1-H}Y(t)$ have the same identical finite-dimensional distributions for all $a > 0$.*

Similarities can be drawn between self-similar long-memory processes and fractals (6). The Hurst exponent H, is a global characteristic of the long-memory process, while the fractal dimension D is a local one. Due to the self-similarity, the local properties are reflected onto the global properties. For a self-similar long-memory process of which the time t is the only variable, thus the space only has $k = 1$ dimension and the relationship

$$D + H = n + 1 \tag{2.44}$$

simplifies to $D = 2 - H$.

2.3.3 Definitions of long-memory

Although the concept of long-memory has always been concerned about the slow decay in the dependence between points (or "memory") of the series, many formal definitions have been used throughout the years. This thesis will present a number of different definitions, all of which are presented and given context in (12), and thoroughly reviewed in (11).

Definition 2.3.4 (Covariance). *A stationary time series, x_t with autocovariance function $\gamma_x(k)$ and Hurst exponent $H \in (0.5, 1)$ has long-memory in the* ***covariance sense*** *if*

$$\gamma_x(k) \xrightarrow{k\to\infty} C_x k^{2H-2} \tag{2.45}$$

for some constant C_x.

Definition 2.3.4 defines the long-memory property as a hyperbolic decay of the ACF, consistent with the general consensus of long-memory. For $H \in (0.5, 1)$ the ACF described in Eq: 2.45 decays so slowly that the sum diverges to infinity,

$$\sum_{k=0}^{\infty} \gamma_H(k) = \infty. \tag{2.46}$$

The covariance definition can also be expressed in the spectral domain, leading to Definition 2.3.5.

Definition 2.3.5 (Spectral)**.** *A stationary time series, x_t with spectral density function $f_x(\lambda)$ and Hurst exponent $H \in (0.5, 1)$ has long-memory in the* ***spectral sense*** *if*

$$f_x(\lambda) \xrightarrow{\lambda \to 0} C_f \lambda^{1-2H} \tag{2.47}$$

for some constant C_f.

The one-to-one correspondence between the time and spectral domain ensures that the spectral and covariance sense of the long-memory definition are equivalent.

Definition 2.3.6 (self-similar)**.** *A stationary time series, x_t with Hurst exponent $H \in (0.5, 1)$ has long-memory in the* ***self-similar sense*** *if*

$$m^{2-2H}\mathrm{Cov}\left(x_t^{(m)}, x_{t+k}^{(m)}\right) \approx C_m k^{2H-2} \tag{2.48}$$

as $k, m \longrightarrow \infty$ where $x_t^{(m)} = \frac{1}{m}(x_{tm-m+1} + \cdots + x_{tm})$ with $m \in \mathbb{N}$ and constant C_m.

The self-similar definition of long-memory originated in (21) where the self-similarity condition was first introduced to long-memory series. It was shown that the fractional Brownian motion, a common long-memory process, possess this property.

2.3.7 Stochastic integration

Stochastic integrals are characterized by a random integrator. For the purposes of this thesis, the integrator denoted B will be the Brownian motion stochastic process. This causes problems as the paths of a Brownian motion are highly irregular, and one cannot therefore assume the trajectories to be differentiable. The paths also do not have bounded variation with probability 1 preventing us from calculating the integral as a pathwise Lebesgue-Stieltjes integral (15).

Consider a continuous function of time $\phi(t)$ assumed to be *simple* on the finite interval $[a, b]$. This means that we can create a strictly increasing sequence of possible input variables $\{t_j\}_{j=a}^{b}$ where we for $t \in (t_j, t_{j+1}]$ denote $\phi(s) = \phi_j$. We then get a corresponding sequence of function evaluation of the sequence of t_j, namely $\{\phi(t_j)\}_{t_j=t_a}^{t_b-1}$. The *stochastic integral* of ϕ on the interval $t \in [a, b]$ has the natural definition:

$$\int_a^b \phi(s)\mathrm{d}B(s) = \sum_{j=0}^{l-1} \phi_j\big(B(t_{j+1}) - B(t_j)\big). \tag{2.49}$$

A sequence of square integrable functions $\{\psi_n\}$ is said to converge in L^2-norm to a square integrable function ψ if

$$\lim_{n\to\infty} \int \big(\psi(s) - \psi_n(s)\big)^2 \mathrm{d}s = 0. \tag{2.50}$$

It can be shown that every square integrable function can be written as a limit in L^2-norm of a sequence of simple functions. If ψ_n are simple for every n we could define the stochastic integral as

$$\int \psi(s)\mathrm{d}B(s) := \lim_{n\to\infty} \int \psi_n(s)\mathrm{d}B(s). \tag{2.51}$$

This turns out to not be generally viable, as we are required to pose very restrictive constraints. A more detailed explanation of stochastic integration can be found in (15) and (4).

2.4 Fractional Brownian motion

Fractional Brownian motion (FBM) is a long-memory generalization of the widely used Brownian motion defined in Def: 2.1.4 and was first introduced in (17). Whereas the Brownian motion process, $B(t)$ at time t assumes independent Gaussian increments,

$$B(t_2) - B(t_1) \sim \mathcal{N}(0, |t_2 - t_1|), \tag{2.52}$$

the FBM, denoted $B_H(t)$ allows for dependence between them. This dependence is explained solely by the Hurst exponent, H. The formal definition of FBMs was originally presented in (21) and is stated in Def: 2.4.1.

Definition 2.4.1 (Fractional Brownian motion)**.** *Let $0 < H < 1$, b_0 be an arbitrary real number and $B(t)$ be a Brownian motion as defined in Def: 2.1.4. We call the following random function $B_H(t)$ the* fractional Brownian motion *with parameter H and starting value b_0 at time $t = 0$. For $t > 0$, $B_H(t)$ is defined by*

$$\begin{aligned} B_H(0) &= b_0 \\ B_H(t) - B_H(0) &= \frac{1}{\Gamma(H+1/2)}\left(\int_{-\infty}^{0} \left[(t-s)^{H-1/2} - (-s)^{H-1/2}\right]\mathrm{d}B(s) \right. \\ &\qquad \left. - \int_{-\infty}^{t} (t-s)^{H-1/2}\mathrm{d}B(s)\right). \end{aligned} \tag{2.53}$$

It can be shown (see (24) for proof) that a process is an FBM if it satisfies the following conditions:

- $B_H(0) = 0$ and $\mathrm{E}[B_H(0)] = 0$ for $t \geq 0$
- $B_H(t)$ has stationary increments, i.e. $B_H(t) - B_H(s) \sim B_H(t-s)$
- $\mathrm{Var}\big(B_H(t)\big) = \sigma^2 |t|^{2H}$
- $\mathrm{Var}\big(B_H(t) - B_H(s)\big) = \sigma^2 |t-s|^{2H}$
- $\mathrm{Cov}\big(B_H(t), B_H(s)\big) = \frac{\sigma^2}{2}\left(|t|^{2H} + |s|^{2H} - |t-s|^{2H}\right)$

H denotes the Hurst exponent and σ^2 is called the *innovation variance.* If $\sigma^2 = 1$ the series is considered a *standard* fractional Brownian motion. From the covariance function it is evident that the FBM increments are positively and negatively correlated for $H > 1/2$ and $H < 1/2$, respectively. For $H = 1/2$, there is no correlation and the FBM reduces to an ordinary Brownian motion process. The distribution of an FBM at time t is

$$B_H(t) \sim \mathcal{N}\left(0, \sigma^2|t|^{2H}\right). \tag{2.54}$$

From 2.54 one finds the FBM to be self-similar,

$$B_H(ct) \sim c^{2H} B_H(t) \tag{2.55}$$

and having long-memory in the self-similar sense with Hurst exponent H.

2.5 Fractional Gaussian noise

Fractional Gaussian noise (FGN) is a stationary long-memory process defined as the increment process of a fractional Brownian motion,

$$X(t) = B_H(t+1) - B_H(t). \tag{2.56}$$

Its conception was motivated by the attempts of (21) to define the concept of the derivative of a Brownian motion process. For ordinary Brownian motion increments it is evident from 2.1.4 that $X(t) \sim \mathcal{N}(0, \sigma^2)$. The autocovariance function of an FGN process with Hurst exponent H (see (14)) is of the form

$$\gamma_H(t) = \frac{\sigma^2}{2}\left[|t-1|^{2H} - 2|t|^{2H} + |t+1|^{2H}\right] \tag{2.57}$$

for $t \in \mathbb{T}$. Equation 2.57 claims independence for the FGN process if $H = 1/2$ and $t \neq 0$. By performing Taylor expansion of 2.57 around the origin we find the Taylor approximation of the autocovariance as $t \to \infty$ to be

$$\gamma_H(t) \sim H(2H-1)t^{2H-2}. \tag{2.58}$$

From 2.58 we can deduct that FGN processes have long-memory with Hurst exponent H, with a persistent ACF for $1/2 < H < 1$, and an anti-persistent ACF for $1/2 < H < 1$. This is displayed by simulations in Fig: 2.3 and Fig: 2.5. In (14) it is established that the spectral density of fractional Gaussian noise is given by

$$f(\lambda) = 2\sin(\pi H)\Gamma(2H+1)(1-\cos\lambda)\left[|\lambda|^{-2H-1} + B(\lambda, H)\right], \tag{2.59}$$

where for $-\pi \leq \lambda \leq \pi$,

$$B(\lambda, H) = \sum_{j=1}^{\infty}\left[(2\pi j + \lambda)^{-2H-1} + (2\pi j - \lambda)^{-2H-1}\right]. \tag{2.60}$$

The spectral density function is represented in Fig: 2.2 where a pole can be located at frequency $\lambda = 0$. The infinite sum of 2.60 is computed for a suitable number of terms as to preserve both accuracy and computational speed.

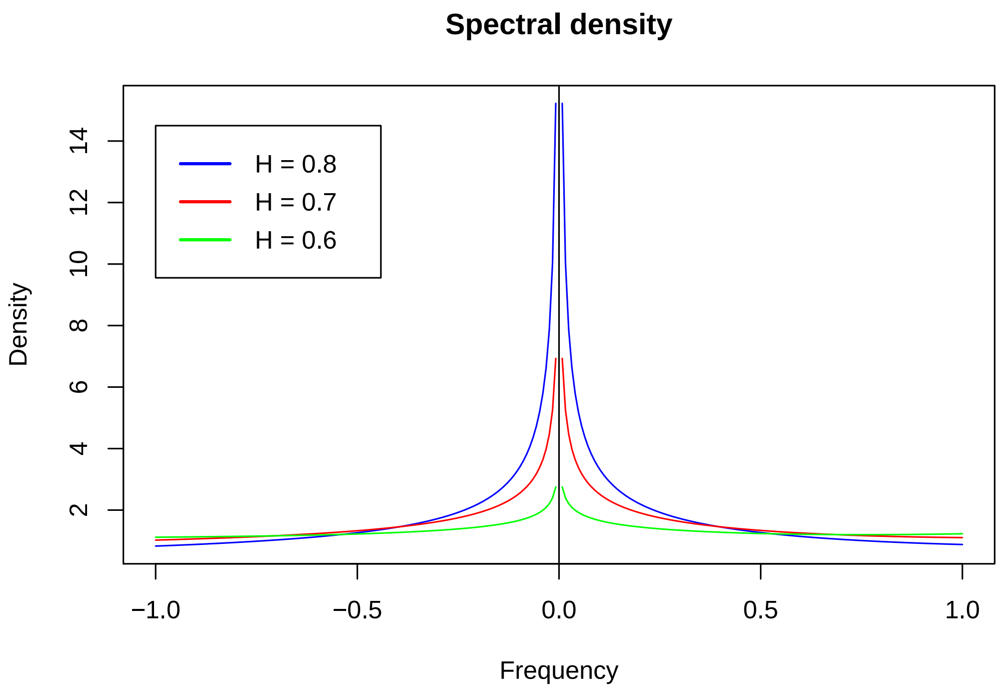

Figure 2.2: The spectral density function of FGN processes with different Hurst exponents. A pole is located at frequency $\lambda = 0$.

2.6 Fractional differencing and ARFIMA models

Let $\{x_t\}$ be an integrated series of order d, denoted as before by

$$x_t \sim I(d). \tag{2.61}$$

This section explores what happens when d is no longer an integer, introducing *fractional integrated series*. To understand the concept of fractional integrated series, consider the example constructed by Granger in 1980 (8), where $\{x_t\}$ is explained by the difference equation

$$(1 - B)^d x_t = \epsilon_t. \tag{2.62}$$

There are no autoregressive or moving average parts in the model, and the only filter acting on the model is the *integrating filter* of order d, $\alpha(B) = (1 - B)^d$. Let $d = 1/2$ and suppose there is a filter $\alpha(B)$ such that when it is applied twice one gets the order one differencing,

$$\alpha(B)^2 x_t = (1 - B)x_t. \tag{2.63}$$

Such a filter does exist, and when it is applied only once it results in 'half-differencing', or *fractional differencing* of order 1/2. This logic can be extended to any $d \in \mathbb{R}$. An integrated series that requires fractional differencing to become a stationary process is called a *fractional integrated series.*

If an ARMA(p, q) process requires fractional differencing of order d to achieve stationarity, then the process is called an autoregressive *fractionally integrated* moving average (ARFIMA) series with parameters p, d and q, defined by the difference equation

$$\phi_p(B)(1-B)^d x_t = \theta_q(B)\epsilon_t, \quad d \in \mathbb{R}\backslash\mathbb{N}. \tag{2.64}$$

Granger (8) shows that the differencing operator $(1-B)$ of an integrated series can be expanded with the generalized binomial coefficient,

$$(1-B)^d = \sum_{k=0}^{\infty} \binom{d}{k}(-B)^k = \sum_{k=0}^{\infty} \frac{\prod_{a=0}^{k-1}(d-a)(-B)^k}{k!}. \tag{2.65}$$

As an example, consider the simple ARFIMA$(0, d, 0)$ process, defined in terms of the differencing operator as

$$(1-B)^d x_t = \epsilon_t, \tag{2.66}$$

where $\epsilon_t \sim \mathcal{N}(0, \sigma_\epsilon^2)$. Using the generalized binomial coefficient one may rewrite equation 2.66 as

$$x_t - dx_{t-1} + \frac{d(d-1)}{2!}x_{t-2} - \ldots = \epsilon_t. \tag{2.67}$$

Granger and Joyeux (9) showed that the autocovariance between x_t and x_{t-k} when $d < 1/2$ is

$$\text{cov}(x_t, x_{t-k}) = \frac{\sigma_\epsilon^2}{2\pi}\sin(\pi d)\frac{\Gamma(k+d)}{\Gamma(k+1-d)}\Gamma(1-2d). \tag{2.68}$$

Thus, the variance increases as d increases and becomes infinite for $d \geq 1/2$, giving rise to long-memory series. The MA(∞) and AR(∞) representation of the model,

$$x_t = \sum_{j=0}^{\infty} b_j \epsilon_{t-j} \quad \text{and} \quad \sum_{j=0}^{\infty} a_j x_{t-j} = \epsilon_t \tag{2.69}$$

are found to have coefficients

$$b_j = \frac{\Gamma(j+d)}{\Gamma(d)\Gamma(j+1)} \quad \text{and} \quad a_j = \frac{\Gamma(j-d)}{\Gamma(d)\Gamma(j+1)} \tag{2.70}$$

respectively. Granger (8) states that by using Sterling's theorem, we can approximate $\Gamma(j+a)/\Gamma(j+b)$ by j^{a-b} for large j. One can then approximate the MA(∞), AR(∞) and the auto-correlation

$$\rho_k = \frac{\Gamma(1-d)}{\Gamma(d)}\frac{\Gamma(k+d)}{\Gamma(k+1-d)}, \tag{2.71}$$

by

$$\rho_j \approx A_1 j^{2d-1} \tag{2.72}$$

$$b_j \approx A_2 j^{d-1} \tag{2.73}$$

$$|a_j| \approx A_3 j^{-(1+d)} \tag{2.74}$$

with some constants A_1, A_2 and A_3. The ARFIMA$(0, d, 0)$ model shows the same hyperbolic decay in its ACF as an FGN process with Hurst exponent $H = d + 1/2$. Definition 2.3.4 then implies that the ARFIMA$(0, d, 0)$ model exhibits long-memory in the covariance sense with Hurst exponent $H = d + 1/2$. Granger also discusses that the same conditions 2.72, 2.73 and 2.74 apply for the more general ARFIMA(p, d, q) model, but with different constants.

2.7 Simulations of long-memory processes

This thesis will mainly focus on estimation techniques and how they compare. In order to measure the accuracy of such techniques it is important to know how to simulate FGN processes. The simulations assume discrete time as we are unable to maintain the assumption of continuous time for a simulation process. Recall that an FGN process $\tilde{X}$ can be obtained from an FBM series $\tilde{B}$ by differencing,

$$\tilde{X}(t) = \tilde{B}(t+1) - \tilde{B}(t) \tag{2.75}$$

and conversely, an FBM can be obtained from an FGN from the cumulative sums

$$\tilde{B}_H(i/n) = \sum_{k=0}^{i} \tilde{X}(i/n), \tag{2.76}$$

where n is the length of a series. One can therefore obtain a simulation of an FGN either by first simulating an FBM and retrieve the FGN process by differencing, or by simply simulating the FGN directly. Due to its stationarity it is preferred to simulate from FGN series rather than FBM series. Also, the FGN has a *Toeplitz* covariance matrix, i.e.

$$\Sigma = \begin{bmatrix} a & b & c & d & e \\ f & a & b & c & d \\ g & f & a & b & c \\ h & g & f & a & b \\ i & h & g & f & a \end{bmatrix} \tag{2.77}$$

which allows for more efficient sampling. To demonstrate the difference, one method of each approach will be conducted and compared.

2.7.1 Stochastic representation

The first method simulates an FGN by obtaining it from a simulated FBM, and the most apparent way to simulate an FBM would be to discretize the stochastic integral from its definition

2.53. We apply the approximation technique illustrated in equation 2.49 which leads to the approximation

$$\tilde{B}_H(n) = C_H \left(\sum_{k=-b}^{0} \left[(n-k)^{H-\frac{1}{2}} - (-k)^{H-\frac{1}{2}} \right] B_1(k) + \sum_{k=0}^{n} (n-k)^{H-\frac{1}{2}} B_2(k) \right) \tag{2.78}$$

where B_1 and B_2 are mutually independent vectors of $b+1$ and $n+1$ i.i.d. standard normal variables respectively. Increasing the number of temporal points will increase the precision at the cost of a more computationally intensive algorithm. This method is generally not an efficient way to generate an FBM due to the required approximations and is mainly included for its simplicity and historical value.

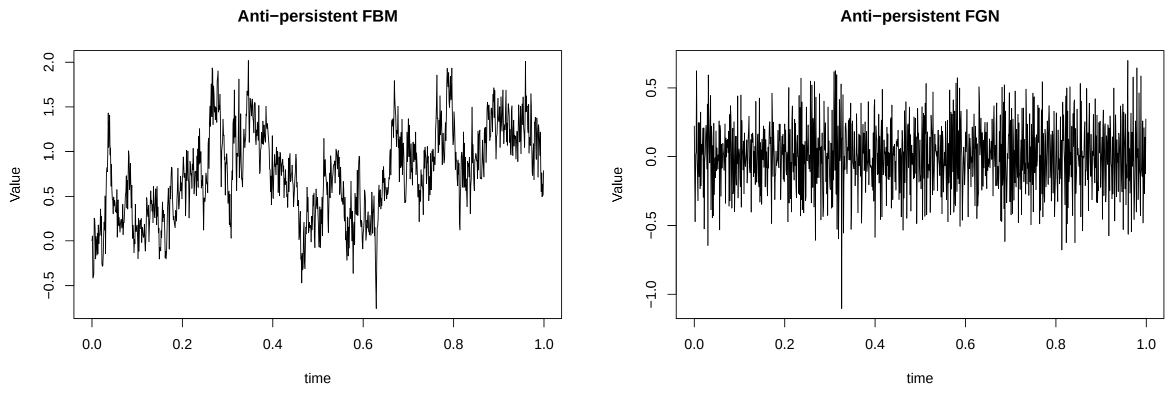

Figure 2.3: $n = 1000$ length simulation of an FBM by the method of stochastic representation for $H = 0.20$ with corresponding FGN.

Illustrations of simulated FBM processes with Hurst exponents $H = 0.20$, $H = 0.50$ and $H = 0.90$ are given in Figures: 2.3, 2.4 and 2.5, along with their corresponding FGN series.

2.7.2 Wood-Chan

Another approach to simulate FGNs was originally proposed by Davis and Harte (3) and later improved by Wood and Chan (33), and is applicable to any stationary Gaussian process. The algorithm is explained thoroughly in (2). The idea is to embed the covariance matrix Σ, in a circulant matrix C of size $m = 2^g, g \in \mathbb{N}^*$ before generating a vector $Y = (Y_0, ..., Y_{m-1})^\top \sim \mathcal{N}(0, C)$. An appropriate construction of C ensures that we are able to generate $(Y_0, ..., Y_{N-1})^\top \sim$

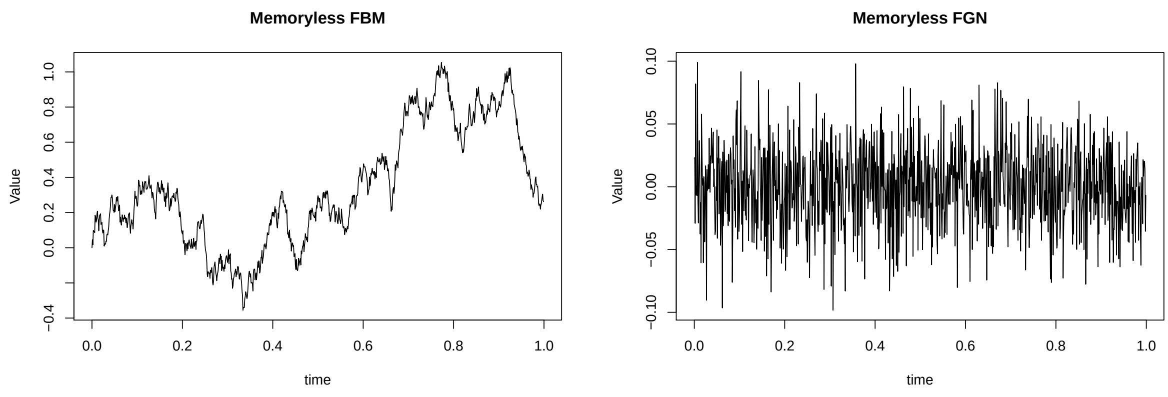

Figure 2.4: $n = 1000$ length simulation of an FBM by the method of stochastic representation for $H = 0.50$ with corresponding FGN.

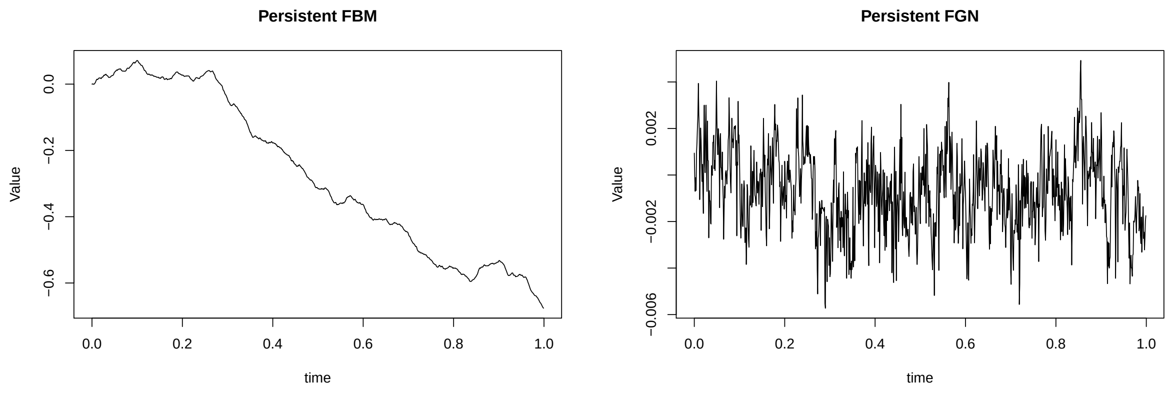

Figure 2.5: $n = 1000$ length simulation of an FBM by the method of stochastic representation for $H = 0.90$ with corresponding FGN.

$\mathcal{N}(0, \Sigma)$. We define the circulant matrix C:

$$C = \begin{bmatrix} c_0 & c_1 & \dots & c_{m-1} \\ c_{m-1} & c_0 & \dots & c_{m-2} \\ \vdots & \vdots & \ddots & \vdots \\ c_1 & c_2 & \dots & c_0 \end{bmatrix} \qquad \text{where} \qquad c_j = \begin{cases} \gamma(\frac{j}{n}), & 0 \le j < m/2 \\ \gamma(\frac{m-j}{n}), & m/2 < j < m-1 \le x \end{cases}.$$

The size of C is explained from the variable $m = 2^g$. This is chosen to be the first power of two, such that C is positive definite,

$$m \geq 2(n-1). \tag{2.79}$$

For an FGN, this condition is satisfied for the value $m = 2 \cdot 2^\theta$, where 2^θ is the first power of two superior to N. A result from (1) is used to decompose C as $C = Q\Lambda Q^*$ with Λ as the diagonal matrix with the eigenvalues of C along the diagonal, and Q is the unitary matrix defined by

$$(Q)_{j,k})m^{-\frac{1}{2}} \exp\left(-2i\pi\frac{jk}{m}\right), \qquad \text{for} \quad j,k = 0,...,m-1. \tag{2.80}$$

Draw $Z \sim \mathcal{N}(0, I_m)$. If $Y = Q\Lambda^{1/2}Q^*Z$ we have that $Y \sim \mathcal{N}(0, C)$ due to Q being unitary. Simulating an FGN can be done by following these summarizing steps:

- Compute estimates of the eigenvalues of C, for example by

$$\lambda_k = \sum_{j=0}^{m-1} c_j \exp\left(-2i\pi\frac{jk}{m}\right), \qquad \text{for } k = 0,...,m-1, \tag{2.81}$$

  which may be calculated using Fast Fourier Transform.

- Fast simulation of $W = Q^*Z$. This can be done by decomposing Q^*Z into both real and imaginary parts. Let the elements of the first half of vector W, denoted W_i for $i \in 1, ...\frac{m}{2}-1$ be the complex conjugate of the elements in the second half $\{W_i : i = \frac{m}{2}, ..., m-1\}$. The first and last element denotes the real value and the purely imaginary value respectively. This is simulated by first generating two independent variables $U, V \sim \mathcal{N}(0,1)$ and letting $W_0 = U$ and $W_{\frac{m}{2}} = V$. The remaining values are sampled similarly by drawing $U_j, V_j \sim \mathcal{N}(0,1)$ for every $1 \leq j < \frac{m}{2}$, then the remaining W_j is computed as follows:

$$\begin{aligned} W_j &= \frac{1}{\sqrt{2}}(U_j + iV_j) \\ W_{m-j} &= \frac{1}{\sqrt{2}}(U_j - iV_j). \end{aligned}$$

- Reconstruct X by calculating

$$X\left(\frac{k}{n}\right) = \frac{1}{\sqrt{m}} \sum_{j=0}^{m-1} \sqrt{\lambda_j} W_j \exp\left(-2i\pi\frac{jk}{m}\right), \qquad \text{for } k = 0,...,m-1. \tag{2.82}$$

  using the Fast Fourier Transform.

With a complexity of $n\log(n)$ this algorithm provides an efficient way of simulating exact FGNs, even for larger values of n. The speed of this algorithm is illustrated by the time plots included in Fig: 2.6 and Fig: 2.7 which compare the speed of Wood-Chan's method to that of the stochastic representation. As can be seen, the Wood-Chan method is far superior in regards to speed compared to the stochastic representation method, especially for larger values of n.

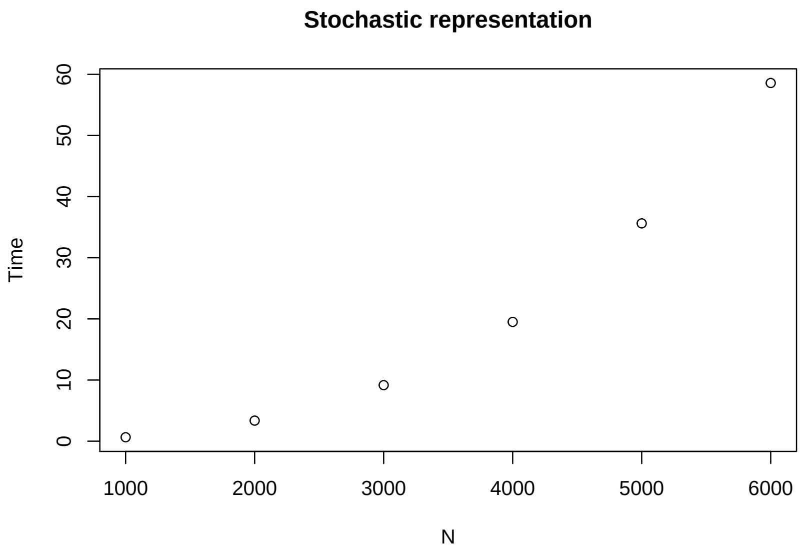

Figure 2.6: Stochastic representation: Number of seconds to complete a simulation versus the length N of the simulated series.

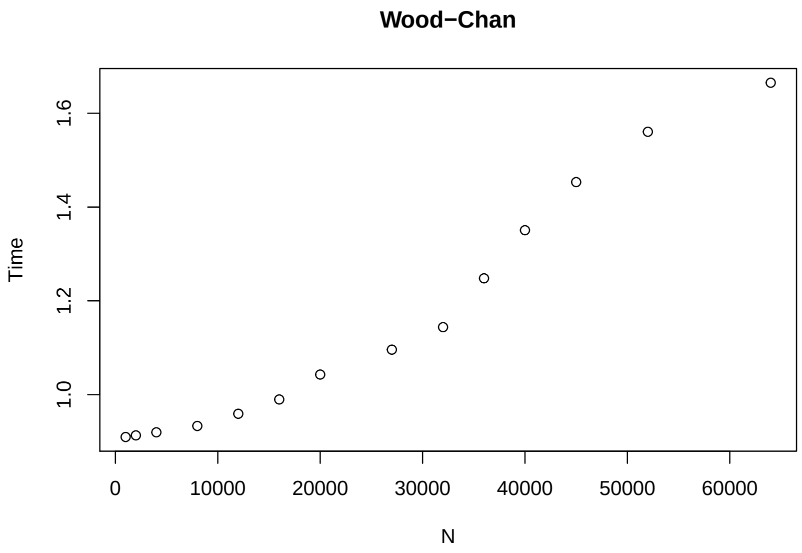

Figure 2.7: Wood-Chan: Number of seconds to complete a simulation versus the length N of the simulated series.

The simulations were generated using the code included in (2) and the resulting FGN simulations and corresponding FBMs for $H = 0.20$, $H = 0.50$ and $H = 0.90$ are included in Figures: 2.8, 2.9and 2.10, respectively.

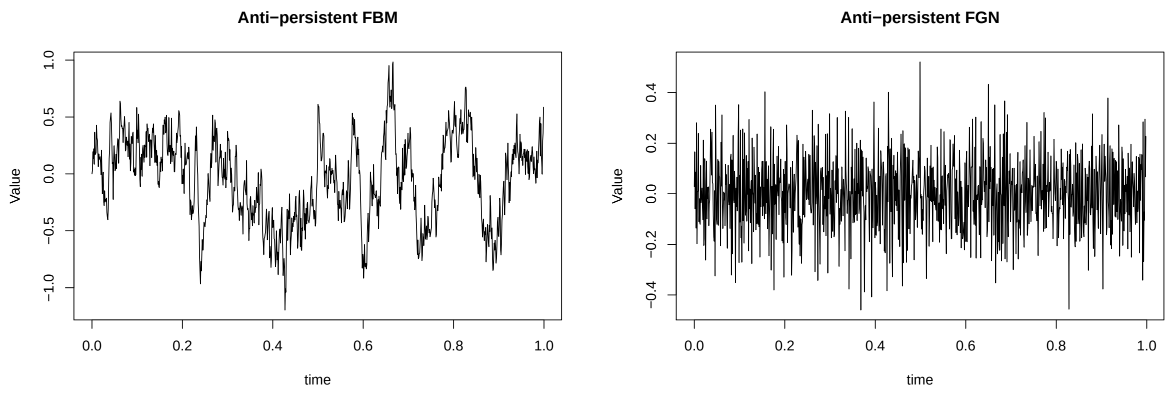

Figure 2.8: $n = 1000$ length simulation of an FGN by the Wood-Chan method for $H = 0.20$ with corresponding FBM.

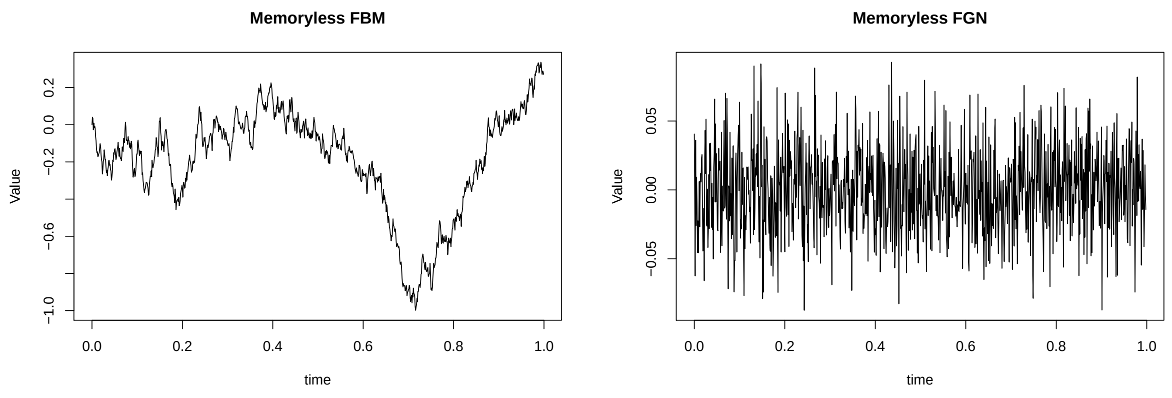

Figure 2.9: $n = 1000$ length simulation of an FGN by the Wood-Chan method for $H = 0.50$ with corresponding FBM.

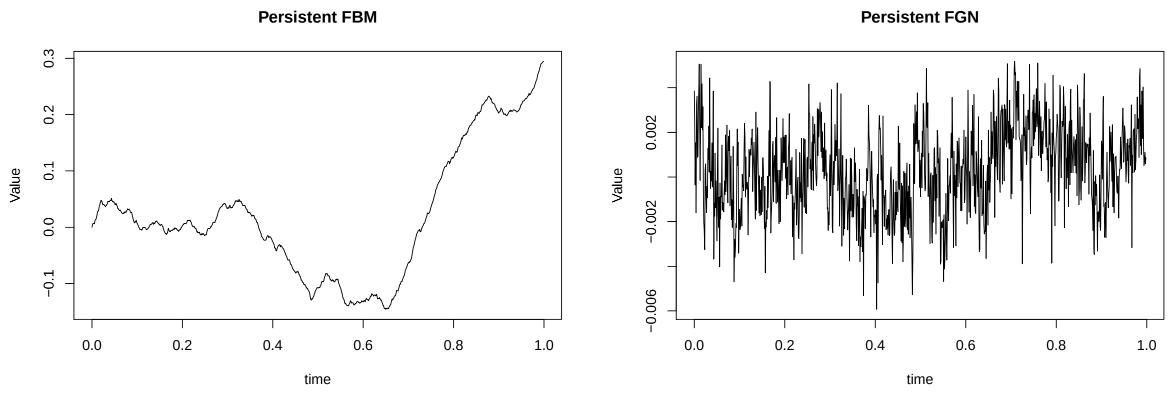

Figure 2.10: $n = 1000$ length simulation of an FGN by the Wood-Chan method for $H = 0.90$ with corresponding FBM.

Chapter 3

Estimation methods for fractional Gaussian noise

3.1 Introduction

This chapter introduces methods to identify fractional Gaussian noise processes by estimating the Hurst exponent and the innovation variance. The simplest method of estimating the Hurst exponent of an FGN is to obtain it directly from its origin, the R/S statistic defined in section 2.3. This method is included in the R package `FGN` as the function `HurstK` and provides a very fast, but sometimes inaccurate estimate of H due to sensitivity to outliers.

This thesis will introduce likelihood based estimators which will be applied to examples and compared. Of particular interest is the use of *integrated nested Laplace approximations* (INLA) (28), available for R at `http://www.r-inla.org` to compute the maximum likelihood estimators.

3.2 Maximum likelihood estimation

The Hurst exponent H and the innovation variance σ^2 can be obtained by maximizing the likelihood function of the FGN process. As FGNs are stationary Gaussian processes the likelihood function for the observations $\mathbf{y} = (y_1, ..., y_n)$ is found to be

$$L(H, \sigma^2; \mathbf{y}) = (2\pi)^{-n/2} \left|\Sigma\left(H, \sigma^2\right)\right|^{-1/2} \exp\left(-\frac{1}{2}\mathbf{y}^\top \Sigma\left(H, \sigma^2\right)^{-1} \mathbf{y}\right), \tag{3.1}$$

where $\Sigma(H, \sigma^2)$ is the autocovariance matrix,

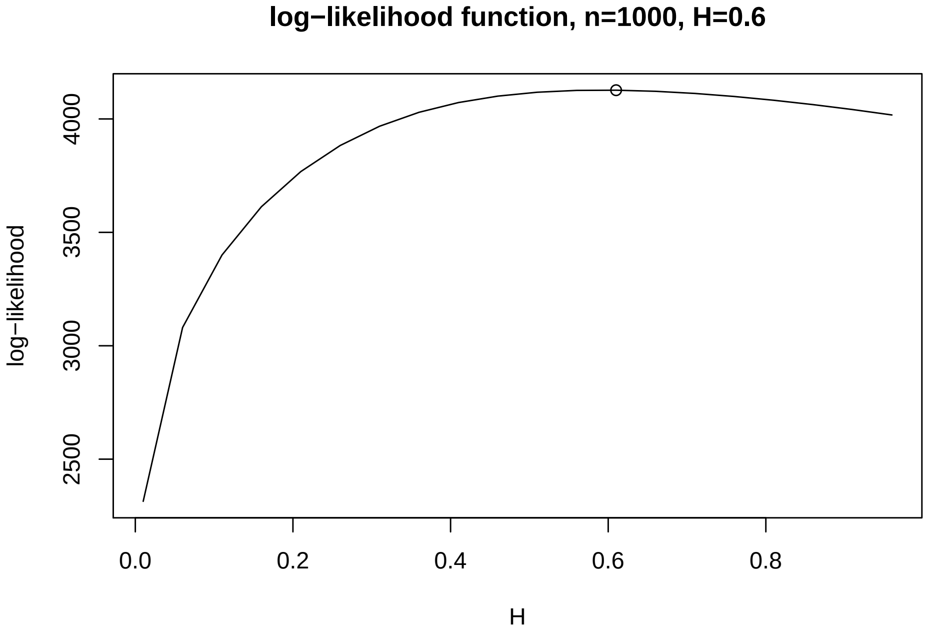

Figure 3.1: The log-likelihood function for a standard fractional Gaussian noise of length $n = 1000$ with true Hurst exponent $H = 0.6$ evaluated for $H \in (0, 1)$.

$$\Sigma(H, \sigma^2) = \begin{bmatrix} \gamma_H(0) & \gamma_H(1) & \dots & \gamma_H(n-1) \\ \gamma_H(1) & \gamma_H(0) & & \gamma_H(n) \\ \vdots & & \ddots & \vdots \\ \gamma_H(n-1) & \gamma_H(n-2) & \dots & \gamma_H(0) \end{bmatrix} \tag{3.2}$$

obtained from the ACF given in Eq: 2.57. The log-likelihood function for an FGN with true Hurst exponent $H = 0.6$ is illustrated in Fig: 3.1. Finding both the innovation parameter and the Hurst exponent is done by using numerical algorithms and can be tedious as one is required to invert the autocovariance matrix and obtain its determinant. This can be avoided by using an approximation rather than evaluating the exact MLE. A common approximation technique for evaluating the MLE of an FGN processes is the *Whittle's approximate likelihood* first introduced by Peter Whittle in 1951 (32) and is explained in greater detail in e.g. (14). The likelihood approximation converges to the exact likelihood as $N \to \infty$. We obtain the estimate for the Hurst exponent by minimizing

$$Q(H) = \int_{-\pi}^{\pi} \frac{I(\lambda)}{f_H(\lambda)} \mathrm{d}\lambda, \tag{3.3}$$

where I is the periodogram described in Eq: 2.34 and f_H is the spectral density stated in Eq: 2.59. This is done by approximating Eq: 3.3 with a Riemann sum where the periodogram and spectral density are evaluated at the Fourier frequencies $\lambda_k = 2\pi k/m$ with, for example $m = N$. Using symmetry in I and f, we obtain $\tilde{H}$ by minimizing

$$\tilde{Q}(H) = \sum_{k=1}^{N/2} \frac{I(\lambda_k)}{f_H(\lambda_k)}. \tag{3.4}$$

The `warfima` function from the `FGN` package in R is used to evaluate Whittle's approximate likelihood for the FGN simulations generated by Wood-Chan's method in section 2.7.2 with length $n = 1000$ and true Hurst exponents $H_1 = 0.20$, $H_2 = 0.50$ and $H_3 = 0.90$. The function yields estimations $\tilde{H}_1 = 0.2015$, $\tilde{H}_2 = 0.4907$ and $\tilde{H}_3 = 0.9316$ for H respectively.

3.3 Bayesian inference

Contrary to classical statistical methods where unknown model parameters are treated as fixed constants, Bayesian models treat the parameters as random variables. Bayesian statisticians also interpret probability as a subjective state of belief rather than an objective truth. This allows for prior knowledge and expertise to be incorporated into the model which will be updated in the light of new data. The prior knowledge about a parameter θ is explained by the *prior distribution*

$$\theta \sim \pi(\theta). \tag{3.5}$$

Since the observations $\mathbf{y} = (y_1, y_2, ..., y_n)$ are associated with a likelihood function with parameter θ

$$\mathbf{y} \sim \pi(\mathbf{y} \mid \theta) = L(\theta \mid \mathbf{y}), \tag{3.6}$$

the beliefs about θ is updated by combining information about the data $\mathbf{y}$ and the prior distribution according to *Bayes' law*,

$$\pi(\theta \mid \mathbf{y}) = \frac{f(\theta, \mathbf{y})}{f(\mathbf{y})} = \frac{\pi(\theta)\pi(\mathbf{y} \mid \theta)}{\int \pi(\mathbf{y} \mid \theta)\pi(\theta)\mathrm{d}\theta}. \tag{3.7}$$

From Eq: 3.7 the *posterior distribution* of θ is retrieved,

$$\pi(\theta \mid \mathbf{y}) \propto \pi(\theta)\pi(\mathbf{y} \mid \theta) \tag{3.8}$$

from which inference about θ is primarily based on. If the prior distribution is uniform, i.e. no prior knowledge is known about θ, then finding the mode of the posterior distribution is equivalent of finding the MLE of θ. The maximum likelihood problem of section 3.2 can be stated as a Bayesian model, and this thesis will present a class of Bayesian models that are particularly suited for such problems, namely the *latent Gaussian models.*

3.3.1 Latent Gaussian modeling

Latent Gaussian models (LGMs) is a class of hierarchical Bayesian models that encompass many widely used statistical models. They are discussed in e.g. (28), but will be stated here for clarity. LGMs are expressed as a three-stage hierarchical model, in which the first stage explains the observations $\mathbf{y} = y_1, .., y_n$, which are assumed to be conditionally independent given an underlying Gaussian random field of unobserved variables $\mathbf{x} = x_1, ..., x_n$ and a set of hyperparameters $\boldsymbol{\theta}_1 = \theta_1, ..., \theta_k$,

$$\pi(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}_1) = \prod_{i=1}^{n} \pi(y_i \mid \mathbf{x}, \boldsymbol{\theta}_1). \tag{3.9}$$

The second stage concerns the specification of the latent Gaussian field,

$$\mathbf{x} \mid \boldsymbol{\theta}_2 \sim \mathcal{N}\big(\boldsymbol{\mu}(\boldsymbol{\theta}_2), \Sigma(\boldsymbol{\theta}_2)\big) \tag{3.10}$$

where the dependence structure of the data is explained. The latent field is controlled by another set of hyperparameters $\boldsymbol{\theta}_2 = \theta_{k+1}, ..., \theta_{k+m}$, and the final stage of the LGM is to assign priors to all hyperparameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$,

$$\theta_i \sim \pi(\theta_i), \quad i = 1, 2, ..., k+m. \tag{3.11}$$

To restate the MLE problem for FGN processes one notes that the likelihood of an FGN process is Gaussian and stated in Eq: 3.1. When working with LGMs it is often preferred to use the *precision* $\kappa = 1/\sigma^2$ and *precision matrix* $Q = \Sigma^{-1}$ rather than the variance and covariance matrix. Let the latent field $\mathbf{x} = (x_1, ..., x_n)$ then be defined as

$$\pi(\mathbf{x} \mid \kappa, H) = (2\pi)^{-n/2} \left|Q(H,\kappa)\right|^{1/2} \exp\left(-\frac{1}{2}\mathbf{x}^\top Q(H,\kappa)\mathbf{x}\right), \tag{3.12}$$

where the innovation precision κ and Hurst exponent H are the hyperparameters of the LGM. If they are assigned uniform priors, then finding the values of H and κ that maximize the likelihood function of an FGN is equivalent to finding the mode of the posterior marginal distribution of H and κ that ensures the best fit of the latent field of the LGM.

For LGMs to be effective in practice, it is important that the number of hyperparameters is small and that the latent field is a sparse *Gaussian Markov random field* (GMRF). GMRFs are Gaussian fields that possess the *Markov property*, i.e. most pairs of variables (x_i, x_j) are conditionally independent given the remaining variables which are denoted $\mathbf{x}_{-ij}$

$$\pi(x_i, x_j \mid \mathbf{x}_{-ij}) = \pi(x_i \mid \mathbf{x}_{-ij})\pi(x_j \mid \mathbf{x}_{-ij}). \tag{3.13}$$

The dependency structure of a GMRF $\mathbf{x} = (x_1, x_2, ..., x_n)^\top$ can be described by the *labelled graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, illustrated in Fig: 3.2. The nodes $\mathcal{V} = \{1, 2, ..., n\}$ correspond to the variables of the GMRF and the edges $\mathcal{E}$ are defined such that there is no edge between node i and j if and only if x_i and x_j are conditionally independent given the remaining variables. If two variables x_i and x_j in a GMRF are conditionally dependent it is shown in e.g. (27) that the corresponding element of the precision matrix is zero $Q_{ij} = 0$. This means that a GMRF with a poorly connected graph will result in a sparse precision matrix that allows for faster computations. Unfortunately, FGNs and other long-memory processes lack the Markov property due to the high degree of interdependence between its points. This results in a fully connected graph and a dense precision matrix that are not eligible for efficient algorithms.

Bayesian inference about the posterior marginal distributions can be achieved by using INLA (28), which is designed to work for latent Gaussian models with sparse latent fields. This method aims to approximate the posterior marginal distributions numerically instead of relying on simulations which is the common approach for such problems. INLA includes a variety of features, but unfortunately none of the built-in model templates supports the specific model described for FGN processes in section 3.3.1. The model must therefore be constructed manually with the customizable `rgeneric` model which is included in INLA. A tutorial for the `rgeneric` model is included in Appendix: A.

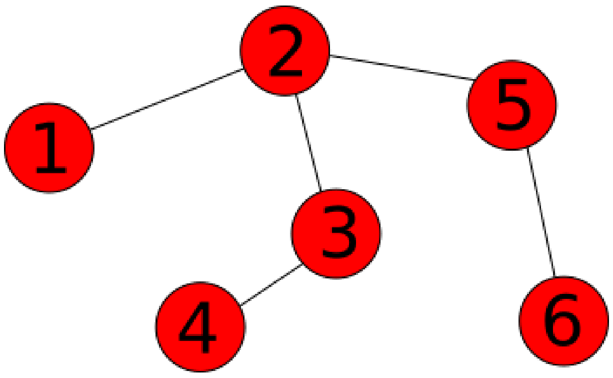

Figure 3.2: Graph of a GMRF with a low degree of conditional independence between its variables.

3.3.2 Implementing an rgeneric model

The latent Gaussian model for general FGN processes constructed in section 3.3.1 will now be defined in `rgeneric`. INLA will then be used to find the best fit for κ and H according to the given data. It is important to note the difference between *internal* and *external* scaling for the hyperparameters. The internal scaling ensures that INLA operates in a stable and unconstrained parameter space on $\mathbb{R}$. This scaling might differ from the external scaling in which the hyperparameters are usually treated. A suitable internal scaling for the innovation precision κ is the log-scale,

$$\theta_1 = \log \kappa \tag{3.14}$$

and for the Hurst exponent the logit-scale is chosen,

$$\theta_2 = \log \frac{H}{1-H}. \tag{3.15}$$

Due to the slow decrease of the autocovariance function we have a dense graph and precision matrix describing the dependencies of the latent variables

$$\mathcal{G} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & & 1 \\ \vdots & & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix}, \qquad Q = \Sigma^{-1}.$$

We need to assign suitable prior distributions for the hyperparameters in order to achieve Bayesian inference for their posterior marginals. Since $H \in (0,1)$ and no prior knowledge is known, the author has chosen a Uniform prior for the Hurst exponential,

$$\pi(H) = \text{Unif}(0,1). \tag{3.16}$$

For the precision parameter a Gamma prior distribution with parameters a and b was chosen,

$$\pi(\kappa) = \text{Gamma}(a,b). \tag{3.17}$$

For this particular `rgeneric` function let $a = 1.0$ and $b = 0.01$. `rgeneric` requires us to compute the joint prior for the hyperparameters in internal scaling. This is found by using the change-of-variables formula and yields,

$$\pi(\theta_1,\theta_2) = \pi\left(e^{\theta_1}\right)\frac{e^{\theta_1+\theta_2}}{(1+e^{\theta_2})^2} = \frac{\kappa H}{1+e^{\theta_2}}\pi(\kappa). \tag{3.18}$$

The code associated with this example is included in listing: 3.1 and should run with an asymptotic runtime of $\mathcal{O}(n^3)$. The model is applied to the Nile water level data (13) discussed in section 2.3, and the resulting plots for the posterior marginal distributions of κ and H are included in Fig: 3.3 and Fig: 3.4 respectively. The estimation obtained for the Hurst exponent is

$$\hat{H}_{\text{INLA}} = 0.8336, \tag{3.19}$$

which is close to what we get from the R/S statistic and improves upon the result from Whittle's approximate likelihood,

$$\hat{H}_{\text{R/S}} = 0.8250 \qquad \hat{H}_{\text{Whittle}} = 0.8992. \tag{3.20}$$

The dense dependence structure makes finding the determinant and computing the inverse matrix very computationally expensive, but by taking advantage of the Toeplitz structure of the autocovariance matrix one hopes to significantly improve the speed of INLA.

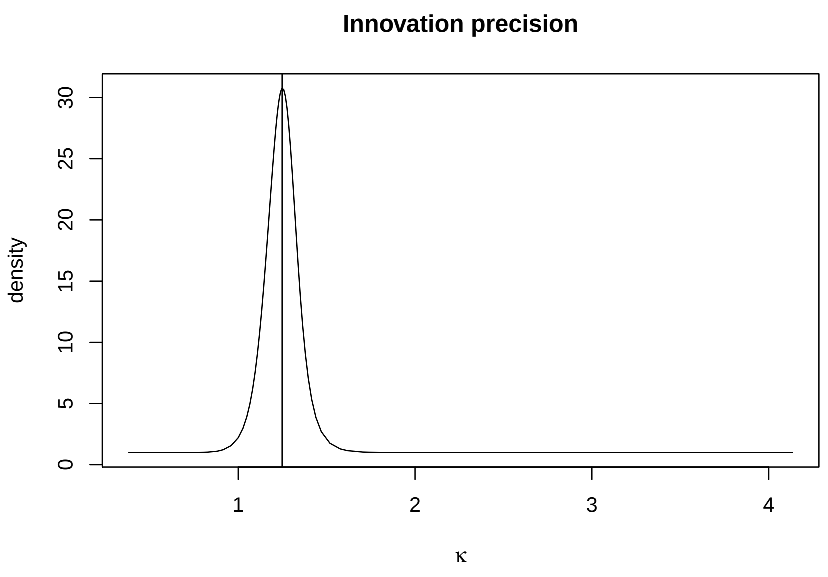

Figure 3.3: Marginal posterior distribution of the innovation precision $\kappa = 1/\sigma^2$ hyperparameter for the Nile data obtained by INLA using the rgeneric model.

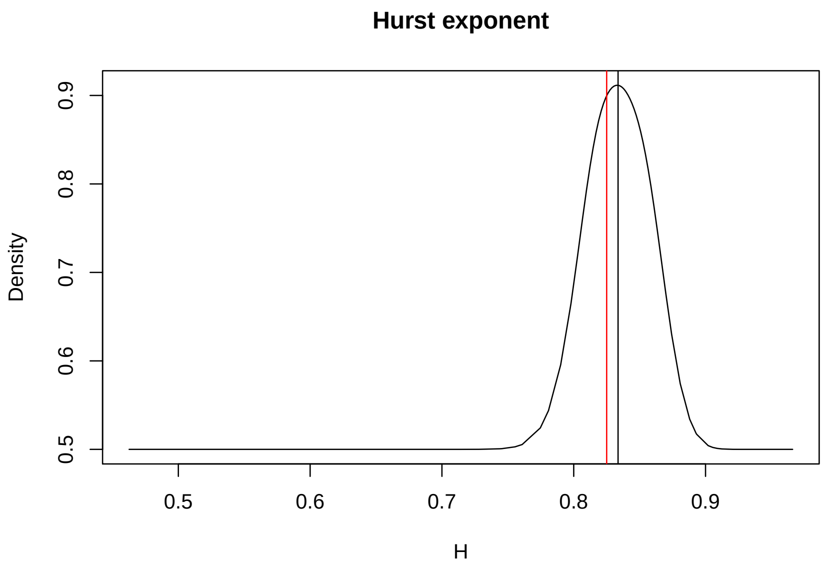

Figure 3.4: Marginal posterior distribution for the Hurst exponent hyperparameter for the Nile data obtained by INLA using the rgeneric model and compared to the R/S estimate represented by the red line.

Listing 3.1: Implementation of the rgeneric function associated with the rgeneric model in INLA.

```
fgn.rgeneric = function(
  cmd = c("graph", "Q","mu", "initial", "log.norm.const",
  "log.prior", "quit"), theta = NULL, args = NULL)
{
  library(ltsa)
  library(HKprocess)
  interpret.theta = function(theta){
    kappa = exp(theta[1])
    H = exp(theta[2])/(1+exp(theta[2]))
    return(list(H=H,kappa=kappa))
  }
  mu = function(n,theta){
    return(numeric(0))
  }
  graph = function(n, theta) {
    return ( matrix(1,nrow=n,ncol=n) )
  }
  Q = function(n, theta){
    params = interpret.theta(theta)
    kappa = params$kappa
    H = params$H
    acorr = acfHKp(H = H, maxlag = n-1)
    Q = toeplitz( c(acorr) )
    return( kappa * solve(Q) )
  }
  log.norm.const = function(n, theta){
    return(numeric(0))
  }
  log.prior = function(n, theta){
    params = interpret.theta(theta)
    return( theta[1] + dgamma(params$kappa,1.0,0.01,log=T)
          + theta[2] - 2*log(1+exp(theta[2])) )
  }
  initial = function(n, theta){
    return ( c(4,0.)  )
  }
  quit = function(n, theta){
    return ()
  }
  cmd = match.arg(cmd)
  val = do.call(cmd, args = list(
    n = as.integer(args$n),
    theta = theta))
  return (val)
}
```

3.4 Improving the rgeneric model

As stated in Eq: 2.77, the autocovariance matrix of an FGN process described in Eq: 3.2 is a Toeplitz matrix. This allows for more efficient algorithms for computing the determinant and inverse matrix, which are the most expensive computations in this program when the precision matrix is dense. The two improvements considered in this thesis is the *Trench inversion* and *Levinson-Durbin* algorithms.

Trench inversion

The Trench inversion algorithm provides a fast way of computing the inverse of a symmetric positive definite Toeplitz matrix. A detailed explanation and proof of the theory behind the algorithm is found in (30) and (7). The algorithm is stated in Alg: 1 and starts by performing the Durbin algorithm (Alg: 2) to achieve the vector $\mathbf{y}$ that solves the linear system $T_{n-1}\mathbf{y} = -(r_0, r_1, ..., r_n)^\top$ for the $n-1 \times n-1$ subblock of T,

$$T_{n-1} = \begin{bmatrix} r_0 & r_1 & r_2 & \cdots & r_{n-1} \\ r_1 & r_0 & r_1 & \cdots & r_{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{n-1} & r_{n-2} & r_{n-3} & \cdots & r_0 \end{bmatrix}. \tag{3.21}$$

The computational complexity for the Trench-Inversion algorithm is $13n^2/4$ flops, including the $2n^2$ flops spent on the Durbin algorithm. The Trench inversion algorithm can be performed by using the `TrenchInverse` function from the `ltsa` package in R.

Algorithm 1 Trench inversion algorithm: Given real numbers $\mathbf{r} = r_0, .., r_n$ such that $T_n = \left(r_{|i-j|}\right) \in \mathbb{R}^{n\times n}$ is positive definite, the algorithm computes $B = T_n^{-1}$.

```
1: procedure TRENCH-INVERSION($T, \mathbf{r}$)
2:     Use Alg: 2 to solve $T_{n-1}\mathbf{y} = -(r_1, ..., r_{n-1})^\top$
3:     $\gamma = 1/\Big(1 + \mathbf{r}\big(1 : (n-1)\big)^\top \mathbf{y}\big(1 : (n-1)\big)\Big)$
4:     $\mathbf{v}\big(1 : (n-1)\big) = \gamma\mathbf{y}\big((n-1) : -1 : 1\big)$
5:     $B(1,1) = \gamma$
6:     $B(1, 2 : n) = \mathbf{v}\big((n-1) : -1 : 1\big)$
7:     for $i$ in $2 :$ floor$\big((n-1)/2\big) + 1$ do
8:         for $j$ in $i : (n-i+1)$ do
9:             $B(i,j) = B(i-1, j-1) + \Big(\mathbf{v}(n+1-j)\mathbf{v}(n+1-i) - \mathbf{v}(i-1)\mathbf{v}(j-1)\Big)/\gamma$
       return $B$
```

Algorithm 2 Durbin algorithm: Given real numbers $\mathbf{r} = r_0, .., r_n$ such that $T_n = \left(r_{|i-j|}\right) \in \mathbb{R}^{n \times n}$ is positive definite, the following algorithm computes $y \in \mathbb{R}^n$ such that $T_n y = -\mathbf{r}$.

```
1: procedure DURBIN($T, \mathbf{r}$)
2:     $\mathbf{y}(1) = -\mathbf{r}(1)$
3:     $\beta = 1$
4:     $\alpha = -\mathbf{r}(1)$
5:     for $k$ in $1 : (n-1)$ do
6:         $\beta = (1 - \alpha^2)\beta$
7:         $\alpha = -(\mathbf{r}(k+1) + \mathbf{r}(k : -1 : 1)^\top \mathbf{y}(1 : k))/\beta$
8:         $\mathbf{z}(1 : k) = \mathbf{y}(1 : k) + \alpha \mathbf{y}(k : -1 : 1)$
9:         $\mathbf{y}(1 : (k+1)) = \begin{bmatrix} z(1:k) \\ \alpha \end{bmatrix}$
       return $\mathbf{y}$
```

Levinson-Durbin recursion

The Levinson-Durbin recurison is an algorithm that recursively finds the solution to a system of linear equations,

$$\mathbf{y} = \mathbf{A}\mathbf{x} \tag{3.22}$$

where $\mathbf{A}$ is a Toeplitz matrix. The procedure was first designed by Norman Levinson in 1947 (20) and then improved by James Durbin in 1960 (5). The version of the algorithm used here is the further improved version by W.F. Trench in 1964 (30). The algorithm is described in Alg: 3 (7) and require $4n^2$ flops. The Levinson-Durbin algorithm is used to find the determinant of the precision matrix more efficiently which can be extracted from the `ltza` function in the R package `HKprocess`.

Algorithm 3 Levinson-Durbin algorithm: Given $b \in \mathbb{R}^n$ and real numbers $\mathbf{r} = r_0, .., r_n$ such that $T_n = \left(r_{|i-j|}\right) \in \mathbb{R}^{n \times n}$ is positive definite, the following algorithm computes $x \in \mathbb{R}^n$ such that $T_n x = b$.

```
1: procedure LEVINSON-DURBIN($T, \mathbf{b}$)
2:     $\mathbf{y}(1) = -\mathbf{r}(1)$
3:     $\mathbf{x}(1) = \mathbf{b}(1)$
4:     $\beta = 1$
5:     $\alpha = -\mathbf{r}(1)$
6:     for $k$ in $1 : (n-1)$ do
7:         $\beta = (1 - \alpha^2)\beta$
8:         $\mu = \left(\mathbf{b}(k+1) - \mathbf{r}(1 : k)^\top \mathbf{x}(k : -1 : 1)\right)/\beta$
9:         $\mathbf{z}(1 : k) = \mathbf{y}(1 : k + \alpha \mathbf{y}(k : -1 : 1))$
10:        $\mathbf{y}(1 : (k+1)) = \begin{bmatrix} z(1:k) \\ \mu \end{bmatrix}$
       return $\mathbf{y}$
```

Comparing efficiency

The Trench inversion algorithm is used to reduce the cost of inverting the autocovariance matrix from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ flops. To make sure that INLA computes the normalizing constant efficiently it is encouraged to implement the `log.norm.const` function manually. With the Levinson-Durbin algorithm one is able to reduce the cost of computing the determinant to $\mathcal{O}(n^2)$ flops. The function `acfHKp` from the `HKprocess` package is used to compute the autocovariances of the FGN process. The R code including the improvements is included in listing: 3.2.

To compare the algorithms we draw different sized samples of data using the Wood-Chan algorithm and use INLA to analyze each set of data. The time spent by each implementation is illustrated in Figures 3.5 and 3.6. As can be seen, the second algorithm was a significant improvement over the original one. However, as the FGN processes are rather long by nature, the rgeneric process still proves too slow to become a viable tool for FGN analysis. Running our improved `rgeneric` model on the Nile data with length $n = 663$ requires a runtime of around 10 minutes on a fairly good computer. It is the density of the graph and precision matrix that is the main concern and primary limitation of the INLA program for such models. Methods of circumventing this problem will be explored in the next chapter.

Listing 3.2: Implementation of the revised rgeneric function associated with the rgeneric model in INLA.

```
fgn.rgeneric2 = function(
  cmd = c("graph", "Q","mu", "initial", "log.norm.const",
  "log.prior", "quit"), theta = NULL, args = NULL)
{
  library(ltsa)
  library(HKprocess)
  interpret.theta = function(theta){
    return(list(kappa = exp(theta[1]),
                 H = exp(theta[2])/(1+exp(theta[2]))))
  }
  mu = function(n,theta){ return(numeric(0))  }
  graph = function(n, theta,ntheta){
    return ( matrix(1,nrow=n,ncol=n) )
  }
  Q = function(n, theta){
    params = interpret.theta(theta)
    kappa = params$kappa
    H = params$H
    acorr = acfHKp(H = H, maxlag = n-1)
    Q = toeplitz( c(acorr) )
    return( kappa * TrenchInverse(Q) )
  }
  log.norm.const = function(n, theta){
    params = interpret.theta(theta)
    acorr = acfHKp(H = params$H, maxlag = n-1)
    logdet = ltza(acorr,rep(0,n))
    ledd1 = -n/2*log(2*pi)
    ledd2 = -0.5*as.numeric(logdet[4] )
    ledd3 = n/2*log(parms$kappa)
    return(ledd1+ledd2+ledd3)
  }
  log.prior = function(n, theta){
    params = interpret.theta(theta)
    return( theta[1] + dgamma(params$kappa,1.0,0.01,log=T)
         + theta[2] - 2*log(1+exp(theta[2])) )
  }
  initial = function(n, theta){
    return( c(4,0.)  ) }
  quit = function(n, theta){ return() }
  cmd = match.arg(cmd)
  val = do.call(cmd, args = list(
    n = as.integer(args$n),
    theta = theta))
  return (val)
}
```

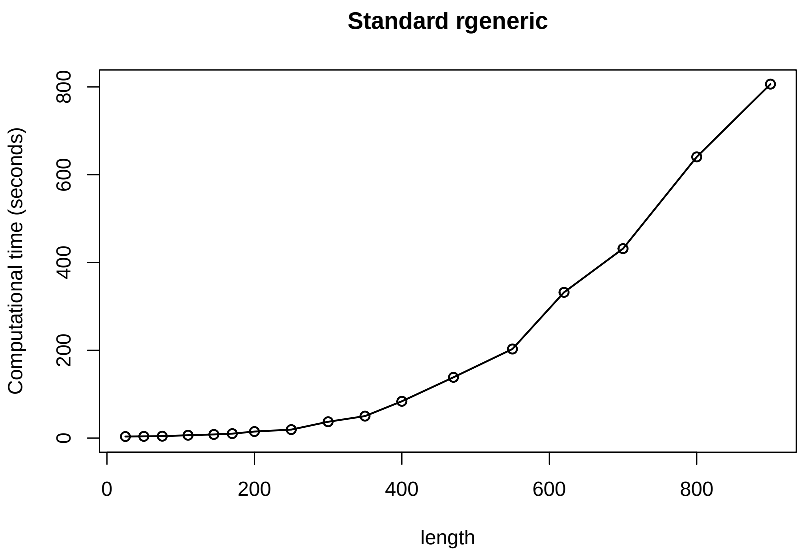

Figure 3.5: Plot of the time consumption of the standard rgeneric model created in section 3.3.2 when applied to simulations of length n.

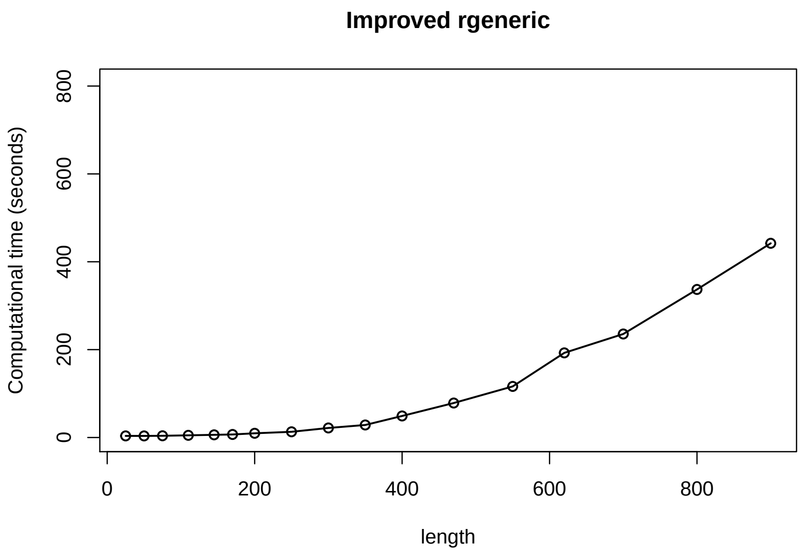

Figure 3.6: Plot of the time consumption of the improved rgeneric model created in section 3.4 when applied to simulations of length n.

Chapter 4

AR1 approximations of FGN models

4.1 Motivation

In 1980, Clive William John Granger (8) explored the aggregation of short term memory processes and discovered that the sum of AR1 processes where the correlation parameter is sampled from a Beta distribution exhibits long-memory properties. This chapter will try to use this result to explore the possibility of explaining long-memory processes by a sum of short term memory AR1 processes.

The goal is to restate any FGN process as a sum of a limited number of AR1 processes by finding the processes that best approximate the FGN process in terms of its Hurst exponent H. If this is possible, the latent Gaussian field described in section 3.3.1 can be replaced with a larger field, but with a sparse corresponding precision matrix due to conditional independence. This could hypothetically decrease the runtime of INLA significantly.

4.2 Aggregation of AR1 processes

The contents of this section is largely based upon the work of C.W.J Granger (8). Recall that an $ARMA(p, q)$ model is a time series model described by

$$\phi_p(B)x_t = \theta_q(B)\epsilon_t, \tag{4.1}$$

where $\phi_p(B) = (1 - \phi_1 B)(1 - \phi_2 B)\cdots(1 - \phi_p B)$ is a polynomial of degree p and $\theta_q(B) = (1 - \theta_1 B)(1 - \theta_2 B)\cdots(1 - \theta_q B)$ is a polynomial of degree q. Granger and Morris established in 1976 (10) that the sum of two ARMA series with parameters (p_1, q_1) and (p_2, q_2) becomes another ARMA model,

$$\text{ARMA}(p_1, q_1) + \text{ARMA}(p_2, q_2) = \text{ARMA}(x, y) \tag{4.2}$$

where the parameters x, y are explained by

$$x \leq p_1 + p_2, \qquad y \leq \max(p_1 + q_2, p_2 + q_1). \tag{4.3}$$

The inequalities of Eq: 4.3 is necessary to account for the possibility of common roots between $\phi_p(B) = 0$ and $\theta_q(B) = 0$ that will cancel each other out. For k mutual roots Eq: 4.3 becomes

$$x = p_1 + p_2 - k \qquad y \leq \max(p_1 + p_2 - k, q_2 + q_1 - k). \tag{4.4}$$

The remaining inequality of 4.4 is still required in some exceptions. To show one situation where this will occur consider the following example by Granger and Morris (10). Suppose

$$\begin{aligned}(1 - \phi B)x_t &= \epsilon_t, \quad \text{i.e. } x_t \sim \text{AR}(1)\\ (1 + \phi B)y_t &= \eta_t, \quad \text{i.e. } y_t \sim \text{AR}(1)\end{aligned}$$

with equal innovations, $\text{var}(\epsilon) = \text{var}(\eta) = \sigma^2$. If $z_t = x_t + y_t$, then

$$(1 - \phi B)(1 + \phi B)z_t = (1 + \phi B)\epsilon_t + (1 - \phi B)\eta_t. \tag{4.5}$$

The right hand side of Eq: 4.5, denoted ψ_t has variance $\text{var}(\psi) = 2(1 + \phi^2)\sigma^2$ with zero autocovariance for any lag $k > 0$. This means that z_t is an AR2 process rather than an ARMA$(2, 1)$ process as Eq: 4.3 would suggest. If two series generated by

$$x_{jt} = \phi_j x_{j,t-1} + \epsilon_{jt}, \qquad j = 1, 2 \tag{4.6}$$

and ϵ_{1t} and ϵ_{2t} are *independent* zero-mean white noise series, then the sum

$$y_t = x_{1t} + x_{2t}, \tag{4.7}$$

follows an ARMA(2,1). Similarly, if N AR(1) models are added as follows,

$$\bar{x}_t = \sum_{j=1}^{N} x_{jt} \tag{4.8}$$

then the aggregate series would follow an ARMA$(N, N - 1)$ model. An important result arises when investigating the spectrum of such aggregations. From Equation 2.32 we find the power spectrum of each x_{jt} is to be (in complex form)

$$f_j(\lambda) = \frac{1}{|1 - \phi_j z|^2} \frac{\text{var}(\epsilon_{jt})}{2\pi}, \qquad z = e^{-i\lambda}. \tag{4.9}$$

As each x_{it} is independent from any other x_jt, then the spectrum of the aggregate series $\bar{x}$ is given by

$$\bar{f}(\lambda) = \sum_{j=1}^{N} f_j(\lambda) \tag{4.10}$$

Assuming further that ϕ_j are random variables (29) such that $\phi \sim F(\phi)$, and that $\text{var}(\epsilon_{jt})$ are drawn from another independent distribution. Then the spectral density of the aggregated process of Eq: 4.10 can be approximated as

$$\bar{f}(\lambda) \approx \frac{N}{2\pi} \text{E}[\text{var}(\epsilon_{jt})] \int \frac{1}{|1 - \phi z|^2} \text{d}F(\phi). \tag{4.11}$$

If ϕ can take only $m < N$ discrete values on the interval $(-1,1)$, then $\bar{f}(\lambda)$ is a spectrum of an ARMA$(m, m-1)$ model. If no such restriction applies and ϕ can take any continuous value on $(-1,1)$, then there are no ARMA models with finite parameters that fits the spectrum $\bar{f}(\lambda)$. To proceed it is assumed that $\phi \in (0,1)$ follows a Beta distribution on the form

$$\mathrm{d}F(\phi) = \begin{cases} \frac{2}{B(p,q)}\phi^{2p-1}(1-\phi^2)^{q-1}\mathrm{d}\phi, & 0 \le \phi \le 1 \\ 0 & \text{otherwise,} \end{cases} \tag{4.12}$$

where $p, q > 0$. As one can write

$$\frac{1}{|1-\phi z|^2} = \frac{1}{(1-\phi)^2}\left[\frac{1+\phi z}{1-\phi z} + \frac{1+\phi \bar{z}}{1-\phi \bar{z}}\right], \tag{4.13}$$

and since

$$\frac{1+\phi z}{1-\phi z} = 1 + 2\sum_{j=1}^{\infty}(\phi z)^j, \tag{4.14}$$

we are able to deduct from Eq: 4.11 and Eq: 4.12 that the coefficient of z^k in $\bar{f}(\lambda)$, denoted here as $\bar{\mu}_k$ has to be

$$\bar{\mu}_k = \frac{2}{B(p,q)}\int_0^1 \phi^{2p+k-1}(1-\phi^2)^{q-2}\mathrm{d}\phi. \tag{4.15}$$

$\bar{\mu}_k$ is known to be the autocovariance of $\bar{x}_t$ at lag k. If $q > 1$ we get

$$\bar{\mu}_k = \frac{\Gamma(q-1)}{B(p,q)} \cdot \frac{\Gamma(p+k/2)}{\Gamma(p+k/2+q-1)}, \tag{4.16}$$

which gives the approximation

$$\bar{\mu}_k = Ck^{1-q} \tag{4.17}$$

for large values of k and a constant C. This means that $\bar{x}_t$ exhibits long-memory in the covariance sense (Def: 2.3.4) and is a fractionally integrated series of order $d = 1 - q/2$,

$$\bar{x}_t \sim I(1-q/2). \tag{4.18}$$

Note that the order of integration, $d = 1 - q/2$ is solely dependent on q, meaning that the shape of $\mathrm{d}F(\phi)$ holds no relevance except for around $\phi = 1$. This discovery was first made by C.W.J. Granger in 1980 (8). Creating long-memory processes by aggregating AR1 processes as described here can be used to simulate fractional Gaussian noise series (12). Let $\{x_{i,t}\}$ denote a set of AR1 series of length n, each defined by the difference equation

$$x_{i,t} = \phi_i x_{i,t-1} + \epsilon_t \quad i = 1, 2, ..., N, \tag{4.19}$$

where ϵ_t is white noise and

$$\phi_i \sim \text{Beta}(p,q). \tag{4.20}$$

Further, define the aggregation series x_t as

$$x_t = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} x_{i,t}. \tag{4.21}$$

From Eq: 4.17 Granger shows that the series defined by Eq: 4.21 exhibit long-memory. It is shown in e.g. Haldrup and Vera-Valddés (12) that the scaled partial sum

$$X_n(\xi) = \sigma_n^{-1} \sum_{t=1}^{[n\xi]} x_t, \tag{4.22}$$

with $\sigma_n^2 = \mathrm{E}\left[\left(\sum_{t=1}^{n} x_t\right)^2\right]$ and $\xi \in [0,1]$, converges in distribution to $B_H(\xi)$, an FBM with Hurst exponent $H = \frac{1}{2}(3-q)$. Hence, the generation of an FBM series with Hurst exponent H by AR1 aggregation can be summarized as follows:

- Sample N AR coefficients $\{\phi_i\}_{i=1}^{N}$ from the density function, in our case it is the Beta distribution with parameters p and $q = 3 - 2H$.
- Generate N AR1 series of length n using the sampled coefficients. The error terms $\epsilon_{i,t}$ are sampled from independent standard Gaussian distributions.
- Aggregate the AR1 series according to Eq: 4.21 to get the FGN simulation.
- For an FBM series one can aggregate the generated series according to equation Eq: 4.22.

As an example we try to create a long-memory FGN with $H = 0.9$. We start by generating $N = 100,000$ AR1 processes of length $n = 400$, with mutual innovation variance equal to $\sigma_\epsilon^2 = 1$. The coefficients are sampled from a beta distribution

$$\phi_i \sim \mathrm{Beta}(1.4, 3-2H) \quad i = 1, 2, ..., N. \tag{4.23}$$

After aggregating the AR1 processes we estimate H by using the INLA method implemented in section 3.4. The posterior marginal distribution was computed by INLA with the improved rgeneric model implemented in listing: 3.2 and is illustrated in Fig: 4.1. The mode was found to be $\hat{H}_{\text{INLA}} = 0.9163$.

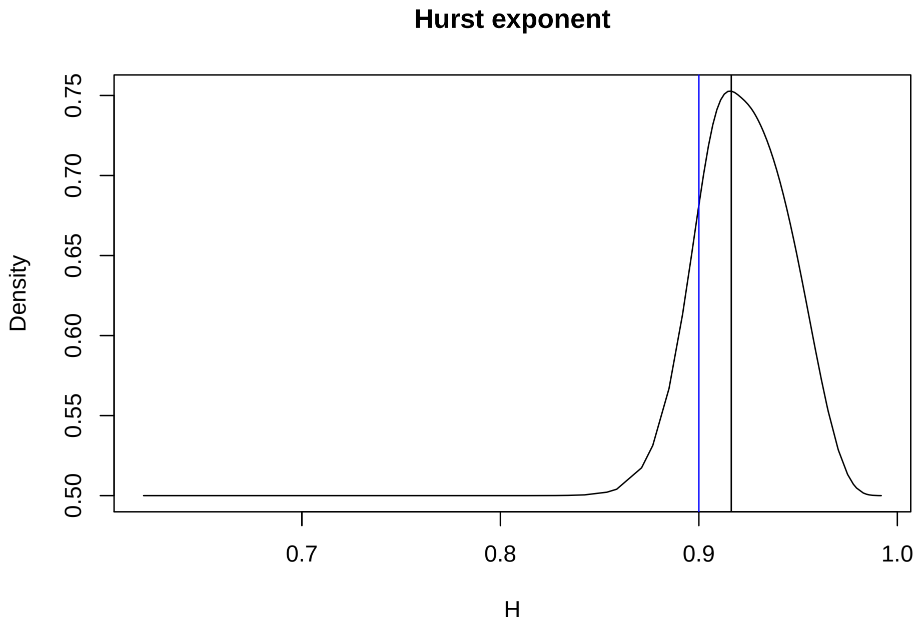

Figure 4.1: Posterior marginal distribution of Hurst exponent for an aggregated series of 100,000 AR1 processes of length 400.

4.3 AR1 approximation with numerical optimization

It has been established that an FGN can be approximated by a sum of N AR1 processes, with parameters drawn from a Beta distributions. As N grows larger, the sum becomes a better approximation, but also more costly in terms of efficiency and memory. The goal now is to disaggregate a long-memory series of unknown Hurst exponent and innovation variance into a sum of N weighted AR1 processes that are scaled to achieve unit variance,

$$\bar{x}_t = \sum_{i=1}^{N} \sqrt{w_i}\,\sqrt{1-\phi_i^2}\,x_{i,t} \tag{4.24}$$

where all weights sum to 1 and the AR1 processes are explained by the difference equation

$$x_{i,t} = \phi_i x_{i,t-1} + \epsilon_t \quad i = 1,2,...,N, \tag{4.25}$$

with noise terms of fixed unit variance. The challenge will be to choose the AR1 series whose sum best replicate an FGN process with Hurst exponent H, while keeping the number of AR1 components sufficiently low.

If we have a mapping between H and the set of parameters and weights, we can use INLA to find the H whose corresponding set of weights and parameters best fit the likelihood of the given data. Such a mapping is difficult to find analytically, but can be approximated by performing numerical optimization methods to find the corresponding set of weights and parameters for a discrete set of values for H as to sufficiently represent the interval $(0.5, 1)$. The continuous mapping is then found by using splines to interpolate the resulting parameters.

For a given set of N parameters with corresponding weights, the Hurst exponent is estimated by finding the average MLE of R replicated aggregation series

$$\hat{H} = \frac{1}{R}\sum_{r=1}^{R} \hat{H}_r, \tag{4.26}$$

where each $\hat{H}_r$ was found by using the `FitFGN` function from the `FGN` package in R. In this thesis, the number of replications was set to $R = 10$ so that a reasonable computational time could be achieved, but it can be increased to improve accuracy. The length of each generated AR1 series was set to $n = 1000$ for the same reason, but can also be increased for a more accurate result. The optimization problem for given N and H consists of finding the set of parameters $\boldsymbol{\phi} = \phi_1, ..., \phi_N$ and weights $\mathbf{w} = w_1, ..., w_N$ that minimize the cost function

$$f(\boldsymbol{\phi}, \mathbf{w}) = (\hat{H} - H)^2 \tag{4.27}$$

subject to constraints

$$\sum_{j=1}^{N} w_j = 1, \quad \phi_i \in (0,1), \quad w_i > 0, \quad i = 1, 2, ..., N. \tag{4.28}$$

To ease the opimization problem the author has chosen to re-parametrise the parameters $\{\phi_i\}_{i=1}^N$ in terms of variables $\{u_i\}_{i=1}^N$ such that $u_i \in (-\infty, \infty)$ and $\phi_1 > \phi_2 > \cdots > \phi_N$. The parametrization chosen for the ϕs is

$$\phi_i = \frac{1}{1 + \sum_{j=1}^{i} e^{-u_j}}. \tag{4.29}$$

For the weights it is important to notice that the sum-to-one constraint reduces the degrees of freedom of the weights to $N - 1$, meaning that one of the weights follow from the value of the others, i.e. $w_i = 1 - \sum_{j \neq i} w_j$. The parametrization for the weights $\{w_i\}_{i=1}^N$ was also chosen to be a function of unconstrained variables $\{v_i\}_{i=1}^N$, where the first variable v_1 is set to 1,

$$w_i = \frac{e^{v_i}}{\sum_{j=1}^{N} e^{v_j}} \qquad \forall i = 1, 2, ..., N. \tag{4.30}$$

From Eq: 4.30 it is evident that the N weights all sum to one and are determined by $N - 1$ unconstrained variables on $\mathbb{R}$. As an example, consider the case of $N = 3$ AR1 components. The parametrization for the parameters is chosen according to Eq: 4.29,

$$\begin{aligned}\phi_1 &= \frac{1}{1+e^{-u_1}}\\ \phi_2 &= \frac{1}{1+e^{-u_1}+e^{-u_2}}\\ \phi_3 &= \frac{1}{1+e^{-u_1}+e^{-u_2}+e^{-u_3}}\end{aligned}$$

resulting in parameters $\phi_1 > \phi_2 > \phi_3$, where $\phi_i \in (0,1)$ for all $i = 1,2,3$. The parametrization for the weights follows from Eq: 4.30 with $v_1 = 1$ becomes

$$\begin{aligned}w_1 &= \frac{e^1}{e^1+e^{v_2}+e^{v_3}}\\ w_2 &= \frac{e^{v_2}}{e^1+e^{v_2}+e^{v_3}}\\ w_3 &= \frac{e^{v_3}}{e^1+e^{v_2}+e^{v_3}}.\end{aligned}$$

To ensure a continuous function the random number generator is seeded the same number (123) for all iterations and with a constant number of 100 burn-in samples. The AR1(ϕ) processes are generated by the `arima.sim` function in R and is scaled as to create a variance of 1,

```
arima.sim(n = 1000, model = list(ar = c(alpha), n.start = 100)
               * sqrt(1 - alpha^2) * sqrt(weight[i]).
```

The optimization is done by the R function `optim` and repeated for all $H = 0.50, 0.51, ..., 0.99$. The domain of H could of course be extended to cover all of (0,1), but was reduced to save time since this thesis focuses primarily on long-memory processes with $H \in (0.5, 1)$. The resulting weights and parameters for $N = 3$ AR1 components are included in tables 4.1 and 4.2 and displayed graphically in Fig: 4.2. The process is repeated for $N = 2, 4, 5, ..., 8$ where the results are displayed in figures 4.3 through 4.8. The sudden spike at $H = 0.99$ when $N = 3$ is due to the convergence difficulties that occur as H approaches unity. This value is omitted for the other values of N as it was deemed too inaccurate to be featured in the results. By interpolating the results as functions of H we have a continuous mapping from the H space to the parameter space, which we are able to use in an `rgeneric` function.

H	ϕ_1	ϕ_2	ϕ_3	w_1	w_2	w_3
0.50	0.009	0.009	0.009	0.197	0.355	0.448
0.51	0.019	0.018	0.017	0.121	0.186	0.693
0.52	0.039	0.033	0.032	0.094	0.132	0.774
0.53	0.064	0.05	0.047	0.085	0.117	0.797
0.54	0.092	0.068	0.063	0.082	0.111	0.807
0.55	0.122	0.086	0.078	0.08	0.107	0.812
0.56	0.154	0.104	0.093	0.079	0.106	0.815
0.57	0.185	0.122	0.107	0.079	0.105	0.816
0.58	0.217	0.14	0.122	0.079	0.105	0.816
0.59	0.248	0.157	0.136	0.079	0.105	0.815
0.60	0.278	0.175	0.15	0.08	0.106	0.814
0.61	0.308	0.193	0.163	0.08	0.107	0.813
0.62	0.336	0.211	0.177	0.081	0.107	0.811
0.63	0.363	0.228	0.19	0.082	0.109	0.809
0.64	0.39	0.246	0.203	0.083	0.11	0.808
0.65	0.415	0.263	0.216	0.084	0.111	0.805
0.66	0.439	0.28	0.229	0.084	0.112	0.803
0.67	0.462	0.296	0.242	0.085	0.114	0.801
0.68	0.484	0.313	0.255	0.086	0.115	0.799
0.69	0.504	0.329	0.267	0.087	0.117	0.796
0.70	0.524	0.346	0.28	0.088	0.118	0.794
0.71	0.543	0.362	0.292	0.089	0.12	0.791
0.72	0.562	0.377	0.305	0.09	0.121	0.789
0.73	0.579	0.393	0.317	0.091	0.123	0.786
0.74	0.596	0.408	0.329	0.092	0.125	0.783

Table 4.1: Values of the parameters for $H \in [0.5, 0.74]$ with $N = 3$ AR1 processes.

H	ϕ_1	ϕ_2	ϕ_3	1	2	3
0.75	0.611	0.423	0.342	0.093	0.126	0.781
0.76	0.627	0.438	0.354	0.094	0.128	0.778
0.77	0.641	0.453	0.366	0.095	0.13	0.775
0.78	0.655	0.468	0.378	0.096	0.132	0.773
0.79	0.668	0.482	0.39	0.097	0.133	0.77
0.80	0.681	0.496	0.402	0.098	0.135	0.767
0.81	0.694	0.51	0.414	0.099	0.137	0.764
0.82	0.705	0.523	0.426	0.1	0.139	0.762
0.83	0.717	0.537	0.438	0.1	0.141	0.759
0.84	0.728	0.55	0.45	0.101	0.143	0.756
0.85	0.738	0.563	0.461	0.102	0.145	0.753
0.86	0.748	0.576	0.473	0.103	0.147	0.75
0.87	0.758	0.588	0.485	0.104	0.149	0.747
0.88	0.767	0.601	0.497	0.105	0.151	0.744
0.89	0.776	0.613	0.509	0.106	0.153	0.741
0.90	0.785	0.625	0.521	0.107	0.155	0.738
0.91	0.794	0.637	0.533	0.108	0.157	0.734
0.92	0.802	0.649	0.545	0.109	0.159	0.731
0.93	0.81	0.661	0.557	0.11	0.162	0.728
0.94	0.819	0.674	0.57	0.111	0.164	0.724
0.95	0.827	0.686	0.583	0.113	0.167	0.721
0.96	0.835	0.699	0.597	0.114	0.17	0.717
0.97	0.845	0.714	0.612	0.115	0.173	0.712
0.98	0.857	0.732	0.633	0.117	0.177	0.707
0.99	0.885	0.78	0.687	0.12	0.186	0.694

Table 4.2: Values of the parameters for $H \in [0.75, 0.99]$ with $N = 3$ AR1 processes.

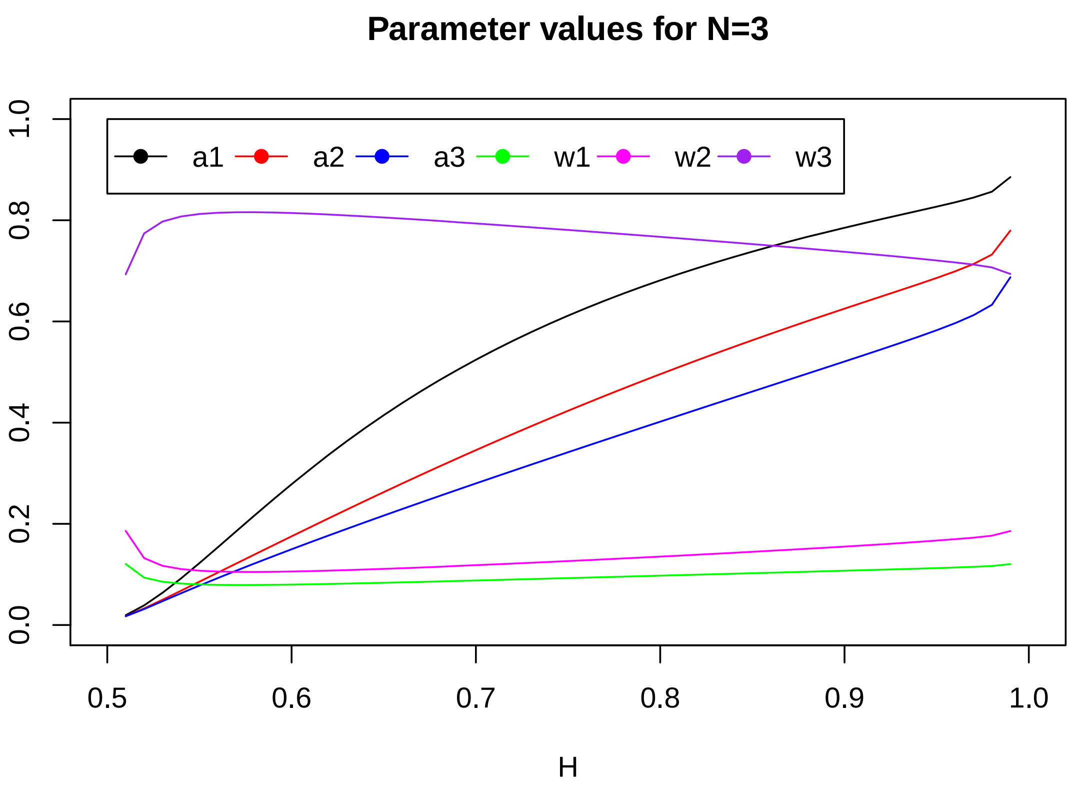

Figure 4.2: The parameters and weights for the sum of $N = 3$ AR1 processes as a function of H.

4.3.1 Kullback-Leibler divergence

The *Kullback-Leibler* divergence (KLD) was introduced by Solomon Kullback and Richard Leibler in 1951 (19) and is a measure of distance between two probability distributions P and Q. It is also used in information theory as a measure of the amount of information lost when using one distribution to approximate another. It is important to note that the KLD is not a true metric as it is neither symmetric nor does the triangle inequality hold. The Kullback-Leibler divergence from the approximate distribution Q to the true distribution P is defined as

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} \mathrm{d}x, \tag{4.31}$$

where p and q are the density functions of P and Q respectively. It can be shown that maximizing the log-likelihood function,

$$\hat{H} = \operatorname*{argmax}_H \sum_{i=1}^{n} \log L(x_i \mid H) \tag{4.32}$$

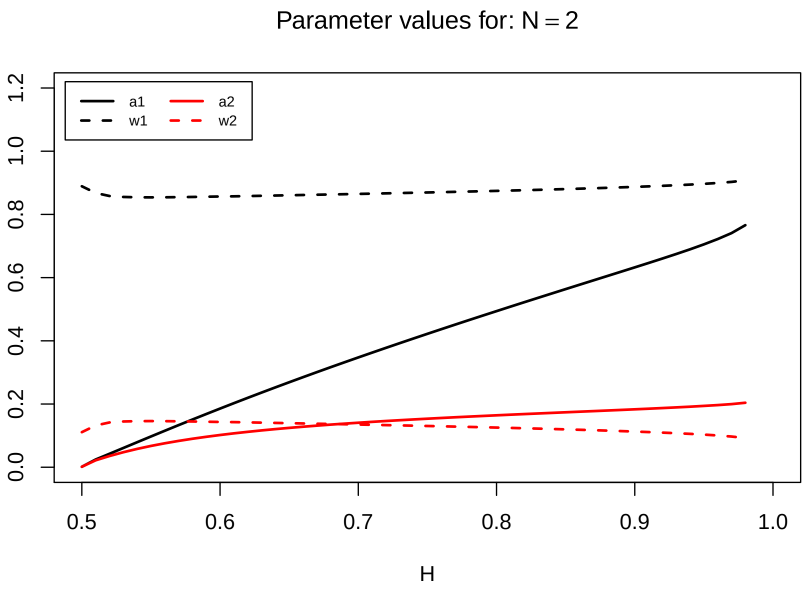

Figure 4.3: Weigts and parameters for $N = 2$ AR1 components.

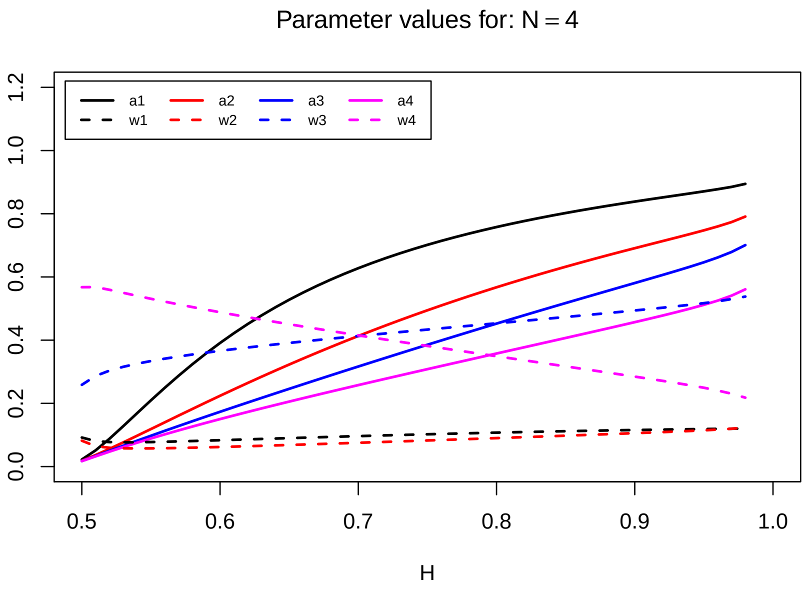

Figure 4.4: Weigts and parameters for $N = 4$ AR1 components.

is equivalent of minimizing the KLD from the estimation $L(x|\hat{H})$ to the true distribution $L(x|H^*)$

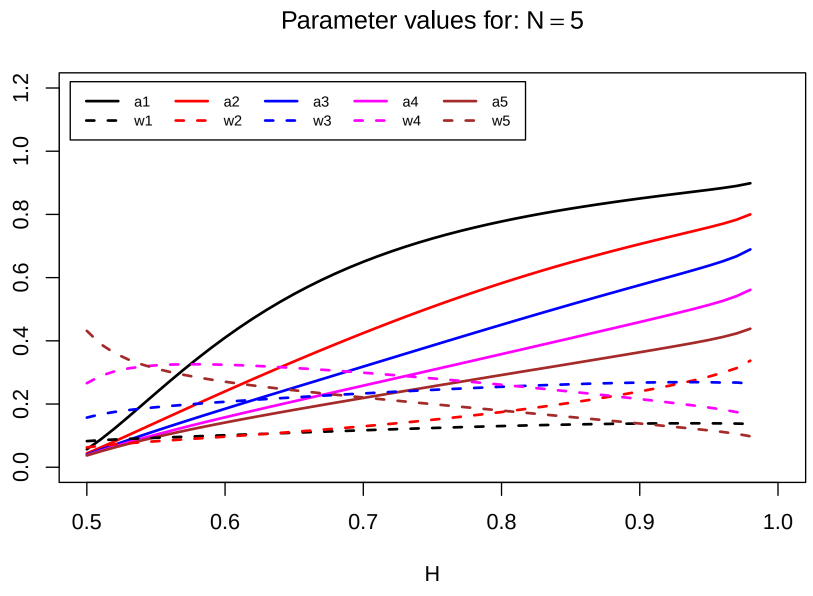

Figure 4.5: Weigts and parameters for $N = 5$ AR1 components.

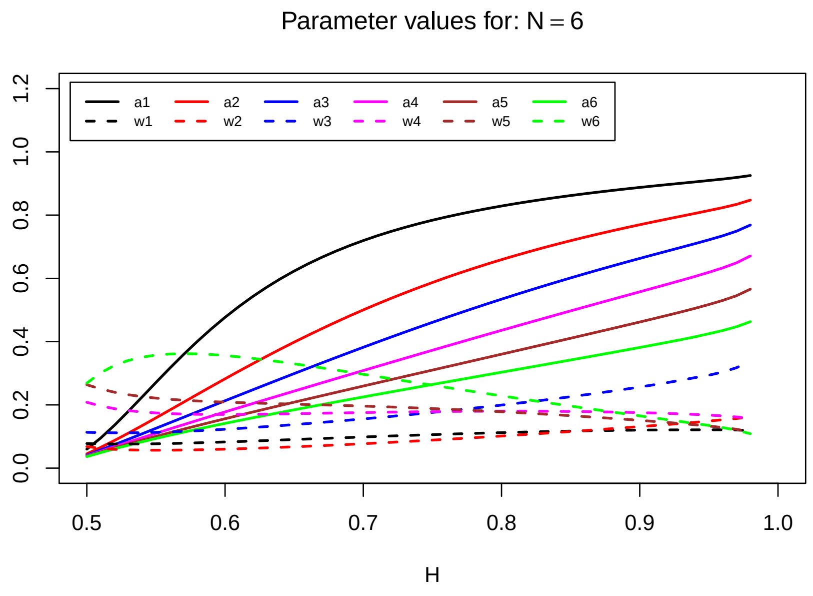

Figure 4.6: Weigts and parameters for $N = 6$ AR1 components.

by using that

$$D_{\mathrm{KL}}\big(L(x \mid H^*) \parallel L(x \mid \hat{H})\big) \overset{\hat{H}}{\propto} -\mathrm{E}\big[\log L(x \mid \hat{H})\big], \tag{4.33}$$

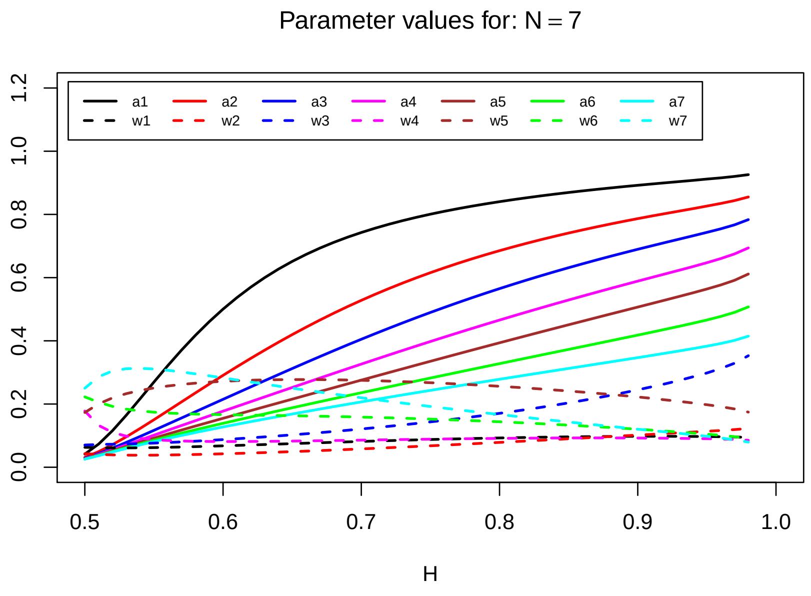

Figure 4.7: Weigts and parameters for $N = 7$ AR1 components.

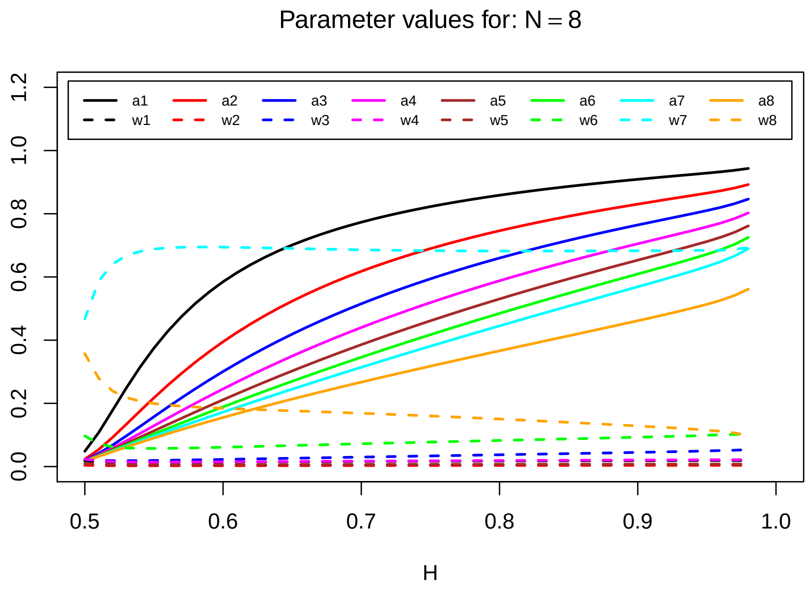

Figure 4.8: Weigts and parameters for $N = 82$ AR1 components.

where $\mathrm{E}\big[\log L(x|\hat{H})\big]$ is the mean of the estimated log-likelihood with respect to the distribution

$L(x|H^*)$. Minimizing the KLD is done by finding the $\hat{H}$ that maximizes

$$\mathrm{E}\big[\log L(x \mid \hat{H})\big] = \int_{-\infty}^{\infty} \log L(x \mid \hat{H}) L(x \mid H^*) \mathrm{d}x. \tag{4.34}$$

The integral of Eq: 4.34 is approximated by a sum with the empirical observations

$$\mathrm{E}\big[\log L(x \mid \hat{H})\big] \approx \frac{1}{n} \sum_{i=1}^{n} \log L(x_i \mid \hat{H}) \tag{4.35}$$

which leads to the same optimization problem as in Eq: 4.32. In our optimization procedure the true distribution corresponds to the sum of AR1 processes, while the approximate distribution is the FGN. This gives the interpretation that the fitted parameters in the sum of AR1 processes minimize the KLD to the FGN. It would be more natural to reverse the role of the sum of AR1's and the FGN, so that the true distribution is the FGN and the sum of AR1's is the approximate distribution. However, with such an approach the fitting procedure would be much more involved and computational demanding, and for these reasons deferred to further work.

4.4 Aggregated rgeneric model

The link between the FGN with Hurst exponent H and set of weighted AR1 processes established in section 4.3 allows us to create an rgeneric model based on the aggregated AR1 components that approximate the previous model, but is more compatible with INLA. To capture the dependence structure between the AR1 components and the aggregated process the latent field is defined in blocks as a *multivariate Gaussian Markov random field* (MGMRF) (27),

$$\mathbf{x} = \left(\mathbf{z}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\right) \tag{4.36}$$

where $\mathbf{x}^{(i)} = (x_1^{(i)}, ..., x_n^{(i)})$ represents the AR1 components,

$$x_t^{(i)} = \phi_i x_{t-1}^{(i)} + \epsilon_t, \quad i = 1, 2, ..., N, \tag{4.37}$$

and

$$\mathbf{z} = \kappa^{-1/2}\left(\sqrt{w_1}\mathbf{x}^{(1)} + ... + \sqrt{w_N}\mathbf{x}^{(N)} + \tau\right) \tag{4.38}$$

is the aggregation series with added small noise τ in the exp(15) range. The MGMRF yields a precision matrix with a block matrix structure,

$$\mathbf{Q} = \begin{bmatrix} Q_{z,z} & Q_{z,1} & \cdots & Q_{z,N} \\ Q_{z,1} & Q_{1,1} & \cdots & Q_{1,N} \\ \vdots & & \ddots & \\ Q_{z,N} & Q_{N,1} & \cdots & Q_{N,N} \end{bmatrix}. \tag{4.39}$$

For simplicity, consider at first the case of only $N = 2$ AR1 processes. The precision matrix is evaluated from the joint distribution

$$\pi\left(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{z}\right) = \pi\left(\mathbf{x}^{(1)}\right)\pi\left(\mathbf{x}^{(2)} \mid \mathbf{x}^{(1)}\right)\pi\left(\mathbf{z} \mid \mathbf{x}^{(1)}, \mathbf{x}^{(2)}\right), \tag{4.40}$$

with

$$\pi\left(\mathbf{x}^{(1)}\right) = (2\pi)^{-n/2}|Q_{\text{AR1}(\phi_1)}| \exp\left(-\frac{1}{2}\mathbf{x}^{(1)\top}Q_{\text{AR1}(\pi_1)}\mathbf{x}^{(1)}\right), \tag{4.41}$$

$$\pi\left(\mathbf{x}^{(2)} \mid \mathbf{x}^{(1)}\right) = (2\pi)^{-n/2}|Q_{\text{AR1}(\phi_1)}| \exp\left(-\frac{1}{2}\mathbf{x}^{(2)\top}Q_{\text{AR1}(\phi_2)}\mathbf{x}^{(2)}\right) \tag{4.42}$$

and

$$\pi\left(\mathbf{z} \mid \mathbf{x}^{(1)}, \mathbf{x}^{(2)}\right) = (2\pi)^{-n/2}\tau^{n/2}\exp\left(\mathbf{u}^\top \tau \mathbf{I}\mathbf{u}\right) \tag{4.43}$$

where $\mathbf{u} = \left(\mathbf{z} - \sigma\left(\sqrt{w}_1\mathbf{x}^{(1)} + \sqrt{w}_2\mathbf{x}^{(2)}\right)\right)$ and $\mathbf{I}$ is the identity matrix. By expanding Eq: 4.40 the blocks of the precision matrix becomes

$$\begin{aligned} Q_{z,z} &= \tau\mathbf{I} \\ Q_{z,1} &= -\kappa^{-1/2}\sqrt{w_1}\tau\mathbf{I} \\ Q_{z,2} &= -\kappa^{-1/2}\sqrt{w_2}\tau\mathbf{I} \\ Q_{1,1} &= Q_{\text{AR1}(\phi_1)} + \kappa^{-1}w_1\tau\mathbf{I} \\ Q_{1,2} &= \kappa^{-1}\sqrt{w_1}\sqrt{w_2}\tau\mathbf{I} \\ Q_{2,2} &= Q_{\text{AR1}(\phi_2)} + \kappa^{-1}w_2\tau\mathbf{I} \end{aligned}$$

The sparsity of $\mathbf{Q}$ is displayed in Fig: 4.9, where it is observed that most blocks are diagonal matrices, with the exception of the blocks that includes the AR1 precision matrix. Extending the example to any N using the same approach, the blocks of Eq: 4.39 are found to be

$$\begin{aligned}
Q_{z,z} &= \tau\mathbf{I} \\
Q_{z,j} &= -\kappa^{-1/2}\sqrt{w_j}\tau\mathbf{I} \\
Q_{i,i} &= Q_{\text{AR1}(\phi_i)} + \kappa^{-1}w_i\tau\mathbf{I} \\
Q_{i,j} &= \kappa^{-1}\sqrt{w_i}\sqrt{w_j}\tau\mathbf{I},
\end{aligned}$$

where $0 \leq i, j \leq N$. This yields a similar sparsity structure as Fig: 4.9 with diagonal matrices $Q_{z,z}, Q_{z,i}$ and $Q_{i,j}$, and the $Q_{i,i}$s are tridiagonal. The latent field for this model contains more variables compared to the previous one, but the precision matrix now has a very sparse structure that allows INLA to perform much better.

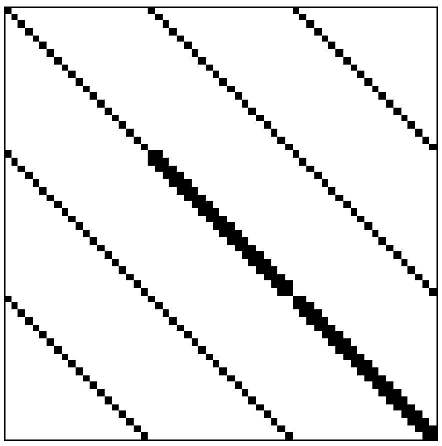

Figure 4.9: The sparsity structure of $\mathbf{Q}$. Block (2,2) and (3,3) are tridiagonal while the other blocks are diagonal matrices.

The parametrization chosen for the internal variables resembles those of section 3.3.2, but it is here assumed that $H \in (0.5, 1)$,

$$\theta_2 = \text{logit}(2H - 1). \tag{4.44}$$

The internal representation of the innovation precision remains the same,

$$\theta_1 = \log(\kappa). \tag{4.45}$$

The prior distributions assigned to the hyperparameters are also the same, but adjusted for the new interval of H

$$\pi(H, \kappa) = \text{Unif}(0.5, 1) \cdot \text{Gamma}(a, b) \tag{4.46}$$

The normalizing constant of $\pi\left(\mathbf{x}^{(1)},\mathbf{x}^{(2)},\mathbf{z}\right)$ is found to be

$$C = (2\pi)^{3n/2}\tau^{n/2}\left(\frac{1}{1-\phi_1^2}\right)^{\frac{n-1}{2}}\left(\frac{1}{1-\phi_2^2}\right)^{\frac{n-1}{2}}, \tag{4.47}$$

whose logarithm is also included in the `rgeneric` definition. Similarly, for a general N the normalizing constant is found similarly to be

$$C = (2\pi)^{(N+1)/2}\tau^{n/2}\left(\frac{1}{1-\phi_1^2}\right)^{\frac{n-1}{2}}\cdots\left(\frac{1}{1-\phi_N^2}\right)^{\frac{n-1}{2}}. \tag{4.48}$$

To reduce the cost of the `rgeneric` function, the interpolated spline functions obtained in section 4.3 is determined outside of the function and then passed on as input, here as object `funks`. The function also requires the number N of AR1 components used in the model and the length n of the FGN data,

```
model = inla.rgeneric.define(rgeneric.general,n=n,N=N,funks=funks).
```

The code associated with the `rgeneric` definition is included in 4.1.

Listing 4.1: `rgeneric` function definition of the disaggregated FGN process.

```
disaggregated.fgn.rgeneric = function(
  cmd = c("graph", "Q","mu", "initial", "log.norm.const",
  "log.prior", "quit"), theta = NULL, args = NULL)
{
  tau = exp(15)

  map = function(H,N,funks) {
    params = numeric(2*N)
    for(i in 1:(2*N)){
      params[i] = funks[[i]](H)
    }
    return(params)
  }
  interpret.theta = function(theta) {
    H = 0.5 + 0.5 / (1 + exp(-theta[2]))
    kappa = exp(theta[1])
    return(list(H = H, kappa = kappa))
  }
  mu = function(n, N,funks,theta) {
    return(numeric(0))
  }
  ar1maker = function(rho, n) {
    tauu = 1 / (1 - rho ^ 2)
    i = c(1L, n, 2L:(n - 1L), 1L:(n - 1L))
    j = c(1L, n, 2L:(n - 1L), 2L:n)
    xx = tauu *
      c(1L, 1L, rep(1 + rho ^ 2, n - 2L), rep(-rho, n - 1L))
    return( sparseMatrix(i = i, j = j, x = xx,
            giveCsparse = FALSE, symmetric = TRUE) )
  }
  graph = function(n, N, funks,theta){
    init = initial(n,N,funks,theta)
    G = Q(n, N,funks,init)
    G[G != 0] = 1
    return (G)
  }
  log.norm.const = function(n,N,funks, theta){
    hyperparams = interpret.theta(theta)
    param = map(hyperparams$H,N,funks)
    sum = n/2*log(tau)
    for (i in 1:N){
      sum = sum -(n-1)/2*log(1-param[i]^2)
    }
    return(sum)
  }
  log.prior = function(n, N,funks,theta){
    params = interpret.theta(theta)
    return(theta[1] + dgamma(params$kappa, 1, 0.01, log = T)
           - theta[2] - 2 * log(1 + exp(theta[2])))
  }
  Q = function(n, N, funks,theta){
     hyperparam = interpret.theta(theta)
     H = hyperparam$H
     kappa = hyperparam$kappa
     param = map(hyperparam$H,N,funks)
     N=length(param)/2
     alphas = param[1:N]
     weights = param[(N+1):(2*N)]
     Q = Diagonal(n, rep(tau, n))
     for (i in 1:N){
       Q = cBind(R1,Diagonal(n, rep(-sqrt(1 / kappa)
            * sqrt(weights[i]) * tau, n)))
     }
     for (i in 2:(N+1)){
       R = numeric(0)
       if(1 <=(i-1)){
         for(j in 1:(i-1)){
           R = cBind(R,Q[(1+(j-1)*n):(j*n),(1+(i-1)*n):(i*n)])
         }
       }
       R = cBind(R,ar1maker(alphas[i-1], n)
            + Diagonal(n, rep(1 / kappa * weights[i-1] * tau, n)) )
       if((N+1)>=(i+1)){
         for (j in (i+1):(N+1)){
           R = cBind(R,Diagonal(n, rep(1 / kappa * sqrt(weights[i-1])
                * sqrt(weights[j-1]) * tau, n)))
         }
       }
       Q = rBind(Q,R)
     }
     return (Q)
  }
  initial = function(n, N,funks,theta){
     return (c(4, 0.))
  }
  quit = function(n, N,funks,theta){
     return ()
  }
  cmd = match.arg(cmd)
  val = do.call(cmd, args = list(
     n = as.integer(args$n), N = as.integer(args$N),
     funks = args$funks, theta = theta))
  return (val)
}
```

Chapter 5

Speed and accuracy analysis

5.1 Time consumption of the AR1 approximation

How the efficiency of INLA with the AR1 approximated latent model depends on the length and Hurst exponent of the data as well as the number of AR1 components is charted by applying the model to simulations where only one of the variables are changing and the others are kept constant. 20 simulations are generated for each step of the variable under consideration from which the mean and 95% confidence intervals are computed.

The time consumption of INLA when using the AR1 approximated model seems to linearly increase in time as the length of the simulated series increases. This behaviour is displayed in Fig: 5.1 when $N = 5$ AR1 components are used and the simulated series are sampled with true Hurst exponent $H = 0.8$.

One can also see a clear trend when the number of AR1 components is varied and the length and Hurst exponent are kept constant. Fig: 5.2 shows the results for simulations with length $n = 700$ and true Hurst exponent $H = 0.8$ and reveals a trend that seems slightly stronger than linear as N increases.

When the length of the simulations and the number of AR1 components are fixated and the Hurst exponent is the only changing variable there does not appear to be any clear trend. The time consumption as a function of H with constant $H = 0.8$ and $N = 5$ is illustrated in Fig: 5.3.

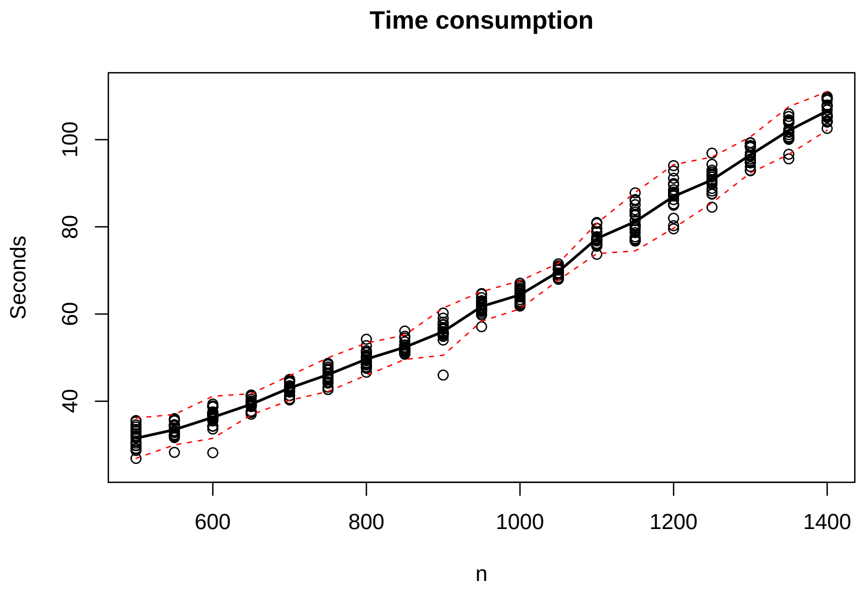

Figure 5.1: The time consumption of the rgeneric function plotted against the length, n of 20 simulated FGN series for fixed $H = 0.8$ and $N = 5$ plotted with the corresponding mean and 95% confidence intervals.

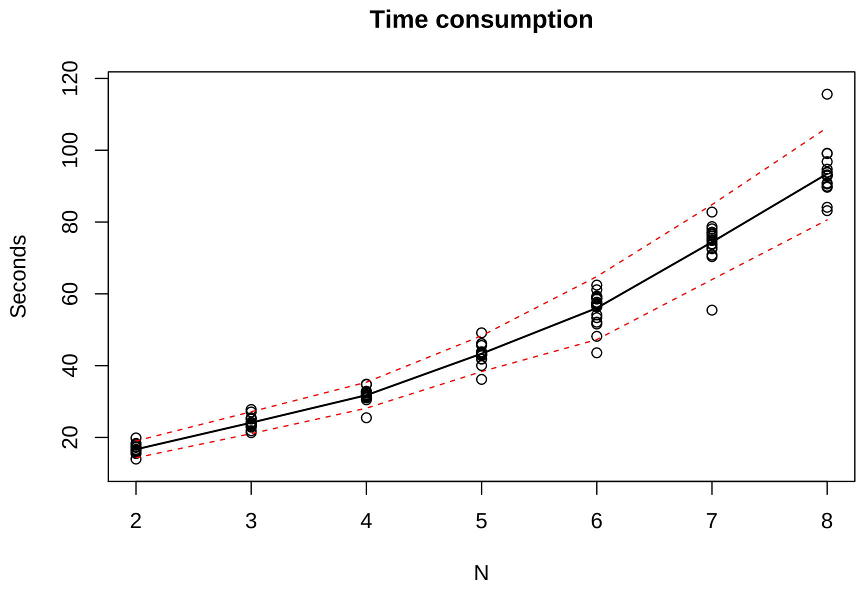

Figure 5.2: The time consumption of the rgeneric function plotted against the number of AR1 components, N of simulated 20 FGN series for fixed $H = 0.8$ and $n = 700$ plotted with the corresponding mean and 95% confidence intervals.

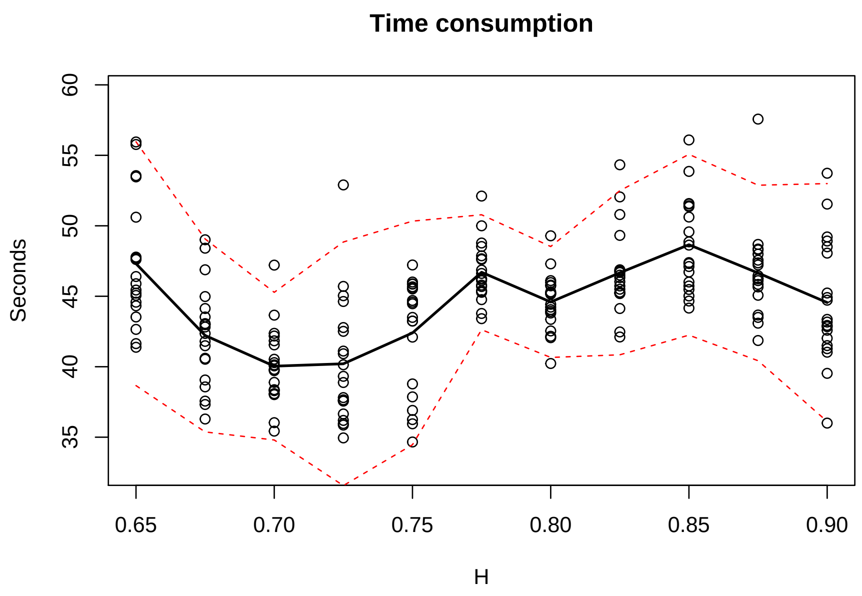

Figure 5.3: The time consumption of the rgeneric function plotted against the Hurst exponent, H of simulated 20 FGN series with fixed $n = 700$ and $N = 5$ plotted with the corresponding mean and 95% confidence intervals.

5.2 Deviation from MLE

A well approximated model should yield a Hurst exponent that approaches the maximum likelihood estimator for H as n and N grows larger. To measure the accuracy, the estimated Hurst exponent obtained from the approximated model is compared with the exact MLE computed from the `FitFGN` function from the R package `FGN`. The deviation from the MLE is examined as a function of the number of AR1 components, the length of the observed FGN and the true Hurst exponent that is to be estimated, and similarly to the time consumption analysis, the procedure is repeated separately with only one variable is changing at a time.

When the length increases with constant Hurst exponent and number of AR1 components, the deviation from the exact MLE reveals consistent overestimation, but since the overestimation remains constant for different lengths it is likely to be caused by one of the other variables. The result of a simulation process with Hurst exponent $H = 0.8$ and $N = 5$ AR1 components is illustrated in Fig: 5.4. 20 replications was computed for each step of the length variable and the associated means and 95% confidence interval is also included in the figure.

The results for both the rgeneric estimation and the exact MLE when the samples are drawn from an increasing Hurst exponent varies with constant length $n = 700$ and with $N = 5$ AR1 components is plotted in Fig: 5.5 together with the MLE and the true value from which the simulations are drawn from. The deviation between the MLE and the rgeneric approximate is illustrated more clearly in Fig: 5.6. Both plots indicate that the results are biased, as the approximated model seems to underestimate the estimation for the Hurst exponent when the true Hurst exponent is low and overestimate if it is larger. The author suggests a more thorough revisit to the optimization procedure in section 4.3 with longer AR1 series and more replications.

To chart the accuracy of the approximated model as the number of AR1 components increases 20 FGN samples were generated and each analyzed using different number of AR1 components. The resulting estimation means from 20 simulations of length $n = 700$ around true Hurst exponent $H = 0.8$ is illustrated in Fig: 5.7 and compares the estimate to the exact MLE. The estimation from approximated model should in theory approach the exact MLE as more components are used, but simulations suggests that the deviation suddenly increase when too many components are used. This is likely due to convergence issues when finding the weights and parameters in section 4.3 with the `optim` function for too many variables. A revisit to the optimization procedure with a more dense discretization of $H \in (0.5, 1)$ is therefore recommended.

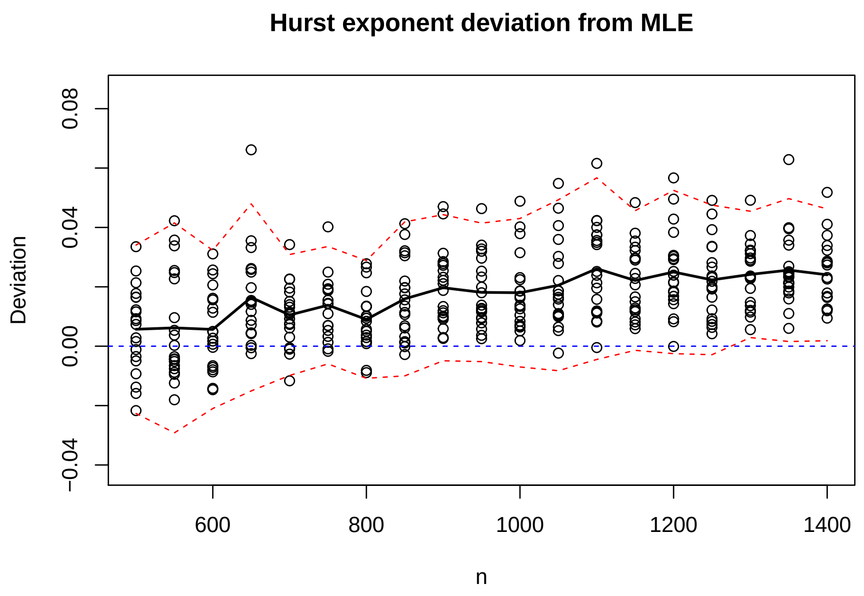

Figure 5.4: The deviation between the MLE and the corresponding $N = 5$ AR1 component rgeneric approximate of 20 simulations of varying length n and fixated Hurst exponent $H = 0.8$. The mean and 95% confidence intervals of the approximations are also included.

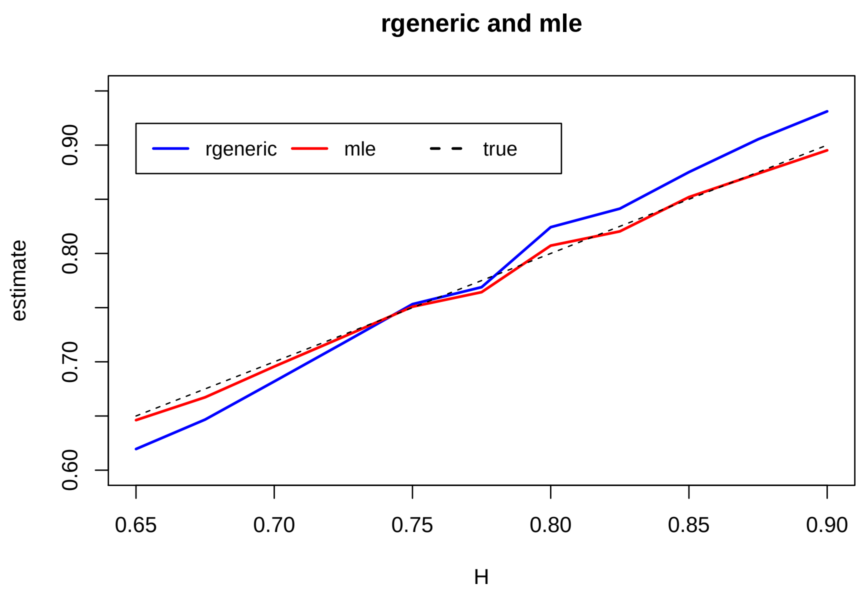

Figure 5.5: The mean of $N = 5$ AR1 component rgeneric approximations for 20 simulations of length $n = 700$ and different values of H plotted with the mean of the corresponding MLE and true Hurst exponent.

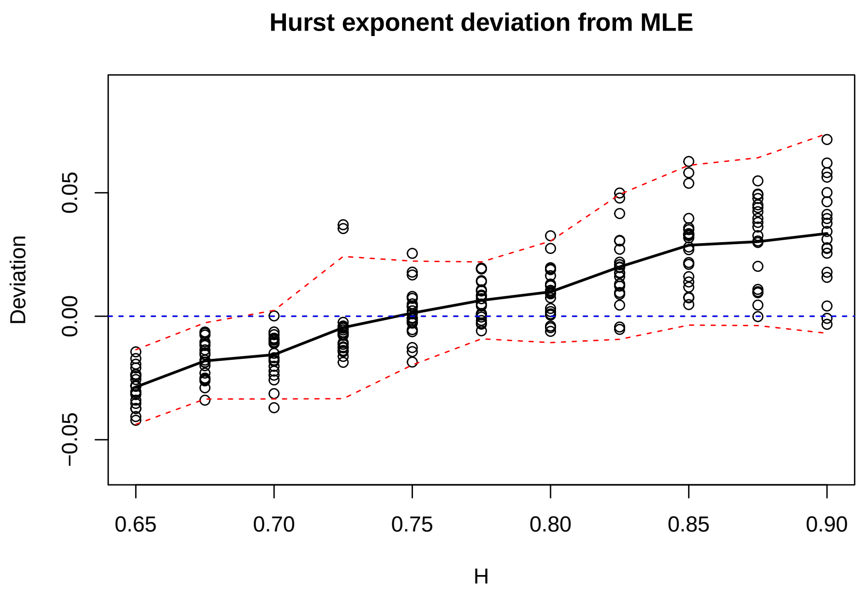

Figure 5.6: The deviation between the MLE and the corresponding $N = 5$ AR1 component rgeneric approximate of 20 simulations of length $n = 700$ and varying Hurst exponent H. The mean and 95% confidence intervals are also included.

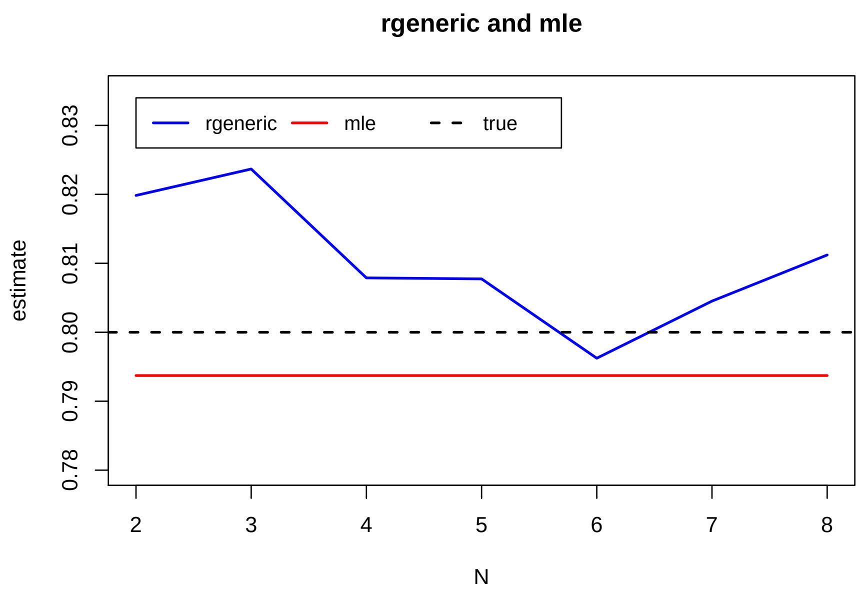

Figure 5.7: The mean of rgeneric approximations with varying number of AR1 components N of the same 20 simulations of length $n = 700$ and true $H = 0.8$ plottet with the mean of the MLE of the simulations.

5.3 Examples

5.3.1 NileMin data

The Nile data from chapter 3 is revisited and analyzed with the new rgeneric model. The marginal posterior distributions for the Hurst exponent and the innovation precision computed with $N = 5$ AR1 components is included in Fig: 5.8 and Fig: 5.9. The modes of the posterior marginal distributions yields

$$\hat{H}_{\text{approx}} = 0.8670 \quad \text{and} \quad \hat{\kappa}_{\text{approx}} = 1.3579 \tag{5.1}$$

which, consistent with the analysis of section 5.2, is an overestimation for both H and κ which was for the original rgeneric model found to be

$$\hat{H}_{\text{INLA}} = 0.8336 \quad \text{and} \quad \hat{\sigma}^2_{\text{INLA}} = 1.2482.$$

The accuracy when increasing the number of AR1 components is illustrated in Fig: 5.10 and Fig: 5.11. The new rgeneric model seems to overestimate regardless of the number of AR1 components, which is consistent with the accuracy analysis of section 5 for Hurst exponents of this size. The speed however, proves to be far superior for the new rgeneric model. The time consumption for different number of AR1 components is included in Fig: 5.12, and shows that the new rgeneric model is far superior to the old in terms of speed for all $N \leq 8$.

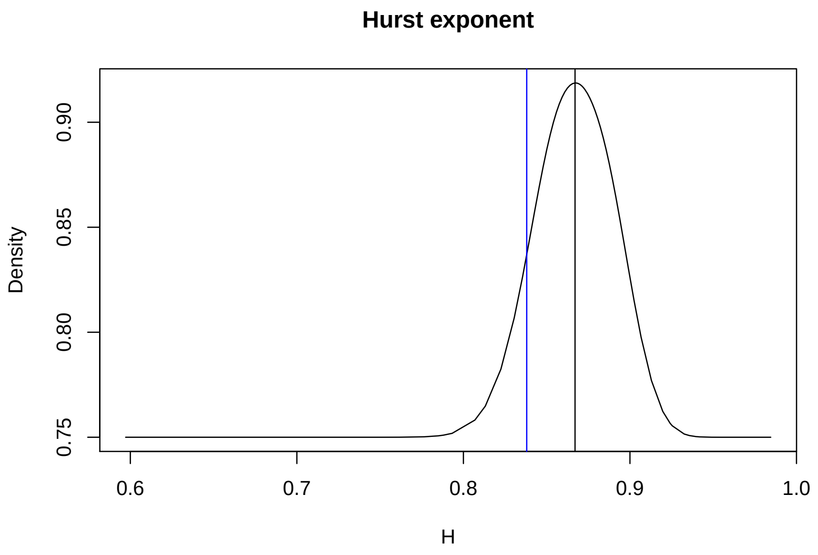

Figure 5.8: NileMin: The posterior marginal distribution of the Hurst exponent generated with the new rgeneric model for $N = 5$ AR1 components. The mode is compared to the corresponding result from the original rgeneric model.

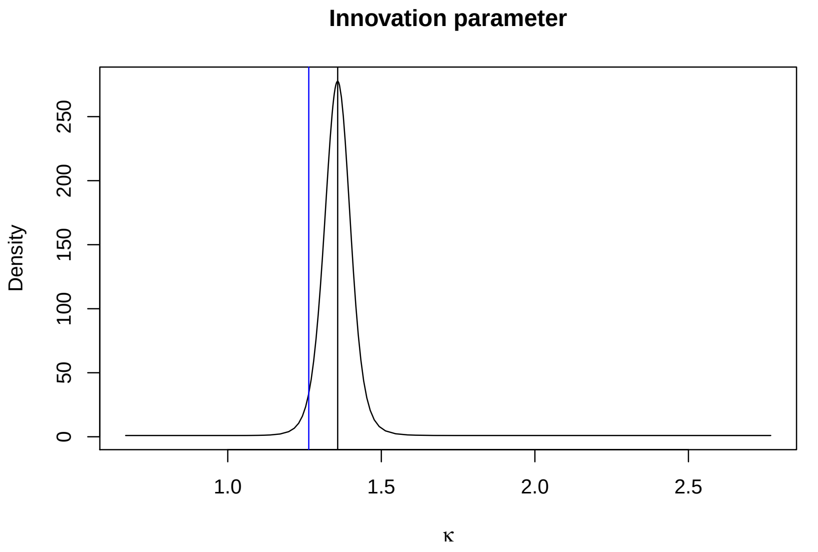

Figure 5.9: NileMin: The posterior marginal distribution of the innovation variance generated with the new rgeneric model for $N = 5$ AR1 components. The mode is compared to the corresponding result from the original rgeneric model.

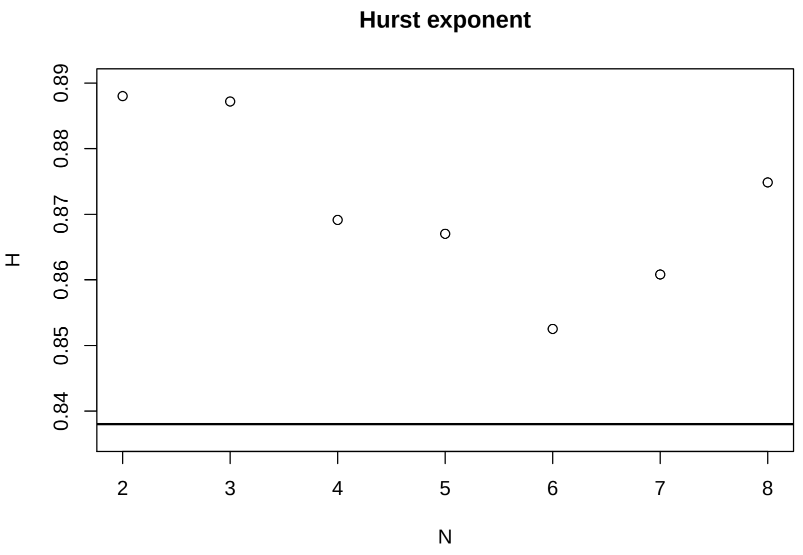

Figure 5.10: NileMin: The computed values for the Hurst exponent when using N AR1 components in the rgeneric model, compared to the corresponding H obtained from the original rgeneric model.

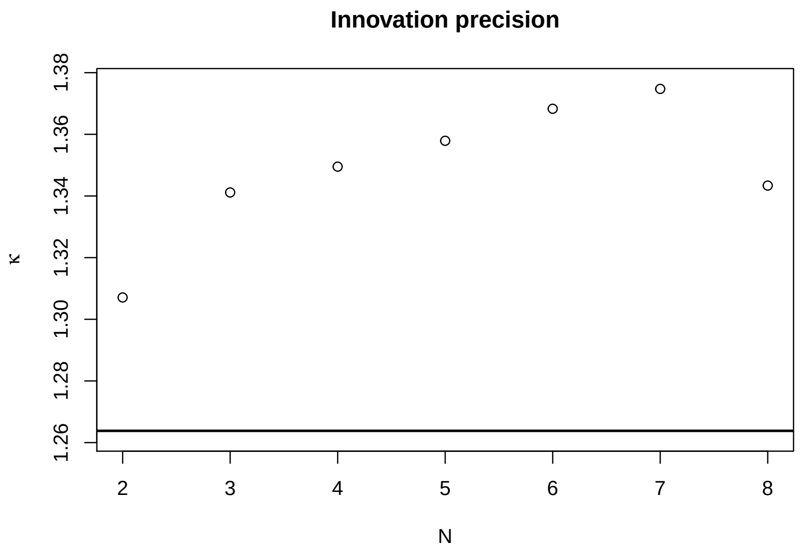

Figure 5.11: NileMin: The computed values for the innovation precision when using N AR1 components in the rgeneric model, compared to the corresponding κ obtained from the original rgeneric model.

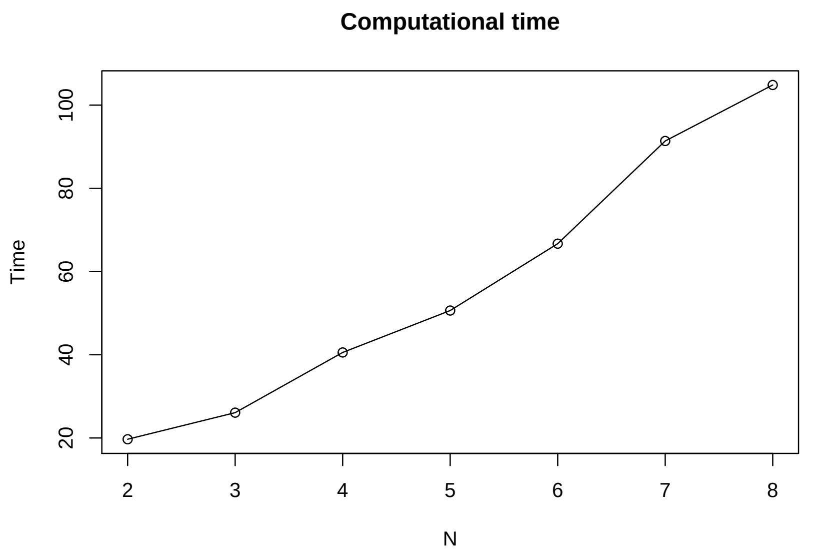

Figure 5.12: NileMin: The time consumption of INLA with the rgeneric model plotted against the number of AR1 components used.

5.3.2 HadCRUT4 data

HadCRUT is a series of datasets of monthly global temperature measurements that combines data of both land and marine temperatures. The dataset containing the sea surface temperature, HadSST3 is compiled by the Hadley Centre of the UK Met Office and the dataset containing land surface air temperature, CRUTEM4 is compiled by the Climatic Research Unit (CRU) of the University of East Anglia.

HadCRUT4 (22) is the most current version and has a more comprehensive error model than the previous versions. HadCRUT4 contains monthly and yearly average of the temperature anomaly computed from the mean temperature value of the period 1961-1990. The HadCRUT4 yearly average temperature anomaly observations for the northern and southern hemisphere as well as the global temperature variations is included in Fig: 5.13 which is available at `https://crudata.uea.ac.uk`.

We wish to analyze the global yearly average temperature. A non-linear trend is observed in the data, so a random walk effect of order 2 is therefore added to the linear predictor describing the observations. The precision hyperparameter of the RW2 model is assigned a Gamma$(1, 0.01)$ prior distribution. The remaining noise is assumed to be an FGN process so an AR1 approximated rgeneric effect is added as well. The formula is constructed as

```
formula = y ~ -1 + f(idy, model = "rw2",scale.model = TRUE,
    hyper = list(prec=list( prior="loggamma",param=c(1,0.01))))
    + f(idx, model=rgenericmodel)
```

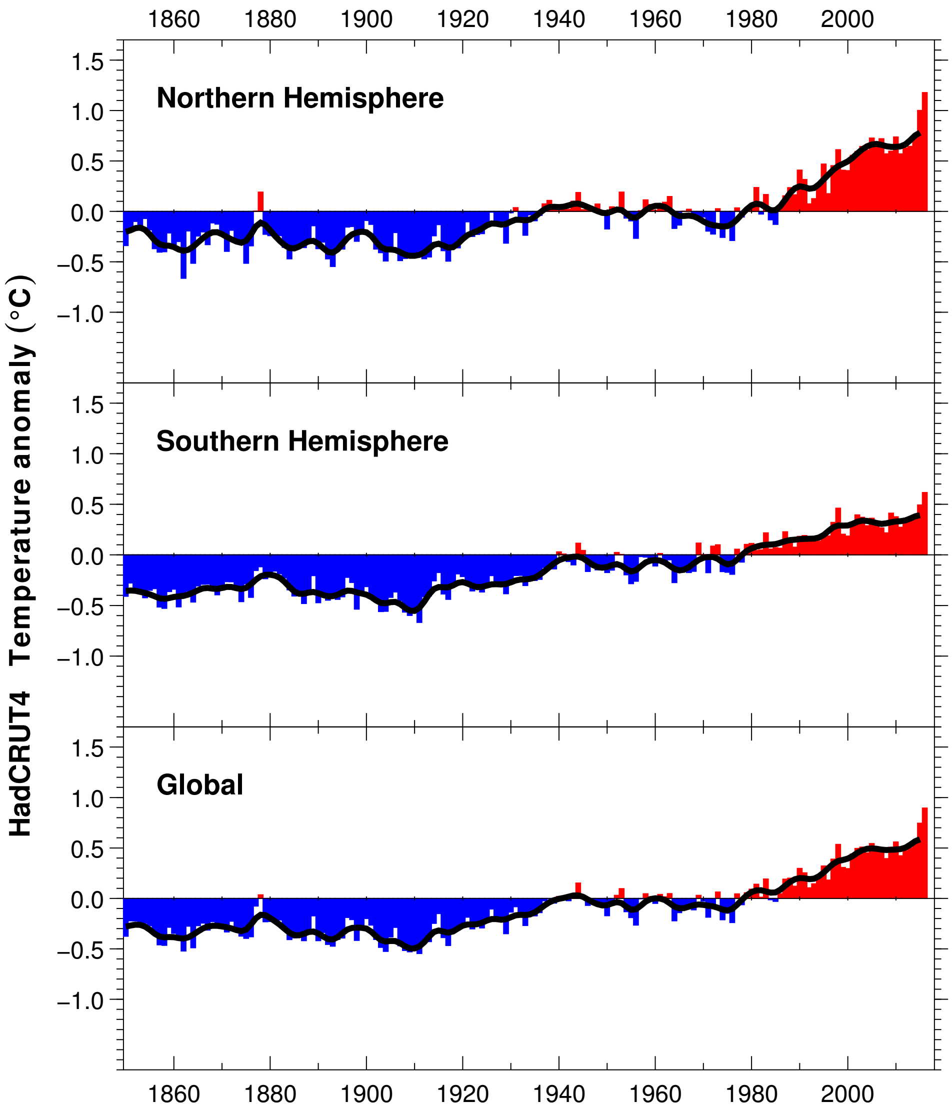

Figure 5.13: HadCRUT4: Contains the yearly average temperature for the northern and southern hemisphere between year 1850 and 2016. The global average is also included. This plot was taken from `https://crudata.uea.ac.uk/cru/data/temperature/HadCRUT4.pdf`.

where the rgeneric model is constructed from the defining function in listing: 4.1. For $N = 4$ components for the rgeneric model the INLA analysis was computed in 9.764 seconds. The mean of the latent variables for the RW2 model is plotted in Fig: 5.14 with corresponding 95% confidence interval included. The RW2 model appears to have captured the trend very well. The rgeneric model of the remaining noise yields a Hurst exponent and innovation variance of $\hat{H} = 0.6914$ and $\hat{\sigma}^2 = 97.613$ respectively. The posterior marginal plots are also included in Fig: 5.15 and Fig: 5.16. The process is repeated for $N = 2, 3, ..., 8$ and the Hurst exponent for each N is displayed in Fig: 5.17.

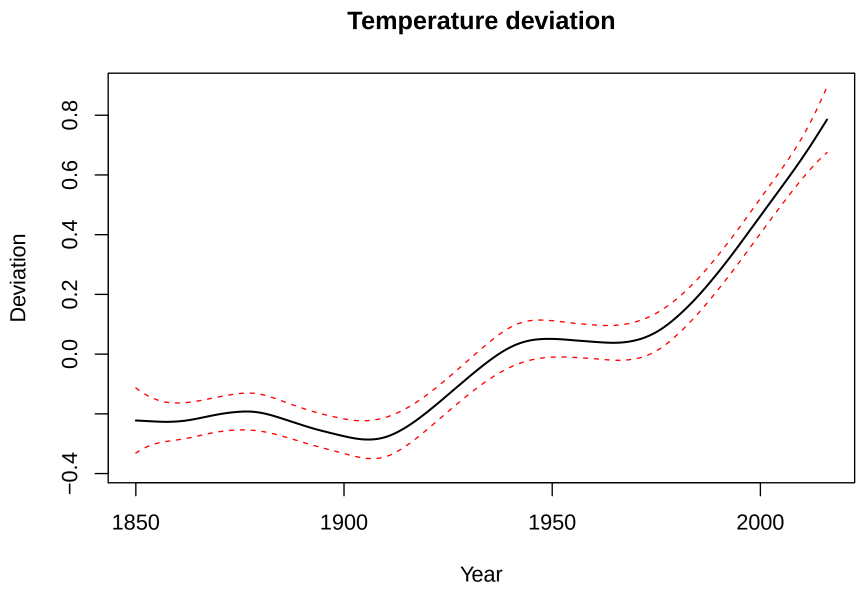

Figure 5.14: HadCRUT4: Posterior marginal means with corresponding 95% confidence interval for the latent RW2 field.

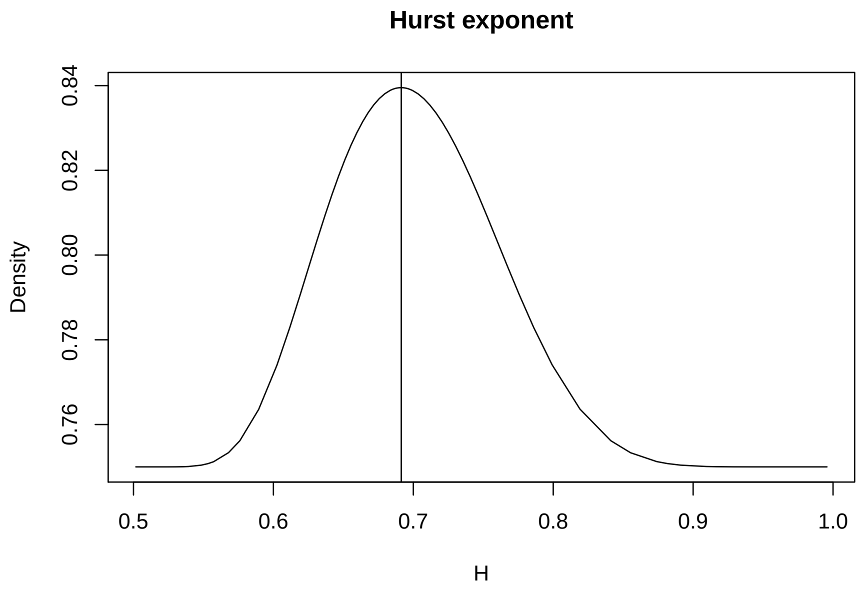

Figure 5.15: Posterior marginals of the Hurst exponent of the RW2 and FGN combined model with the HadCRUT4 dataset.

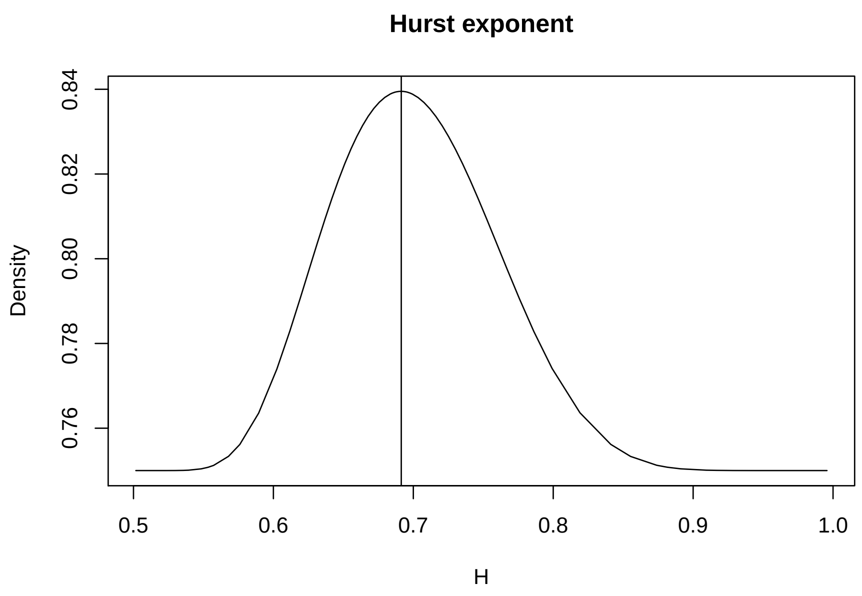

Figure 5.16: Posterior marginals of the innovation variance of the RW2 and FGN combined model with the HadCRUT4 dataset.

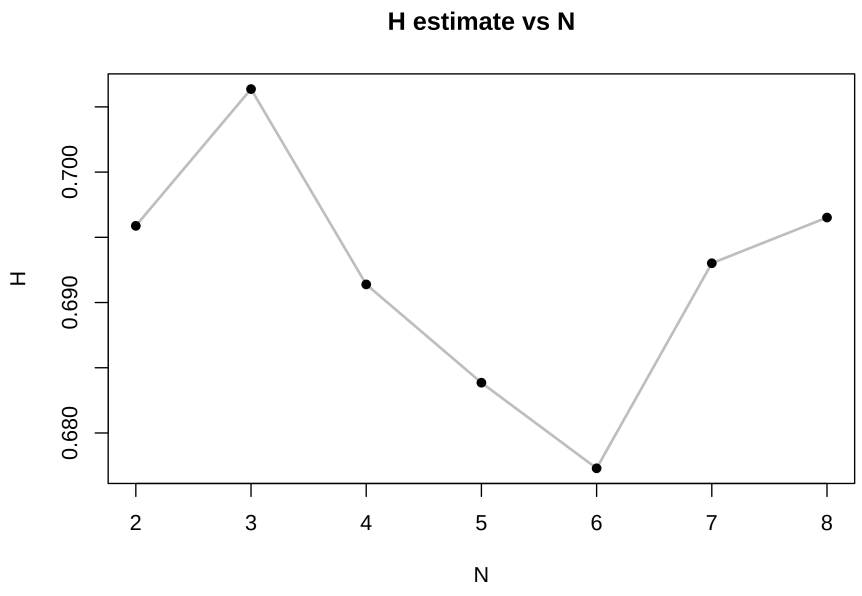

Figure 5.17: HadCRUT4: Hurst estimate for different number of AR1 components.

Chapter 6

Concluding remarks

In this thesis, attempts to create a viable tool for performing Bayesian inference for fractional Gaussian noise have been made. Restating the likelihood of an FGN as a latent field in a latent Gaussian model allows the MLE to be found using the integrated nested Laplace approximation method. However, this proves too slow to be used in practice, especially for longer time series. The efficiency was improved by using numerical optimization methods to find a continuous mapping from $H \in (0.5, 1)$ to a set of corresponding weights $\{w_i\}$ and parameters $\{\phi_i\}$ such that the sum of N AR1 series given by

$$\bar{x}_t = \sum_{i=1}^{N} \sqrt{w_i} \sqrt{1 - \phi_i^2}\, x_{i,t}, \tag{6.1}$$

yields the MLE $\hat{H} = H$. The latent field of the latent Gaussian model is then replaced to contain all AR1 series and the weighted sum, yielding a sparse precision matrix that significantly improves the computational efficiency of the INLA method. The mapping and the new latent field allows INLA to find the H whose corresponding weights and parameters best fit the data.

There are two primary benefits of this approach. First, it reduces the runtime of INLA to be linear with respect to the length of the FGN series, making Bayesian inference easily obtainable even for long FGN series. Second, it is compatible with the R-INLA framework which makes it possible to combine the FGN term with other terms in the linear predictor, like seasonal effects, non-linear effects of covariates etc.

The speed and accuracy of the new approach was thoroughly tested, and while the speed displayed a remarkable improvement, the method proved unsuccessful in achieving sufficient accuracy. The analysis revealed bias in the estimation results for the Hurst exponent, underestimating if the estimated Hurst exponent was small, and overestimating if it was high.

The inaccuracy of the model is likely to stem from the restrictions imposed on the optimization problem of section 4.3 to ensure a reasonable convergence rate. Due to technical limitations and time constrictions, the length of the sampled of AR1 processes was only $n = 1,000$. Also, the mean of the MLE was only computed from $R = 10$ MLE replications. Increasing either the length of the AR1 processes or the number of replications for the MLE should improve the approximation. Also, the accuracy can be further improved if a more dense discretization can be chosen to represent the domain of the Hurst exponent, $H \in (0.5, 1)$.

Despite the inaccuracy, the AR1 approximation approach does seem promising. Implementing support for long-memory processes and FGN series in particular, provides a new computational Bayesian framework to analyze long-memory processes, both in a temporal and spatial context. This methodology will have a wide range of applications, among others, in analyzing climate data. The results of this thesis should be perceived more as a 'proof-of-concept' rather than a complete and rigorous procedure as there is still work to be done for this method to be a viable tool for FGN analysis.

Appendix A

rgeneric tutorial

`rgeneric` is a model in INLA that aims to construct any latent Gaussian model compatible with INLA. It allows the user to define custom latent models that are not yet implemented in INLA either as a built-in model templates or from one of the simpler generic functions so that it can be included in the linear predictor. It was previously only available for Linux and Mac platforms, but support for Windows has also been introduced. The properties of the rgeneric model is evaluated in an external C-program which communicates with R by pipes. This will result in a somewhat slower performance compared to the more traditional models in INLA.

To use this model we first define an rgeneric function which includes our definition of the model, this will then be used with the `inla.rgeneric.define` function to create an rgeneric model compatible with INLA. For a simple rgeneric function called 'my.rgeneric' the model is obtained from

```
model = inla.rgeneric.define(my.rgeneric, <args>),
```

where <args> are parameters of the rgeneric function, for instance the dimension size or the number of parameters. The model is then passed on as any other model in INLA

```
formula = y ~ f(x, model = model).
```

A.1 rgeneric function composition

The rgeneric function must contain the functions that each return necessary information depending on the hyperparameters. These functions are

- `graph`, which returns the graph of the latent field
- `Q`, which returns the precision matrix of the latent field given the hyperparameters
- `mu`, which returnts the mean of the latent field
- `log.norm.const`, which returns the logarithm of the normalizing constant of the latent field
- `log.prior`, which returns the logarithm of the joint prior of the internal hyperparameters
- `initial`, which returns the initial values for the internal hyperparameters

Internal hyperparameters refers to the hyperparameters of the model in its internal scaling. It is important to distinct between the internal and external scaling for the hyperparameters as the most efficient operating space might differ from the space in which the hyperparameters are most familiar. It is important that the internal scaling allows INLA to operate in a stable and unconstrained parameter space on $\mathbb{R}$. It is often useful to create a function that transforms the hyperparameters from the internal-scale to the user-scale. As an example, consider a model where the precision $\kappa > 0$ is the only hyperparameter, e.g. a random walk model. The common internal scaling chosen for the precision is the log-scale, and it can be returned to the usual scaling by equation A.1.

$$\kappa = \exp(\theta) \tag{A.1}$$

A function that returns the internal hyperparameter to the original scale according to Equation A.1 is included in listing: A.1.

Listing A.1: Function that returns the precision parameter from log-scale.

```
interpret.theta = function(theta) #theta is a list
{
    return( kappa = exp(theta[1]) )
}
```

The complete skeleton-function for the rgeneric function is provided in listing: A.2. The functions that make up `cmd` must be implemented and are explained in greater detail below.

Listing A.2: R skeleton function of the my.rgeneric.function.

```
my.rgeneric.function = function(
    cmd = c("graph", "Q", "mu", "log.norm.const",
    "log.prior", "initial", "quit"),
    theta = NULL, args = NULL )
    {
        graph = function(n, theta){ ... }
        Q = function(n, theta){ ... }
        mu = function(n, theta){ ... }
        log.norm.const = function(n, theta){ ... }
        log.prior = function(n, theta){ ... }
        initial = function(n, theta){ ... }
        quit = function(n, theta){ ... }

        cmd = match.arg(cmd)
        val = do.call(cmd, args = list(n = as.integer(args$n),
            theta = theta))
        return (val)
    }
```

`graph`

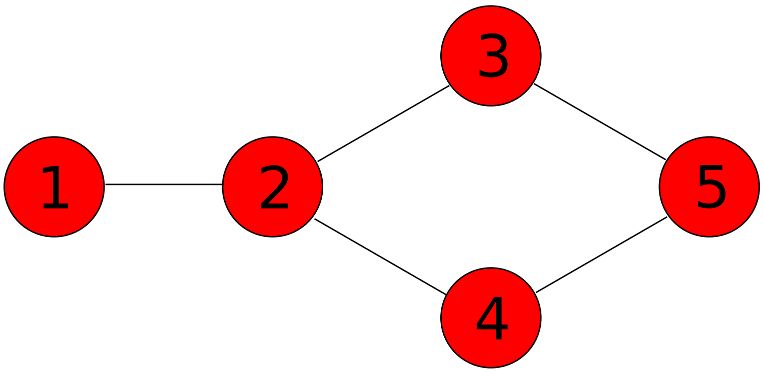

Figure A.1: Graph of a latent GMRF.

The `graph` function must return a matrix representing the graph of the latent GMRF. This information does not depend on the hyperparameters. These matrices are symmetric for all purposes and INLA requires only the lower triangular to be implemented. Consider for example, the graph shown in Fig: A.1. The corresponding matrix to be implemented is

$$\mathcal{G} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

`Q`

This function returns the precision matrix, $Q(\boldsymbol{\theta})$ and tells us how each latent variable depend on each other. As for the `graph` function this matrix is returned as a matrix with the lower triangular included. To show this with an example, consider the 1st order random walk model with precision matrix:

$$Q = \kappa \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix}.$$

The precision parameter, κ is extracted from its internal-scale by the `interpret.theta` function defined in listing: A.1. The function setting up this matrix is implemented in listing: A.3.

Listing A.3: `Q` function for the RW1 model. function

```
Q = function(n, theta)
    {
      params = interpret.theta(theta)
      kappa = params$kappa
      i = c(1:n,1:(n-1) ,1)
      j = c(1:n,   2:n   ,n)
      x = kappa * c(rep(2,n),rep(-1,n-1), -1 )

      Q = sparseMatrix(i=i , j=j , x=x, giveCsparse=F, symmetric=T)
    }
    return (Q)
  }
```

`mu`

The `mu` function returns the mean vector of the model. For most cases, the mean is zero, but models with non-zero mean values that can also depend on the hyperparameters are also supported by rgeneric. To return a zero mean one could return `numeric(0)` as is shown in listing: A.4, or equivalently `rep(0,n)`. An example with this function is included in section: A.2.3.

Listing A.4: `mu` function for a zero mean model.

```
mu = function(n,theta)
{
    return( numeric(0) )
}
```

`log.norm.const`

This function returns the logarithm of the normalizing constant. If the precision matrix is not improper, i.e. of full rank, we can make INLA compute the normalizing constant rather than providing that information here. This is done by letting this function return `numeric(0)`. For Intrinsic GMRFs however, we are required to provide the log normalizing constant for the purpose of providing information about the rank deficiency of the precision matrix. It is of particular importance to include how the normalizing constant depends on the hyperparameters, but if one wishes to compute the marginal likelihood one needs to add any constant terms as well.

As an example we consider the normalizing constant of the iid model,

$$(2\pi)^{-\frac{n}{2}}|Q|^{1/2}\exp\left(-\frac{1}{2}\mathbf{x}^\top\mathbf{Q}\mathbf{x}\right). \tag{A.2}$$

The precision matrix is here a diagonal matrix with the precision hyperparameter, κ along its diagonal. As we are only concerned with the terms depending on the hyperparameter we neglect the constant term, $(2\pi)^{-\frac{n}{2}}$ and include only the determinant $|Q|^{1/2} = \kappa^{n/2}$. The transformation between the internal-scale and the user-scale for the iid model coincides with that of the RW1 model. We can therefore reuse the function for extracting this parameter that we implemented in listing: A.1. The implementation of `log.norm.const` is included below in listing: A.5.

Listing A.5: `log.norm.const` function to obtain the log normalizing constant for the internal hyperparameter.

```
log.norm.const = function(n,theta)
{
    params = interpret.theta(theta)
    kappa = params$kappa
    return( n/2*log(kappa) )
}
```

`log.prior`

This function returns the logarithm of the joint prior distribution assigned to the internal hyperparameters. If there are no hyperparameters the function must return `numeric(0)`. It is important that the prior is assigned to the hyperparameters in internal scaling and proper transformation techniques must be applied if the prior is instead provided for the hyperparameter(s) in external scaling. Consider the case of a single hyperparameter, the precision κ. If we have assigned a $\pi_\kappa(\kappa)$ prior to the precision κ, then the change-of-variable formula yields the prior for $\theta = \log(\kappa)$,

$$\pi_\theta(\theta) = \pi_\kappa(e^\theta)e^\theta. \tag{A.3}$$

The logarithm of $\pi_\theta(\theta)$ is then returned. An example with more than one hyperparameter can be found in Section A.2.5 where the AR1 model is implemented.

`initial`

This function returns the initial values for the internal hyperparameters, θ. If there are no hyperparameters this function must return `numeric(0)`. An example with two hyperparameters with initial values 1 and 2 is included in listing A.6.

Listing A.6: An `initial` function that sets the initial values for both hyperparameters to 1.

```
initial = function(n,theta)
{
    return( rep(1,2) )
}
```

`quit`

This functions explains what to do after all computations are done before closing the C-program. If there are nothing to be done here (and there usually isn't) this function returns nothing.

Listing A.7: The trivial `quit` function.

```
quit = function(n,theta)
{
    return (invisible())
}
```

A.2 Examples

A.2.1 iid model

For the iid model all variables in the latent field are independent. This leads to a diagonal graph and precision matrix, as well as a very simple joint density,

$$\pi(\mathbf{x} \mid \tau) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} \sqrt{\tau} \exp\left(\frac{\tau}{2} x_i^2\right). \tag{A.4}$$

Eq: A.4 yields the log normalizing constant to be

$$\log C = \frac{n}{2} \log(\tau) + \text{const.} \tag{A.5}$$

We assume for the following example, a gamma distribution for the latent hyperparameter τ. The following implementation is written by Håvard Rue and can be found in the current documentation for the rgeneric model.

Listing A.8: The rgeneric function for an iid model.

```
inla.rgeneric.iid.model = function(cmd = c("graph", "Q", "mu",
                  "initial", "log.norm.const", "log.prior", "quit"),
                  theta = NULL, args = NULL)
{
  interpret.theta = function(n, theta){
    return (list(prec = exp(theta[1L])))
  }
  graph = function(n, theta){
    G = Diagonal(n, x= rep(1, n))
    return (G)
  }
  Q = function(n, theta){
    prec = interpret.theta(n, theta)$prec
    Q = Diagonal(n, x= rep(prec, n))
    return (Q)
  }
  mu = function(n,theta){
    return(numeric(0))
  }
  log.norm.const = function(n, theta){
    prec = interpret.theta(n, theta)$prec
    val = sum(dnorm(rep(0, n), sd = 1/sqrt(prec), log=TRUE))
    return (val)
  }
  log.prior = function(n, theta){
    prec = interpret.theta(n, theta)$prec
    val = dgamma(prec, shape = 1, rate = 0.01, log=TRUE) + theta[1L]
    return (val)
  }
  initial = function(n, theta){
    return (1)
  }
  quit = function(n, theta){
    return (invisible())
  }
  cmd = match.arg(cmd)
  val = do.call(cmd, args=list(n=as.integer(args$n), theta=theta))
  return (val)
}
```

We now test this implementation on the example from the documentation of the iid model in INLA. We observe realizations from a binomial distribution

$$y \sim \text{binom}(n, p). \tag{A.6}$$

The probability of success parameter p is given through the logit link by the latent hyperparameter, η

$$p = \frac{e^{\eta}}{1 + e^{\eta}} \tag{A.7}$$

with each η drawn from a zero-mean Gaussian distribution standard deviation of 0.5. We are interested in η and perform analysis with INLA using an rgeneric model. The R code associated with the example follows, and the result is included in Fig: A.2.

```
n=30
Ntrials = sample(c(80:100), size=n, replace=TRUE)
eta = rnorm(n,0,0.5)
prob = exp(eta)/(1 + exp(eta))
y = rbinom(n, size=Ntrials, prob = prob)
data=data.frame(y=y,z=1:n)

model = inla.rgeneric.define(inla.rgeneric.iid.model,n=n,ntheta=1)
formula = y ~ f(z, model = model)
result=inla(formula,data=data,family="binomial",Ntrials=Ntrials,
            control.family = list(link="logit") )
```

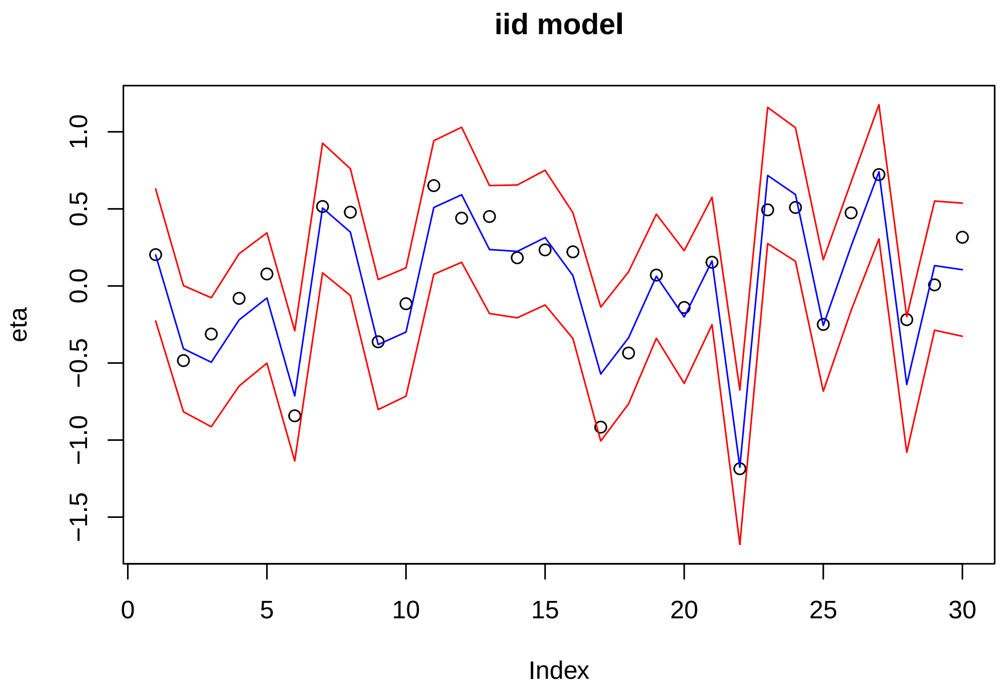

Figure A.2: Plot of the realizations of iid η, observed by binomial realizations through a logit link.

A.2.2 Linear model

The linear model is the simplest of all models supported by INLA. The model aims to specify a fixed linear effect

$$\eta = \beta z_i \tag{A.8}$$

where β is assigned a normal distribution with constant mean μ and precision τ. For this model β is considered the only latent variable, and there are no hyperparameters since μ and τ are deterministic. The function defining the linear model follows for a $\mathcal{N}(0, 1)$ prior for β follows.

Listing A.9: Implementation of the rgeneric function for the linear example.

```
    my.rlinear = function(
    cmd = c("graph", "Q", "mu", "initial",
            "log.norm.const", "log.prior", "quit"),
    theta = NULL, args = NULL)
{
    graph = function(n){ return (Diagonal(1,x=1) }
    Q = function(n){ return (Diagonal(1, x=1)) }
    mu = function(n){ return(numeric(0)) }
    log.norm.const = function(n){
        return (dnorm(0, sd = 1, log = TRUE))
    }
    log.prior = function(n){ return(numeric(0)) }
    initial = function(n){ return(numeric(0)) }
    quit = function(n){ return (invisible()) }
    cmd = match.arg(cmd)
    val = do.call(cmd, args = args)
    return (val)
}
```

The model is used to perform a linear fit of data drawn from a uniform distribution, observed with Gaussian noise with $\sigma = 0.1$. The call is included below, and a plot of the result is included in Fig: A.3.

```
n = 50
z = runif(n)
y = z + rnorm(n, sd = 0.1)
data = data.frame(y,z, idx = rep(1, n))

model = inla.rgeneric.define(my.rlinear,n=n)
formula = y ~ f(idx, z, model = model)
result = inla(formula,data=data,family="gaussian")
```

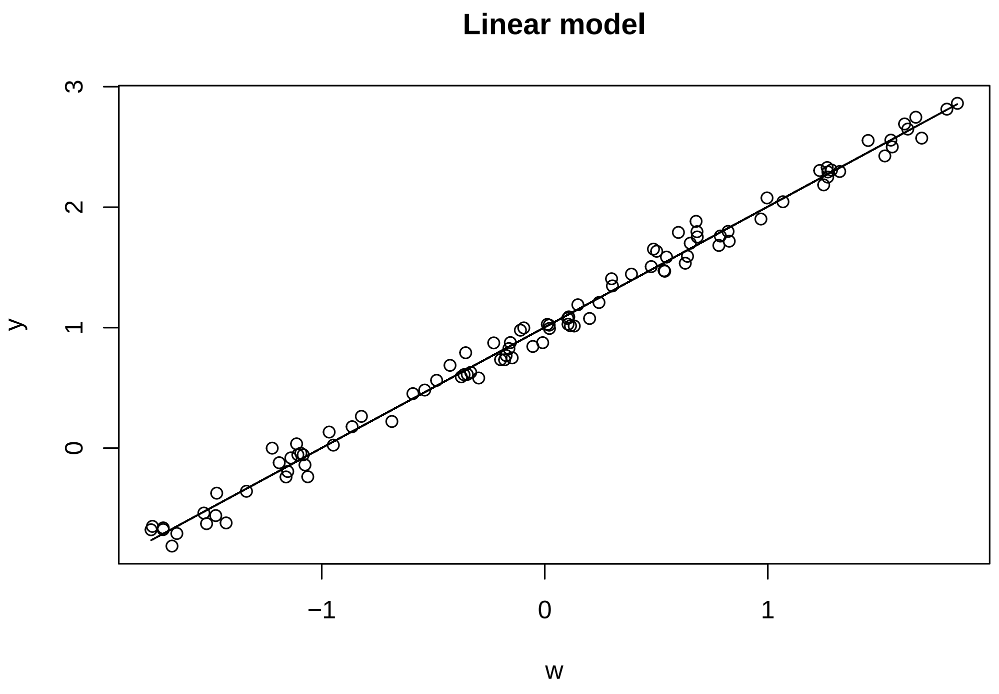

Figure A.3: Plot of linear fit against observations with Gaussian noise.

A.2.3 Non-zero mean model

As an alternative to model a behaviour in terms of complicated links, graphs and precision matrices, one can implement a model where the mean is selected to express the desired behaviour directly. For this example, define the model for the latent field

$$\mathbf{x} \sim \mathcal{N}\big(\boldsymbol{\mu}(\mathbf{x} \mid \theta), \tau^{-1}\mathbf{I}\big), \tag{A.9}$$

where the precision τ is a known large constant and

$$\boldsymbol{\mu}(x \mid \theta) = \begin{cases} 0, & 0 \le x \le 10 \\ 1 - e^{\theta x}, & x > 10 \end{cases}$$

is the mean of the model. The mean depends on the covariate x and the hyperparameter θ. The latent field models the behaviour of function $1 - e^{\theta x}$ starting at $z = 11$ with small noise of precision τ. There is no dependence between the latent parameters. For a mean model with precision $\kappa = 1000$ and a Gamma$(1, 0.01)$ the rgeneric function has been implemented and included in listing: A.10.

Listing A.10: Implementation of the rgeneric function modelled after $1 - e^{\theta z}$ as a mean model.

```
inla.rgeneric.mu.model = function(
        cmd = c("graph", "Q", "mu", "initial",
        "log.norm.const", "log.prior", "quit"),
        theta = NULL, args = NULL)
{
  interpret.theta = function(theta){
    return ( list( prec = exp(theta[1L]) ) )
  }
  graph = function(n, theta){
    return (Diagonal(n,x = rep(1,n)))
  }
  Q = function(n, theta){
    prec = 1000#Precision
    Q = Diagonal(n, x= rep(prec, n))
    return (Q)
  }
  mu = function(n,theta){
      prec = interpret.theta(theta)
      split = n%/%3
      z1 = rep(0,split)
      z2 = seq(-0.5,0,length.out = n-split)
      z = c(z1,z2)
      means0 = rep(0,split)
      means1 = 1-exp(z2*theta)
      means = c(means0,means1)
      stopifnot(length(means)==n)
      return(means)
  }
  log.norm.const = function(n, theta){ return (numeric(0)) }
  log.prior = function(n, theta){
    prec = interpret.theta(theta)
    val = dgamma(prec, shape = 1, rate = 0.01, log=TRUE) + theta[1L]
    return (val)
  }
  initial = function(n, theta){ return (rep(1, 1)) }
  quit = function(n, theta){ return (invisible()) }
  cmd = match.arg(cmd)
  val = do.call(cmd, args =
          list(n = as.integer(args$n), theta = theta))
  return (val)
}
```

This model is used to fit a sample of $n = 30$ observations drawn according to Eq: A.9. The results are included in figure A.4 with 95% confidence intervals included.

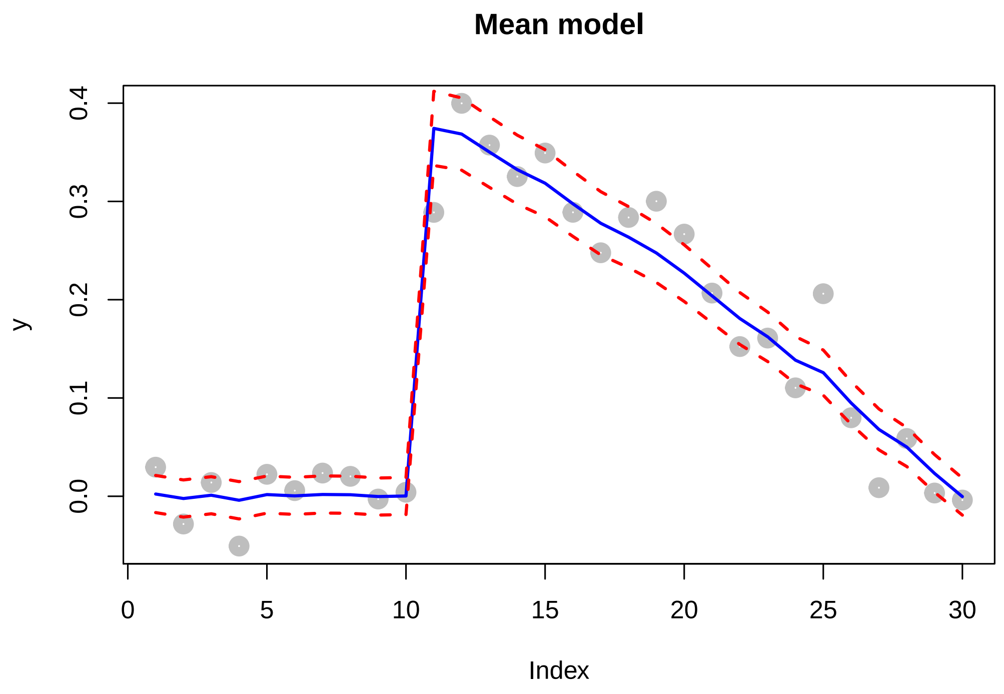

Figure A.4: Plot of the marginal means with observations and 95% confidence interval included.

A.2.4 Cyclic 2nd order random walk

The cyclic RW2 model is a model of a single hyperparameter and a rank deficiency of 1. The example here will revolve around analyzing the famous Tokyo Rainfall data (16) where we will assume a $\mathcal{G}(1, 0.0001)$ prior distribution for the precision parameter κ. Before implementing the rgeneric function to define the model some key values are stated. The cyclic RW2 model has graph and precision matrix is defined as

$$\mathcal{G} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & \cdots & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & \cdots & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & \cdots & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 1 & 0 & 0 & \dots & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & \dots & 0 & 1 & 1 & 1 \end{bmatrix}, \quad \mathbf{Q} = \kappa \begin{bmatrix} 6 & -4 & 1 & 0 & \cdots & 1 & -4 \\ -4 & 6 & -4 & 1 & \cdots & 0 & 1 \\ 1 & -4 & 6 & -4 & \cdots & 0 & 0 \\ 0 & 1 & -4 & 6 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 1 & 0 & 0 & \dots & -4 & 6 & -4 \\ -4 & 1 & 0 & \dots & 1 & -4 & 6 \end{bmatrix}.$$

The log normalizing constant can be shown to be

$$\ln C = \frac{n-1}{2}\ln(\kappa) + \text{const}, \tag{A.10}$$

where the constant term is ignored in the implementation of the rgeneric function which follows in listing: A.11.

Listing A.11: Implementation of the rgeneric function for the Tokyo Rainfall data under the RW2C model assumptions.

```
my.rgeneric = function(
cmd = c("graph", "Q", "initial", "mu","log.norm.const",
        "log.prior", "quit"),
theta = NULL, args = NULL)
{
  interpret.theta = function(theta){ return( exp(theta[1]) ) }
  mu = function(n,theta){ return(numeric(0)) }
  graph = function(n,  theta){
    G = Q(n, N,funks,initial(n,N,funks,theta))
    G[G != 0] = 1
    return (G)
  }
  Q = function(n, theta){
    kappa = interpret.theta(theta)
    i = c(1:n,1:(n-1),1:(n-2), 1  ,1,2)
    j = c(1:n,   2:n   ,  3:n   ,n-1,n,n)
    x = kappa * c(rep(6,n),rep(-4,n-1),rep(1,n-2), 1,-4,1 )
    Q = sparseMatrix(i=i,j=j,x=x, giveCsparse=F,symmetric=T)
    return (Q)
  }
  log.norm.const = function(n, theta){
    kappa = interpret.theta(theta)
    return( -(n-1)/2*log(2*pi) + (n-1)/2*log(kappa) )
  }
  log.prior = function(n, theta){
    kappa = interpret.theta(theta)
    return(dgamma(kappa,shape=1, rate=0.0001,log=T) + log(kappa))
  }
  initial = function(n, theta){ return (rep(1, 1)) }
  quit = function(n, theta){ return () }
  cmd = match.arg(cmd)
  val = do.call(cmd, args = list(
  n = as.integer(args$n),
  theta = theta))
  return (val)
}
```

The rgeneric model is used to fit the binary Tokyo Rainfall data, where a probit link has been used between the probability parameter and the latent field. A graph of the results are included in figure A.5, and the associated call is included below.

```
data("Tokyo")
time = Tokyo$time

model = inla.rgeneric.define(my.rgeneric,n=366)
formula = y ~ -1 + f(time, model = model)
result = inla(formula,family="binomial",
    Ntrials=Tokyo$n,data=Tokyo,
    control.family=list(link="probit")  )
```

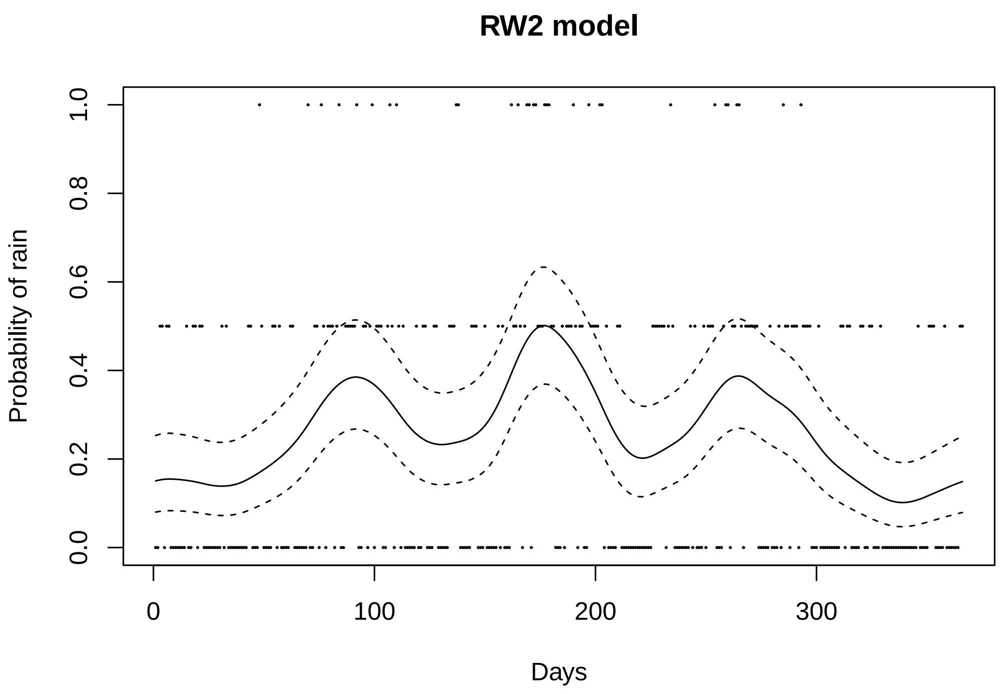

Figure A.5: Plot of the mean marginals computed by INLA using the rgeneric model. The 95% confidence intervals are also included along with observations.

A.2.5 Autoregressive model of order 1

To study a model with more than one hyperparameter we consider the AR1 model. The marginal precision τ and the lag-one correlation ρ is transformed from the internal-scale by the equations A.11 and A.12.

$$\tau = \exp(\theta_1) \tag{A.11}$$

$$\rho = 2\frac{\exp(\theta_2)}{1+\exp(\theta_2)} - 1. \tag{A.12}$$

Further, the graph and precision matrix is defined as

$$\mathcal{G} = \begin{bmatrix} 1 & 1 & & & & \\ 1 & 1 & 1 & & & \\ & 1 & 1 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 1 & 1 \\ & & & & 1 & 1 \end{bmatrix}, \quad \mathbf{Q} = \kappa \begin{bmatrix} 1 & -\rho & & & & \\ -\rho & 1+\rho^2 & -\rho & & & \\ & -\rho & 1+\rho^2 & -\rho & & \\ & & \ddots & \ddots & \ddots & \\ & & & -\rho & 1+\rho^2 & -\rho \\ & & & & -\rho & 1 \end{bmatrix}.$$

The log normalizing constant is here

$$\log C = \frac{n}{2}\log(\tau_I) + \frac{1}{2}\log(1-\rho^2) \tag{A.13}$$

where τ_I is the innovation precision, defined as $\tau_I = \tau/(1-\rho^2)$. The precision is assigned a Gamma(a, b) prior whilst the θ_2 parameter is assigned a $\mathcal{N}(\mu, \kappa)$ prior. As the Gamma prior is not scaled to fit the internal-scale for the hyperparameter we use the change-og-variable formula to obtain the prior for θ_1. The joint prior, $\pi(\theta = (\theta_1, \theta_2))$ becomes

$$\pi(\theta) = \text{Gamma}(\exp(\theta_1)\ ;\ a, b)\exp(\theta_1) \times \mathcal{N}(\theta_2\ ;\ \mu, \kappa). \tag{A.14}$$

For this example, we use $a = b = 1, \mu = 0$ and $\kappa = 1$. The rgeneric definition follows in listing A.12 and is a trimmed version of the example found in the INLA documentation for the AR1 model.

Listing A.12: Implementation of the rgeneric function for an AR1 example.

```
my.rgeneric = function(cmd = c("graph", "Q", "mu", "initial",
        "log.norm.const", "log.prior", "quit"),
        theta = NULL, args = NULL)
{
  interpret.theta = function(n, theta){
    return (list(prec = exp(theta[1L]),
                 rho = 2*exp(theta[2L])/(1+exp(theta[2L])) - 1.0))
  }
  graph = function(n, theta){
    i = c(1L:(n-1L))
    j = c(1L, n, 2L:(n-1L),2L:n)= 1
    G = sparseMatrix(i=i, j=j, x=x, giveCsparse = FALSE)
    return(G)
  }
  Q = function(n, theta){
    param = interpret.theta(n, theta)
    i = c(1L, n, 2L:(n-1L),1L:(n-1L))
      j = c(1L, n, 2L:(n-1L),2L:n)
      x = param$prec/(1-param$rho^2) *
        c( 1L, 1L, rep(1+param\$rho^2, n-2L),rep(-param\$rho, n-1L))
      Q = sparseMatrix(i=i, j=j, x=x, giveCsparse=FALSE)
    return (Q)
  }
  mu = function(n, theta){ return(numeric(0)) }
  log.norm.const = function(n, theta){
    param = interpret.theta(n, theta)
    prec.innovation = param$prec / (1.0 - param$rho^2)
    val = n * (- 0.5 * log(2*pi) + 0.5 * log(prec.innovation))
        + 0.5 * log(1.0 - param$rho^2)
    return (val)
  }
  log.prior = function(n, theta){
    param = interpret.theta(n, theta)
    val = (dgamma(param$prec, shape = 1, rate = 1, log=TRUE)
    + theta[1L] + dnorm(theta[2L], mean = 0, sd = 1, log=TRUE))
    return (val)
  }
  initial = function(n, theta){
    return (rep(1, 2))
  }
  quit = function(n, theta){ return (invisible()) }
  cmd = match.arg(cmd)
  val = do.call(cmd, args =
        list(n = as.integer(args\$n), theta = theta))
  return (val)
}
```

The rgeneric is used to fit a set of data points drawn from an AR1 process, observed with Gaussian noise. The results are displayed in figure A.6.

```
n = 100
rho=0.9
x = arima.sim(n, model = list(ar = rho)) * sqrt(1-rho^2)
y = x + rnorm(n, sd = 0.1)
model = inla.rgeneric.define(inla.rgeneric.ar1.model, n=n)
formula = y ~ -1 + f(idx, model=model)
r = inla(formula, data = data.frame(y, idx = 1:n), family = "gaussian")
```

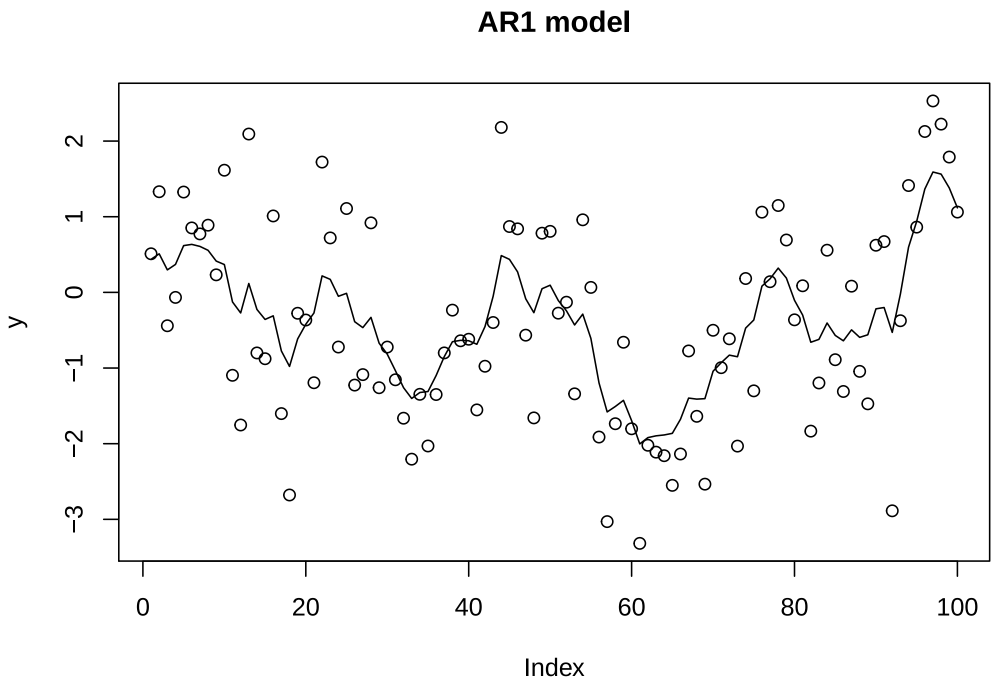

Figure A.6: Plot of the marginal means computed by INLA using the rgeneric model. The 95% confidence intervals are also included along with observations.

Bibliography

[1] P. Brockwell and R. Davis. *Time Series: Theory and Methods.* Springer-Verlag New York, 2 edition, 1991.

[2] J.F Coeurjolly. Simulation and identification of the fractional brownian motion: A bibliographical and comparative study. *Journal of Stat. Software*, 2000.

[3] R.B. Davies and D.S. Harte. Tests for hurst effect. *Biometrika*, 74:95–101, 1987.

[4] Ton Dieker. Simulation of fractional brownian motion. Master's thesis, University of Twente, 2004.

[5] J. Durbin. The fitting of time series models. *Rev. Inst. Int. Stat.*, 28:233–243, 1960.

[6] T. Gneiting and M. Schlather. Stochastic models which separate fractal dimension and hurst effect. *arXiv:physics/0109031 [physics.data-an]*, 2001.

[7] G.H. Golub and C.F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, third edition, 1983.

[8] C.W.J. Granger. Long memory relationships and the aggregation of dynamic models. *Journal of econometrics*, 14:227–238, 1980.

[9] C.W.J. Granger and R. Joyeux. An introduction to long memory time series and fractional differencing. *Journal of Time Series Analysis 1*, pages 1–15, 1980.

[10] C.W.J. Granger and Morris M.J. Time series modelling and interpretation. *Journal of the Royal Statistical Society*, 1976.

[11] D. Guegan. How can we define the concept of long-memory? an econometric survey. *Econometric Reviews*, 24 (2):113–149, 2005.

[12] N. Haldrup and Vera-Valdés J.E. Timelong memory, fractional integration and cross-sectional aggregation. *CREATES Research Papers*, 2015.

[13] H. E. Hurst. Long-term storage capacity of reservoirs. *Transactions of American Society of Civil Engineers*, 16:770–799, 1951.

[14] Beran J. *Statistics for Long-Memory Processes.* Chapman & Hall/CRC, 1 edition, 1994.

[15] I. Karatzas and S. Schreve. *Brownian Motion and Stochastic Calculus.* Springer-Verlag New York, 2 edition, 1998.

[16] G. Kitagawa. *Non-Gaussian state-space modeling of nonstationary time series*, volume 82. Taylor & Francis Group, 1987.

[17] A. Kolmogorov. *Wienersche Spiralen und einige andere interessante Kurven im Hilbertschen Raum.* Dokl. Akad. Nauk SSSR, 1940.

[18] A. Kolmogorov. Dissipation of energy in a locally isotropic turbulence. *Proceedings of the USSR Academy of Sciences*, pages 16–18, 1941.

[19] S. Kullback and R.A. Leibler. On information and sufficiency. *Annals of Mathematical statistics*, 22(1):79–86, 1951.

[20] N. Levinson. The wiener rms error criterion in filter design and prediction. *J. Math. Phys*, 25:261–278, 1947.

[21] B. Mandelbrot and J.W. Ness. Fractional brownian motions, fractional noises and applications. *SIAM Review*, 18:1088–1107, 1968.

[22] C.P. Morice, J.J. Kennedy, N.A. Rayner, and P.D. Jones. Quantifying uncertainties in global and regional temperature change using an ensemble of observational estimates: the hadcrut4 data set. *Journal of Geophysical Research - Atmospheres*, D8, 2012.

[23] Lars Tjensvold Olsen. Estimators of long range dependence - a survey of finite samples and robustness. Master's thesis, Universitety of Agder, 2012.

[24] W. Palma. *Long-memory time series - Theory and mehod*, volume 1. Wiley, 2007.

[25] M.B. Priestly. *Spectral Analysis and Time Series*, volume I and II. Academic Press, 1982.

[26] S.M. Ross. *Introduction to probability models.* Academic Press, 11 edition, 2014.

[27] H. Rue and L. Held. *Gaussian Markov Random Fields: Theory and Applications.* Chapman & Hall/CRC, first edition, 2005.

[28] H. Rue, S. Martino, and N. Chopin. Approximate bayesian inference for latent gaussian models using integrated nested laplace approximations (with discussion). *Journal of the Royal Statistical Society, Series B*, 2009.

[29] H. Theil. *Linear Aggregation of Economic Relations*, volume VII. North-Holland Publishing Co., Amsterdam, 1954.

[30] W.F. Trench. An alorithm for the inversion of finite toeplitz matrices. *Journal of the Society for Industrial and Applied Mathematics*, 12:515–522, 1964.

[31] W. Wei. *Time Series Analysis: Univariate and Multivariate Methods.* Pearson Addison Wesley, second edition, 2006.

[32] P. Whittle. *Hypothesis testing in time series analysis.* Uppsala, Almqvist & Wiksells boktrykkeri, 1951.

[33] A. Wood and G. Chang. Simulation of stationary gaussian processes in $[0,1]^d$. *Journal of Comutational and Graphical Statistics*, 3, 1994.