Stockholms universitet

Wavelet and Fourier Methods for Nonparametric Regression

Pontus Masip

Kandidatuppsats i matematisk statistik
Bachelor Thesis in Mathematical Statistics

Kandidatuppsats 2025:10
Matematisk statistik
Juni 2025

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Stockholm University

Mathematical Statistics
Stockholm University
Bachelor Thesis **2025:10**
http://www.math.su.se

Wavelet and Fourier Methods for Nonparametric Regression

Pontus Masip*

June 2025

Abstract

Fourier series and transforms have proved useful in many applications to statistics like Fourier regression and frequency domain analysis among others. One inherit limitation of them is their loss of localization. Wavelet transforms do not have this limitation and can give us information in both the time and frequency domain. We present the theory behind Fourier series- and transforms as well as for wavelets and wavelet transforms, as described in among others [11, 10]. This theory is used to cover two approaches to nonparametric regression, Fourier smoothing as proposed in [2], and wavelet thresholding using one method of global thresholding and two methods for level-dependent thresholding. We conclude by applying the theory in a simulation study of a selection of test functions. We found that both methods show promising results, but also that Fourier smoothing was overall outperformed by the wavelet thresholding method.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: masippontus@gmail.com. Supervisor: Johannes Heiny Ola Hössjer.

Abstract

Fourier series and transforms have proved useful in many applications to statistics like Fourier regression and frequency domain analysis among others. One inherit limitation of them is their loss of localization. Wavelet transforms do not have this limitation and can give us information in both the time and frequency domain. We present the theory behind Fourier series- and transforms as well as for wavelets and wavelet transforms, as described in among others [11, 10]. This theory is used to cover two approaches to nonparametric regression, Fourier smoothing as proposed in [2], and wavelet thresholding using one method of global thresholding and two methods for level-dependent thresholding. We conclude by applying the theory in a simulation study of a selection of test functions. We found that both methods show promising results, but also that Fourier smoothing was overall outperformed by the wavelet thresholding method.

Sammanfattning

Fourier serier och transformer har visat sig användbara i många tillämpningar inom statistiken, bland annat i form av Fourier regression och analys i frekvensdomänen. En av metodernas begränsningar är deras förlust av lokalisering. Wavelet transformer har inte denna begränsning och kan ge oss information i både tids-och frekvensdomänerna. Vi presenterar teorin bakom Fourier serier- och transformer samt wavelets och wavelet transformer, som bland andra beskrivs i [11, 10]. Denna teori används för att täcka två sorters icke-parametrisk regression, Fourier smoothing som tagits fram i [2], och wavelet thresholding där en metod av global thresholding och två metoder för nivå-beroende thresholding använts. Vi avslutar med att tillämpa teorin i en simuleringsstudie för ett urval av testfunktioner. Vi fann att båda metoderna visade lovande resultat, men att Fourier smoothing generellt överträffades av wavelet thresholding.

Acknowledgement

I would like to thank my supervisors Johannes Heiny and Ola Hössjer for their valuable feedback and suggestions during the writing of this thesis. The only way artificial intelligence was used in the writing of this thesis, was for suggestions of sources in the initial literature search.

Contents

1 Introduction

Fourier methods have been used in many fields for a long time and are a very useful tool when analyzing data containing periodicity as well as for function approximation. A newer method is that of wavelet transforms and wavelet analysis, which is rigorously derived in [3]. Both Fourier and wavelet methods are powerful statistical tools, which is why we will try to introduce them from the ground up to an audience who might be unfamiliar with them. As well as introduce applications of the theory to nonparametric regression using Fourier smoothing and wavelet thresholding. We will conclude by a simulation study were we apply the methods to a selection of test functions in order to compare them and investigate their practical use and limitations.

2 Fourier and Wavelet Transform

In this section, we introduce the theory needed in order to apply these methods to nonparametric regression. Starting with Fourier series and working up to the wavelet transform.

2.1 Preliminaries

Here we present some necessary definitions that will be used in the theory for Fourier series- and transforms as well as for wavelets and wavelet transforms, all of them can be found in [6]

Definition 2.1 (Linear Vector Space)**.** A *linear vector space* over the field $\mathscr{F}$, consists of a space X on which the two mappings $(x, y) \to x + y$,$(\lambda, x) \to \lambda x$ called vector addition and scalar multiplication are defined. These mappings are defined on $X \times X \to X$ and $\mathscr{F} \times X \to X$ respectively, and such that they satisfy the conditions

i) The group $(X, +)$ is commutative,

ii) The associative law $\lambda(\mu x) = (\lambda\mu)x$ holds,

iii) The distributive law $(\lambda + \mu)x = \lambda x + \mu x,\ \lambda(x + y) = \lambda, x + \lambda y$ hold

iv) $1x = x$.

We shall denote by $X(\mathscr{F})$ the linear vector space X over the field $\mathscr{F}$. If a vector space X is said to be real or complex it should be understood that the vector space is $X(\mathbb{R})$ or $X(\mathbb{C})$ respectively.

Definition 2.2 (Inner Product)**.** An *inner product* on a linear vector space X over the field $\mathscr{F}$. Is a mapping $\langle \cdot, \cdot \rangle : X \times X \to \mathscr{F}$, satisfying the conditions

i) $\langle x, x \rangle \geq 0$ and $\langle x, x \rangle = 0$ iff. $x = 0$ (Positive-definite)

ii) $\langle \lambda x + \mu y, z \rangle = \lambda \langle x, z \rangle + \mu \langle y, z \rangle \ \forall x, y, z \in X$ and $\forall \lambda, \mu \in \mathscr{F}$ (Linear in the first argument)

iii) $\langle x, y \rangle = \overline{\langle y, x \rangle}$ (Conjugate Symmetry)

Definition 2.3 (Inner Product Space)**.** An *inner product space* is a linear vector space X over the field $\mathscr{F}$, equipped with an inner product $\langle \cdot, \cdot \rangle : X \times X \to \mathscr{F}$

Definition 2.4 (Hilbert Space)**.** A nonempty set H is called a *Hilbert space* if H is a complete inner product space over the complex numbers. That is for every $\{x_n\} \subset H$ such that $\lim_{n,m\to\infty} \langle x_n - x_m, x_n - x_m \rangle \to 0$ there exists an element $x \in H$ such that $\lim_{n\to\infty} \langle x_n - x, x_n - x \rangle = 0$.

We define the norm $||\cdot||$ associated with the inner product $\langle \cdot, \cdot \rangle$ as $||x|| = \langle x, x \rangle^{1/2}$.

Definition 2.5 (Orthonormal Set)**.** A set M in a Hilbert space H is said to be *orthonormal* if the conditions

i) $||x|| = 1$,

ii) $\langle x, y \rangle = 0$ if $x \neq y$,

hold for all x, y in M.

Definition 2.6 (Orthonormal Basis)**.** A set M in a Hilbert space H is said to be an *orthonormal basis* of H if M is an orthonormal set and if for any $x \in H$

$$x = \sum_{y \in M_x} \langle x, y \rangle y, \text{ where } M_x := \{y \in M | \langle x, y \rangle \neq 0\}.$$

2.2 Fourier series

Denote by E the linear space of complex-valued piecewise continuous functions defined on the interval $[-\pi, \pi]$. To turn E into a inner product space we need to equip it with an inner product $\langle \cdot, \cdot \rangle : E \to \mathbb{C}$. For each $f, g \in E$ define $\langle \cdot, \cdot \rangle$ by

$$\langle f, g \rangle = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) \overline{g(x)} \, dx.$$

Where $\overline{g(x)}$ denotes the complex conjugate of $g(x)$. Since $f \cdot \overline{g}$ is piecewise continuous, we can write the integral as a sum of integrals between the discontinuities of $f \cdot \overline{g}$ on which the function is continuous which guarantees the existence of the integral.

Every function f in a Hilbert space H can be decomposed into a projection of an orthonormal basis β of H. This follows from the Definition 2.6 of an orthonormal basis. Let f be a periodic function on $L^2[a, b], -\infty < a < b < \infty$ with the inner product $\langle f, g \rangle = \frac{1}{b-a} \int_a^b f \overline{g} \, dx$. One orthonormal basis for $L^2[a, b]$

is $\beta = \{\mathrm{e}_n = e^{\frac{2in\pi x}{b-a}} \mid n \in \mathbb{Z}\}$, where $i = \sqrt{-1}$ is the imaginary unit. Which means we can express f as a projection onto β as

$$f = \sum_{n=-\infty}^{\infty} \langle f, \mathrm{e_n} \rangle \mathrm{e_n} = \langle f, \mathrm{e_0} \rangle \mathrm{e_0} + \sum_{n=1}^{\infty} \left(\langle f, \mathrm{e_n} \rangle \mathrm{e_n} + \langle f, \mathrm{e_{-n}} \rangle \mathrm{e_{-n}} \right). \tag{1}$$

The coefficients $\langle f, \mathrm{e_n} \rangle$ are called the Fourier Coefficients of f. We will begin by finding a nicer expression for the Fourier Coefficients. As $\mathrm{e_0} = e^{\frac{2i0\pi x}{b-a}} = 1$ the zeroth coefficient becomes

$$\langle f, \mathrm{e_0} \rangle = \frac{1}{b-a} \int_a^b f(x)\, dx. \tag{2}$$

The nth coefficient can be expressed as

$$\langle f, \mathrm{e_n} \rangle = \frac{1}{b-a} \int_a^b f(x) \overline{e^{\frac{2in\pi x}{b-a}}}\, dx = \frac{1}{b-a} \int_a^b f(x) \left(\overline{\cos\left(\frac{2n\pi x}{b-a}\right) + i \sin\left(\frac{2n\pi x}{b-a}\right)} \right) dx$$

$$= \frac{1}{b-a} \int_a^b f(x) \left(\cos\left(\frac{2n\pi x}{b-a}\right) - i \sin\left(\frac{2n\pi x}{b-a}\right) \right) dx$$

$$= \frac{1}{b-a} \int_a^b f(x) \cos\left(\frac{2n\pi x}{b-a}\right) dx - i \frac{1}{b-a} \int_a^b f(x) \sin\left(\frac{2n\pi x}{b-a}\right) dx, \tag{3}$$

and the $-n$th coefficient can be written as

$$\langle f, \mathrm{e_{-n}} \rangle = \frac{1}{b-a} \int_a^b f \overline{e^{-\frac{2in\pi x}{b-a}}}\, dx$$

$$= \frac{1}{b-a} \int_a^b f \left(\cos\left(\frac{2n\pi x}{b-a}\right) + i \sin\left(\frac{2n\pi x}{b-a}\right) \right) dx$$

$$= \frac{1}{b-a} \int_a^b f \cos\left(\frac{2n\pi x}{b-a}\right) dx + i \frac{1}{b-a} \int_a^b f \sin\left(\frac{2n\pi x}{b-a}\right) dx. \tag{4}$$

Substituting the expressions (2),(3) and (4) into (1) gives us

$$f = \frac{1}{b-a}\int_a^b f(x)\,dx + \sum_{n=1}^{\infty}\left[\frac{e^{\frac{2in\pi x}{b-a}}}{b-a}\int_a^b f(x)\cos\left(\frac{2n\pi x}{b-a}\right)\,dx - i\frac{e^{\frac{2in\pi x}{b-a}}}{b-a}\int_a^b f(x)\sin\left(\frac{2n\pi x}{b-a}\right)\,dx + \ldots\right.$$

$$\left.\frac{e^{-\frac{2in\pi x}{b-a}}}{b-a}\int_a^b f(x)\cos\left(\frac{2n\pi x}{b-a}\right)\,dx + i\frac{e^{-\frac{2in\pi x}{b-a}}}{b-a}\int_a^b f(x)\sin\left(\frac{2n\pi x}{b-a}\right)\,dx\right]$$

$$= \frac{1}{b-a}\int_a^b f(x)\,dx + \sum_{n=1}^{\infty}\left[\frac{e^{\frac{2in\pi x}{b-a}}+e^{-\frac{2in\pi x}{b-a}}}{b-a}\int_a^b f(x)\cos\left(\frac{2n\pi x}{b-a}\right)\,dx + \ldots\right.$$

$$\left.i\frac{-e^{\frac{2in\pi x}{b-a}}+e^{-\frac{2in\pi x}{b-a}}}{b-a}\int_a^b f(x)\sin\left(\frac{2n\pi x}{b-a}\right)\,dx\right]$$

$$= \frac{1}{b-a}\int_a^b f(x)\,dx + \sum_{n=1}^{\infty}\left[\frac{2\cos\left(\frac{2n\pi x}{b-a}\right)}{b-a}\int_a^b f(x)\cos\left(\frac{2n\pi x}{b-a}\right)\,dx + \frac{2\sin\left(\frac{2n\pi x}{b-a}\right)}{b-a}\int_a^b f(x)\sin\left(\frac{2n\pi x}{b-a}\right)\,dx\right]. \tag{5}$$

For simplicity of notation, we introduce the coefficients a_0, a_n and b_n, defined as

$$a_0 = \frac{2}{b-a}\int_a^b f(x)\,dx \tag{6}$$

$$a_n = \frac{2}{b-a}\int_a^b f(x)\cos\left(\frac{2n\pi x}{b-a}\right)\,dx$$

$$b_n = \frac{2}{b-a}\int_a^b f(x)\sin\left(\frac{2n\pi x}{b-a}\right)\,dx. \tag{7}$$

Using this notation (5) turns into

$$f = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left[a_n\cos\left(\frac{2n\pi x}{b-a}\right) + b_n\sin\left(\frac{2n\pi x}{b-a}\right)\right]. \tag{8}$$

From (8) it becomes much easier to see that the Fourier Series of f is a sum of sinusoids. As the function f was chosen as an arbitrary periodic function in $L^2[a,b]$ $-\infty < a < b < \infty$, (8) shows how to decompose any such periodic function into a sum of sinusoids.

This series expansion is very useful and appears frequently in engineering subjects such as electrical engineering and signal processing, but we will focus on using it as a tool for approximating functions. The series (8) can be shown to converge pointwise to f on $[a,b]$, but this requires an infinite number of terms. Luckily the coefficients a_n, b_n are decreasing which means that we can sum a finite number of terms and still end up with an accurate approximation. If

$f, f' \in E$ then the Fourier series of f converges uniformly to f on any subinterval that does not contain any discontinuity point of f.

Although one problem arises as most if not all of the time series that we observe are in the form of discrete observations $\{x_t\}$, so equation (8) can not be applied directly. This is not that hard of a task if we make the assumption that the sequence $\{x_t\}$ has length N and is periodic, that is $x_{t+N} = e_t, \forall t$. Then we can define the Fourier coefficients of $\{x_t\}$ using either complex exponentials or using sines and cosines as in (6)-(7). As most time series are real-valued we will be defining them using sines and cosines, which do not include the imaginary identity i. So assume we have a sequence $\{x_t \mid t = 0, \ldots, N-1\}$ and assume further that it's periodic so that $x_{t+N} = x_t, \forall N$. Then the discrete Fourier coefficients of the sequence are given by

$$a_n = 2 \sum_{k=0}^{N-1} x_k \cos\left(\frac{2\pi kn}{N}\right)$$
$$b_n = 2 \sum_{k=1}^{N-1} x_k \sin\left(\frac{2\pi kn}{N}\right).$$

The frequencies $\{-\pi < \omega_k = \frac{2\pi k}{N} \leq \pi, k = 0, \ldots, N-1\}$ are called the *Fourier frequencies* of $\{x_t \mid t = 0, \ldots, N-1\}$. The original sequence can then be reconstructed using the discrete Fourier series as

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} a_k \cos\left(\frac{2\pi kn}{N}\right) + b_k \sin\left(\frac{2\pi kn}{N}\right).$$

2.3 Fourier transform

As many signals are non-periodic the Fourier series expansion is not always the obvious choice, we can then instead use the Fourier transform, as defined in [11].

Definition 2.7 (Fourier transform)**.** The Fourier transform $\mathcal{F}[\cdot]$ of a function $f : \mathbb{R} \to \mathbb{C}$ is defined as,

$$\mathcal{F}[f](\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(x) e^{-i\omega x} dx, \qquad \omega \in \mathbb{R} \tag{9}$$

when the integral exists. When it exists we will denote the Fourier transform of the function $f(x)$ by $\hat{f}(\omega)$.

The Fourier transform has many applications in signal processing or for solving differential equations. We can view the transform as a transformation from the time domain into the frequency domain. More specifically, if $f(t)$ is a function of time in seconds then the Fourier transform $\hat{f}(\omega)$ of $f(t)$ can be seen as a function of frequency in Hz(1/s). Which maps the frequency ω to its (complex) amplitude. In other words $\hat{f}(\omega)$ gives us the amplitude with which the complex exponential $e^{-i\omega t}$ is "present" in $f(t)$. The Fourier transform has

many useful properties which make it so powerful. To this end let $G(\mathbb{R})$ be the space of all functions f that are piecewise continous on all of $\mathbb{R}$ and that are absolutely integrable. The the functions $f \in G(\mathbb{R})$ have some nice properties, most of which will be stated without proof. The next result is Theorem 3.1 of [11].

Theorem 2.3.1 (Properties of functions in $G(\mathbb{R})$)**.** *For each $f \in G(\mathbb{R})$*

1. *$\hat{f}(\omega)$ is defined for all $\omega \in \mathbb{R}$,*
2. *$\hat{f}(\omega) \in C^0(\mathbb{R})$,*
3. *$\lim_{\omega \to \pm\infty} \hat{f}(\omega) = 0$.*

Knowing that for all $f \in G(\mathbb{R})$ the Fourier transform exists and is continuous, we can begin to state some properties of the transform

Property 2.3.2 (Linearity of Fourier transform)**.** For each $f, g \in G(\mathbb{R})$ and $\forall a, b \in \mathbb{C}$, $af + bg \in G(\mathbb{R})$ and $\mathcal{F}[af + bg](\omega) = a\mathcal{F}[f](\omega) + b\mathcal{F}[g](\omega)$.

The linearity of the transform is a direct consequence of the linearity of the integral.

Property 2.3.3 (Shift formula for Fourier transform)**.** Let $f \in G(\mathbb{R})$ then $\forall a, b \in \mathbb{R}, a \neq 0$, $f(ax + b) \in G(\mathbb{R})$ and the Fourier transform of the shifted function $f(ax + b)$ is $\mathcal{F}[f(ax + b)](\omega) = \frac{1}{|a|} e^{\frac{i\omega b}{a}} \mathcal{F}[f](\frac{\omega}{a})$.

There are also many useful theorems regarding the transform but only a handful will be mentioned here, once again without proof which can be found in [11].

Before stating the first theorem we will define the concept of convolution, which has many applications. Two of which are calculating the Probability Density Function (PDF) of sums of random variables and defining the notion of a filter. Let f and g be two functions defined on all of $\mathbb{R}$. Then the convolution $f * g$ is defined as $(f * g)(x) = \int_{-\infty}^{\infty} f(x - t)g(t)\,dt = \int_{-\infty}^{\infty} f(t)g(x - t)\,dt$. It is shown in [11] that if $f, g \in G(\mathbb{R})$ then $f * g$ exists and is absolutely integrable, which leads us to Theorem 3.5 of [11]:

Theorem 2.3.4 (Convolution Theorem)**.** *Let $f, g \in G(\mathbb{R})$ and $\hat{f}, \hat{g}$ denote their respective Fourier transforms. Then*

$$\mathcal{F}[(f * g)](\omega) = 2\pi \hat{f}(\omega) \cdot \hat{g}(\omega).$$

In some situations we want to return from the frequency domain to the time domain, in other words we are interested in going from the Fourier transform $\hat{f}$ to the underlying function f. Formally the inverse Fourier transform is defined as $f(x) = \mathcal{F}^{-1}[\mathcal{F}[f]] = \int_{-\infty}^{\infty} \hat{f}(\omega) e^{i\omega x}\,d\omega$. The inverse transform is very similar to the Fourier transform with the difference that there is no constant in front of the integral and that we have $e^{i\omega x}$ inside the integral.

But when does this inverse transform exist and what does it evaluate to at discontinuity points? This is covered in Theorem 3.3 of [11]:

Theorem 2.3.5 (Inverse Fourier transform)**.** *If $f \in G(\mathbb{R})$, then for every point $x \in \mathbb{R}$ where the one sided derivatives of f exist*

$$\frac{f(x+) - f(x-)}{2} = \lim_{M \to \infty} \int_{-M}^{M} \hat{f}(\omega) e^{i\omega x} \, d\omega.$$

As in the case of Fourier series a problem arises when using the Fourier transform on real data, as most if not all of the time series we observe are not in the form of a function f, but instead as a sequence $\{x_n\}$ of observations at discrete times. So how can we modify the definition (9) to be applicable to series instead of functions? We will be defining the Discrete Fourier transform (DFT) and the Discrete Time Fourier transform (DTFT) similarly to how they are defined in [10].

Definition 2.8 (Discrete Time Fourier Transform)**.** Let $\{a_t | t = \dots, -1, 0, 1, \dots\}$ denote some real or complex valued sequence that satisfies $\sum_{-\infty}^{\infty} |a_t|^2 < \infty$. Then the DTFT $\hat{a}(\omega)$ of $\{a_t\}$ is defined as

$$\hat{a}(\omega) = \mathcal{F}_{DTFT}\left[\{a_t\}\right] = \sum_{t=-\infty}^{\infty} a_t e^{-i\omega t}. \tag{10}$$

From (10) we can see that the DTFT maps an infinite sequence in discrete time to a complex valued function of frequency ω. This is reflected in the name as only the time and not the frequency is discrete.

The DTFT is periodic with period 2π so that $\hat{a}(\omega + 2\pi n) = \hat{a}(\omega)$ for any frequency ω and any integer n.

The definition (10) is easily extended to finite sequences. Say we have a finite sequence $\{b_t | t = 0, \dots, N-1\}$ of length N. We can then define the infinite sequence $\{a_t | a_t = b_t \text{ if } 0 \leq t \leq N-1, a_t = 0 \text{ otherwise}\}$. Then the DTFT becomes $\hat{a}(\omega) = \mathcal{F}_{DTFT}[\{a_t\}] = \sum_{t=-\infty}^{\infty} a_t e^{-i\omega t} = \sum_{t=0}^{N-1} a_t e^{-i\omega t}$.

But once again we are presented with an issue, as even for finite sequences the frequency in the DTFT is a continuous variable. As we would like to be able to calculate and analyze the transform using computers. It would be beneficial if the transform of our sequence mapped onto another finite sequence. To handle this we define the DFT.

Definition 2.9 (Discrete Fourier transform)**.** Let $\{a_t\}_{t=0}^{N-1}$ be a real or complex sequence of length N. Then the DFT of $\{a_t\}_{t=0}^{N-1}$ is the sequence $\{\hat{a}_k\}_{k=0}^{N-1}$. Where the kth element is defined as

$$\hat{a}_k = \sum_{t=0}^{N-1} a_t e^{-itk\omega_0}.$$

Note that $\hat{a}_k$ corresponds to the frequency $\omega_k = k\omega_0$, where ω_0 is the fundamental frequency $\omega_0 = \frac{2\pi}{N}$. We will write the DFT of the entire sequence as

$$\{\hat{a}_k\} = \mathcal{F}_{DFT}\left[\{a_t\}\right].$$

In order to keep the periodic property we define for $t > N-1$ or $t < 0$ a_t by periodic extension, so that $a_{-i} = a_{N-i}$, and $a_{N-1+i} = a_i$ for integers $i \geq 1$. We can denote this is as $a_{t \bmod N}$.

As for the Continuous Fourier transform (CFT) we can easily define the convolution of two DFT's. But first we will need to define the concept of circular convolution. Let $\{a_t\}$ and $\{b_t\}$ be two sequences of length N, then the convolution or circular convolution can be defined as $a * b_t = \sum_{u=0}^{N-1} a_u b_{t-u}$, which with the notation for periodic extension becomes $a*b_t = \sum_{u=0}^{N-1} a_u b_{t-u \bmod N}$. We can now state the Convolution Theorem for the DFT, as stated in [10].

Theorem 2.3.6. *Discrete Convolution Theorem Let $\{a_t\}$ and $\{b_t\}$ be two sequences of length N. Then the DFT of $a * b_t$ is*

$$\mathcal{F}_{DFT}[\{a * b_t\}](k) = \sum_{t=0}^{N-1} a * b_t e^{-itk\omega_0} = \hat{a}_k \hat{b}_k$$

Having defined how we can calculate the DFT of convolutions, we will define the notion of a filter similarly to how it is defined in [10].

Let $\{a_t\}$ and $\{b_t\}$ be two arbitrary real or complex valued sequences and let$\{\hat{a}_t\}$, $\{\hat{b}_t\}$ be their respective DFT. If we regard $\{a_t\}$ as our input and $\{b_t\}$ as the filter we want to use. The operation of filtering $\{a_t\}$ with $\{b_t\}$ is equivalent to convolving the two. Using Theorem 2.3.6 we know that in the frequency domain this is equivalent to multiplication of $\{\hat{a}_t\}$ and $\{\hat{b}_t\}$. Let us also introduce the notion of width for a filter, assume that the filter $\{b_t\}_{t=0}^{N-1}$ is such that $b_K \neq 0$, $b_{K+L} \neq 0$ for some positive integers L and K. Assume further that $b_t = 0$ for all $t < K$ and $t > K+L$, then as the filter b_t is identically zero for all but L adjacent terms. We say that the filter has width L.

2.4 Wavelets

Fourier transforms only contain information about the global frequency contents of a function or signal. This of course means a total loss of time dependent information. That is, the transform gives us information about all frequencies present in the signal or function, but no information about when they are present.

An alternative to the Fourier transform is the so called wavelet transform, which contains information about both frequency and time. The main difference between Fourier and Wavelet transforms is that the former uses a basis of complex exponential functions. Which are periodic and repeat infinitely in time, whereas the latter uses a wavelet basis which is localized in time.

We will begin by introducing the notion of a wavelet in more depth, as with much of the theory in this section we will be using the definition of [10].

Definition 2.10 (Wavelet)**.** A real valued function ψ defined on the extended real numbers and satisfying the properties

(i) The integral of ψ is zero:

$$\int_{-\infty}^{\infty} \psi(x)\, dx = 0$$

(ii) The square of ψ integrates to one:

$$\int_{-\infty}^{\infty} \psi^2(x)\, dx = 1$$

Is called a wavelet.

To get a basic understanding of what a wavelet function is and how they can look we will introduce three examples . One of the simplest and oldest examples of a wavelet is the Haar wavelet, defined as:

$$\psi_H(x) := \begin{cases} -1/\sqrt{2}, & -1 < x < 0 \\ 1/\sqrt{2}, & 0 < x < 1 \\ 0, & \text{otherwise} \end{cases}$$

We will continue by defining two wavelets based on the PDF of the normal distribution $\mathcal{N}(0, \sigma^2)$ which we here denote by $f(x)$. Taking the first and second derivatives of $f(x)$ w.r.t x we obtain

$$f'(x) = -\frac{x}{\sigma^3\sqrt{2\pi}} e^{-x^2/2\sigma^2}, \text{ and } f''(x) = \frac{\left(x^2 - \sigma^2\right) e^{-x^2/2\sigma^2}}{\sigma^5\sqrt{2\pi}}.$$

Multiplying the two derivatives and normalizing them with the constants a_1,a_2 in order to satisfy (ii), results in (cf. [10])

$$\psi_{FDG}(x) = \frac{f'(x)}{a_1} = -\frac{\sqrt{2}x}{\sigma^{3/2}\pi^{1/4}} e^{-x^2/2\sigma^2}, \text{ and}$$
$$\psi_R(x) = \frac{f''(x)}{a_2} = \frac{2\left(1 - \frac{x^2}{\sigma^2}\right) e^{-x^2/2\sigma^2}}{\sqrt{3\sigma}\pi^{1/4}}.$$

We will call the second wavelet, ψ_R, the Ricker wavelet but its also commonly referred to as the Marr wavelet or Mexican hat wavelet.
The three wavelets introduced so far, can be seen in figure 1. From the figure we can clearly see that these wavelets are nonzero only on a small region around the origin.

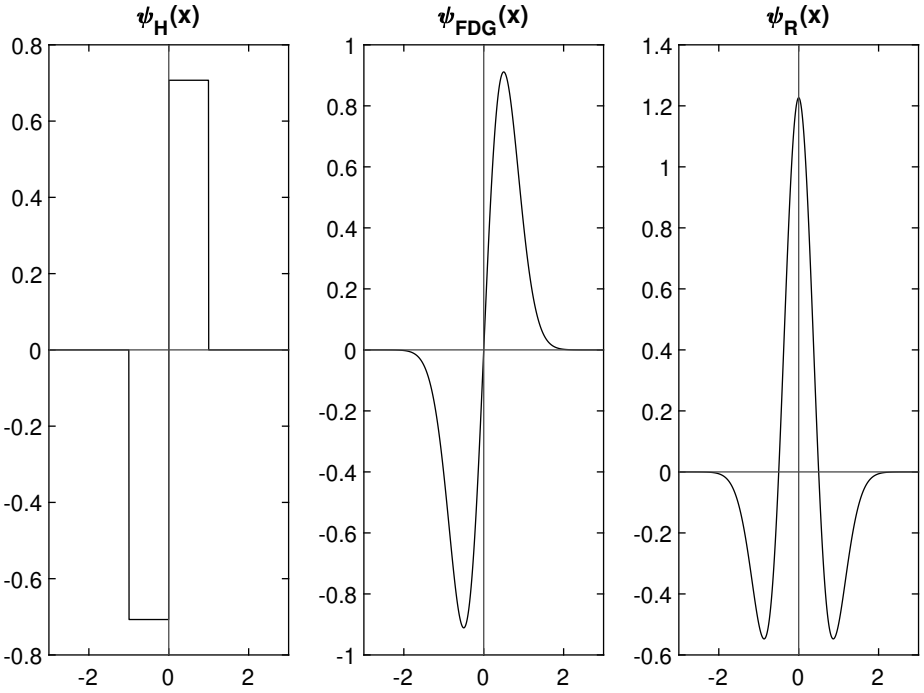

Figure 1: The three example wavelets $\psi_H(x)$, $\psi_{FDG}(x)$ and $\psi_R(x)$ plotted around the origin.

Before defining the wavelet transform we will mention a common additional condition for wavelets, as defined in [4].

Definition 2.11 (Admissibility Condition)**.** A wavelet function ψ is said to be admissible, if the number C_ψ defined by

$$C_\psi = 2\pi \int |\omega|^{-1} |\hat{\psi}(\omega)|^2 \, d\omega, \tag{11}$$

where $\hat{\psi}$ is the Fourier transform of ψ, satisfies $C_\psi < \infty$.

2.4.1 Wavelet transform

Denote by $\psi^{a,b}$ the wavelet function with parameters a, b defined by

$$\psi^{a,b}(x) = \frac{1}{\sqrt{a}} \psi \left(\frac{x-b}{a} \right) \tag{12}$$

Recall that the three example wavelets in figure 1 were localized around the origin. If we vary the parameter b in (12) we can translate the wavelet to instead be centered around b. If we instead vary the parameter a in (12) we can scale the wavelet to be either more or less spread out around b.

It was shown that the scaled and translated wavelet (12) still satisfies the conditions of Definition 2.10 , in [10]. Using the notation (12) we can now define the Continuous Wavelet Transform (CWT).

Definition 2.12 (Continuous wavelet transform)**.** The CWT $W(a,b)$ of a signal $x(t)$ using the wavelet $\psi^{a,b}(x)$ is given by

$$W(a,b) = \int_{-\infty}^{\infty} \psi^{a,b}(u)x(u)\,du.$$

If the wavelet $\psi^{a,b}(x)$ additionally satisfies the admissibility condition, and the signal $x(t)$ is square integrable, that is $x(t)$ satisfies

$$\int_{-\infty}^{\infty} x(t)^2\,dt < \infty.$$

Then the CWT preserves all information about the signal, we can then define the inverse continuous wavelet transform as in [10].

Definition 2.13 (Inverse continuous wavelet transform)**.** If the wavelet $\psi^{a,b}(x)$ is admissible, and the signal $x(t)$ is square integrable. We can define the Inverse Continuous Wavelet Transform (ICWT) of the CWT $W(a,b)$ as the integral

$$x(t) = \frac{1}{C_\psi}\int_0^{\infty}\left[\int_{-\infty}^{\infty} W(a,b)\psi^{a,b}(u)\,du\right]\frac{da}{a^2},$$

where C_ψ is defined in (11).

2.4.2 Orthonormal transforms

An $N\times N$ matrix $\mathcal{O}$ is said to be orthonormal if $\mathcal{O}^\top\mathcal{O} = \mathbb{I}_N$l, where $\mathbb{I}_N$ is the $N\times N$ identity matrix. We can then decompose a time series $\{X_t : t = 0,1,...,N-1\}$ represented as the $N\times 1$ dimensional vector $\mathbf{X} = \left[X_0, X_1, ..., X_{N-1}\right]^\top$ using $\mathcal{O}$ by left multiplication of $\mathbf{X}$ with $\mathcal{O}$. Denote the decomposition of X using $\mathcal{O}$ by $\mathbf{O}$, that is $\mathbf{O} = \mathcal{O}X$ which is also a $N\times 1$ dimensional vector.

Given $\mathbf{O}$ we can use the orthonormal property of $\mathcal{O}$ to reconstruct or synthesize the original time series $\mathbf{X}$. This is done by left multiplication of $\mathbf{O}$ with $\mathcal{O}^\top$

$$\mathcal{O}^\top\mathbf{O} = \mathcal{O}^\top\mathcal{O}\mathbf{X} = \mathbb{I}_N\mathbf{X} = \mathbf{X}. \tag{13}$$

Notice that we can write the ith element o_i of $\mathbf{O}$ as $\langle X, \mathcal{O}_{i,\bullet}\rangle$ where $\mathcal{O}_{i,\bullet}$ is the transpose of row i in $\mathcal{O}$, i.e a $N\times 1$ dimensional vector. The synthesis (13) can then be rexpressed as

$$\mathbf{X} = \sum_{i=0}^{N-1}\langle \mathbf{X}, \mathcal{O}_{i,\bullet}\rangle\mathcal{O}_{i,\bullet}. \tag{14}$$

As the vectors $\mathcal{O}_{i,\bullet}$ form a basis of $\mathbb{R}^N$, (14) holds for an arbitrary choice of $\mathbf{X}$. An important fact about orthonormal transforms is that they preserve

energy in that the squared norm of the coefficients $\mathbf{O}$ equals that of the original time series $\mathbf{X}$, this fact can be seen from

$$||\mathbf{O}||^2 = \mathbf{O}^\top \mathbf{O} = (\mathcal{O}\mathbf{X})^\top (\mathcal{O}\mathbf{X}) = \mathbf{X}^\top \mathcal{O}^\top \mathcal{O}\mathbf{X} = \mathbf{X}^\top \mathbb{I}_N \mathbf{X} = \mathbf{X}^\top \mathbf{X} = ||\mathbf{X}||^2.$$

One prominent example of an orthonormal transform is the Orthonormal Discrete Fourier transform (ODFT) which differs from the DFT only by a factor $\frac{1}{\sqrt{N}}$. Let $\mathcal{O} = F$ be the transformation matrix of the ODFT then the (k,t)th element (row k, column t) of F is $\frac{1}{\sqrt{N}} e^{-itk\omega_0}$. This element is associated with the value X_t at time t and with the frequency $\omega_k = k\omega_0$. For the regular DFT which is an orthogonal transform the (k,t)th element of it's transformation matrix is $e^{-itk\omega_0}$.

Using each row we can group the transform by frequency and split the synthesis of $\mathbf{X}$ into a low pass part $\mathcal{S}_{\mathcal{F},k}$ called the kth order Fourier smooth and a high pass part $\mathcal{R}_{\mathcal{F},k}$ called the kth order Fourier rough. Using these parts we can express the synthesis of $\mathbf{X}$ as $\mathbf{X} = \mathcal{S}_{\mathcal{F},k} + \mathcal{R}_{\mathcal{F},k}$. We can also define the detail $\mathcal{D}_{\mathcal{F},k}$ of order k, which is the difference between smooths or roughs of adjacent orders, that is $\mathcal{D}_{\mathcal{F},k} = \mathcal{S}_{\mathcal{F},k} - \mathcal{S}_{\mathcal{F},k-1} = \mathcal{R}_{\mathcal{F},k-1} - \mathcal{R}_{\mathcal{F},k}$. We will not go into how the smooth, rough or detail are calculated yet.

2.4.3 Discrete wavelet transform

Let $\mathbf{X}$ be the vector representation of a time series $\{X_t : t = 0, 1, \ldots, N-1\}$, where N is a multiple of some integer 2^{J_0}. The partial Discrete Wavelet Transform (DWT) of $\mathbf{X}$ is an orthonormal transform given by $\mathbf{W} = \mathcal{W}\mathbf{X}$, where $\mathbf{W}$ is an N dimensional vector of DWT coefficients and $\mathcal{W}$ is a real $N \times N$ matrix defining the transform. The vector $\mathbf{W}$ and matrix $\mathcal{W}$ can be partitioned as

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_1 \\ \mathbf{W}_2 \\ \vdots \\ \mathbf{W}_{J_0} \\ \mathbf{V}_{J_0} \end{bmatrix}, \quad \mathcal{W} = \begin{bmatrix} \mathcal{W}_1 \\ \mathcal{W}_2 \\ \vdots \\ \mathcal{W}_{J_0} \\ \mathcal{V}_{J_0} \end{bmatrix},$$

so that the ith element $\mathbf{W}_i = \mathcal{W}_i\mathbf{X}$ and $\mathbf{V}_{J_0} = \mathcal{V}_{J_0}\mathbf{X}$. Here $\mathbf{W}_i$ is an $N_i = N/2^i$ dimensional vector of wavelet coefficients associated with changes on the scale 2^{i-1}, $\mathcal{W}_i$ is an $N_i \times N$ dimensional matrix and $\mathbf{V}_{J_0}$ is an N_{J_0} dimensional vector of scaling coefficients associated with averages on the scale 2^{J_0}. If moreover $N = 2^J$ for some integer J we can similarly define the full wavelet transform by setting $J_0 = J$.

In practice the vector $\mathbf{W}$ of wavelet coefficients is calculated using the pyramid algorithm introduced in [9]. Unlike matrix multiplication which for multiplication of a $N \times N$ matrix and $N \times 1$ vector requires N^2 multiplications, this algorithm allows $\mathbf{X}$ to be calculated using only $O(N)$ multiplications. This makes calculation of the DWT using the pyramid algorithm more efficient than calculation of the DFT using the Fast Fourier transform (FFT) algorithm which requires $O(N \log_2 N)$ multiplications.

The pyramid algorithm requires two filters h, g called the *wavelet* and *scaling filters* respectively. Let h be a filter of width L where we define $h_0 \neq 0$, $h_{L-1} \neq 0$ and $h_l = 0$ for $l < 0$ and $l \geq L$. In order for h to be a wavelet filter L must be even and the filter needs to satisfy the three following properties, as explained in [10].

$$\sum_{l=0}^{L-1} h_l = 0$$
$$\sum_{l=0}^{L-1} h_l^2 = 1,$$
$$\text{and} \quad \sum_{l=0}^{L-1} h_l h_{l+2n} = \sum_{l=-\infty}^{\infty} h_l h_{l+2n} = 0.$$

Two commonly used examples of wavelet filters are the Haar and Daubechies filters defined by $\{h_0 = \frac{1}{\sqrt{2}}, h_1 = -\frac{1}{\sqrt{2}}\}$, and $\{h_0 = \frac{1-\sqrt{3}}{4\sqrt{2}}, h_1 = \frac{-3+\sqrt{3}}{4\sqrt{2}}, h_2 = \frac{3+\sqrt{3}}{4\sqrt{2}}, h_3 = \frac{-1-\sqrt{3}}{4\sqrt{2}}\}$ respectively. As this specific example of a Daubechies filter has width 4 it is usually denoted by $D(4)$, but note that there are several others. The scaling filter g cannot be chosen freely and corresponds to the choice of wavelet filter through the relation

$$g_l = (-1)^{l+1} h_{L-1-l}.$$

To explain the pyramid algorithm is beyond the scope of this thesis but an in depth explanation can be found in [9], or a somewhat less in depth one in [10]. Similiar to what we did with the ODFT we can also define the ith level wavelet smooth, rough and detail denoted $\mathcal{S}_i, \mathcal{R}_i$ and $\mathcal{D}_i$ respectively. We can calculate the J_0th level wavelet smooth as $\mathcal{S}_{J_0} = \mathcal{V}_{J_0}^\top \mathbf{V}_{J_0}$ which is associated with the scale 2^{J_0} and the ith level wavelet detail as $\mathcal{D}_i = \mathcal{W}_i^\top \mathbf{W}_i$ which is associated with the scale 2^{i-1}.We can then use the J_0th level smooth and the ith level detail to succesively calculate the ith level wavelet smooth and rough as

$$\mathcal{S}_i = \mathcal{S}_{J_0} + \sum_{k=i+1}^{J_0} \mathcal{D}_k \qquad \text{and} \qquad \mathcal{R}_i = \sum_{k=1}^{i} \mathcal{D}_k.$$

As the DWT is an orthonormal transform we can synthesize $\mathbf{X}$ with (13), using $\mathcal{W}$ and $\mathbf{W}$. This can be rewritten using $\mathcal{S}_{J_0}$ and the $\mathcal{D}_i$'s as

$$\mathbf{X} = \mathcal{W}^\top \mathbf{W} = \sum_{i=1}^{J_0} \mathcal{W}_i^\top \mathbf{W}_i + \mathcal{V}_{J_0}^\top \mathbf{V}_{J_0} = \sum_{i=1}^{J_0} \mathcal{D}_i + \mathcal{S}_{J_0}. \tag{15}$$

We call the rightmost expression in (15) a Multi Resolution Analysis (MRA) of $\mathbf{X}$, or in other words an additive decomposition of $\mathbf{X}$ using components

associated with different scales. Another consequence of the DFT being orthonormal is that it preserves the energy of $\mathbf{X}$. Using the MRA we can write the energy decomposition as

$$||\mathbf{X}||^2 = \sum_{i=1}^{J_0} ||\mathcal{D}_i||^2 + ||\mathcal{S}_{J_0}||^2 \tag{16}$$

Since the sample variance $\hat{\sigma}^2_{\mathbf{X}}$ of $\mathbf{X}$ can be expressed using the squared norm and sample mean $\bar{\mathbf{X}}$ as $\hat{\sigma}^2_{\mathbf{X}} = \frac{1}{N}||X||^2 - \bar{\mathbf{X}}^2$. We can then use the MRA energy decomposition (16) to decompose the sample variance as

$$\begin{aligned}\hat{\sigma}^2_{\mathbf{X}} &= \frac{1}{N}||X||^2 - \bar{\mathbf{X}}^2 = \frac{1}{N}\left(\sum_{i=1}^{J_0} ||\mathcal{D}_i||^2 + ||\mathcal{S}_{J_0}||^2\right) - \bar{\mathbf{X}}^2 \\ &= \frac{1}{N}\sum_{i=1}^{J_0} ||\mathcal{D}_i||^2 + \frac{||\mathcal{S}_{J_0}||^2}{N} - \bar{\mathbf{X}}^2.\end{aligned}$$

3 Methods

3.1 Non-Parametric Regression

In simple linear regression we use the model

$$Y_i = \alpha + \beta X_i + \varepsilon_i, \qquad \varepsilon_i \sim \mathcal{N}(0, \sigma^2)$$

to describe the relationship between the response variable Y and the regressor $X \in [0, \pi]$. Where the points $X_1, ..., X_N$ are assumed to be non-random, and the noise ε_i is independent. This model is useful to describe many different phenomena or to find a trend in noisy data. The simple linear regression model together with multiple linear regression and other models, where we make an assumption on the relationship between response and regressor variables, are called parametric models or parametric regression. Parametric regression is a very useful technique when the relationship between response and regressor is known, or when it is reasonable to make assumptions about the nature of the relation.

But what if the relationship between response variable Y and the regressors are unknown, and no meaningful assumptions can be made? This is a quite common situation when trying to understand different phenomena. Mathematically this can be formulated as

$$Y_i = g(t_i) + \varepsilon_i, \qquad \varepsilon_i \sim \mathcal{N}(0, \sigma^2) \quad i = 1, \ldots, N, \tag{17}$$

where $t_i = X_i \in [0, \pi]$ and ε_i are independent, and σ^2 is finite. The function g is a bounded and continuous function. It describes the unknown relationship between X and Y, and is what we intend to estimate. In this situation a technique called nonparametric regression can be used, where the goal is to estimate the unknown function g.

This problem is a hard one as there are infinitely many families of functions that g could belong to. As mentioned in [2], when the errors are normal the log-likelihood function for (σ^2, g) is

$$l(\sigma^2, g) = C - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - g(t_i))^2, \tag{18}$$

where C is some constant. Clearly the g that maximizes (18) is any function that interpolates the points y_i, that is $g(t_i) = y_i$. So some further restrictions are needed.
We will cover two approaches to nonparametric regression, Fourier Smoothing as proposed in [2], and wavelet thresholding covered in [1].

3.1.1 Fourier Smoothing

As proposed in [2] one restriction on the set of possible g is using Grenander's method of sieves. Loosely speaking the method of sieves involves restricting the maximization of the likelihood to a subset of the parameter space, where the subset is chosen increasingly dense with increasing sample size. As mentioned in [8] using this method leads to consistent nonparametric estimators, the sequence of subsets is what is called the sieve. Different choices of sieve leads to different estimators. The sieve proposed in [2] was

$$S_\mu = \left\{ T \mid ||T^{(2)}||_{L_2[0,\pi]} \leq \frac{1}{\mu} \right\}, \tag{19}$$

where T is defined for a fixed K and some constants $\mathbf{a} = (b, \frac{1}{2}a_0, a_1, ..., a_K)^\top$ as

$$T(t) = bt + \frac{1}{2}a_0 + \sum_{k=1}^{K} a_k \cos(kt).$$

We can see that T is a the sum of a linear function and a cosine expansion. Using the sieve (19), the problem was reduced in [2], for some $\mu_0 > \mu$ to

$$\underset{\mathbf{a}\in\mathbb{R}^{K+2}}{\text{Minimize}} \quad \frac{1}{N}\sum_{i=1}^{N}(y_i - T(t_i))^2 + \lambda P(\mathbf{a}), \tag{20}$$

where $P(\mathbf{a}) := \sum_{k=1}^{K} k^4 a_k^2$ and λ is a Lagrange multiplier. As $\lambda \to \infty$ the function T will converge to the least squares line and as $\lambda \to 0$ T will interpolate all y_i for large enough K.

For fixed λ the normal equations to (20) become $\mathbf{M}_\lambda \mathbf{a} = \mathbf{b}$ where $\mathbf{M}_\lambda$ and

b are defined as

$$\mathbf{M}_\lambda = \left(\frac{1}{N}\mathbf{X}^\top\mathbf{X} + \lambda\tilde{\mathbf{D}}\right),\ \mathbf{b} = \frac{1}{N}\mathbf{X}^\top\mathbf{y}$$

where

$$\mathbf{X} = \begin{pmatrix} t_1 & 1 & \cos(t_1) & \cos(2t_1) & \dots & \cos(Kt_1) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ t_N & 1 & \cos(t_N) & \cos(2t_N) & \dots & \cos(Kt_N) \end{pmatrix}$$

and

$$\tilde{\mathbf{D}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1^4 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 2^4 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 3^4 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & K^4 \end{pmatrix},$$

so that the coefficients **a** can be obtained by solving the system as $\mathbf{a}(\lambda) = \mathbf{M}_\lambda^{-1}\mathbf{b}$. With $\mathbf{a}(\lambda)$ obtained we can finally estimate g at time $t \in (0, \pi)$ as

$$\tilde{g}_\lambda(t) = \mathbf{x}(t)\mathbf{a}(\lambda) = b(\lambda)t + \frac{1}{2}a_0(\lambda) + \sum_{k=1}^{K} a_k(\lambda)\cos(kt), \tag{21}$$

where $\mathbf{x}(t) = (t, 1, \cos(t), \cos(2t), \dots, \cos(Kt))^\top$. So that the estimate at each point t_i can be obtained as the vector $\tilde{\mathbf{g}}_\lambda = \mathbf{X}\mathbf{a}(\lambda)$. For a fixed value of λ and some linear smoother $\mathbf{S}_\lambda$ with smoothing parameter λ the estimate $\tilde{\mathbf{g}}_\lambda$ satisfies $\tilde{\mathbf{g}}_\lambda = \mathbf{S}_\lambda\mathbf{y}$. The matrix $\mathbf{S}_\lambda$ can easily be expressed as $\mathbf{S}_\lambda = \frac{1}{N}\mathbf{X}\mathbf{M}_\lambda\mathbf{X}^\top\mathbf{y}$. It was shown in [2] that the nonzero eigenvalues of $\mathbf{S}_\lambda$ are $\left\{1, 1, (1+\lambda\gamma_k)^{-1} \mid k = 1, \dots, K\right\}$ where the γ_k's are the inverses to the eigenvalues of the matrix $\mathbf{D}^{-1/2}(\mathbf{C} - \mathbf{B}\mathbf{A}^{-1}\mathbf{B}^\top)\mathbf{D}^{-1/2}$. Here **A** is a 2×2 matrix, **B** is a $K \times 2$ matrix and **C** is a $K \times K$ matrix all given by the relation

$$\frac{1}{n}\mathbf{X}^\top\mathbf{X} = \begin{pmatrix} \mathbf{A} & \mathbf{B}^\top \\ \mathbf{B} & \mathbf{C} \end{pmatrix}.$$

The matrix **D** is simply obtained by partitioning $\tilde{\mathbf{D}}$ as

$$\tilde{\mathbf{D}} = \begin{pmatrix} 0_{2\times 2} & 0_{2\times K} \\ 0_{K\times 2} & \mathbf{D} \end{pmatrix}.$$

The nonzero eigenvalues where then used to show that for a fixed $\lambda > 0$, the spectral radius ρ of $\mathbf{S}_\lambda$ (maximum absolute value of the eigenvalues of $\mathbf{S}_\lambda$) satisfies $\rho(\mathbf{S}_\lambda) \leq 1$. But also that tr $\mathbf{S}_\lambda = 2 + \sum_{k=1}^{K} \frac{1}{1+\lambda\gamma_k}$ is a strictly decreasing convex function of λ bounded above by $K + 2$. With these relations a method

of choosing an "optimal" smoothing level was proposed in [2], i.e choosing λ by minimizing the Mean Squared Error of Cross Validation (MSECV) denoted by $CV(\lambda)$. This quantity $CV(\lambda)$ is defined as

$$CV(\lambda) = \frac{1}{N}\sum_{i=1}^{n}(y_i - \tilde{g}_{\lambda,i})^2,$$

where $\tilde{g}_{\lambda,i}$ denotes the smoother (21) fitted to the data with (t_i, y_i) removed. In order to not have to refit the model N times Theorem 4.2.1 of [12] was used to evaluate $CV(\lambda)$ with complexity $O(N)$ as

$$CV(\lambda) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i - \tilde{g}(t_i)}{1 - S_\lambda(i,i)}\right)^2,$$

where $S_\lambda(i,i)$ is the element on the ith row and ith column of $\mathbf{S}_\lambda$. Finally the value of λ was calculated iteratively with the scheme

$$\begin{aligned}\lambda_0 &= 0\\ \lambda_{i+1} &= \lambda_i + \frac{2(\text{tr }\mathbf{S}_\lambda - 2) - \text{df}_i}{\sum_{k=1}^{K}\gamma_k(1+\lambda_i\gamma_k)^{-2}},\end{aligned}$$

where $\text{df}_i = 2 + i,\ i = 0, 1, \ldots .$ The scheme was terminated when there was no further decrease in $CV(\lambda_i)$, leading to a smoothing parameter$\lambda = \lambda(\mathbf{y})$ that is a function of the response variable $\mathbf{y}$.

3.1.2 Wavelet Thresholding

A different approach to nonparametric regression of the model (17) is the method called wavelet shrinkage or wavelet thresholding. It is based on the sparseness of the DWT which allows for the assumption that only a few large wavelet coefficients $\mathbf{W}_{i,k}$ will contain information about g, while small $\mathbf{W}_{i,k}$ are attributed to noise. This assumption comes from the fact that the DWT of white noise is a new independent white noise variable, as mentioned in [1]. The DWT of $\mathbf{y}$ results in the so called empirical scaling and wavelet coefficients $\tilde{\mathbf{V}}_{J_0,k}$ and $\tilde{\mathbf{W}}_{i,k}$ respectively. We will assume that the length of $\mathbf{y}$ is $N = 2^J$ for some positive integer J. The empirical coefficients are then given by

$$\begin{aligned}\tilde{\mathbf{V}}_{J_0,k} &= \mathbf{V}_{J_0,k} + \sigma\varepsilon_{J_0,k}, \qquad k = 0, 1, \ldots 2^{J_0} - 1\\ \tilde{\mathbf{W}}_{i,k} &= \mathbf{W}_{i,k} + \sigma\varepsilon_{i,k}, \qquad i = 1, \ldots, J_0,\ k = 0, 1, \ldots, 2^{J-i},\end{aligned} \tag{22}$$

where $\varepsilon_{J_0,k}$ and $\varepsilon_{i,k}$ are independent $\mathcal{N}(0,1)$ random variables. The wavelet thresholding method then revolves around finding a way to classify and discard the wavelet coefficients $\mathbf{W}_{i,k}$ attributed to noise.

The two main thresholding methods are called soft and hard thresholding respectively, where they differ in how the thresholding of the coefficients is done.

The hard and soft thresholding functions with thresholding level λ are defined as

$$\delta_\lambda^H(\tilde{\mathbf{W}}_{i,k}) = \begin{cases} 0 & \text{if } |\tilde{\mathbf{W}}_{i,k}| \leq \lambda, \\ \tilde{\mathbf{W}}_{i,k} & \text{if } |\tilde{\mathbf{W}}_{i,k}| > \lambda, \end{cases}$$

$$\delta_\lambda^S(\tilde{\mathbf{W}}_{i,k}) = \begin{cases} 0 & \text{if } |\tilde{\mathbf{W}}_{i,k}| \leq \lambda, \\ \tilde{\mathbf{W}}_{i,k} - \lambda & \text{if } \tilde{\mathbf{W}}_{i,k} > \lambda, \\ \tilde{\mathbf{W}}_{i,k} + \lambda & \text{if } \tilde{\mathbf{W}}_{i,k} < -\lambda. \end{cases}$$

The reason for using only the wavelet coefficient $\tilde{\mathbf{W}}_{i,k}$ in the definition is that it is proposed in [1] that the scaling coefficients $\tilde{\mathbf{V}}_{J_0,k}$ are left intact. It is mentioned in [1] that hard thresholding results in larger variance for the same threshold level while soft thresholding creates unnecessary bias when the true coefficients are large as it shifts the coefficients by λ even if they contain no noise.
To account for these drawbacks a firm thresholding was proposed in [7] , the firm thresholding function is defined as

$$\delta_{\lambda_1,\lambda_2}^F(\tilde{\mathbf{W}}_{i,k}) = \begin{cases} 0 & \text{if } |\tilde{\mathbf{W}}_{i,k}| \leq \lambda_1, \\ \operatorname{sign}\left(\tilde{\mathbf{W}}_{i,k}\right) \frac{\lambda_2\left(|\tilde{\mathbf{W}}_{i,k}| - \lambda_1\right)}{\lambda_2 - \lambda_1} & \text{if } \lambda_1 < |\tilde{\mathbf{W}}_{i,k}| \leq \lambda_2, \\ \tilde{\mathbf{W}}_{i,k} & \text{if } |\tilde{\mathbf{W}}_{i,k}| > \lambda_2, \end{cases}$$

with the lower and upper thresholding levels λ_1 and λ_2 respectively.

The drawback of firm thresholding is that two thresholding levels need to be estimated instead of just one.
In order to use these thresholding methods it remains to estimate the thresholding level(s) λ. But before we can estimate λ we have to decide if the thresholding will be global or level-dependent. Or in other words if the same threshold level λ will be used for all wavelet coefficients $\tilde{\mathbf{W}}_{i,k}$ or if a possibly separate threshold level λ_j will be used for each resolution level $j = J_0, \ldots, J-1$ of wavelet coefficients.

We will cover one method of global thresholding and two methods for level-dependent thresholding. All of these methods require an estimate of the noise level σ. As the signal $g(X_i)$ is unknown and could possibly have a large standard deviation by itself, the standard deviation of the noisy signal $g(X_i)+\sigma\varepsilon_i$ is not a good estimator in general, unless as mentioned in [1], the signal $g(X_i)$ in itself is flat. But this is an assumption on g which we would like to avoid. As we saw in equation (22), the wavelet coefficients contain independent noise with the same noise level as the original signal (17). We also already assumed that only a few wavelet coefficients will contain information about the original signal while the rest are mostly noise. We could then estimate the noise level σ from the wavelet coefficients. In [5] the absolute median deviation of the wavelet coefficients on the finest resolution level was used. The reason for using the finest resolution level was as mentioned in [5] that they contain with a few exceptions basically

pure noise. The absolute median deviation noise estimator is defined as

$$\tilde{\sigma} = \frac{\text{median } \left(\left\{ |\tilde{\mathbf{W}}_{1,k}|;\ k = 0, 1, \ldots, 2^{J-1} - 1 \right\} \right)}{0.6745}, \tag{23}$$

the noise estimator (23) will be used in the following estimations of the threshold level λ.
The global thresholding level was selected as the universal threshold mentioned in [1] and proposed in [5]. The universal threshold λ^U for sample size N is given by

$$\lambda^U = \tilde{\sigma}\sqrt{2 \log N},$$

and it is very computationally inexpensive to calculate.
The two level-dependent methods we will cover are thresholding as a multiple hypothesis testing problem and SureShrink both described in [1]. The method of thresholding as a hypothesis test, works by for each coefficient $\tilde{\mathbf{W}}_{i,k} \sim N(\mathbf{W}_{i,k}, \hat{\sigma}^2)$testing the hypothesis

$$H_0 : \mathbf{W}_{i,k} = 0 \quad H_1 : \mathbf{W}_{i,k} \neq 0.$$

If H_0 is rejected the coefficient $\mathbf{W}_{i,k}$ is kept, and otherwise it is dropped from the model. If this hypothesis test is performed on an individual level the chance of wrongly keeping a coefficient is high, but if instead the test is performed simultaneously for all parameters the chance is low. The method for performing the test simultaneously is based on the False Discovery Rate (FDR) as described in [1]. The result of applying the method is a thresholding level λ^{FDR}, which can then be used with one of the thresholding techniques described.

3.2 Simulation

To evaluate the nonparametric regression methods covered in the previous section a small simulation study was performed for a number of test functions.

The code used for this simulation study was partially written for this thesis and otherwise freely available. The implementation of the Fourier smoothing method was done by the author of this thesis. The implementation of the DWT and the SureShrink method was available from the R library `waveslim`. The thresholding as a multiple hypothesis test was implemented by the author as well.

The test functions used for the simulation study were a subset of those used in [1] and [5]. The idea was to cover a wide variety of functions that are relevant in this field. The test functions used were

1. Corner

$$g_1(t) = 623.87t^3(1 - 4t)\mathcal{I}_{[0,0.5]}(t) + 187.161(0.125 - t^3)t^4\mathcal{I}_{(0.5,0.8]}(t) + 3708.470441(t - 1)^3\mathcal{I}_{(0.8,1]}(t).$$

2. Spikes

$$g_2(t) = 15.6676e^{-500(t-0.23)^2} + 2e^{-2000(t-0.33)^2} + 4e^{-8000(t-0.47)^2} + 3e^{-16000(t-0.69)^2} + e^{32000(t-0.83)^2}.$$

3. Angles

$$g_3(t) = (2t+0.5))\mathcal{I}_{[0,0.15]}(t) + (-12(t-0.15)+0.8)\mathcal{I}_{(0.15,0.2]}(t) + 0.2\mathcal{I}_{(0.2,0.5]}(t) + (6(t-0.5)+0.2)\mathcal{I}_{(0.5,0.6]}(t) + (-10(t-0.6)+0.8)\mathcal{I}_{(0.6,0.65]}(t) + (-5(t-0.65)+0.3)\mathcal{I}_{(0.65,0.85]}(t) + (2(t-0.85)+0.2)\mathcal{I}_{(0.85,1]}(t).$$

4. Bumps

$$g_4(t) = \sum_{j=1}^{11} h_j K\left(\frac{t-t_j}{w_j}\right), \quad K(t) = (1+|t|)^{-4},$$
$$t_j = (0.1,\ 0.13,\ 0.15,\ 0.23,\ 0.25,\ 0.4,\ 0.44,\ 0.65,\ 0.76,\ 0.78,\ 0.81),$$
$$h_j = (4,\ 5,\ 3,\ 4,\ 5,\ 4.2,\ 2.1,\ 4.3,\ 3.1,\ 5.1,\ 4.2),$$
$$w_j = (0.005,\ 0.005,\ 0.006,\ 0.01,\ 0.01,\ 0.03,\ 0.01,\ 0.01,\ 0.005,\ 0.008,\ 0.005).$$

5. Blocks

$$g_5(t) = \sum_{j=1}^{11} h_j K(t-t_j), \quad K(t) = \frac{1+\mathrm{sgn}(t)}{2},$$
$$t_j = (0.1,\ 0.13,\ 0.15,\ 0.23,\ 0.25,\ 0.4,\ 0.44,\ 0.65,\ 0.76,\ 0.78,\ 0.81),$$
$$h_j = (4,\ -5,\ 3,\ -4,\ 5,\ -4.2,\ 2.1,\ 4.3,\ -3.1,\ 2.1,\ -4.2).$$

6. Heavisine

$$g_6(t) = 4\sin(4\pi t) - \mathrm{sgn}(t-0.3) - \mathrm{sgn}(0.72-t).$$

7. Doppler

$$g_7(t) = \left[t(1-t)^{\frac{1}{2}}\sin\left(2\pi\frac{1.05}{t+1.05}\right)\right].$$

Where $\mathcal{I}_{[a,b]}(t)$ denotes the indicator function on the interval $[a,b]$. These test functions were defined for $t \in [0,1]$ so the values $t_i \in [0,\pi]$ were scaled by $\frac{1}{\pi}$ when used as inputs to the functions. As in [5] the functions were scaled so that the signal to noise ratio was 7, where the signal to noise ratio is defined as $\sqrt{\mathrm{Var}\ [f]}/\sigma$ where σ is the noise level, and $\mathrm{Var}(f) = \mathrm{Var}[f(X)]$, when X is uniformly distributed on $[0,\pi]$. Plots of the test functions can be found in Appendix A.

The wavelet filters used for the simulations were,

1. Haar
$$\{h_0 = 0.7071068,\ h_1 = -0.707106\}$$

2. D(4)
$$\{h_0 = -0.129410,\ h_1 = -0.224144,\ h_2 = 0.836516,\ h_3 = -0.482963\}$$

3. D(8)
$$\{h_0 = -0.010597,\ h_1 = -0.032883,\ h_2 = 0.030841,\ h_3 = 0.187035,\\ h_4 = -0.027984,\ h_5 = -0.630881,\ h_6 = 0.714847,\ h_7 = -0.230378\}$$

4. D(16)
$$\{h_0 = -0.000117,\ h_1 = -0.000675,\ h_2 = -0.000392,\ h_3 = 0.004870,\\ h_4 = 0.008746,\ h_5 = -0.013981,\ h_6 = -0.044088,\ h_7 = 0.017369,\\ h_8 = 0.128747,\ h_9 = -0.000472,\ h_{10} = -0.284016,\ h_{11} = 0.015829,\\ h_{12} = 0.585355,\ h_{13} = -0.675631,\ h_{14} = 0.312872,\ h_{15} = -0.054416\}$$

5. LA(8)
$$\{h_0 = -0.075766,\ h_1 = -0.029636,\ h_2 = 0.497619,\ h_3 = 0.803739,\\ h_4 = 0.297858,\ h_5 = -0.099220,\ h_6 = -0.012604,\ h_7 = 0.032223,\ \}$$

6. LA(16)
$$\{h_0 = -0.003382,\ h_1 = -0.000542,\ h_2 = 0.031695,\ h_3 = 0.007607,\\ h_4 = -0.143294,\ h_5 = -0.061273,\ h_6 = 0.481360,\ h_7 = 0.777186,\\ h_8 = 0.364442,\ h_9 = -0.051946,\ h_{10} = -0.027219,\ h_{11} = 0.049137,\\ h_{12} = 0.003809,\ h_{13} = -0.014952,\ h_{14} = -0.000303,\ h_{15} = 0.001890\}$$

For the Fourier smoothing $K = 5, 10, 15, \ldots, 60, 65, 70, 72, 75, 77, 80, 82, 85, 87, 90, 92, 95, 97, 100$ cosine terms were used.
For all choices of test function, threshold method and wavelet filter 100 simulations were performed and the evaluation criterion was averaged over these runs. The same was done for the Fourier smoothing method but instead simulations were repeated for each choice of test function and K. .

The evaluation criteria used were

1. **RMSE**: Root Mean Squared Error
$$\mathrm{RMSE}(\mathbf{y}, \tilde{\mathbf{y}}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\mathbf{y}_i - \tilde{\mathbf{y}}_i)^2}$$

2. **MAE**: Mean Absolute Error

$$\text{MAE}(\mathbf{y}, \tilde{\mathbf{y}}) = \frac{1}{N} \sum_{i=1}^{N} |\mathbf{y}_i - \tilde{\mathbf{y}}_i|$$

3. **sMAPE**: symmetric Mean Absolute Percentage Error

$$\text{sMAPE}(\mathbf{y}, \tilde{\mathbf{y}}) = \frac{1}{N} \sum_{i=1}^{N} \frac{|\mathbf{y}_i - \tilde{\mathbf{y}}_i|}{|\mathbf{y}_i| + |\tilde{\mathbf{y}}_i|}$$

Where $\tilde{\mathbf{y}}_i = \tilde{g}(t_i)$ is the ith value estimated from the test function with added noise and $\mathbf{y}_i = g(t_i)$ is the ith true value of the test function without any added noise.

4 Results

The results from the simulation study as described in Section 3.2 will be presented in the form of figures with the evaluation criteria. In figure 2 the evaluation criteria for the Fourier Smoothing is plotted as a function of K. From the figure we can see that all of the evaluation criteria decrease with increasing K, except for $K = 80$. For this choice of K the value of all three criteria spike for the functions, Spikes, Heavisine, Corner, as well as a much less noticeable rise for Blocks.

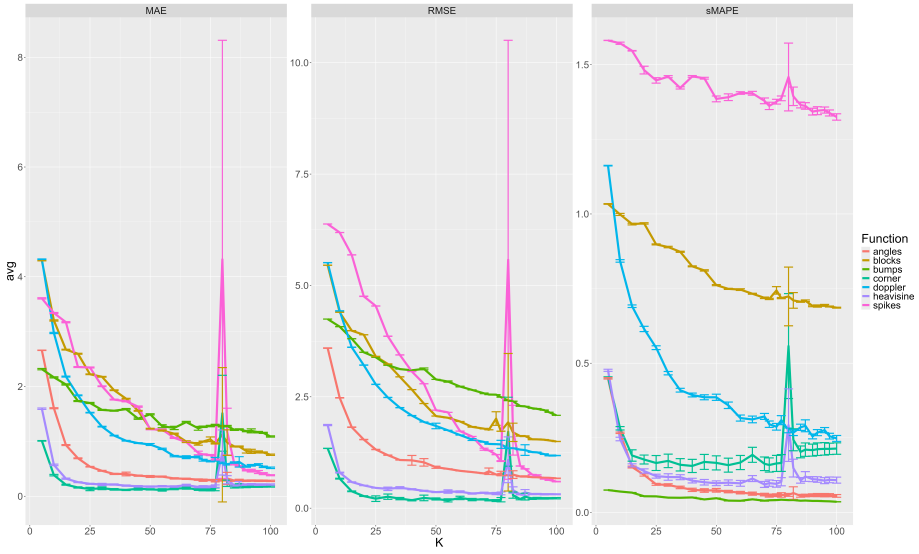

Figure 2: Evaluation criteria MAE, RMSE and sMAPE as a function of the number of cosine terms K for Fourier Smoothing.

In figure 3 we can see the smoothing parameters for the Fourier Smoothing method. We can see that the spikes around $K = 80$ in figure 2 are not present in the plot of CV in figure 3. This is probably due to the fact that the errors in figure 2 were calculated against the true value of $\mathbf{y}$ containing no added noise, where CV was calculated using the noisy data that the model was fitted to.

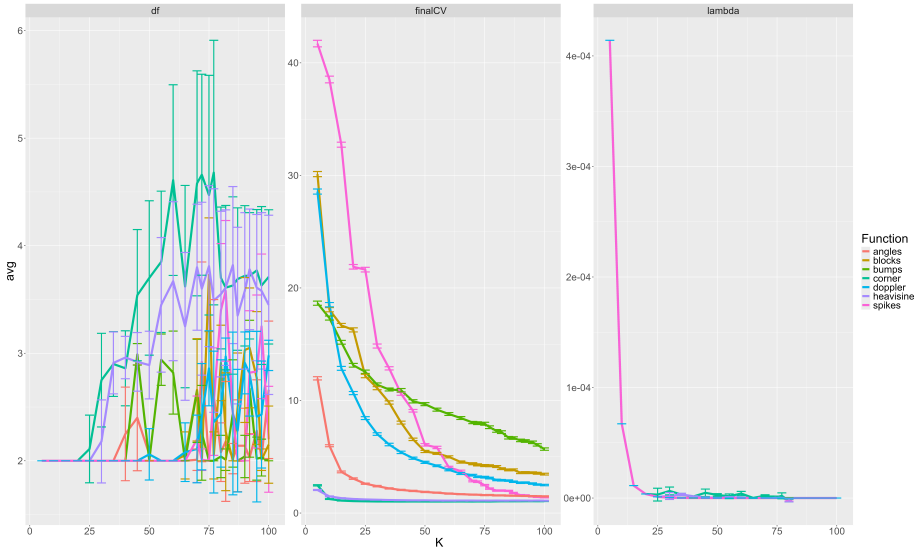

Figure 3: The parameters df,λ and CV as a function of K for Fourier Smoothing.

In figures 4-9 we can see the evaluation criterion for different wavelet filters in the columns and different choice of thresholding parameter λ in the rows. Figures 4,6 and 8 show the three evaluation criteria using hard thresholding, while Figures 5,7 and 9 show the three evaluation criteria using soft thresholding.

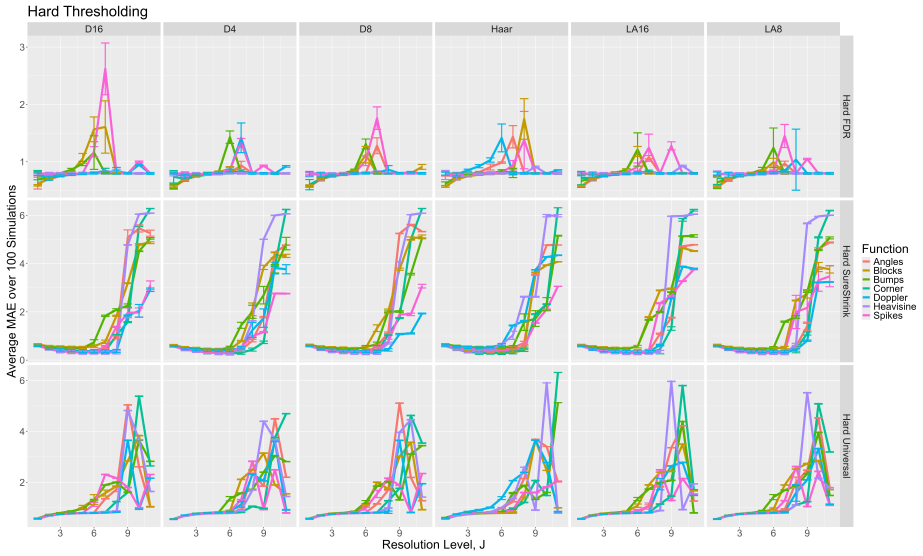

Figure 4: MAE as a function of Resolution Level J, for Hard Thresholding. The first row uses the thresholding level λ^{FDR} the second uses SureShrink and the third uses λ^{U}

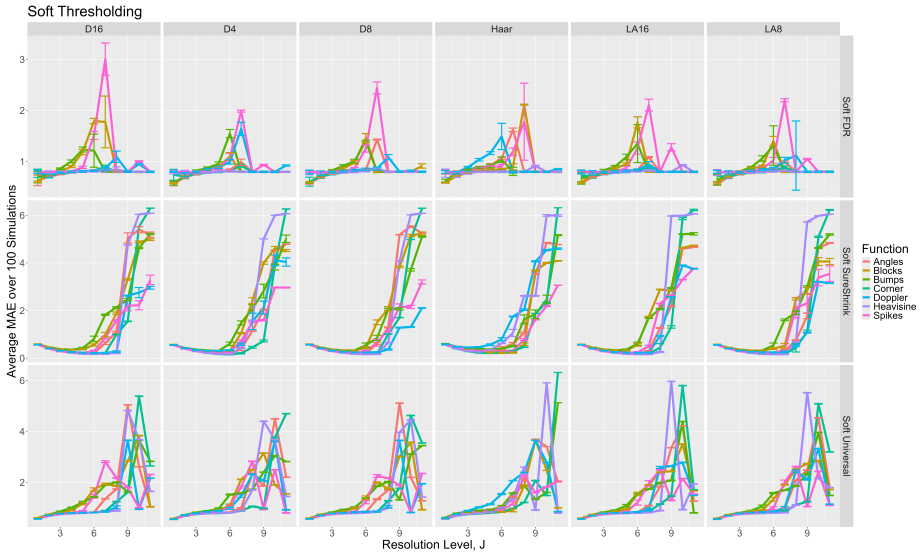

Figure 5: MAE as a function of Resolution Level J, for Soft Thresholding. The first row uses the thresholding level λ^{FDR} the second uses SureShrink and the third uses λ^{U}.

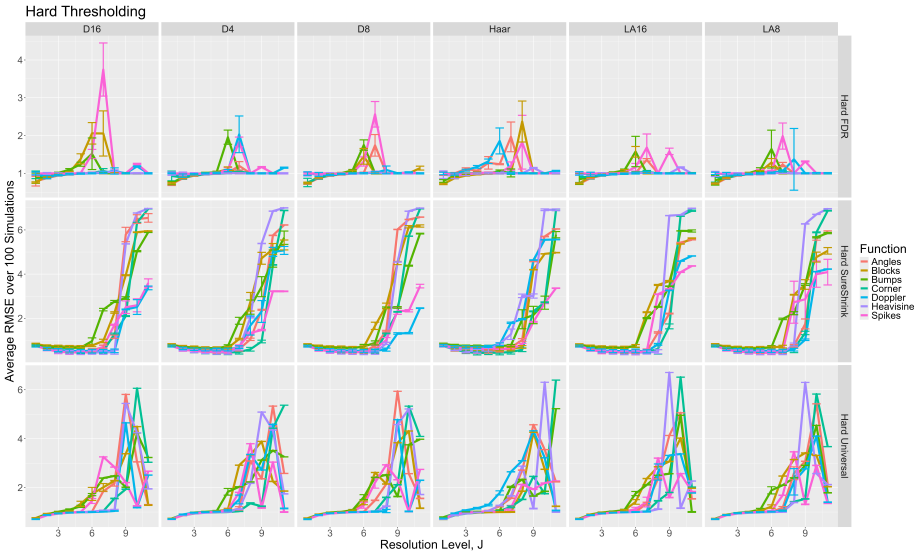

Figure 6: RMSE as a function of Resolution Level J, for Hard Thresholding. The first row uses the thresholding level λ^{FDR} the second uses SureShrink and the third uses λ^{U}.

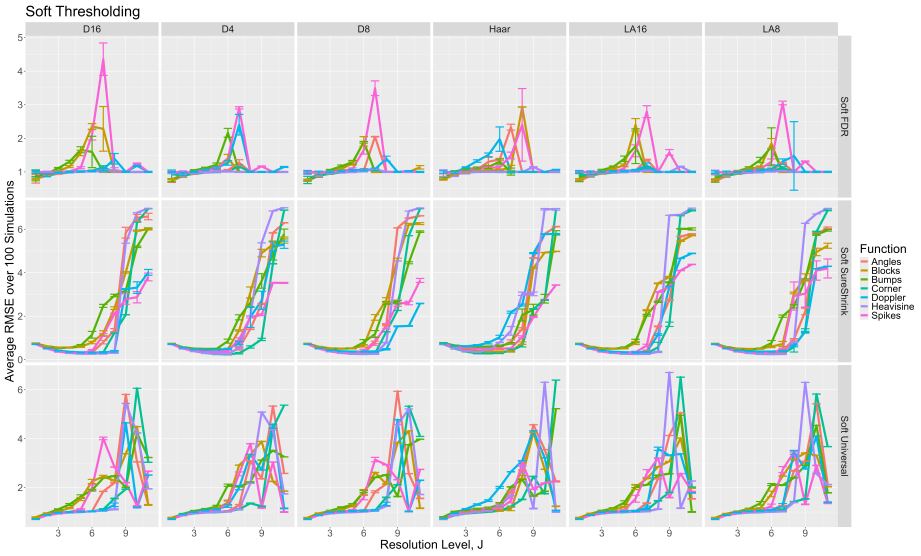

Figure 7: RMSE as a function of Resolution Level J, for Soft Thresholding. The first row uses the thresholding level λ^{FDR} the second uses SureShrink and the third uses λ^{U}

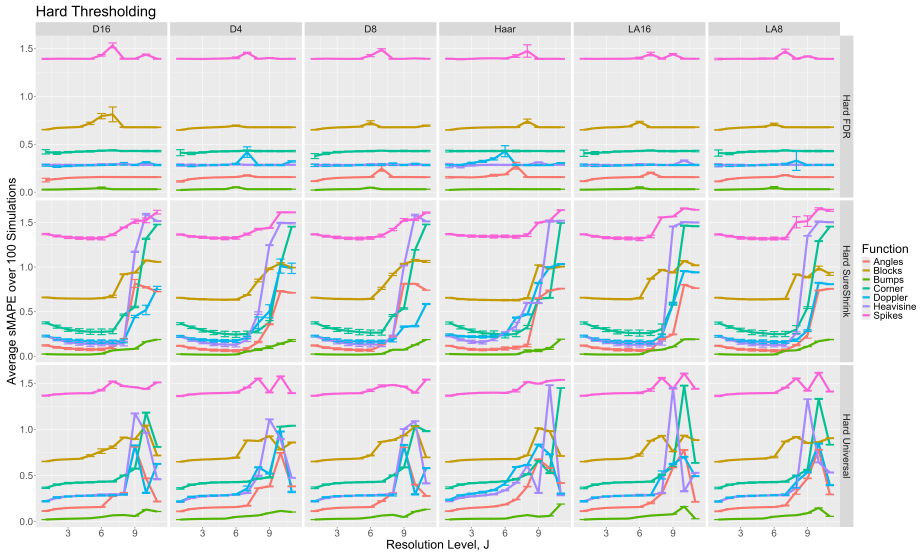

Figure 8: sMAPE as a function of Resolution Level J, for Hard Thresholding. The first row uses the thresholding level λ^{FDR} the second uses SureShrink and the third uses λ^{U}

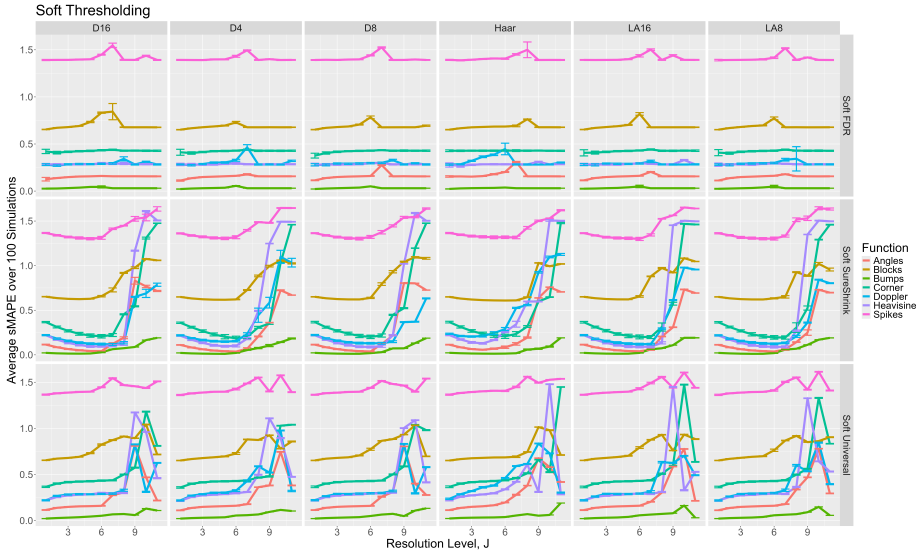

Figure 9: sMAPE as a function of Resolution Level J, for Soft Thresholding. The first row uses the thresholding level λ^{FDR} the second uses SureShrink and the third uses λ^{U}

For the SureShrink thresholding we can see that the values of all evaluation

criteria decrease when the resolution level of the transform increases up until around $J = 7$, where there is a sharp rise. But for the universal and FDR thresholds the evaluation criterion look more or less the same if not increasing. Similar spikes as for SureShrink are seen, but at what resolution level varies more. The universal threshold seems to have the sharpest increase for $J \approx 6$ but it slightly varies between evaluation methods and choice of wavelet filter. Although the evaluation criteria also seem to drop down to the value before the spike when resolution level is increased further. For most subplots the FDR threshold seems to have its peak at around $J = 8$.

These sharp peaks with subsequent drop are hard to explain, and could have several causes. As they are present in both figure 2 and figures 4-9 it is likely not only due to the nature of the models. We can also see in for example 2 that the rise in MAE and RMSE is by far the largest for the function spike which can be explained by the characteristics of the function itself, but the location of the spike is hard to explain.

We will now present the methods that performed best for each of the test functions with respect to the three different criterion. As well as presenting the parameters i.e number of cosine terms K, choice of wavelet filter and resolution level that achieved this.

1. Corner

   Comparing table 1 and 2 we can see that the Fourier smoothing method outperformed the best wavelet methods in all criteria. We can also see that the number of cosine terms K that performed best depended on the criterion, but the wavelet and thresholding method were the same for all criteria.

Table 1: Best performing Fourier method with respect to the three evaluation criterion, for the test function corner.

K	RMSE	MAE	sMAPE
75	0.1705478	-	-
55	-	0.1149294	-
40	-	-	0.1559033

Table 2: Best performing wavelet method with respect to the three evaluation criterion, for the test function corner.

Wavelet Filter	Thresholding Method	Resolution Level	RMSE	MAE	sMAPE
D4	Soft SureShrink	6	0.2404435	-	-
D4	Soft SureShrink	6	-	0.1574096	-
D4	Soft SureShrink	6	-	-	0.1851969

2. Spikes

Looking at table 3 and 4 we can see that the best wavelet thresholding methods outperformed Fourier smoothing in all criteria. We can also see that the Fourier smoothing using the maximum number of cosine terms K performed best for each criteria. But that different resolution levels and wavelet filters were used for the best performance on each criteria.

Table 3: Best performing Fourier method with respect to the three evaluation criterion, for the test function spikes.

K	RMSE	MAE	sMAPE
100	0.5969305	-	-
100	-	0.3832771	-
100	-	-	1.324863

Table 4: Best performing wavelet method with respect to the three evaluation criterion, for the test function spikes.

Wavelet Filter	Thresholding Method	Resolution Level	RMSE	MAE	sMAPE
LA8	Soft SureShrink	5	0.2940816	-	-
LA8	Soft SureShrink	5	-	0.2148403	-
LA16	Soft SureShrink	6	-	-	1.299789

3. Angles

Comparing table 5 and 6 we can see that as for the function spikes, the wavelet thresholding method outperformed Fourier smoothing in all criteria. The best performing Fourier method also used the maximum number of cosine terms just as for Spikes, and the best performing wavelet methods all used the same wavelet filter and thresholding method.

Table 5: Best performing Fourier method with respect to the three evaluation criterion, for the test function angles.

K	RMSE	MAE	sMAPE
100	0.6701377	-	-
100	-	0.2797862	-
100	-	-	0.05542024

Table 6: Best performing wavelet method with respect to the three evaluation criterion, for the test function angles.

Wavelet Filter	Thresholding Method	Resolution Level	RMSE	MAE	sMAPE
D4	Soft SureShrink	6	0.2714624	-	-
D4	Soft SureShrink	6	-	0.1792679	-
D4	Soft SureShrink	6	-	-	0.03626487

4. Bumps

   From table 7 and 8 we can see that the Fourier smoothing method was significantly outperformed w.r.t RMSE and MAE compared to wavelet thresholding, and somewhat less w.r.t sMAPE. Once again as for the functions spikes and angles the maximum number of cosine terms were used for the Fourier Smoothing, whereas different wavelet filters and resolution levels were used for the wavelet methods.

Table 7: Best performing Fourier method with respect to the three evaluation criterion, for the test function bumps.

K	RMSE	MAE	sMAPE
100	2.084805	-	-
100	-	1.089198	-
100	-	-	0.03578664

Table 8: Best performing wavelet method with respect to the three evaluation criterion, for the test function bumps.

Wavelet Filter	Thresholding Method	Resolution Level	RMSE	MAE	sMAPE
LA8	Soft SureShrink	4	0.4700393	-	-
LA8	Soft SureShrink	4	-	0.3102521	-
D4	Soft SureShrink	5	-	-	0.01086009

5. Blocks

   Comparing table 9 and 10 we can see that wavelet thresholding performed the best for all criteria, especially for RMSE and MAE. The Fourier method using the maximum number of cosine terms performed the best on all criteria. Unlike the other test functions the Haar wavelet using resolution level 6 resulted in the best performance w.r.t all evaluation criteria.

Table 9: Best performing Fourier method with respect to the three evaluation criterion, for the test function blocks.

K	RMSE	MAE	sMAPE
100	1.497180	-	-
100	-	0.7563130	-
100	-	-	0.6860814

Table 10: Best performing wavelet method with respect to the three evaluation criterion, for the test function blocks.

Wavelet Filter	Thresholding Method	Resolution Level	RMSE	MAE	sMAPE
Haar	Soft SureShrink	6	0.3694584	-	-
Haar	Soft SureShrink	6	-	0.2254934	-
Haar	Soft SureShrink	6	-	-	0.6089576

6. Heavisine

   From table 11 and 12 we can see that Fourier smoothing was outperformed slightly on all evaluation criteria, and that different number of cosine terms performed the best w.r.t different evaluation criteria. The wavelet thresholding methods all used the same resolution level of 6 but used different wavelet filters.

Table 11: Best performing Fourier method with respect to the three evaluation criterion, for the test function heavisine.

K	RMSE	MAE	sMAPE
85	0.3092343	-	-
70	-	0.1693416	-
70	-	-	0.09396342

Table 12: Best performing wavelet method with respect to the three evaluation criterion, for the test function heavisine.

Wavelet Filter	Thresholding Method	Resolution Level	RMSE	MAE	sMAPE
D8	Soft SureShrink	6	0.2541370	-	-
LA8	Soft SureShrink	6	-	0.1646360	-
LA8	Soft SureShrink	6	-	-	0.08805781

7. Doppler

   From table 13 and 14 we can see that the wavelet thresholding outperformed the Fourier smoothing w.r.t all evaluation criterion, and that the maximum or almost maximum number of cosine terms performed the best. For the wavelet thresholding methods the same wavelet filter but slightly different resolution levels resulted in the best performance.

Table 13: Best performing Fourier method with respect to the three evaluation criterion, for the test function doppler.

K	RMSE	MAE	sMAPE
97	1.180594	-	-
100	-	0.5180512	-
100	-	-	0.2480506

Table 14: Best performing wavelet method with respect to the three evaluation criterion, for the test function doppler.

Wavelet Filter	Thresholding Method	Resolution Level	RMSE	MAE	sMAPE
D16	Soft SureShrink	6	0.3239850	-	-
D16	Soft SureShrink	6	-	0.2166013	-
D16	Soft SureShrink	7	-	-	0.1167665

5 Conclusions

Both the Fourier smoothing method and wavelet thresholding methods showed promising results for several of the test functions. Although the Fourier methods were outperformed for all test functions except corner where it outperformed the wavelet methods in all evaluation criteria. This was a surprising result as some of the test functions like heavisine and Doppler are variations of sine waves. But them not just being superpositioned sine waves could explain why the wavelet methods performed better on them. Another surprising result was that the best performing wavelet methods all used soft SureShrink thresholding for all test functions and w.r.t all three criteria. The wavelet filters that seemed to perform best on the most test function were the D4 and LA8 filters, but it was also the D4 wavelet filter that was outperformed by Fourier smoothing as seen in table 1 and 2. The best performing resolution level was also quite consistent at mostly 6 but 4, 5 or 7 also performed quite good. The Haar wavelet filter performed the best only for the test function blocks which was not surprising as it is "blocky" by nature. Further work should be done on investigating the reason for the spikes at $K = 80$ in figure 2, and the spikes in figures 4-9, but is due to time limitations outside the scope of this thesis.

References

[1] Anestis Antoniadis, Jeremie Bigot **and** Theofanis Sapatinas. "Wavelet Estimators in Nonparametric Regression: A Comparative Simulation Study". **in***Journal of Statistical Software*: 6.6 (2001), **pages** 1–83. DOI: 10.18637/jss.v006.i06. URL: https://www.jstatsoft.org/index.php/jss/article/view/v006i06.

[2] Martin Bilodeau. "Fourier smoother and additive models". **in***Canadian Journal of Statistics*: 20.3 (1992), **pages** 257–269. DOI: https://doi.org/10.2307/3315313. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.2307/3315313. URL: https://onlinelibrary.wiley.com/doi/abs/10.2307/3315313.

[3] Ingrid Daubechies. *Ten Lectures on Wavelets*. Society for Industrial **and** Applied Mathematics, 1992. DOI: 10.1137/1.9781611970104. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611970104. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611970104.

[4] Ingrid Daubechies. *Ten Lectures on Wavelets*. Society for Industrial **and** Applied Mathematics, 1992. DOI: 10.1137/1.9781611970104. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611970104. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611970104.

[5] David L Donoho **and** Iain M Johnstone. "Ideal spatial adaptation by wavelet shrinkage". **in***biometrika*: 81.3 (1994), **pages** 425–455.

[6] A. Friedman. *Foundations of Modern Analysis*. Dover Books on Mathematics Series. Dover, 1982. ISBN: 9780486640624.

[7] Hong-Ye Gao **and** Andrew G. Bruce. "WAVESHRINK WITH FIRM SHRINKAGE". **in***Statistica Sinica*: 7.4 (1997), **pages** 855–874. ISSN: 10170405, 19968507. URL: http://www.jstor.org/stable/24306159 (**urlseen** 13/04/2025).

[8] S. Geman. "The Method of Sieves." **in***Encyclopedia of Statistical Sciences*: 5 (1985), **pages** 473–476.

[9] S.G. Mallat. "A theory for multiresolution signal decomposition: the wavelet representation". **in***IEEE Transactions on Pattern Analysis and Machine Intelligence*: 11.7 (1989), **pages** 674–693. DOI: 10.1109/34.192463.

[10] D.B. Percival **and** A.T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2000. ISBN: 9780521640688. URL: https://books.google.se/books?id=XF4urPW362AC.

[11] A. Pinkus **and** S. Zafrany. *Fourier Series and Integral Transforms*. Fourier Series and Integral Transforms. Cambridge University Press, 1997. ISBN: 9780521597715. URL: https://books.google.se/books?id=uZgLLAS4vfoC.

[12] G. Wahba. *Spline Models for Observational Data*. CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial **and** Applied Mathematics, 1990. ISBN: 9780898712445. URL: https://books.google.se/books?id=ScRQJEETs0EC.

Appendix A Test functions

1. Corner

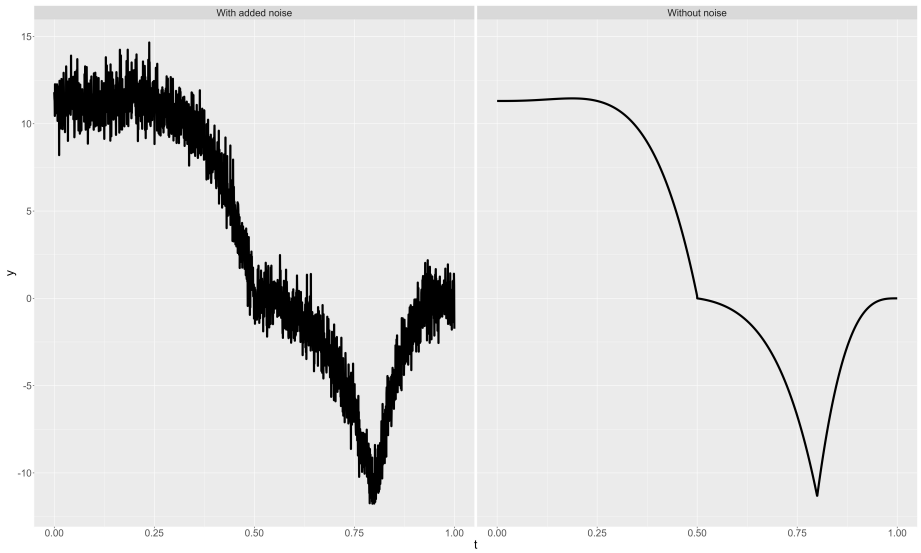

Figure 10: The test function Corner, rescaled so that the signal to noise ratio was 7 when noise was added. On the left we see the value of the test function with added noise, and on the right rescaled but without noise.

2. Spikes

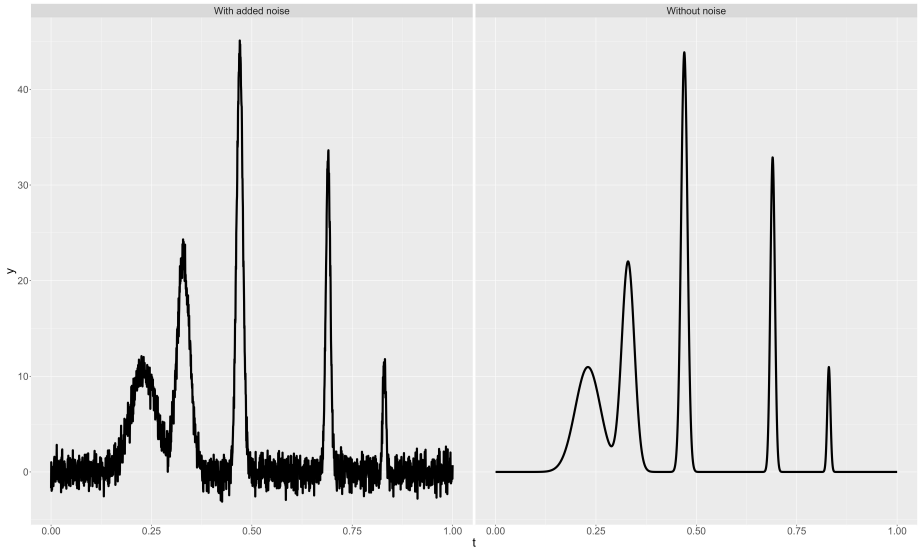

Figure 11: The test function Spikes, rescaled so that the signal to noise ratio was 7 when noise was added. On the left we see the value of the test function with added noise, and on the right rescaled but without noise.

3. Angles

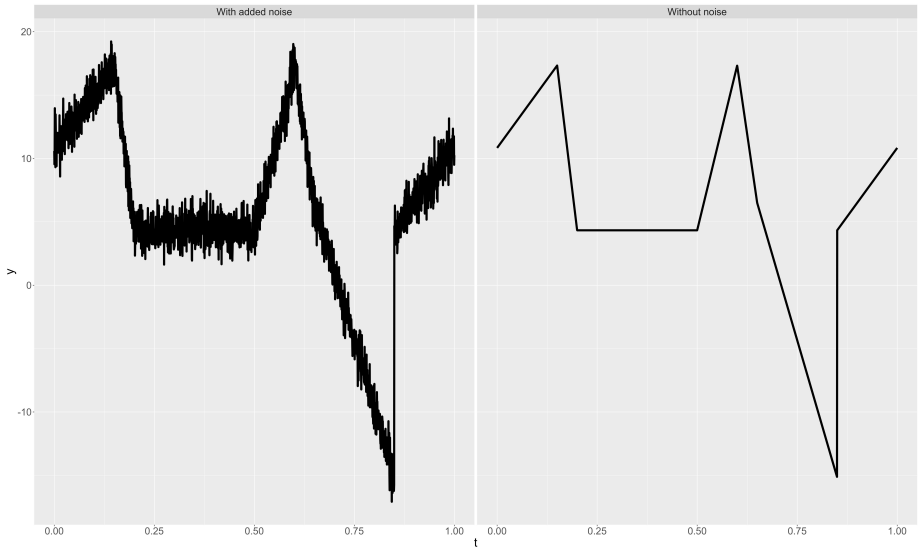

Figure 12: The test function Angles, rescaled so that the signal to noise ratio was 7 when noise was added. On the left we see the value of the test function with added noise, and on the right rescaled but without noise.

4. Bumps

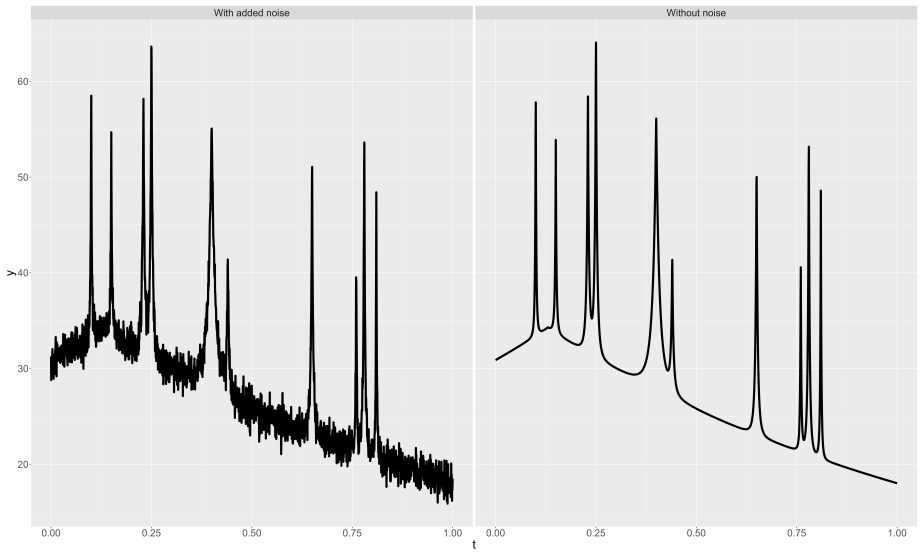

Figure 13: The test function Bumps, rescaled so that the signal to noise ratio was 7 when noise was added. On the left we see the value of the test function with added noise, and on the right rescaled but without noise.

5. Blocks

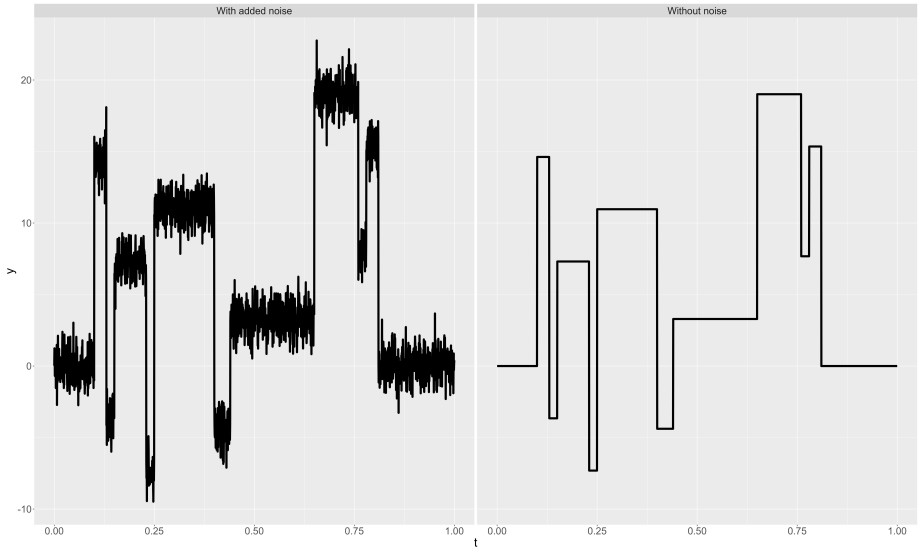

Figure 14: The test function Blocks, rescaled so that the signal to noise ratio was 7 when noise was. On the left we see the value of the test function with added noise, and on the right rescaled but without noise.

6. Heavisine

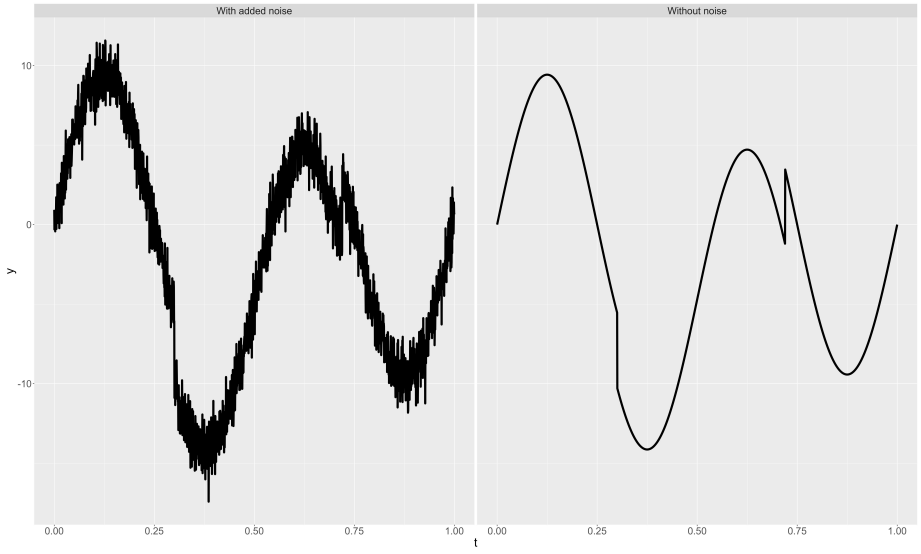

Figure 15: The test function Heavisine, rescaled so that the signal to noise ratio was 7 when noise was added. On the left we see the value of the test function with added noise, and on the right rescaled but without noise.

7. Doppler

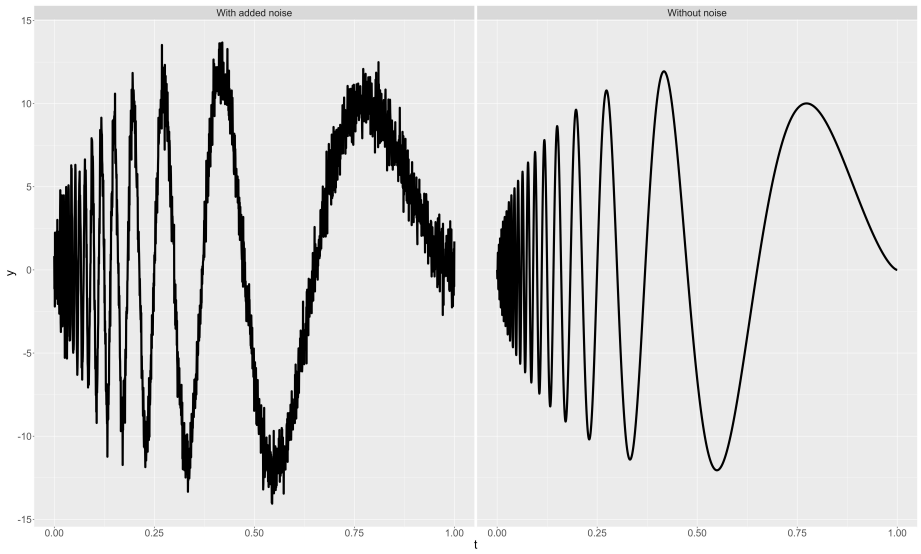

Figure 16: The test function Doppler, rescaled so that the signal to noise ratio was 7 when noise was added. On the left we see the value of the test function with added noise, and on the right rescaled but without noise.

Appendix B Nomenclature

Table 15: Some commonly used symbols and their meaning

Symbol	Description
	Sets
$\mathbb{Z}$	the integers $\{0,\ \pm 1,\ \pm 2, \dots\}$
$\mathbb{R}$	the real numbers
$\mathbb{C}$	the complex numbers
	Variables
$x,\ y,\ z$	arbitrary variables
t	time domain variable
$\omega,\ \zeta$	frequency domain variables
$m,\ n$	integer variables
$X,\ Y$	random variables
	Matrices
$\mathbb{I}_N$	the $N \times N$ identity matrix
$A^\top$	the transpose of the matrix A
	Functions
ψ	wavelet function
$\psi^{a,b}$	wavelet with parameters a and b
$f,\ g,\ h$	functions
$\hat{f},\ \hat{g},\ \hat{h}$	the fourier transform of functions $f,\ g,\ h$
$\overline{f},\ \overline{g},\ \overline{h}$	the complex conjugates of functions $f,\ g,\ h$
$\mathcal{I}_A(x)$	the indicator function on the set A

B.1 Acronyms

WFT Windowed Fourier transform

PDF Probability Density Function

DFT Discrete Fourier transform

DTFT Discrete Time Fourier transform

FFT Fast Fourier transform

DWT Discrete Wavelet Transform

ODFT Orthonormal Discrete Fourier transform

MRA Multi Resolution Analysis

CFT Continuous Fourier transform

CWT Continuous Wavelet Transform

ICWT Inverse Continuous Wavelet Transform

MSECV Mean Squared Error of Cross Validation

FDR False Discovery Rate

Appendix C Code

functions.R

```
#########################################################
#Test Functions
#########################################################
corner =function(x){
  y=vector(length=length(x))
  idx<-x<=0.5
  y[idx]=623.87*(x[idx])^3*(1-4*x[idx])
  y[idx]=(0.6/(max(y[idx])-min(y[idx])))*y[idx]+0.6
  idx=(x>0.5)&(x<=0.8)
  y[idx]=187.161*(0.125-x[idx]^3)*(x[idx]^4)
  y[idx]=(0.6/(max(y[idx])-min(y[idx])))*y[idx]
  idx=(x>0.8)&(x<=1)
  y[idx]=3708.470441*(x[idx]-1)^3
  y[idx]=(0.6/(max(y[idx])-min(y[idx])))*y[idx]

  return(y)
}
spikes=function(x)
{
  y=15.6676*(exp(-500*(x-0.23)^2)+2*exp(-2000*(x-0.33)^2)
      +4*exp(-8000*(x-0.47)^2)+3*exp(-16000*(x-0.69)^2)+
      exp(-32000*(x-0.83)^2))
  y=(0.6/(max(y)-min(y)))*y
  return(y)
}
angles=function(x)
{
  y=vector(length=length(x))
  idx<-x<=0.15
  y[idx]=(2*x[idx]+0.5)
  idx<-(x>0.15)&(x<=0.2)
  y[idx]=(-12*(x[idx]-0.15)+0.8)
  idx<-(x>0.2)&(x<=0.5)
  y[idx]=0.2
  idx<-(x>0.5)&(x<=0.6)
  y[idx]=(6*(x[idx]-0.5)+0.2)
```

```
  idx<-(x>0.6)&(x<=0.65)
  y[idx]=(-10*(x[idx]-0.6)+0.8)
  idx<-(x>0.65)&(x<=0.85)
  y[idx]=(-5*(x[idx]-0.65)+0.3)
  idx<-(x>0.85)&(x<=1)
  y[idx]=(2*(x[idx]-0.85)+0.2)
  return(y)
}
bumps=function(x)
{


  h=c(4,5,3,4,5,4.2,2.1,4.3,3.1,5.1,4.2)
  w=c
      (0.005,0005,0.006,0.01,0.01,0.03,0.01,0.01,0.005,0.008,0.005)

  tj=c
      (0.1,0.13,0.15,0.23,0.25,0.4,0.44,0.65,0.76,0.78,0.81)

  K=function(t)1/((1+abs(t))^4)
  t_mat=matrix(nrow=length(tj),ncol=length(x))

  y=vector(length=length(x))
  for (i in 1:length(tj))
  {

    y=y+h[i]*K((x-tj[i])/w[i])
  }


  return(y)
}
blocks=function(x){

  h=c(4,-5,3,-4,5,-4.2,2.1,4.3,-3.1,2.1,-4.2)

  tj=c
      (0.1,0.13,0.15,0.23,0.25,0.4,0.44,0.65,0.76,0.78,0.81)

  K=function(t)(1+sign(t))/2
  t_mat=matrix(nrow=length(tj),ncol=length(x))
  for (i in 1:length(tj))
  {
    t_mat[i,]=(x-tj[i])
    t_mat[i,]=h[i]*K(t_mat[i,])
```

```
  }

  y=colSums(t_mat)
  return(y)}
heavisine=function(x){
  y=4*sin(4*pi*x)-sign(x-0.3)-sign(0.72-x)
  return(y)
}
doppler=function(x,epsilon=0.05){
  y=((x*(1-x))^(1/2))*sin(2*pi*(1+epsilon)/(x+epsilon))
  return(y)
}

############################################################
#Helper Functions
############################################################
rescale_add_noise=function(g_func,noise_level,signal_to_
    noise_ratio,n_points,tstart=0,tend=1){
  t=seq(from=tstart,to=tend,length.out=n_points)
  ytrue=g_func(t)
  noise=rnorm(n=n_points)*noise_level
  ytrue=ytrue*(7*noise_level/sd(ytrue))
  y_out=ytrue+noise
  return(list(yout=y_out,points=t,ytrue=ytrue,noise=noise
      ))
}
rescale_x=function(x,epsilon,wrapstart=0,wrapend=pi){
  start=wrapstart+epsilon
  end=wrapend*(1-epsilon)
  scaled_x=start+(x-min(x))*((end-start)/(max(x)-min(x)))
  return(scaled_x)
}
############################################################
#FOURIER SMOOTHING SETUP
############################################################
create_X_mat=function(t,K){
  ang_vel_vec=seq(from=0,to=K,length.out=K+1)
  X_row<-function(xi,ang_vel_vec){return(cos(xi*ang_vel_
      vec))}
  X_out=matrix(nrow=length(t),ncol=K+2)
  X_out[,0:K+2]=t(apply(X=t,MARGIN=1,FUN=function(xi)X_
      row(xi,ang_vel_vec)))
  X_out[,1]=t
  return(X_out)}
create_I_k=function(K){
  return(seq(from=1,to=K,length.out=K)^4)
```

```
}

create_D_tilde=function(K){
  vec=vector(length=K+2)
  vec[1:K+2]=create_I_k(K)
  return(diag(vec))
}

create_M_lambda=function(t,X,D,lambda)
{
  n=length(t)
  M_lambda=((1/n)*t(X)%*%X+lambda*D)
  return(M_lambda)
}

create_b_vec=function(X,y)
{
  return((1/length(y))*t(X)%*%y)
}


create_a_vec=function(M,b)
{
  return(inv(M)%*%b)
}
g_est=function(a,t,X)
{

  return(X%*%a)
}
create_S_lambda=function(n,X,M)
{
  S_lambda=(1/n)*X%*%solve(M)%*%t(X)
  return(S_lambda)
}
calculate_CV=function(n,y,ghat,Slambda){
  summand_vec=(y-ghat)/(1-diag(Slambda))
  return((1/n)*sum(summand_vec^2))
}
calculate_gamma=function(n,X,K){
  ninv_XtX=(1/n)*t(X)%*%X

  A=ninv_XtX[0:2,0:2]
  B=ninv_XtX[3:(2+K),0:2]
  C=ninv_XtX[3:(2+K),3:(2+K)]
  Ik=create_I_k(K)
```

```
  return(1/eigen(inv(sqrtm(diag(Ik)))%*%(C-B%*%inv(A)%*%t
      (B))%*%inv(sqrtm(diag(Ik))),only.values = TRUE)$
      values)
}
iterate_lambda=function(gamma_vec,df,lambdai){
  upper_sum=2*sum(1/(1+lambdai*gamma_vec))-df
  lower_sum=sum(gamma_vec/(1+lambdai*gamma_vec)^2)
  return(lambdai+upper_sum/lower_sum)
}
#####################################################
# Thresholding
#####################################################
soft_threshold=function(x_dwt,lambda) {

  wavelet_names<-names(x_dwt)[grepl("d(\\d+)",names(x_dwt
      ))]

  y=x_dwt
  for(i in length(wavelet_names)){

    x_i<-x_dwt[[wavelet_names[i]]]

    lower_cut<-x_i<(-lambda)
    upper_cut<-x_i>lambda
    cutoff<-abs(x_i)<=lambda

    y[[wavelet_names[i]]][upper_cut]=y[[wavelet_names[i
        ]]][upper_cut]-lambda
    y[[wavelet_names[i]]][lower_cut]=y[[wavelet_names[i
        ]]][lower_cut]+lambda
    y[[wavelet_names[i]]][cutoff]=0

  }
  return(y)
}
FDR_threshold=function(x_dwt,noise_estimate,alpha=0.05) {
  wavelet_coeffs<-unlist(x_dwt[grepl("d(\\d+)",names(x_
      dwt))])
  p<-2*(1-pnorm(abs(as.vector(wavelet_coeffs))/noise_
      estimate))
  p<-sort(p,decreasing=FALSE)

  m=length(p)-1
  idx=seq(1,length(p))
  ok<-alpha*idx/m
  ok<-(p<ok)
```

```
  idx[!ok]=0
  k=max(idx)
  lambda_fdr<-noise_estimate*qnorm(1-p[k]/2)



  return(lambda_fdr)
}
hard_threshold=function(x_dwt,lambda) {

  wavelet_names<-names(x_dwt)[grepl("d(\\d+)",names(x_dwt
      ))]

  y=x_dwt
  for(i in length(wavelet_names)){

    x_i<-x_dwt[[wavelet_names[i]]]
    cutoff=(abs(x_i)<=lambda)

    y[[wavelet_names[i]]][cutoff]=0

  }

  return(y)
}
MAD_noise_est=function(y_dwt){
  wavelet_names<-names(y_dwt)[grepl("d(\\d+)",names(y_dwt
      ))]

  n_coeffs=0
  for(i in length(wavelet_names)){
    n_coeffs=n_coeffs+length(y_dwt[[wavelet_names[i]]])


  }
  y=vector(length = n_coeffs)
  start_idx=0
  end_idx=0
  for(i in length(wavelet_names)){
    end_idx=end_idx+length(y_dwt[[wavelet_names[i]]])
    y[start_idx+1:end_idx]=y_dwt[[wavelet_names[i]]]
    start_idx=end_idx



  }
```

```
  noise_est=median(abs(y))/0.6745
  return(noise_est)
}
```

functions.R

```
---
title: "Evaluation"
author: "Pontus Masip"
date: "`r Sys.Date()`"
output: pdf_document
---

```{r setup, include=FALSE}
library(waveslim)
library(ggplot2)
library(expm)
library(matlib)
library(tidyr)
library(dplyr)
library(Metrics)
library(doParallel)
library(tibble)
source('functions.R')
```

```{r message=TRUE, warning=TRUE}
fit_fourier_smoother=function(y_obs,t_obs,K=20,epsilon_t
    =0.05){
  t_wrapped=rescale_x(as.array(t_obs),
                      epsilon=epsilon_t,
                      wrapstart=0,
                      wrapend=pi)
  n_points=length(t_obs)

  lambda_old=0
  df=2


  X=create_X_mat(t_wrapped,K)
  D=create_D_tilde(K)
  M=create_M_lambda(t_wrapped,X,D,lambda_old)
  b=create_b_vec(X,y_obs)
  a=create_a_vec(M,b)
  S_lambda=create_S_lambda(n_points,X,M)
  g_hat=g_est(a,t_wrapped,X)
```
```

```
  gamma=calculate_gamma(n_points,X,K)
  old_crossval=calculate_CV(n_points,y_obs,g_hat,S_lambda
      )
  lambda_old=iterate_lambda(gamma,df,lambda_old)

  finished=FALSE

  while(!finished)
  {
    M=create_M_lambda(t_wrapped,X,D,lambda_old)
    a=create_a_vec(M,b)
    S_lambda=create_S_lambda(n_points,X,M)
    g_hat=g_est(a,t_wrapped,X)

    new_crossval=calculate_CV(n_points,y_obs,g_hat,S_
        lambda)

    lambda_new=iterate_lambda(gamma,df,lambda_old)

    if(new_crossval<=old_crossval){
      df=df+1
      lambda_old=lambda_new
      old_crossval=new_crossval

    }
    else{
      finished=TRUE
    }
  }
  return(list(lambda=lambda_old,df=df,S_lambda=S_lambda,g
      _hat=g_hat,M_lambda=M,old_CV=old_crossval,new_CV=new
      _crossval))
}
```

```{r}
fourier_smoother_df=tibble(Function=character(),K=numeric
    (),Iteration=numeric(),RMSE=numeric(),MSE=numeric(),
                           MAE=numeric(),R2=numeric(),
                               MAPE=numeric(),MASE=numeric
                               (),sMAPE=numeric(),RAE=
                               numeric(),
                           RSE=numeric(),RRSE=numeric(),
                               SSE=numeric(),RMSLE=numeric
                               (),MSLE=numeric(),lambda=
                               numeric(),finalCV=numeric()
```

```
                                        ,df=numeric())
function_names=c("Spikes","Blocks","Waves","Doppler")
i=1
ytrue=test$ytrue
ypred=out$g_hat
K_current=20
funcs=list(functions=c(spikes,corner,angles,bumps,blocks,
    heavisine,doppler),func_names=c("spikes","corner","
    angles","bumps","blocks","heavisine","doppler"))
j=1

```

```{r}
fit_fourier_and_return_df=function(i,noise_level,n_points
    ,ytrue,t,K_current,epsilon_t,iter_per_func,current_
    func_name){

  noise=noise_level*rnorm(n_points)
  y_noisy=ytrue+noise
  fitted_smoother=fit_fourier_smoother(y_obs=y_noisy,t_
      obs=t,K=K_current,epsilon_t = epsilon_t)
  ypred=fitted_smoother$g_hat
  df_list=tibble_row(Function=current_func_name,
                                  Iteration=i,
                                  K=K_current,
                                  RMSE=rmse(actual=ytrue,
                                      predicted=ypred),
                                  MSE=mse(actual=ytrue,
                                      predicted=ypred),
                                  MAE=mae(actual=ytrue,
                                      predicted=ypred),
                                  R2=1-sum((ypred - ytrue)^2)
                                      /sum((ytrue - mean(ytrue))
                                       ^2),
                                  MAPE=mape(actual=ytrue,
                                      predicted=ypred),
                                  MASE=mase(actual=ytrue,
                                      predicted=ypred),
                                  sMAPE=smape(actual=ytrue,
                                      predicted=ypred),
```

```
                                    RAE=rae(actual=ytrue,
                                        predicted=ypred),
                                    RSE=rse(actual=ytrue,
                                        predicted=ypred),
                                    RRSE=rrse(actual=ytrue,
                                        predicted=ypred),

                                    SSE=sse(actual=ytrue,
                                        predicted=ypred),
                                    RMSLE=rmsle(actual=ytrue,
                                        predicted=ypred),

                                    MSLE=msle(actual=ytrue,
                                        predicted=ypred),
                                df=fitted_smoother$df,
                                lambda=fitted_smoother$lambda,
                                finalCV=fitted_smoother$old_CV)


  return(df_list)
}
```

```
```{r message=FALSE, warning=FALSE}
fourier_smoother_df=tibble(Function=character(),K=numeric
    (),Iteration=numeric(),RMSE=numeric(),MSE=numeric(),
                                MAE=numeric(),R2=numeric(),
                                    MAPE=numeric(),MASE=numeric
                                    (),sMAPE=numeric(),RAE=
                                    numeric(),
                                RSE=numeric(),RRSE=numeric(),
                                    SSE=numeric(),RMSLE=numeric
                                    (),MSLE=numeric(),lambda=
                                    numeric(),finalCV=numeric()
                                    ,df=numeric())
iter_per_func=100
K_current=20
dataf=fourier_smoother_df
yactual=test$yout
t=seq(from=0,to=1,length.out=2^11)

epsilon_t=0.05
noise_level=1
n_points=length(t)
K_list=c
    (5,10,15,20,25,30,35,40,45,50,55,60,65,70,72,75,77,80,82,85,87,90,92,95,97,10
```
```

```
 nThreads <- detectCores()-1

clusterOfThreads <- makeCluster(nThreads)


clusterEvalQ(clusterOfThreads, {

  library(Metrics)
  library(matlib)
  library(expm)
  source('functions.R')

})
doParallel::registerDoParallel(clusterOfThreads, cores=
    nThreads)
set.seed(1972)
for (ij in 1:length(K_list)){
  K_current=K_list[ij]
  print("K:")
  print(K_current)
for (j in 1:length(funcs$func_names)){
  current_func=funcs$functions[j][[1]]
  current_func_name=funcs$func_names[j]
  ytrue=current_func(t)

  noise=rnorm(n=n_points)*noise_level
  ytrue=ytrue*(7*noise_level/sd(ytrue))
  print("Function:")
  print(current_func_name)


  clusterExport(clusterOfThreads, varlist = c("fit_
      fourier_and_return_df",
                                              "fit_fourier_
                                                  smoother",
                                              "rmse", "mse"
                                                  , "mae", "
                                                  mape", "
                                                  mase", "
                                                  smape",
                                              "rae", "rse",
                                                   "rrse", "
                                                  sse", "
                                                  rmsle", "
```

```
                                                    msle",
                                                "noise_level"
                                                    , "tibble_
                                                    row","n_
                                                    points","
                                                    ytrue","t"
                                                    ,"K_
                                                    current","
                                                    epsilon_t"
                                                    ,"iter_per
                                                    _func","
                                                    current_
                                                    func_name"
                                                    ,"rescale_
                                                    x","
                                                    rescale_
                                                    add_noise"
                                                    ))
  list_of_df=clusterApply(cl=clusterOfThreads,
                          x=seq(length.out=iter_per_func
                              ),
                          fun=function(i) fit_fourier_
                              and_return_df(i,noise_level
                              =noise_level,n_points=n_
                              points,ytrue=ytrue,t=t,K_
                              current=K_current,epsilon_t
                              =epsilon_t,iter_per_func=
                              iter_per_func,current_func_
                              name = current_func_name ))
  dataf=dataf%>%bind_rows(list_of_df)

}}
stopCluster(clusterOfThreads)


write.csv(dataf,"fourier_smoothing_stats.csv")
```
```
```{r}
test_df=dataf

test_df=test_df%>%group_by(Function,K)%>%summarise(RMSE=
    mean(RMSE),
                                   MSE=mean(MSE),
                                   MAE=mean(MAE),
                                   R2=mean(R2),
                                   MAPE=mean(MAPE),
```
```

```
                                    MASE=mean(MASE),
                                    sMAPE=mean(sMAPE),
                                    RAE=mean(RAE),
                                    RSE=mean(RSE),
                                    RRSE=mean(RRSE),

                                    SSE=mean(SSE),
                                    RMSLE=mean(RMSLE),

                                    MSLE=mean(MSLE),lambda=mean(
                                        lambda),df=median(df),
                                        finalCV=mean(finalCV))
test_df
ggplot(data=test_df,mapping=aes(x=K,y=RMSE,color=Function
    ))+geom_line()

```

````
```{r}

fit_wavelet_and_return_df=function(i,current_func_name,
    ytrue,current_wf,resolution_level,boundary){

  noise=noise_level*rnorm(n_points)
  y_noisy=ytrue+noise
  y_dwt=dwt(y_noisy,wf=current_wf,n.levels=resolution_
      level,boundary=boundary)

  noise_estimate<-MAD_noise_est(y_dwt)
  universal_threshold_lambda=noise_estimate*sqrt(2*log(n_
      points))

  FDR_threshold_lambda<-FDR_threshold(y_dwt,noise_
      estimate,alpha=0.01)

  y_s_univ_dwt<-soft_threshold(y_dwt,universal_threshold_
      lambda)
  y_h_univ_dwt<-hard_threshold(y_dwt,universal_threshold_
      lambda)
  y_s_fdr_dwt<-soft_threshold(y_dwt,FDR_threshold_lambda)
  y_h_fdr_dwt<-hard_threshold(y_dwt,FDR_threshold_lambda)
  y_s_sure_dwt<-sure.thresh(y_dwt,max.level=min(
      resolution_level,12),hard=FALSE)
  y_h_sure_dwt<-sure.thresh(y_dwt,max.level=min(
````
````

```
    resolution_level,12),hard=TRUE)

y_s_univ_pred=idwt(y_s_univ_dwt)
y_h_univ_pred=idwt(y_h_univ_dwt)
y_s_fdr_pred=idwt(y_s_fdr_dwt)
y_h_fdr_pred=idwt(y_h_fdr_dwt)
y_s_sure_pred=idwt(y_s_sure_dwt)
y_h_sure_pred=idwt(y_h_sure_dwt)

ypred_list=list(s_univ=y_s_univ_pred,h_univ=y_h_univ_
    pred,s_fdr=y_s_fdr_pred,h_fdr=y_h_fdr_pred,s_sure=y_
    s_sure_pred,h_sure=y_h_sure_pred)
ypred_names=c("s_univ","h_univ","s_fdr","h_fdr","s_sure
    ","h_sure")

df=tibble()
for (k in 1:length(ypred_names)){
  ypred=ypred_list[[ypred_names[k]]]
  df=df%>%bind_rows(tibble_row(Function=current_func_
      name,
                               Iteration=i,
                               Resolution_Level=resolution_
                                   level,
                               Wavelet_Filter=current_wf,
                               Thresholding_Method=ypred_
                                   names[k],
                               RMSE=rmse(actual=ytrue,
                                   predicted=ypred),
                               MSE=mse(actual=ytrue,
                                   predicted=ypred),
                               MAE=mae(actual=ytrue,
                                   predicted=ypred),
                               R2=1-sum((ypred - ytrue) ^ 2)
                                   /sum((ytrue - mean(ytrue))
                                     ^ 2),
                               MAPE=mape(actual=ytrue,
                                   predicted=ypred),
                               MASE=mase(actual=ytrue,
                                   predicted=ypred),
                               sMAPE=smape(actual=ytrue,
                                   predicted=ypred),
                               RAE=rae(actual=ytrue,
                                   predicted=ypred),
                               RSE=rse(actual=ytrue,
                                   predicted=ypred),
                               RRSE=rrse(actual=ytrue,
```

```
                                    predicted=ypred),

                                SSE=sse(actual=ytrue,
                                    predicted=ypred),

                            ))

                                }
  return(df)
}

```

```{r}

wavelet_smoother_df=tibble(Function=character(),
    Resolution_Level=numeric(),
                                Wavelet_Filter=character(),
                                    Iteration=numeric(),
                                    Thresholding_Method=
                                    character(),RMSE=numeric()
                                    ,MSE=numeric(),
                               MAE=numeric(),R2=numeric(),
                                   MAPE=numeric(),MASE=numeric
                                   (),sMAPE=numeric(),RAE=
                                   numeric(),
                               RSE=numeric(),RRSE=numeric(),
                                   SSE=numeric())
iter_per_func=100

datafram=wavelet_smoother_df

t=seq(from=0,to=1,length.out=2^11)


noise_level=1
n_points=length(t)
boundary="periodic"
wavelet_filter_list=c("haar","la8","la16","d4","d8","d16"
    )


nThreads <- detectCores()-1

clusterOfThreads <- makeCluster(nThreads)
```

```
clusterEvalQ(clusterOfThreads, {

  library(Metrics)
  library(matlib)
  library(dplyr)
  library(waveslim)
  library(expm)
  source('functions.R')

})
doParallel::registerDoParallel(clusterOfThreads, cores=
    nThreads)



set.seed(1968)
for (ii in 1:11){
  resolution_level=ii
  print("Resolution Level")
  print(resolution_level)
for (jj in 1:length(wavelet_filter_list)){
  current_wf=wavelet_filter_list[jj]

for (j in 1:length(funcs$func_names)){
  current_func=funcs$functions[j][[1]]
  current_func_name=funcs$func_names[j]
  ytrue=current_func(t)

  noise=rnorm(n=n_points)*noise_level
  ytrue=ytrue*(7*noise_level/sd(ytrue))

  clusterExport(clusterOfThreads, varlist = c("fit_
      wavelet_and_return_df",

                                                "MAD_noise_
                                                    est","
                                                    ytrue",
                                                "noise_level"
                                                    , "tibble_
                                                    row","n_
                                                    points","
                                                    iter_per_
                                                    func","
                                                    current_
                                                    func_name"
```

```
                                                    ,"rescale_
                                                    x","
                                                    rescale_
                                                    add_noise"
                                                    ,"boundary
                                                    ","
                                                    resolution
                                                    _level","
                                                    current_wf
                                                    "))
  list_of_df=clusterApply(cl=clusterOfThreads,
                          x=seq(length.out=iter_per_func
                              ),
                          fun=function(i) fit_wavelet_
                              and_return_df(i=i,current_
                              func_name=current_func_name
                              ,ytrue=ytrue,current_wf=
                              current_wf,resolution_level
                              =resolution_level ,boundary
                              =boundary ))

  datafram=datafram%>%bind_rows(list_of_df)

}}}
stopCluster(clusterOfThreads)
write.csv(datafram,"wavelet_periodic_smoothing_stats.csv"
    )
```

```
---
title: "Evaluation_Plots"
output: html_document
date: "`r Sys.Date()`"
---

```{r setup, include=FALSE}
library(waveslim)
library(ggplot2)
library(expm)
library(matlib)
library(tidyr)
library(dplyr)
library(Metrics)
library(stringr)
source('functions.R')
PLOTWIDTH=25
```
```

```
PLOTHEIGHT=15
LINEWIDTH=2
ERRORLINEWIDTH=1
ERRORWIDTH=0.75
FIG_DPI=500
standard_theme=theme(text=element_text(size=24))
```



```{r}
fourier_df=read.csv("fourier_smoothing_stats_2.csv")%>%
    select(!X)

long_fourier_df<-fourier_df%>%pivot_longer(cols=-c("
    Function","K","Iteration"),names_to="metric")%>%group_
    by(Function,K,metric)%>%mutate(sd=sd(value),avg=mean(
    value))
long_fourier_df=long_fourier_df%>%filter(metric %in%c("
    RMSE","MAE","sMAPE","R2","df","finalCV","lambda"))%>%
    mutate(Function = str_to_title(Function),metric=recode
    (metric,finalCV="Final-CV"))
wavelet_df=read.csv("wavelet_periodic_smoothing_stats.csv
    ")%>%select(!X)

long_wavelet_df<-wavelet_df%>%pivot_longer(cols=-c("
    Function","Resolution_Level","Iteration","Wavelet_
    Filter","Thresholding_Method"),names_to="metric")%>%
    filter(metric %in%c("RMSE","MAE","sMAPE","R2"))%>%
    group_by(Function,Wavelet_Filter,Resolution_Level,
    Thresholding_Method,metric)%>%mutate(sd=sd(value),avg=
    mean(value))
long_wavelet_df=long_wavelet_df%>%mutate(Function = str_
    to_title(Function),Wavelet_Filter=recode(Wavelet_
    Filter, d4 = "D4", haar = "Haar",d8="D8",d16="D16",la8
    ="LA8",la16="LA16"),Thresholding_Method=recode(
    Thresholding_Method,s_univ="Soft-Universal",s_fdr="
    Soft-FDR",s_sure="Soft-SureShrink",h_univ="Hard-
    Universal",h_fdr="Hard-FDR",h_sure="Hard-SureShrink"))
long_wavelet_df
```


```{r}

df_CV_lambda_plot=ggplot(data=long_fourier_df%>%filter(
    metric %in% c("df","Final-CV","lambda")),mapping=aes(x
```

```
=K, y=avg, color=Function))+geom_line(linewidth=
LINEWIDTH)+geom_errorbar(aes(ymin=avg-sd, ymax=avg+sd),
linewidth=ERRORLINEWIDTH, width=5*ERRORWIDTH)+facet_
wrap(facets="metric", scales="free_y")+standard_theme

RMSE_sMAPE_MAE_plot=ggplot(data=long_fourier_df%>%filter(
metric %in% c("RMSE","sMAPE","MAE")),mapping=aes(x=K, y
=avg, color=Function))+geom_line(linewidth=LINEWIDTH)+
geom_errorbar(aes(ymin=avg-sd, ymax=avg+sd), linewidth=
ERRORLINEWIDTH, width=5*ERRORWIDTH)+facet_wrap(facets="
metric", scales="free_y")+standard_theme

ggsave("Figures/Results/df_CV_lambda_plot.png", df_CV_
lambda_plot, width=PLOTWIDTH, height=PLOTHEIGHT, dpi=FIG_
DPI)

ggsave("Figures/Results/RMSE_sMAPE_MAE_plot.png",RMSE_
sMAPE_MAE_plot, width=PLOTWIDTH, height=PLOTHEIGHT, dpi=
FIG_DPI)
```
```

```{r}
ggplot(data=fourier_df%>%filter(Function!="doppler"&
Function!="heavisine"),mapping=aes(x=K, y=R2, color=
Function))+geom_line(aes(y=R2))+geom_errorbar(aes(ymin
=mean(R2)-sd(R2),ymax=mean(R2)+sd(R2)))
```
```
```{r}
soft_thresholding_sMAPE_plot=ggplot(data=long_wavelet_df
%>%filter(metric=="sMAPE",Function!="bumps",
Thresholding_Method%in%c("Soft-Universal","Soft-FDR","
Soft-SureShrink")),mapping=aes(x=Resolution_Level, y=
avg, color=Function))+geom_line(linewidth=LINEWIDTH)+
geom_errorbar(aes(ymin=avg-sd, ymax=avg+sd), linewidth=
ERRORLINEWIDTH, width=ERRORWIDTH)+facet_grid(
Thresholding_Method~Wavelet_Filter, scales="free_y")+
labs(x="Resolution-Level-J", y="Average-sMAPE-over-100
-Simulations", title="Soft-Thresholding")+standard_
theme

hard_thresholding_sMAPE_plot=ggplot(data=long_wavelet_df
%>%filter(metric=="sMAPE",Function!="bumps",
Thresholding_Method%in%c("Hard-Universal","Hard-FDR","
Hard-SureShrink")),mapping=aes(x=Resolution_Level, y=
avg, color=Function))+geom_line(linewidth=LINEWIDTH)+
geom_errorbar(aes(ymin=avg-sd, ymax=avg+sd), linewidth=
```

```
    ERRORLINEWIDTH, width=ERRORWIDTH)+facet_grid(
    Thresholding_Method~Wavelet_Filter, scales="free_y")+
    labs(x="Resolution Level, J", y="Average sMAPE over 100
    Simulations", title="Hard Thresholding")+standard_
    theme

sure_threshold_sMAPE_plot=ggplot(data=long_wavelet_df%>%
    filter(metric=="sMAPE", Function!="bumps", Thresholding_
    Method%in%c("Soft SureShrink","Hard SureShrink")),
    mapping=aes(x=Resolution_Level, y=avg, color=Function))+
    geom_line(linewidth=LINEWIDTH)+geom_errorbar(aes(ymin=
    avg-sd, ymax=avg+sd), linewidth=ERRORLINEWIDTH, width=
    ERRORWIDTH)+facet_grid(Thresholding_Method~~Wavelet_
    Filter, scales="free_y")+labs(x="Resolution Level, J", y
    ="Average sMAPE over 100 Simulations", title="
    SureShrink Threshold")+standard_theme

ggsave("Figures/Results/soft_thresholding_sMAPE_plot.png"
    , plot=soft_thresholding_sMAPE_plot, width=PLOTWIDTH,
    height=PLOTHEIGHT, dpi=FIG_DPI)

ggsave("Figures/Results/hard_thresholding_sMAPE_plot.png"
    , plot=hard_thresholding_sMAPE_plot, width=PLOTWIDTH,
    height=PLOTHEIGHT, dpi=FIG_DPI)

ggsave("Figures/Results/sure_threshold_sMAPE_plot.png",
    plot=sure_threshold_sMAPE_plot, width=PLOTWIDTH, height=
    PLOTHEIGHT, dpi=FIG_DPI)


```
```{r}
soft_thresholding_RMSE_plot=ggplot(data=long_wavelet_df
    %>%filter(metric=="RMSE", Function!="bumps",
    Thresholding_Method%in%c("Soft Universal","Soft FDR","
    Soft SureShrink")), mapping=aes(x=Resolution_Level, y=
    avg, color=Function))+geom_line(linewidth=LINEWIDTH)+
    geom_errorbar(aes(ymin=avg-sd, ymax=avg+sd), linewidth=
    ERRORLINEWIDTH, width=ERRORWIDTH)+facet_grid(
    Thresholding_Method~Wavelet_Filter, scales="free_y")+
    labs(x="Resolution Level, J", y="Average RMSE over 100
    Simulations", title="Soft Thresholding")+standard_theme

hard_thresholding_RMSE_plot=ggplot(data=long_wavelet_df
    %>%filter(metric=="RMSE", Function!="bumps",
    Thresholding_Method%in%c("Hard Universal","Hard FDR","
```

```
    Hard-SureShrink")),mapping=aes(x=Resolution_Level,y=
    avg,color=Function))+geom_line(linewidth=LINEWIDTH)+
    geom_errorbar(aes(ymin=avg-sd,ymax=avg+sd),linewidth=
    ERRORLINEWIDTH,width=ERRORWIDTH)+facet_grid(
    Thresholding_Method~Wavelet_Filter,scales="free_y")+
    labs(x="Resolution-Level,-J",y="Average-RMSE-over-100-
    Simulations",title="Hard-Thresholding")+standard_theme

sure_threshold_RMSE_plot=ggplot(data=long_wavelet_df%>%
    filter(metric=="RMSE",Function!="bumps",Thresholding_
    Method%in%c("Soft-SureShrink","Hard-SureShrink")),
    mapping=aes(x=Resolution_Level,y=avg,color=Function))+
    geom_line(linewidth=LINEWIDTH)+geom_errorbar(aes(ymin=
    avg-sd,ymax=avg+sd),linewidth=ERRORLINEWIDTH,width=
    ERRORWIDTH)+facet_grid(Thresholding_Method~~Wavelet_
    Filter,scales="free_y")+labs(x="Resolution-Level,-J",y
    ="Average-RMSE-over-100-Simulations",title="SureShrink
    -Threshold")+standard_theme

ggsave("Figures/Results/soft_thresholding_RMSE_plot.png",
    plot=soft_thresholding_RMSE_plot,width=PLOTWIDTH,
    height=PLOTHEIGHT,dpi=FIG_DPI)

ggsave("Figures/Results/hard_thresholding_RMSE_plot.png",
    plot=hard_thresholding_RMSE_plot,width=PLOTWIDTH,
    height=PLOTHEIGHT,dpi=FIG_DPI)

ggsave("Figures/Results/sure_threshold_RMSE_plot.png",
    plot=sure_threshold_RMSE_plot,width=PLOTWIDTH,height=
    PLOTHEIGHT,dpi=FIG_DPI)
```
```

```{r}
soft_thresholding_MAE_plot=ggplot(data=long_wavelet_df%>%
    filter(metric=="MAE",Function!="bumps",Thresholding_
    Method%in%c("Soft-Universal","Soft-FDR","Soft-
    SureShrink")),mapping=aes(x=Resolution_Level,y=avg,
    color=Function))+geom_line(linewidth=LINEWIDTH)+geom_
    errorbar(aes(ymin=avg-sd,ymax=avg+sd),linewidth=
    ERRORLINEWIDTH,width=ERRORWIDTH)+facet_grid(
    Thresholding_Method~Wavelet_Filter,scales="free_y")+
    labs(x="Resolution-Level,-J",y="Average-MAE-over-100-
    Simulations",title="Soft-Thresholding")+standard_theme

hard_thresholding_MAE_plot=ggplot(data=long_wavelet_df%>%
    filter(metric=="MAE",Function!="bumps",Thresholding_
```
```

```
    Method%in%c("Hard Universal","Hard FDR","Hard
    SureShrink")),mapping=aes(x=Resolution_Level,y=avg,
    color=Function))+geom_line(linewidth=LINEWIDTH)+geom_
    errorbar(aes(ymin=avg-sd,ymax=avg+sd),linewidth=
    ERRORLINEWIDTH,width=ERRORWIDTH)+facet_grid(
    Thresholding_Method~Wavelet_Filter,scales="free_y")+
    labs(x="Resolution Level, J",y="Average MAE over 100
    Simulations",title="Hard Thresholding")+standard_theme

sure_threshold_MAE_plot=ggplot(data=long_wavelet_df%>%
    filter(metric=="MAE",Function!="bumps",Thresholding_
    Method%in%c("Soft SureShrink","Hard SureShrink")),
    mapping=aes(x=Resolution_Level,y=avg,color=Function))+
    geom_line(linewidth=LINEWIDTH)+geom_errorbar(aes(ymin=
    avg-sd,ymax=avg+sd),linewidth=ERRORLINEWIDTH,width=
    ERRORWIDTH)+facet_grid(Thresholding_Method~~Wavelet_
    Filter,scales="free_y")+labs(x="Resolution Level, J",y
    ="Average MAE over 100 Simulations",title="SureShrink
    Threshold")+standard_theme

ggsave("Figures/Results/soft_thresholding_MAE_plot.png",
    plot=soft_thresholding_MAE_plot,width=PLOTWIDTH,height
    =PLOTHEIGHT,dpi=FIG_DPI)

ggsave("Figures/Results/hard_thresholding_MAE_plot.png",
    plot=hard_thresholding_MAE_plot,width=PLOTWIDTH,height
    =PLOTHEIGHT,dpi=FIG_DPI)

ggsave("Figures/Results/sure_threshold_MAE_plot.png",plot
    =sure_threshold_MAE_plot,width=PLOTWIDTH,height=
    PLOTHEIGHT,dpi=FIG_DPI)
```

```
title: "Plots of test functions"
output: html_notebook
```

```
```{r}
library(tidyverse)
library(ggplot2)
source('functions.R')
PLOTWIDTH=25
PLOTHEIGHT=15
LINEWIDTH=2
ERRORLINEWIDTH=1
ERRORWIDTH=0.75
```
```

```
FIG_DPI=300
standard_theme=theme(text=element_text(size=24))
```

```{r}
current_func=funcs$functions[j][[1]]
  current_func_name=funcs$func_names[j]
  ytrue=current_func(t)
  funcs=list(functions=c(spikes,corner,angles,bumps,
      blocks,heavisine,doppler),func_names=c("Spikes","
      Corner","Angles","Bumps","Blocks","Heavisine","
      Doppler"))

t_vec=seq(from=0,to=1,length.out=2^11)
  for (j in 1:length(funcs$func_names)){
  current_func=funcs$functions[j][[1]]
  current_func_name=funcs$func_names[j]
  ytrue=current_func(t_vec)

  noise=rnorm(n=n_points)*noise_level
  ytrue=ytrue*(7*noise_level/sd(ytrue))
  y_noisy=ytrue+noise
  plot_df=bind_rows(tibble(t=t_vec,value=ytrue,type="
      Without noise",func=current_func_name),tibble(t=t_
      vec,value=y_noisy,type="With added noise",func=
      current_func_name))
  func_plot=ggplot(data=plot_df,mapping=aes(x=t,y=value))
      +geom_line(linewidth=LINEWIDTH)+facet_wrap("type")+
      standard_theme+labs(x="t",y="y")
  filename=paste("Figures/test_functions/",paste(current_
      func_name,"_plot.jpg",sep=""))
 ggsave(filename,func_plot,height = PLOTHEIGHT,width =
     PLOTWIDTH,dpi=FIG_DPI)
  }
```