Hydraulic Data Analysis Using Python

Master thesis by Teresa Schnellbach
Date of submission: August 18, 2022

1. Review: Prof. Dr.-Ing. habil. Boris Lehmann
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

WASSERBAU HYDRAULIK

Department of Civil and Environmental Engineering
Institute of Hydraulic and Water Resources Engineering
Chair of Hydraulic Engineering

Hydraulic Data Analysis Using Python

Master thesis by Teresa Schnellbach

1. Review: Prof. Dr.-Ing. habil. Boris Lehmann

Date of submission: August 18, 2022

Darmstadt

Institut für Wasserbau und Wasserwirtschaft
Fachgebiet Wasserbau und Hydraulik

Leiter: Prof. Dr.-Ing. habil. Boris Lehmann
Franziska-Braun-Straße. 7, 64287 Darmstadt, Tel. +49 6151 16 – 21165

TECHNISCHE UNIVERSITÄT DARMSTADT

Master-Arbeit für Frau Teresa SCHNELLBACH (Matr.-Nr. 2564175)

Hydraulische Messdatenanalyse mit Python

Hydraulic Data Analysis Using Python

Veranlassung und Thema

Messdaten sind ein wichtiger Bestandteil wasserbaulicher Projekte - ob zur Kalibrierung und Validierung von numerischen Modellen oder zur Planung von Anlagen. Aufgrund der gerätespezifischen Messprinzipien können die Daten jedoch fehlerbehaftet sein, was bei der Auswertung entsprechend berücksichtigt werden muss. Hierbei gibt es verschiedene Methoden, um beispielsweise Fehler auszusortieren und Daten zu bereinigen. Ebenso können aus den empfangenen Signalen der Messgeräte durch eine geeignete Filterung Rückschlüsse auf Strömungsphänomene gezogen werden z.B. auf periodisch wiederkehrenden Schwankungen der Geschwindigkeit. Zusätzlich stellt sich vor einer Messung immer wieder die Frage nach der erforderlichen Messdauer, welche abhängig vom Untersuchungsziel gewählt werden muss und auf welche in der Ergebnisdarstellung bei vielen Studien oft nicht in ausreichender Tiefe eingegangen wird.

Im Rahmen der Masterthesis sollen daher verschiedene Möglichkeiten zur Filterung und Analyse von Zeitreihen hydraulischer Messdaten zusammengestellt, erläutert und angewendet werden. Ebenso sind abschließend Empfehlungen zur Messdauer zu erarbeiten.

Aufgabenstellung

Die zu untersuchenden hydraulischen Daten beziehen sich im Wesentlichen auf Messgeräte, welche am Fachgebiet Wasserbau und Hydraulik aktuell vielfach im Einsatz sind (bspw. Acoustic Doppler Velocimeter, Acoustic Doppler Current Profiler und Ultraschall Distanz Sensor). Dabei sollen zunächst mögliche allgemeine und gerätespezifische Fehlerarten genannt sowie auf die statistischen Analysen der Zeitreihen (unter Berücksichtigung der GUM - Guide to the Expression of Uncertainty in Measurement) eingegangen werden. Anschließend sollen verschiedene Möglichkeiten zur Filterung von Daten beschrieben werden. Diese Filterung kann zum einen daraus bestehen, schlechte Daten auszusortieren, als auch im Sinne einer Signalanalyse (FFT-Analyse, Tiefpassfilterung etc.) Rückschlüsse auf hydraulische Phänomene und Charakteristika der Strömung (z.B. Wirbelgrößen) ziehen zu können. Dabei stellt sich auch die Frage ob und ggf. wie fehlerhafte Daten ersetzt werden können.

Je nach zu untersuchendem Parameter ist außerdem die Messdauer entsprechend zu wählen, um z.B. relevante periodische Schwankungen unter Berücksichtigung des Nyquist-Shannon-Abtasttheorem richtig zu erfassen oder für gemittelte Größen belastbare Werte zu erhalten. Hierfür soll ein geeignetes Vorgehen erarbeitet und Empfehlungen gegeben werden, um hydraulische Analysen von Messdaten zweckmäßig und effizient durchzuführen. Alle Analysen sollen dabei in Python programmiert und dargestellt werden.

Sollte sich im Laufe der Bearbeitung eine abweichende Vorgehensweise als sinnvoll erweisen, kann in Absprache mit den Ansprechpersonen am Fachgebiet Wasserbau und Hydraulik von den oben dargelegten Punkten abgewichen werden.

Modalitäten

Grundsätzlich gelten die Bestimmungen zu Abschlussarbeiten in der Allgemeinen Prüfungsordnung TU Darmstadt und in der Studienordnung des Fachbereiches 13. Diese Bestimmungen beziehen sich u.a. auf die Aspekte

- Betreuung und Bewertung von Abschlussarbeiten und
- besondere Regelungen bei externen Arbeiten.

Die Kandidatin hat dafür selbstständig Sorge zu tragen, dass diese Bestimmungen eingehalten werden. Darüber hinaus gelten folgende Randbedingungen:

Referent	Prof. Boris Lehmann FG Wasserbau und Hydraulik lehmann@wb.tu-darmstadt.de	Empfohlene Kenntnisse	Grundlagen der Hydromechanik
Ansprechpartner	Katharina Bensing, M.Sc. FG Wasserbau und Hydraulik k.bensing@wb.tu-darmstadt.de		
Bearbeitungszeitraum	Auslösung nach Vereinbarung, Leistungszeitraum 26 Wochen	Arbeitsbedingungen	Heimarbeit

Darmstadt, den 22.10.2021

[illegible]

Prof. Boris Lehmann

Abstract

Acoustic Doppler velocimeter (ADV) data is prone to high uncertainty in measurement. In this thesis, technical literature that proposes data analysis methods to reduce error effects is reviewed, and subsequently, three methods are implemented using the programming language Python. The reduction of uncertainty in measurement is evaluated by categorising statistical parameters and analysing time-series and Kolmogorov energy spectra for 160 ADV samples in turbulent flow. The results show that out of the examined data analysis methods, kernel density estimation despiking in combination with lowpass Butterworth filtering is the most promising way to reduce the uncertainty in measurement. Furthermore, a procedure to find the optimal sampling time for ADV measurements is realised. The implementation shows that statistical equivalence testing is adequate for finding the optimum sampling time. Still, the procedure needs further development to provide significance regarding higher statistical moments. Ultimately, a systematic workflow for handling ADV data is proposed.

Keywords: Turbulence; ADV; Uncertainty; Data processing; Sampling time.

Daten, die mit dem akustischen Doppler Velocimeter (ADV) aufgenommen werden, sind anfällig für hohe Messunsicherheiten. In dieser Thesis werden Datenanalysemethoden aus der Fachliteratur, die zur Reduzierung von Fehlereffekten entwickelt wurden, geprüft. Anschließend werden drei Datenanalysemethoden in der Programmiersprache Python implementiert. Die Reduzierung der Messunsicherheit wird dann durch die Kategorisierung statistischer Parameter und durch die Analyse von Zeitreihen und Kolmogorov Energiespektren bei 160 ADV-Messungen in turbulenter Strömung bewertet. Die Ergebnisse zeigen, dass von den untersuchten Datenanalysemethoden das Kerndichteschätzer-Despiking in Kombination mit dem Butterworth Tiefpassfilter die vielversprechendste Methode zur Reduzierung der Messunsicherheit ist. Darüber hinaus wird ein Verfahren zur Ermittlung einer optimalen Abtastdauer für ADV-Messungen realisiert. Die Umsetzung veranschaulicht, dass statistische Äquivalenztests geeignet sind, um die optimale Abtastdauer zu finden. Jedoch muss das Vorgehen noch weiterentwickelt werden, um auch in Bezug auf höhere statistische Momente Signifikanz zu schaffen. Schließlich wird ein systematischer Arbeitsablauf für die Handhabung von ADV-Daten vorgeschlagen.

Schlüsselworte: Turbulenz; ADV; Datenaufbereitung; Messwertverarbeitung; Messdauer.

Contents

List of Figures

List of Acronyms

ADV	Acoustic Doppler Velocimeter
AR	Autoregressive
ARMA	Autoregressive Moving Average
BIC	Bayesian Information Criterion
CLT	Central Limit Theorem
COR	Correlation Score
DFT	Discrete Fourier Transformation
DN	Diamètre Nominal
DSP	Digital Signal Processing
EMD	Empirical Mode Decomposition
ENU	Earth-North-Up
FFT	Fast Fourier Transformation
FIR	Finite Impulse Response
Gau	Gauss Filter
GUI	Graphical User Interface
GUM	Guide to the Uncertainty in Measurement
IDE	Integrated Development Environment
IDFT	Inverse Discrete Fourier Transformation
IFFT	Inverse Fast Fourier Transformation
IIR	Infinite Impulse Response
JCGM	Joint Committee for Guides in Metrology

KDE	Kernel Density Estimation
LLN	Law of Large Numbers
LMMSE	Linear Minimum Mean Square Error
LTI	Linear Time Invariant
MA	Moving Average
NAC	Noise Auto-correlation
POD	Proper Orthogonal Decomposition
P-SAT	Python Statistical Analysis of Turbulence
RANS	Reynolds Averaged Navier-Stokes
RMS	Root Mean Square
SNR	Signal-to-Noise Ratio
SSA	Singular Spectrum Analysis
TKE	Turbulent Kinetic Energy
TOST	Two One-sided T-test

1. Introduction

Hydraulic measurements are an integral part of hydraulic laboratory practice. Recorded parameters range from the most apparent geometric quantities (e.g. lengths, widths, flow depths and radii) to kinematic quantities (e.g. flow velocities) and dynamic parameters (e.g. pressures, local forces and stresses). They are needed to conduct hydromechanical experiments or design, plan and construct hydraulic structures. For hydraulic structure models, hydraulic data is necessary to transfer actual flow conditions to a true-to-scale representation. Numerical models are verified and calibrated using recorded hydraulic data, which can help understand ecological or ethohydraulic interactions in the laboratory and the field. (Morgenschweis, 2018)

Therefore, every researcher in the hydraulic laboratory and many other professionals in the industry and the public sector are familiar with measuring hydraulic data. Regardless of the contemplated use of the recorded data, measurement device users generally aim to generate the most accurate data possible with the available resources. They calibrate their measurement devices, minimise systematic errors, and record repetitively to reduce random errors. Sometimes, they manipulate their data to eliminate known errors inherent to the measurement technique, the device, or measurand from their data.

This thesis focuses on flow velocity measurements in turbulent flow. It highlights the particularities of a measurement technique called acoustic Doppler velocimetry. Specifically, it concentrates on a device known as the acoustic Doppler velocimeter (ADV). Although ADV devices are not new to the market of velocity measurement devices, they are still prone to uncertainties in measurement.

Resulting from these uncertainties in measurement, velocity data recorded with ADVs needs to undergo extensive data processing efforts to be usable without a doubt. This need for processing leads to a significant flaw of this otherwise practical and precise measurement device: data processing activities are not standardised. Authors rarely report transparently and in detail on their processing efforts, and there are only a few computational open-source solutions to process recorded ADV data efficiently. Because of this lack of standardised procedure, this thesis attempts to identify critical data analysis methods for ADV data by reviewing technical literature and further tries to implement several reviewed methods in the open-source programming language Python. Then, the effects of said data analysis methods on 160 ADV samples are evaluated.

Besides the absence of standardised protocols for processing ADV data, there also seems to be a considerable disparity in recommendations and studies concerning the necessary sampling time for ADV measurements. This ambiguity is surprising, as the sampling time is not only one of the first considerations needed when recording data but also a rather fundamental one. The undertaken efforts to apply hypothesis testing to find an optimal sampling time for ADV measurements are specified.

Ultimately, the goal is to give pointers to standardise the ADV data recording and processing protocols in the hydraulic laboratory at Technische Universität Darmstadt and provide provisional open-source tools that facilitate the realisation of this protocol.

Fundamental scientific principles needed to realise the abovementioned goals are gathered in chapter 2 of this thesis. The chapter briefly summarises the basics of statistics, fluid dynamics, signal processing, and digital filters that are relevant here. Detailed information on ADV devices is outlined in chapter 3. Chapter 4 reviews the technical literature on ADV data analysis methods. While chapter 5 describes the process of implementing promising methods in Pyhon, chapter 6 explains the evaluation methodology and evaluates the velocity measurements at hand. Chapter 7 illustrates the efforts to develop a procedure for finding optimal sampling times. Finally, chapter 8 critically reviews the methodology used in this thesis, points out shortcomings and flaws and gives recommendations for standardising laboratory work with the ADV. The chapter brings forward suggestions to further improve the implementations in Python. It highlights other details concerning ADV measurements that can be examined and evaluated for standardisation in future projects.

2. Fundamentals

This chapter outlines scientific fundamentals that need to be grasped to understand the less common concepts picked up in chapter 4 and comprehend the details of the code implementations in chapter 5, chapter 6 and chapter 7.

Section 2.1 goes into statistic fundamentals that need to be well-defined to evaluate hydraulic measurement data statistically and introduces ways to test recorded data regarding their statical parameters. The second section (see subsection 2.2), gives insight into basic concepts of fluid dynamics to understand details on flow and flow conditions. Furthermore, this section is meant to draw a bow to the modus operandi of acoustic Doppler velocimetry (see chapter 3). Lastly, sections 2.3 and 2.4 introduce the basics of signal processing and the operation of digital filters. These fundamentals are critical for comprehending how digital filtering can reduce error terms in hydraulic measurements (see subsection 4.3.2).

2.1. Basics of Statistics

Hypothesis testing is a conventional method to prove the statistical significance of data. This concept can be perfectly transferred to hydraulic data and helps understand uncertainty in measurements and gain confidence in recorded data. Therefore, this subsection first gives pointers on the Gauss distribution and the Student's t-distribution, then outlines the basics of hypothesis testing and highlights the differences between traditional hypothesis testing and equivalence testing.

2.1.1. Probability Distributions

Probability distributions of experiments describe the probability of occurrence of the experiment's different possible outcomes. Probability distributions can be specified by several statistical parameters, such as the mean, the variance, the standard deviation, the skewness, the kurtosis, the central moment of order, the modus, quantiles and the median. (Lange &

Mosler, 2017, pp. 32–36) There are many common probability distributions. However, this study will only talk about the Gauss distribution (also normal distribution) and Student's t-distribution as only those are needed within its scope.

Gauss distribution calls a random variable normally distributed if it obeys the probability density function of

$$f(x) = \frac{1}{\sqrt{2\pi} \cdot \sigma} \cdot e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}, x \in \mathbb{R} \tag{2.1}$$

$f(x)$	– probability density function,
x	– random variable,
σ	– standard deviation,
μ	– mean. (Lange & Mosler, 2017, p. 56)

Talking about normal distribution also calls for mentioning the central limit theorem (CLT). It implies that the sum of an adequate number of independent random variables is normally distributed by approximation. This assumption holds even when the original variables do not follow a normal distribution. (Bonamente, 2013, pp. 54–58) In practice, it means that the normal distribution of a random variable can be assumed if the threshold of an adequate number of measurements is reached, which is important for multiple statistical applications, e.g. hypothesis testing. The rule of thumb to assume normal distribution is at a sample size of 30-50 (Ross, 2010, Schiefer and Schiefer, 2021, p. 76). It is important to distinguish the CLT from the law of large numbers (LLN) as they draw different conclusions. While the CLT determines normal distribution for sample sizes tending to infinity, the LLN asserts that the sample mean equals to the population mean for sample sizes tending to infinity (Bonamente, 2013, p. 62). If the standard deviation of a population is unknown and must be approximated with the sample standard deviation, the sample mean of normally distributed data is not normally distributed anymore but follows a different distribution called Student's t-distribution. (Waldi, 2019, pp. 241–242) Generally, the population standard deviation is unknown in hands-on data analysis. This is why, the Student's t-distribution is rather important for further statistical operations such as hypothesis testing (see subsection 2.1.3)

2.1.2. Statistical Hypothesis Testing

The general approach of hypothesis testing is to compare distribution parameter values of a sample to a reference value of the same distribution parameter. Consequently, these tests are also called parameter-bound tests as they use statistical parameters to test for differences in the mean (t-test), the dispersion (F-test) or the distribution (χ^2-test). (Schiefer & Schiefer, 2021, p. 69)

Statistical hypothesis testing quantitatively determines whether a formulated assumption describing a sample holds or must be rejected at a given confidence level (also p-value or confidence interval). This assumption is called the null hypothesis. If the null hypothesis is rejected for a given confidence level, the alternate hypothesis holds. The significance level gives the probability that the null hypothesis is rejected erroneously, which is called a type I error (also α-error). Therefore the confidence level describes the probability of not rejecting an accepted null hypothesis. The significance level is usually set at $\alpha = 5\,\%$ or lower. For

$$\alpha = 0.05 \tag{2.2}$$

follows

$$p = 1 - \alpha = 0.95 \tag{2.3}$$

α – significance level,
p – confidence level.

Type II error (also β-error) happens if the test does not reject the null hypothesis although the alternate hypothesis holds. As there is no safeguard to reduce the occurrence of type II error, hypothesis testing produces more reliable results if the test design proves a statement by rejecting the null hypothesis instead of proving a statement by accepting the null hypothesis. Thus, one chooses the null hypothesis accordingly to produce robust test results. (Frost, 2017, pp. 9–10, 18)

There are two basic types of hypothesis tests: one-tailed (also one-sided) and two-tailed (also two-sided) tests. The test either compares whether the sample values are the same as the reference value or not (two-tailed, term 2.4) or it compares whether the sample values are smaller (right-tailed, term 2.5) or bigger (left-tailed, term 2.6) than the reference value:

$$H_0 : x = x_0 \wedge H_1 : x \neq x_0 \tag{2.4}$$

$$H_0 : x \leq x_0 \wedge H_1 : x > x_0 \tag{2.5}$$

$$H_0 : x \geq x_0 \wedge H_1 : x < x_0 \tag{2.6}$$

H_0 – null hypothesis,
H_1 – alternate hypothesis,
x – sample value,
x_0 – reference value.

While two-tailed tests are also called tests with a point hypothesis that refers to a permissible value (see figure 2.1b), one-tailed tests consider the distribution function from the right or the left side (see figure 2.1a). One-tailed tests are perceived to produce more significant results than two-tailed tests. Still, two-tailed tests are a standard approach when extreme discrepancies between values are unlikely (Waldi, 2019, p. 226). Generally, hypothesis testing becomes more sensitive as sample sizes increase. (Schiefer & Schiefer, 2021, p. 72) Hypothesis

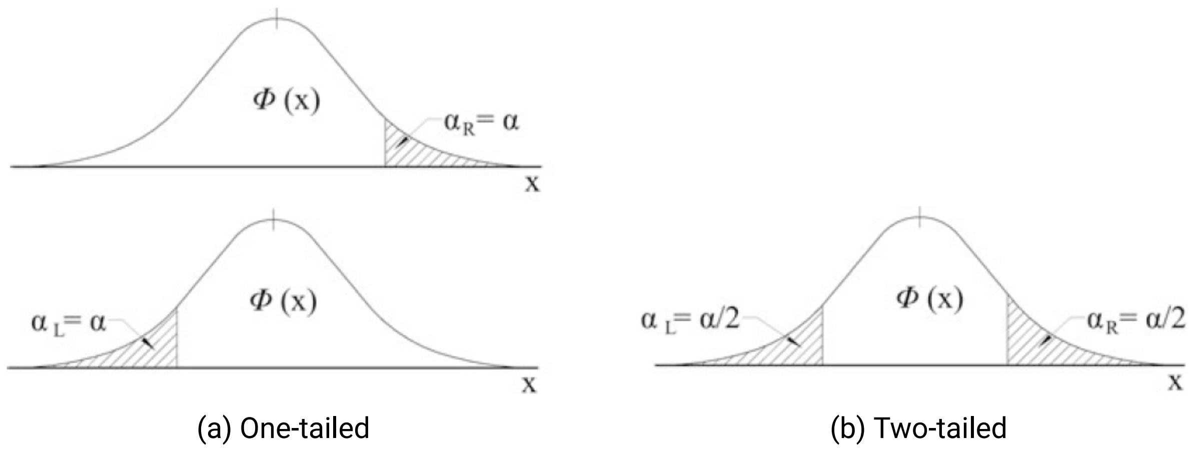


Figure 2.1.: Basic types of hypothesis tests (Schiefer & Schiefer, 2021, p. 73).

tests usually call for requirements, like independency of measurements and samples, normal distribution of the examined characteristic or variance homogeneity of samples, to be met. However, if some conditions are not met, one can use test modifications or test extensions for these exceptions. (Schiefer & Schiefer, 2021, p. 75)

2.1.3. T-Test

T-tests are hypothesis tests used to determine whether the mean values of a sample and a reference value (from a population) or the mean values of two samples differ and whether their difference can be deemed significant or random.

There are several requirements for performing a t-test, which are listed in the following:

– Samples are taken at random from the population.

– Samples and measurements are independent of each other.

– The examined characteristic is interval-scaled.

– The examined characteristic is normally distributed, or the sample size allows for the CLT (see subsection 2.1.1) to be applied.

– The variance of the examined characteristics (from two samples) are the same, or the test must be modified (Welch's t-test). (Schiefer & Schiefer, 2021, p. 75)

By replacing the population standard deviation of the normal distribution with the sample standard deviation, the Student's t-distribution's test statistic with $n - 1$ degrees of freedom is formulated:

$$S = \sqrt{\frac{1}{n-1}\sum(X_i - \bar{X})^2} \tag{2.7}$$

$$T = \frac{\bar{X} - \mu}{S}\sqrt{n} \tag{2.8}$$

S – sample standard deviation,
T – test statistic,
n – sample size,
X_i – random variable,
$\bar{X}$ – sample mean,
μ – population mean. (Frost, 2017, p. 8)

Then, a so-called p-value can be calculated or determined using specific tables and taking into account the beforehand calculated test statistic, the respective degrees of freedom and the chosen significance level. The null hypothesis is rejected if the p-value is smaller than the selected significance level. The null hypothesis is accepted if the p-value is greater than the chosen significance level.

Altogether, to carry out a t-test (as for every hypothesis test), several steps are required:

– Drafting the null hypothesis and the alternate hypothesis.

– Defining the significance level.

– Determining the rejection region of the test.

– Calculating the test statistic and the p-value.

– Interpreting the results: reject or accept the null hypothesis. (Frost, 2017, p. 9)

2.1.4. Equivalence Testing

When hypothesis testing, the null hypothesis represents the opposite of the researcher's intent, so the usual goal is to reject the null hypothesis. However, sometimes the research question might require not to look for differences but a certain level of equivalence instead. Carrying out a traditional two-tailed t-test does not provide evidence on this question. So-called equivalence tests offer a solution: they reverse the null and alternate hypotheses and test for equivalence of samples or values within the scope of a constant, previously-set equivalence margin. (Walker & Nowacki, 2011, p. 1)

Equivalence testing was first introduced to improve pharmaceutical studies which is why it is still particularly prevalent in the medical field, and recommendations for use, such as Piaggio et al., 2006, are given in this context. However, there are efforts to adopt equivalence testing to other research questions and fields, e.g. in Rose et al., 2018 for behavioural and ecological studies and Siebert and Ellenberger, 2020 for automatic passenger counting in public transport.

The following figure 2.2 shows the difference between traditional comparative hypothesis testing and equivalence testing by picking up a problem from the data analysis part of this thesis (see section 7.2):

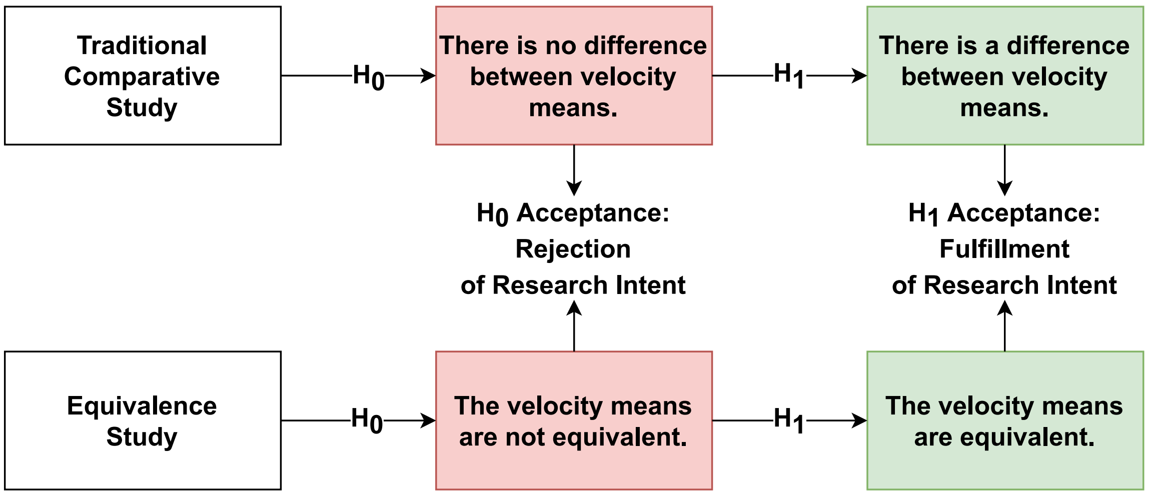

Traditional Comparative Study
H_0
There is no difference between velocity means.
H_1
There is a difference between velocity means.
H_0 Acceptance: Rejection of Research Intent
H_1 Acceptance: Fulfillment of Research Intent
Equivalence Study
H_0
The velocity means are not equivalent.
H_1
The velocity means are equivalent.

Figure 2.2.: Difference in hypothesis formulation between hypothesis testing and equivalence testing (adapted from Walker and Nowacki, 2011, p. 2).

A simple and common approach to equivalence testing is the two one-sided t-test (also TOST) by Westlake, 1976 and Schuirmann, 1987. The TOST tests the difference in means of two samples against a preset range, the equivalence margin δ, limited by a lower boundary and an upper boundary. The TOST's major flaw is choosing the equivalence margin prior to testing.

Thus, the researcher must assume from the data or estimate from experience, which might tamper with test results or make them intransparent. Consequently, accurate protocolling of methods is indispensable when conducting the TOST. (Juzek & Kizach, 2019, p. 2)

Two one-tailed t-tests are carried out for the TOST, so two null hypotheses are specified, one for each boundary. The two alternate hypotheses translate to one final alternate hypothesis, which indicates the positive test outcome of similarity within the set equivalence margin at a chosen significance level:

$$H_{0,\mathrm{a}} : \mu_1 - \mu_2 \leq -\delta \wedge H_{1,\mathrm{a}} : \mu_1 - \mu_2 > -\delta \tag{2.9}$$

$$H_{0,\mathrm{b}} : \mu_1 - \mu_2 \geq +\delta \wedge H_{1,\mathrm{b}} : \mu_1 - \mu_2 < +\delta \tag{2.10}$$

$$H_1 : -\delta < \mu_1 - \mu_2 < +\delta \tag{2.11}$$

$H_{0,\mathrm{a}}$	– null hypothesis of test a,
$H_{1,\mathrm{a}}$	– alternate hypothesis of test a,
$H_{0,\mathrm{b}}$	– null hypothesis of test b,
$H_{1,\mathrm{b}}$	– alternate hypothesis of test b,
H_1	– final alternate hypothesis,
$-\delta$	– lower boundary,
$+\delta$	– upper boundary,
μ_1	– mean sample 1,
μ_2	– mean sample 2. (Juzek & Kizach, 2019, p. 3)

The test statistics comply with those of the t-test (see equation 2.8), but as the TOST consists of two one-tailed t-tests, the confidence level is:

$$p = 1 - 2\alpha \tag{2.12}$$

α	– significance level,
p	– confidence level.

2.2. Basics of Fluid Dynamics

Acquisition of hydraulic data in the laboratory or in situ is essential for many projects in hydraulic engineering. In both settings, measurements aim to observe hydraulic characteristics and occurring phenomena. However, to observe hydraulic characteristics and phenomena, one must take the properties of fluids and flow into account. Therefore, the following sections address the transport properties of fluids and the kinematic properties of flow. As most technical flows are turbulent, the subsequent sections introduce turbulent flow characteristics and approaches to its description.

2.2.1. Kinematic Properties of Flow

It is essential to introduce technical concepts to describe flow conditions. The most basic concept considers flow as an accumulation of single moving fluid particles and conceptualising equations considering an individual particle and its particular movement.

Initial coordinates, which form planes (and thus functions) of space coordinates, are attributed to every particle in its initial position. Because each particle keeps the attributed coordinates when changing position, all planes comprise an invariant set of particles. Then, space coordinates are given as a function of the initial coordinates and time to identify the so-called pathlines. The Lagrangian approach (Song, 2018, pp. 49–50) (also Lagrangian description or material description (Spurk & Aksel, 2020, p. 8)) is used:

$$\begin{aligned} x &= F_1(a,b,c,t) \\ y &= F_2(a,b,c,t) \\ z &= F_3(a,b,c,t) \end{aligned} \tag{2.13}$$

x, y, z – space coordinates,
a, b, c – initial coordinates,
t – time. (Oertel, 2017, p. 46)

Connecting all points on the trajectories of the particles that passed through a set point in the flow field within a set timeframe yields so-called streaklines. One can observe streaklines when inserting a tracer into a fluid in motion at a fixed point. (Oertel et al., 2015, p. 70) The velocity components are defined as a function of space coordinates and time to consider a change in flow conditions over time:

$$\begin{aligned} u &= f_1(x, y, z, t) \\ v &= f_2(x, y, z, t) \\ w &= f_3(x, y, z, t) \end{aligned} \tag{2.14}$$

u, v, w – velocity components. (Oertel, 2017, p. 46)

This representation is called the Eulerian approach (Song, 2018, p. 50) (also Eulerian description or spatial description (Spurk & Aksel, 2020, p. 9)).

Streamlines are another concept to describe flow conditions: by placing a tangent to the pathlines, the slope field of the velocity vector is given at a specific time. Because the tangent lines are always parallel to the velocity vector, the velocity components can be derived using the cross product between the velocity vector and an infinitesimal section of the pathline. (Bollrich, 2013, p. 87)

$$\vec{v} \times d\vec{x} = 0 \tag{2.15}$$

$\vec{v}$ – velocity vector,
$d\vec{x}$ – infinitesimal section of the pathline. (Oertel et al., 2015, p. 69)

This leads to the system of differential equations for streamlines (Bollrich, 2013, p. 88):

$$\begin{aligned} v \cdot dz &= w \cdot dy \\ w \cdot dx &= u \cdot dz \\ u \cdot dy &= v \cdot dx. \end{aligned} \tag{2.16}$$

For stationary flow, the pathlines accord with the corresponding streamlines. For unsteady flow, the curves differ. Still, to give precise information for a compressible flow on hand, one must additionally state the pressure and the density. (Oertel, 2017, p. 46)

2.2.2. Transport Properties of Fluids

Interactions between molecules determine the transport properties of fluids. However, with varying definitions and approaches, theories, and many fields of application, it is a complex phenomenon to grasp. Mass conservation, momentum conservation, and energy conservation translate to three fluid transport properties: mass transfer, friction and heat transfer. (Oertel et al., 2015, p. 48)

Literature sometimes distinguishes between outer friction (also skin friction), which describes the interaction of fluid molecules and adjacent solids, and inner friction (also viscosity), which occurs because of molecule exchange within the fluid (Bollrich, 2013, p. 26). However, skin friction only occurs because of viscosity (Schlichting & Gersten, 2006, p. 1).

Viscosity is a physical property that describes the molecules' resistance to move in a fluid in motion relatively to said motion (Bollrich, 2013, p. 26). A practical consequence of viscosity is the no-slip condition, which assumes zero relative motion for viscous fluid layers in contact with solid boundaries (Day, 1990, p. 3). The no-slip condition serves as a basis for the development of fluid dynamic boundary layers near solid structures (Schlichting & Gersten, 2006, p. 27).

For transport properties, the viscosity is relevant as it induces momentum transport normal to the main flow direction of the fluid. Newton's law of viscosity represents this concept and defines shear stress:

$$\tau = \mu \frac{du}{dy} \tag{2.17}$$

τ – shear stress $[\frac{\mathrm{N}}{\mathrm{m}^2}]$,
μ – dynamic viscosity $[\frac{\mathrm{kg}}{\mathrm{m\,s}}]$,
$\frac{du}{dy}$ – shear rate (also velocity gradient) $[\frac{1}{\mathrm{s}}]$. (Schlichting & Gersten, 2006, p. 3)

Equation 2.17 is recognised as the definition of viscosity. Nevertheless, one must note that it showcases a particular case, the simple Couette configuration. Stokes' law generalises this formulation. (Schlichting & Gersten, 2006, p. 3)

Analogous to friction inducing momentum transport across a velocity gradient (from high to low velocity), one can envision mass transfer as a flux across a concentration gradient (from high to low density) which can be described using Fick's law of diffusion, and heat transfer as a flux across a temperature gradient (from high temperature to low temperature), respectively described by Fourier's law of heat conduction. (Oertel, 2017, p. 666)

2.2.3. Turbulence

Measuring flow velocities and interpreting flow velocity data do not get by without basic knowledge of turbulence. With varying definitions and approaches, theories and many fields of application, it is, however, a complex phenomenon to grasp. Tsinober, 2009 tries to apprehend this complexity by linking three problems when describing turbulence, i.e. nonlinearity, nonintegrability and nonlocality to the Navier-Stokes equations (Tsinober, 2009, p. 24). While in subsection 2.2.2 Stokes' law is mentioned as the generalisation of the simple Couette configuration, it itself is a derivation from these partial differential equations, which render it possible to describe flow fields completely. Although the Navier-Stokes equations offer universal validity and a deterministic approach to turbulence, turbulence remains difficult to describe because of its randomness, absence of reproducibility, and sensitivity to disruption. Therefore most turbulence models are deduced from solving the Navier-Stokes equations for special cases or using numerical models, which again does not lead to a general understanding of turbulence. (Tsinober, 2009, pp. 24–30)

A classic, empirical approach to turbulence is to distinguish two basic types of flow: laminar and turbulent flow regimes, linked by a transition between both states. This approach goes back to O. Reynolds (1883), who observed the regimes in pipe flow and introduced a parameter called the Reynolds number that specifies laminar flow at small values and turbulent flow for values surpassing a critical Reynolds number:

$$Re = \frac{u \cdot L}{\nu} \tag{2.18}$$

and

$$Re_{\text{crit,pipe}} = 2040 \pm 10 \tag{2.19}$$

Re	– Reynolds number [-],
u	– mean velocity [$\frac{\text{m}}{\text{s}}$],
L	– characteristic length [m],
ν	– kinematic viscosity [$\frac{\text{m}^2}{\text{s}}$],
$Re_{\text{crit,pipe}}$	– critical Reynolds number for pipe flow [-] (Avila et al., 2011, p. 5).

The critical Reynolds number differs for every geometry. (Bollrich, 2013, pp. 90, 99) As noted in equation 2.18, to determine the Reynolds number, usually the mean velocity is used for its calculation. This approximation is necessary as the random nature of turbulence causes fluctuation in instantaneous velocity measurements, which prompts a statistical description of the velocity field.

The Reynolds Decomposition facilitates this approach by splitting the velocity components into the sum of the mean velocity and the turbulence-induced fluctuation velocity:

$$u(x, y, z, t) = \overline{u}(x, y, z) + u'(x, y, z, t) \tag{2.20}$$

$u(x, y, z, t)$ – instantaneous velocity $[\frac{m}{s}]$,
$\overline{u}(x, y, z)$ – mean velocity term $[\frac{m}{s}]$,
$u'(x, y, z, t)$ – fluctuating velocity term $[\frac{m}{s}]$. (Spurk & Aksel, 2020, p. 226)

Combining the Reynolds Decomposition and the Navier-Stokes equations leads to a concept called Reynolds-averaged Navier-Stokes (RANS) Equations where the velocity components in the Navier-Stokes equations are substituted with the Reynolds decomposition as showcased in equation 2.20. Consequently, the fluctuation terms cancel out in the Navier-Stokes equations, which means that on average, they are as often and overall in the same magnitude positive as they are negative, leaving only the mean velocity components to be determined for a time-averaged solution. On the contrary, the nonlinear terms in the Navier-Stokes equations (the correlations between the fluctuating velocity components) do not zero out, as they are not statistically independent. (Schlichting & Gersten, 2006, pp. 505–510)

Recognised as Reynolds' stresses, they form the elements of the so-called Reynolds stress tensor instead. Thus, turbulent fluctuating motion increases momentum flux, but also heat and diffusion flux, which are picked up in subsection 2.2.2. Only the no-slip condition (see subsection 2.2.2) makes the Reynolds stress terms become zero in a layer called the viscous sublayer, where viscous stresses dominate. (Spurk & Aksel, 2020, pp. 226–230)

Following up on this statistical take on turbulence and bearing the basics of statistics outlined in section 2.1 in mind, one considers viewing flow velocities and flow velocity fluctuations in the light of probability distributions and respective consequences for data analysis that incorporates statistical parameters or normality. While normality is defined explicitly by the probability density function 2.1, there exist multiple notions on whether turbulent velocities can be accepted as normally distributed.

Many references speak of near-Gaussian distributions with only the tails of the distribution deviating slightly (Chanson & Larrarte, 2008, p. 69) and analytical solutions supporting this hypothesis but empirical findings straying off normality (She, 1991, p. 3). Others argue that error analysis in experiments ultimately leads to an approximation of normal distribution (Doroudian et al., 2010, p. 7) and Gaussian distribution to be given as a deduction from the CLT (see subsection 2.1.1) (Tennekes & Lumley, 1972, p. 218).

In practice, slight deviations from the Gauss distribution are sometimes neglected for assumptions but kept in mind when evaluating overall (Cea et al., 2007, p. 9, Strom and Papanicolaou, 2007, p. 4). On the other hand, there seems to be a consensus concerning velocity derivatives, which are viewed as clearly non-Gaussian (Li and Meneveau, 2005, p. 1, Bailly and Comte-Bellot, 2015, p. 11).

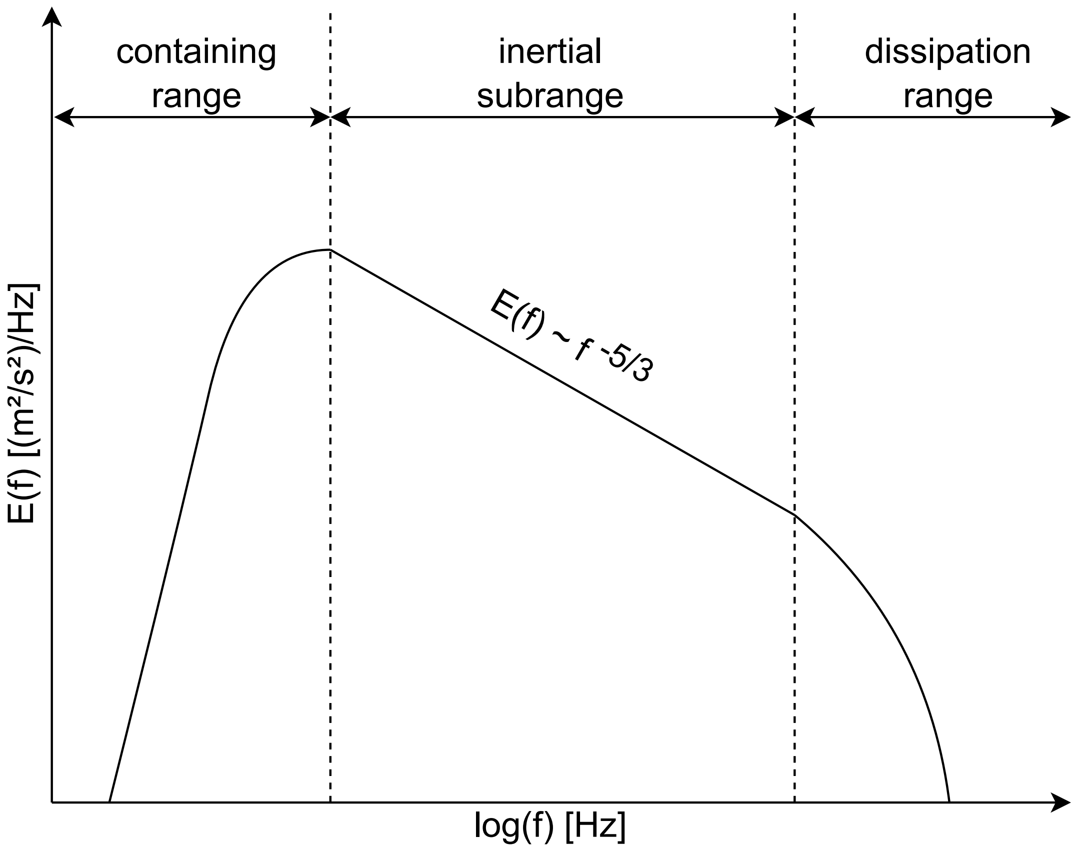


Figure 2.3.: Kolmogorov energy spectrum of turbulent flow (adapted from von Kusserow, 2018, p. 172).

Further, time and length scales can be deduced by integrating timely and spatial correlations from the Reynolds-averaged Navier-Stokes equations. These lead to different types of spectra: wavenumber or frequency spectra that describe underlying patterns and periodicity of turbulence. Three different length scales are considered to find laws for these reoccurring phenomena via data correlation. Then, it is assumed that there is reproducibility along the largest, the smallest and intermediate scales of turbulence. Based on these assumptions (separation of scales and similarity), A. Kolmogorov formulated Kolmogorovs' law in 1941 (see, e.g. in Kolmogorov, 1991), describing the energy spectrum of turbulent flow in relation to the wavenumber. (Reynolds, 1974, pp. 71–102, Skiadas, 2016, p. 10)

Figure 2.3 shows Kolmogorov's energy spectrum of turbulent flow in relation to the frequency. In this form, the spectrum is often used to compare spectra of turbulent flow measurements against it; the spectrum is deemed universal for every turbulent flow. While the features of the spectrum in the containing range are neglected, the distinct $-\frac{5}{3}$ slope in the inertial subrange and the plummeting energy levels in the dissipation range of Kolmogorov's spectrum are ideal indicators of turbulent flow spectra.

Statistical parameters quantify turbulence intensity and the turbulent kinetic energy (TKE). View subsection 3.3.1 to gain insight into the calculation of basic statistical parameters for velocity measurements and their connection to the uncertainty in measurements, as these parameters are needed for the calculation of turbulence parameters with

$$I_\mathrm{u} = \frac{\sqrt{\frac{1}{2}\left(\sigma_\mathrm{u}^2 + \sigma_\mathrm{v}^2 + \sigma_\mathrm{w}^2\right)}}{\overline{u}} \tag{2.21}$$

I_u – turbulence intensity in u-velocity component [-] (other components, respectively),
σ_u^2 – variance in u-velocity component $[\frac{\mathrm{m}^2}{\mathrm{s}^2}]$,
σ_v^2 – variance in v-velocity component $[\frac{\mathrm{m}^2}{\mathrm{s}^2}]$,
σ_w^2 – variance in w-velocity component $[\frac{\mathrm{m}^2}{\mathrm{s}^2}]$,
$\overline{u}$ – mean of u-velocity component $[\frac{\mathrm{m}}{\mathrm{s}}]$(other components, respectively),

and

$$\mathrm{TKE} = \frac{1}{2} \cdot \left(\overline{u'}^2 + \overline{v'}^2 + \overline{w'}^2\right) \tag{2.22}$$

TKE – turbulence kinetic energy $[\frac{\mathrm{m}^2}{\mathrm{s}^2}]$,
$\overline{u'}$ – mean of u-velocity fluctuating term $[\frac{\mathrm{m}}{\mathrm{s}}]$,
$\overline{v'}$ – mean of v-velocity fluctuating term $[\frac{\mathrm{m}}{\mathrm{s}}]$,
$\overline{w'}$ – mean of w-velocity fluctuating term $[\frac{\mathrm{m}}{\mathrm{s}}]$.

Because it is normalised, the turbulence intensity is an appropriate parameter to compare flow fields. The TKE is the energy represented in Kolmogorov's energy spectrum. (Adam and Lehmann, 2011, p. 101, Schlichting and Gersten, 2006, pp. 312, 406)

2.3. Basics of Signal Processing

To understand filter operations in signal processing, one needs to know the basic properties of signals and signal recording. This section first focuses on distinguishing between analog and digital signals by quantising and sampling. The Nyquist-Shannon sampling theorem then showcases the most fundamental principle of sampling. The last two subsections emphasise two signal processing operations needed for nearly every signal processing operation: convolution and transformation.

2.3.1. Signals

When dealing with signal processing, it is evident to define the term signal. Definitions from technical literature contain three basic keywords: physical phenomenon, information and mathematical function (Veloni, 2019, p. 3, Meyer, 2021, p. 1, Puthusserypady, 2021, p. 3, Stein, 2000, p. 42, Sundararajan, 2021, p. 1). As they are almost identical, only one is cited here:

> ”Signal is a physical phenomenon that carries information. This physical phenomenon is described by mathematical functions, and usually the signal and its mathematical function are used for one another, i.e., synonymous.”
> - Gazi, 2018, p. 1

The distinction of signals is between continuous and discrete, deterministic and random, periodic and aperiodic and real and complex signals. The most basic classification derives from sampling and quantising functions in time and values. While sampling means to record values of a continuous quantity only at set intervals of time and therefore reducing the resolution in time, quantising means to assign values to defined value ranges, thus reducing the resolution of the signal in amplitude. Consequently, sampling and quantising are always attended by a loss of information. The processes give four basic types of signals which are depicted in figure 2.4. This classification includes the definition of two often-used categories, i.e. analog and digital signals. (Veloni, 2019, pp. 6–7)

Periodic signals repeat their values after a period of time in the time domain. Therefore, they can be completely specified by only knowing the details of one period. Aperiodic signals do not follow a pattern in the time domain. (Puthusserypady, 2021, p. 9)

The values of deterministic signals can be predicted exactly without uncertainty. The fluctuation of physical phenomena characterises random signals, and therefore their measurement and further analysis also contain uncertainty. Often, a statistical analysis helps quantify and describe random signals. (Meyer, 2021, p. 16)

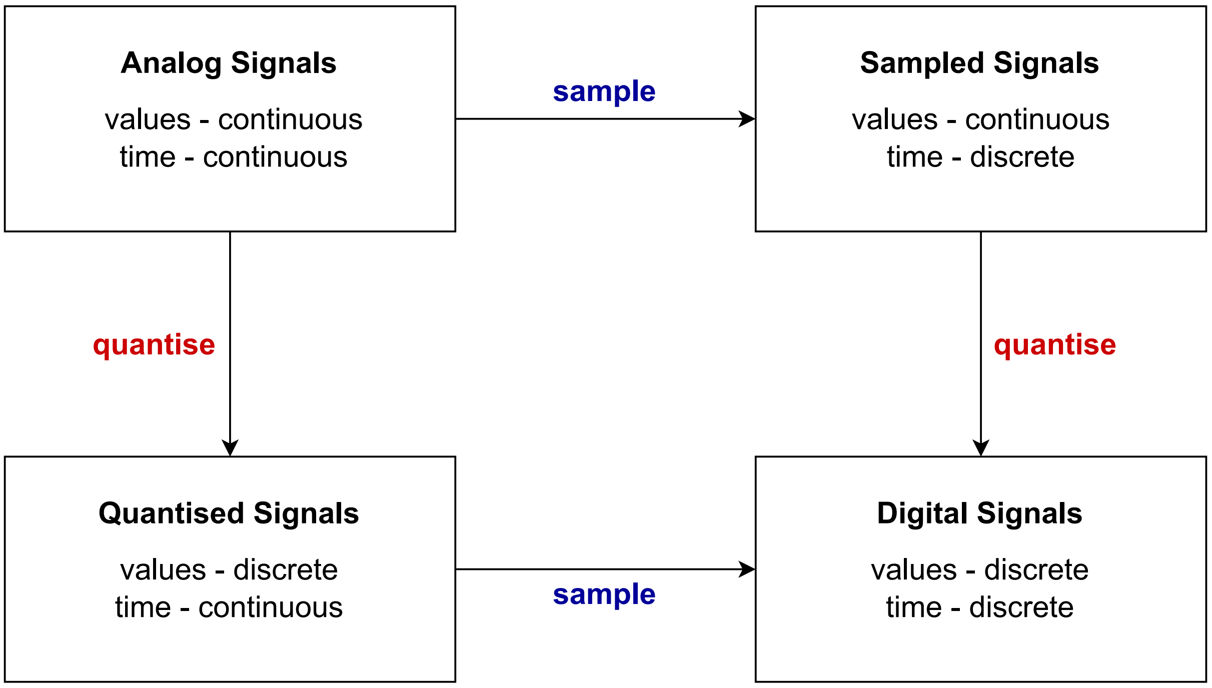


Figure 2.4.: Analog and digital signals (adapted from Meyer, 2021, p. 15).

There are some more classifications, such as real and complex signals, bound and unbound signals, even and odd signals and causal and noncausal signals, which can be neglected in our context. Hence, they are not further explained after this (Sundararajan, 2021, pp. 7–12).

2.3.2. Nyquist-Shannon Sampling Theorem

As shortly outlined in the previous subsection 2.3.1, sampling is a necessary operation when recording and working with signals. It reduces the complexity of a signal, so one must handle sampling with care to still record signals with enough detail. While undersampling a signal causes a loss of information, oversampling is less worrisome concerning data quality but still imposes an increased strain on resources (time, personnel, computational). For this, the Nyquist-Shannon sampling theorem proposes an optimum sampling frequency so that a signal which is continuous in time can be reconstructed entirely from recorded samples. It states that the sampling frequency must be chosen so that it is at least twice the maximum frequency of the signal:

$$f_\mathrm{s} \geq 2 f_\mathrm{max} \tag{2.23}$$

f_s – sampling frequency [Hz],
$f_{\max}$ – maximum signal frequency [Hz].

If the sampling frequency is chosen without respecting the Nyquist-Shannon sampling theorem, the signal will not be reconstructible without ambiguity, which causes effects known as aliasing and folding. (Veloni, 2019, p. 11)

Figure 2.5 shows an original signal (black line) that is sampled at a sampling frequency that does not respect the Nyquist-Shannon sampling theorem. The sampled values do not allow for a reconstruction of the signal without loss of information. In fact, in this case, it is impossible to reconstruct the original signal from the sampled values; instead, an aliased signal is constructed (red line).

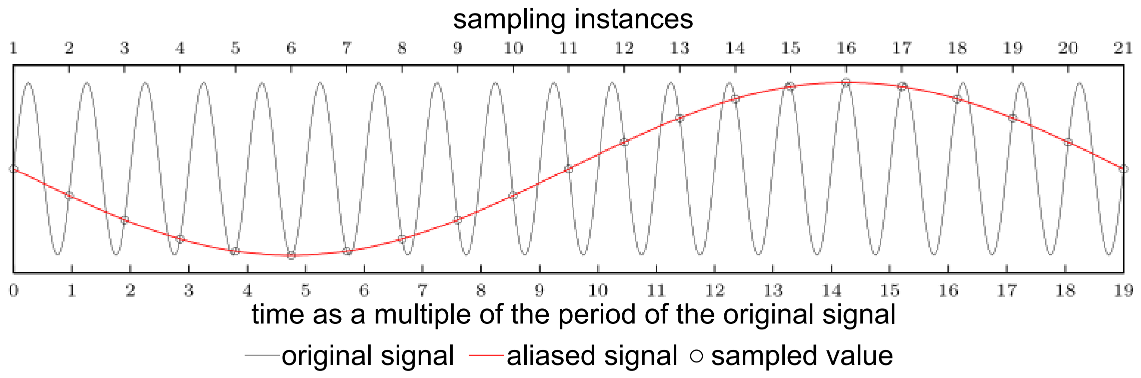


Figure 2.5.: Aliasing resulting from sampling at a lower sampling frequency than stipulated by the Nyquist-Shannon sampling theorem (adapted from Mrtz, 2004).

2.3.3. Convolution

Convolution is a mathematical operation to manipulate a function using another function. This operation is usually indicated by using the star symbol. For discrete data, it forms the sum of products of the elements of two finite sequences, while one sequence is time-reversed and the other remains unmodified. The operation is commutative, associative and distributive.

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k) = \sum_{k=-\infty}^{\infty} h(k)x(n-k) \tag{2.24}$$

$$n = N + M - 1 \tag{2.25}$$

$y(n)$	– output sequence,
$x(n)$	– input sequence,
$h(n)$	– input sequence,
k	– control variable,
n	– length of output sequence,
N	– length of first input sequence,
M	– length of second input sequence. (Sundararajan, 2021, pp. 44–46)

Convolution is the basic mathematical principle behind filter operations in signal processing, which are covered in section 2.4 of this thesis. It is essential to understand that convolution in the time domain corresponds to multiplication in the frequency domain to use the concept for filters. This statement can be proven by Fourier transforming the components of the convolution operation (Meyer, 2021, p. 36). Transformation operations are picked up in the following subsection 2.3.4.

2.3.4. Transformations

Most real physical signals are arbitrary in their amplitude profiles making them generally difficult to process in the time domain. Therefore, appropriate signal representation is needed, which can be reached by transforming signals using mathematical operations. Transformations - such as the Fourier transformation or the wavelet transformation - represent the signal in the frequency domain. They facilitate the manipulation of the signals and improve computational efficiency. (Sundararajan, 2021, p. 65)

Fourier Transform

The Fourier analysis is a mathematical method to approximate a signal in the time domain by sums of trigonometric functions. Fourier transformation is the core operation of said method. The result of Fourier transformation is called the Fourier transform; it gives all frequency components of the signal. There are four different types of the Fourier transform, but in the context of this thesis, only the discrete Fourier transformation (DFT) is of importance. (Meyer, 2021, pp. 164–166)

The DFT separates a sequence into harmonic trigonometric components and a direct component (which corresponds to the mean of the sequence) by expressing the data in the form of complex exponentials using Euler's formula:

$$e^{\pm j\omega} = cos(\omega) \pm jsin(\omega) \tag{2.26}$$

j – imaginary unit,
ω – angular frequency.

While the DFT of a signal can be calculated using

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-jk\frac{2\pi}{N}n}, k = 0, 1, 2, ..., (N-1) \tag{2.27}$$

the inverse discrete Fourier transformation (IDFT) transforms the Fourier transform back to the signal in the time domain

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{jk\frac{2\pi}{N}n}, n = 0, 1, 2, ..., (N-1) \tag{2.28}$$

$X(k)$	– sequence of complex amplitudes,
$x(n)$	– signal in time domain,
k	– DFT length,
n	– input sequence length,
N	– window length of signal. (Sundararajan, 2021, pp. 66–67)

In practice, numerical solutions often use the fast Fourier transform (FFT) and its counterpart, the inverse fast Fourier transform (IFFT), to calculate the DFT and the IDFT. The FFT and the IFFT use an algorithm designed by Cooley and Tukey in 1965. The algorithm calculates redundant parts of the DFT only once and hereby reduces the complexity of multiplication from order N^2 to order $Nlog(N)$. (Meyer, 2021, p. 175)

Wavelet Transform

The wavelet transform is a transformation operation that was introduced out of the need to represent and manipulate a signal in the time-frequency domain rather than only in the frequency domain like in Fourier analysis. Instead of decomposing a signal into trigonometric components, wavelet theory uses impulse signals in the time domain. These impulse signals are then compared to a base function called the mother wavelet. (Teolis, 2017, pp. 59–62)

This thesis only broaches the subject of wavelet transformation, as it is frequently picked up by publications in denoising contexts for a threshold approach proposed by Donoho, 1995 - the universal wavelet threshold.

2.4. Basics of Filters in Signal Processing

Filters in signal processing are tools to manipulate input data by applying a convolution operation. Filters can be interpreted as black-box systems, which makes their output data a result of the systems impulse response to the input. Generally, they work in both the time and the frequency domain but are usually applied in the frequency domain as a usual goal is to let specific frequencies pass while filtering out others. A function of frequency will therefore relate input and output. (Gazi, 2018, p. 233)

There are two basic types of filters: analog and digital filters. Their concepts correspond to the ones of analog and digital signals, i.e. analog filters can process continuously in time and amplitude, and digital filters handle naturally time-varying values as a quantised series. As signal processing is a field originating from electrical engineering, filters stem from manipulating signals in electrical circuits. (Meyer, 2021, pp. 123, 128)

Therefore, analog filters are realised using mechanical, electrical or electro-mechanical components such as capacitors, inductors or resistors. In electrical circuits, digital filter systems are realised using microprocessors, programmable logic blocks or chips, which carry out the needed mathematical operations of the digital filter after converting the analog signal to a digital signal. After the digital filter operations are successfully executed, the signal is reconverted to the analog signal. Nowadays, processing with digital filters is possible with standard computers after sampling. (Meyer, 2021, pp. 128, 150–151)

In analogy to analog filters in hydraulic contexts, one can think of screens or honeycombs in a flume. In the entryway of a flume, these structures can be used to eliminate, e.g. unwanted spin induced by pumps and inlet pipes. Another possibility to include analog filters in the hydraulic setup is to add electrical components in the electrical circuit of the ADV device. Processing the data with digital filters on the computer is the same as for electrical circuits.

The following subsections only discuss inherent digital filters and digital filters deduced from analog filters by approximation. However, this section on digital filters is not exhaustive, as picking up every detail would exceed the reasonable periphery of this thesis.

2.4.1. Filter Characteristics

Filter characteristics describe how filter design can be altered to generate the desired filter effect. Generally, the goal of a filter operation needs to be before defining filter characteristics. Four basic characteristics can differentiate between filters:

- The passband specifies the desired frequencies allowed to pass through the filter.

- The cutoff frequency (f_c) defines the boundary of the passband.
- The stopband specifies the undesired frequencies which are held back by the filter.
- The transition frequencies between the passband and the stopband are called the skirt response. Here, the reduction in signal amplitude changes rapidly, which is referred to as attenuation. (Winder, 2002, pp. 22–23)

Another filter property is filter order. It describes the steepness of the stopband's slope which defines the delay present in the filter operation. The rate of attenuation is defined as 20 dB per decade. All previously explained characteristics can be determined by viewing the frequency response function of a filter. (Winder, 2002, p. 27) Frequency responses can be categorised following four prototypes: the lowpass, highpass, bandpass, and bandstop ideal filters (Alessio, 2016, p. 185). The four prototypes are displayed in the following figure (see figure 2.6), with a colour scheme indicating the four characteristics. The digital filters picked up in this section

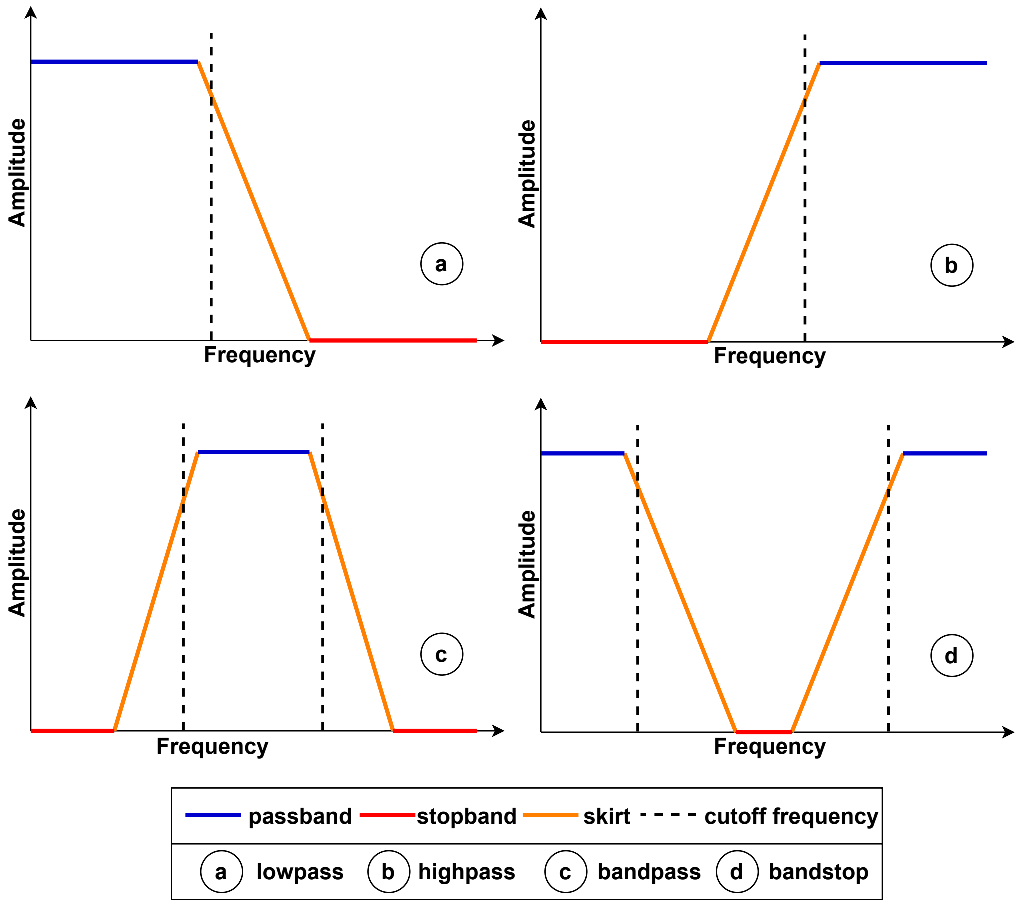


Figure 2.6.: Filter prototypes (adapted from Winder, 2002, p. 23).

are all linear time-invariant (LTI) systems. The formal characteristics of these filters are time invariance, linearity, causality and stability. While time invariance grants constant system coefficients for the investigation period, linearity ensures superposition and scaling principles of linear systems, respectively, additivity and homogeneity. Causality states that the system is not anticipatory; thus, the output only depends on the input, and stability specifies that sequences with limited absolute values will produce output sequences within the bounds of this limitation. (Haslwanter, 2021, p. 73, Alessio, 2016, pp. 36–37)

Filter operations always consider a neighbourhood for every point of the signal. This means that they use previous and subsequent measurements of every time step. For each tuple of neighbouring values, the filter function is applied, which causes a slight phase delay in the output signal of filter operations. (Haslwanter, 2021, p. 82, Alessio, 2016, p. 193)

From all the above information, the general expression for a filter operation can be written as follows:

$$y[n] = h[n] * x[n] \tag{2.29}$$

$y[n]$ – signal output function,
$h[n]$ – filter impulse response,
$x[n]$ – signal input function (Alessio, 2016, p. 38).

This expression - just as all the following equations on digital filters - follows a numerical, computational notation rather than a mathematical one. The advantages of using this notation will become apparent when viewing and following the paragraphs on the digital signal processing (DSP) representations.

2.4.2. Finite Impulse Response Filters

Finite impulse response (FIR) filters are typically nonrecursive linear digital filters. Their characterising feature is that they generate an impulse response of finite nature. They can be implemented both digitally or analogously but are mainly used as digital filters.

$$y[n] = \sum_{k=0}^{M} b_k \cdot x[n-k] \tag{2.30}$$

$y[n]$ – signal output function,
k – control variable,
M – order of magnitude of input sequence,
b_k – filter coefficients,
$x[n]$ – signal input function (Haslwanter, 2021, p. 73). (Gazi, 2018, pp. 290–291)

The scheme for an FIR filter in figure 2.7 shows how computers process the digital filter operations. This DSP representation highlights the finite nature of the filter as it works by solely processing data from the input sequence. FIR filters are easy to realise because their design is linear, and the desired optimisation typically follows straight-forward optimisation goals. Their finite nature guarantees stable operation, i.e. convergence toward a constant. The disadvantage to FIR filters is that they require large memory contingencies, and for complex problems, the process can become time-consuming. Common FIR filters are median filters, moving average filters (also known as boxcar filters) and Savitzky-Golay-Filters. (Stein, 2000, pp. 602–603)

2.4.3. Infinite Impulse Response Filters

Contrary to FIR filters, infinite impulse response (IIR) filters are usually recursive, and their impulse response does never exactly reach zero but instead lasts indefinitely. This property results from the impulse response of IIR filters considering the most recent output values ongoing from the first output. (Haslwanter, 2021, p. 75)

$$y[n] = \sum_{k=0}^{M} b_k \cdot x[n-k] - \sum_{k=1}^{N} a_k \cdot y[n-k] \tag{2.31}$$

$y[n]$	– signal output function,
k	– control variable,
M	– order of magnitude of input sequence,
b_k	– filter coefficients,
$x[n]$	– signal output function,
N	– order of magnitude for most recent output values,
a_k	– filter coefficients for most recent output values (Haslwanter, 2021, p. 75).

The scheme for an IIR filter in figure 2.8 shows how computers process the digital filter operations. This DSP representation highlights the infinite nature of the filter as it works by processing data from the input sequence and additionally taking recent output data into account. The design of IIR filters is also linear, but while the desired optimisation by the impulse response of IIR systems often approaches a constant (usually zero), it does not always. Sometimes, the infinite nature of IIR filters makes the operations become unstable and produce unbound outcomes. IIR filters require less memory than FIR filters and are more computationally efficient and easier to implement than FIR filters. However, FIR filters can be easier to design when specific frequency responses are required or when a form aside from the beforementioned four prototypes is needed. Common IIR filters are Butterworth filters, Chebyshev filters, Cauer filters and Bessel filters. (Stein, 2000, pp. 602–603)

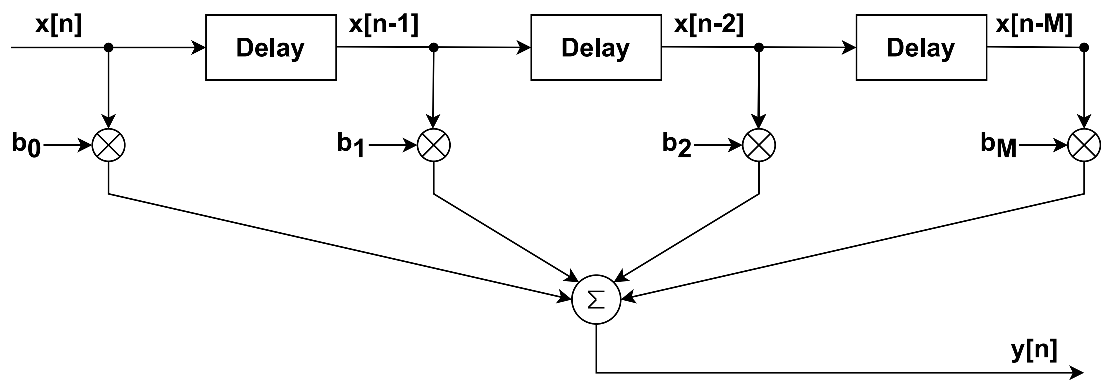


Figure 2.7.: Digital signal processing representation of a finite impulse response filter (adapted from Haslwanter, 2021, p. 74).

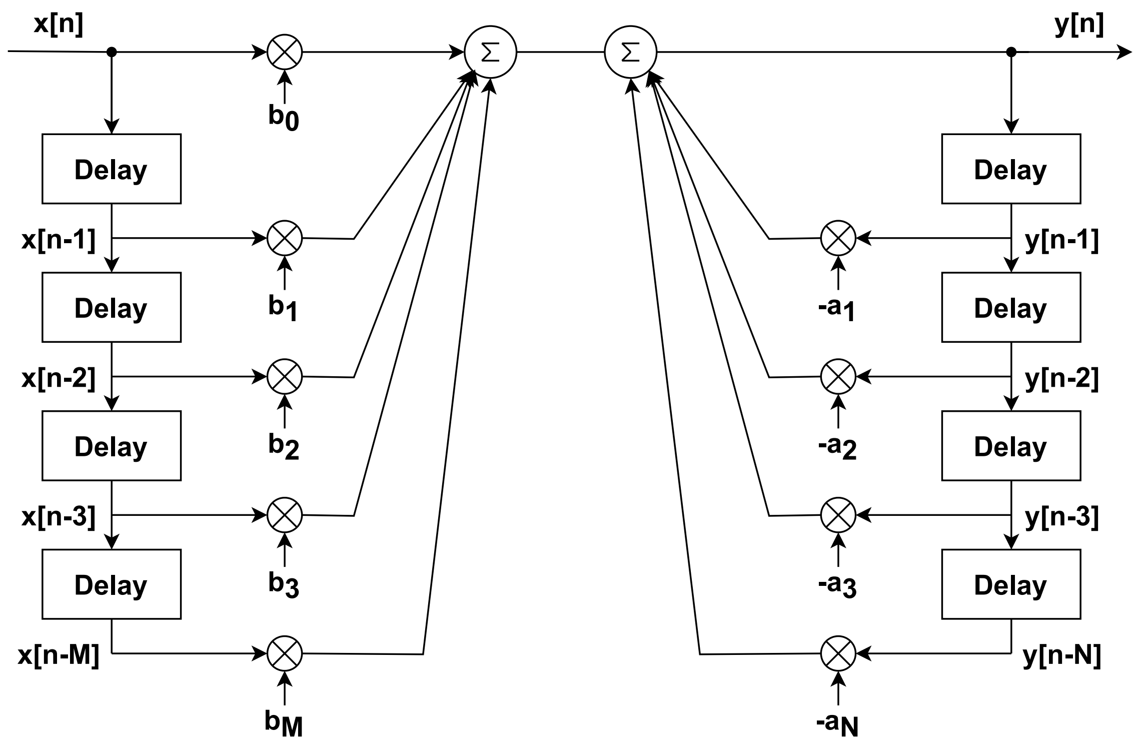


Figure 2.8.: Digital signal processing representation of an infinite impulse response filter (adapted from Haslwanter, 2021, p. 75).

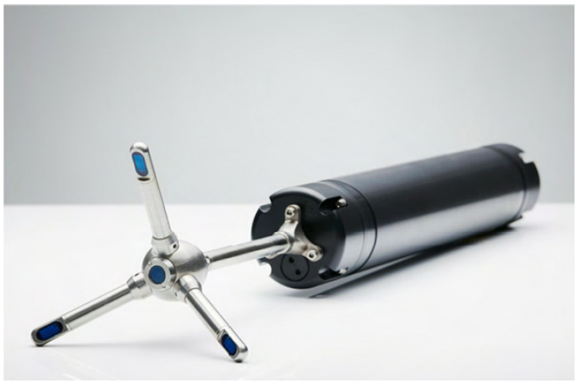
3. Acoustic Doppler Velocimeter

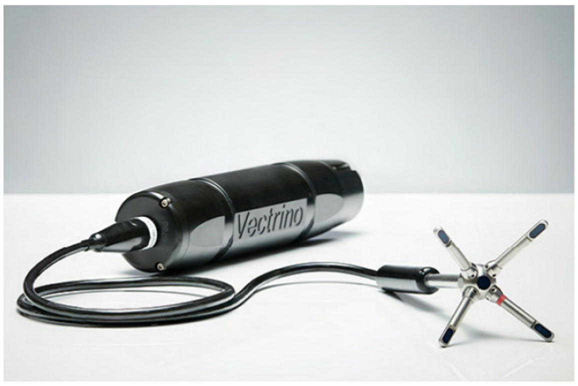
ADVs are measurement devices predominantly used to record instantaneous three-dimensional single-point velocity data in fluids. They are commonly used for laboratory and field measurements. Notable manufacturers of ADV devices are the Nortek Group, Norway (devices: Vector, Vectrino and Vectrino+) and SonTek, USA (devices: microADV, Argonaut, FlowTracker). Since the Chair of Hydraulic Engineering at TU Darmstadt only uses Nortek devices, this paper emphasises those.

The following sections describe the fundamentals of ADVs in theory and practice: components, the basic principles of acoustic Doppler velocimetry, errors, and commonly applied data analysis methods for ADV data.

3.1. Components

An ADV comprises a pressure case, a probe with several transducers, power and communication cables and sensors. The pressure case connects to a computer and an external power source with a power and communication cable. It contains a single circuit board that manages the power supply and signal processing. This electronic module acts as the interface between the probe and the connected computer. A cable or a rigid rod links the probe to the pressure case. The probe mounts two types of transducers: one transmitter and, varying by model, several receivers. While the transmitter sits at the centre of the probe head, the receivers are at the tip of each angled receiver arm. Vertical and horizontal operation of probes is possible. All the Nortek models have a temperature sensor. The Vector also comes with a pressure sensor to collect pressure data, tilt sensor to detect the tilt of the probe against a set axis, and a compass or an inertial motion unit to translate velocity measurements into earth-north-up (ENU) coordinates. Nortek also offers the Vectrino as a sturdier field probe. The pictures in figure 3.1 show two models manufactured by Nortek with a differing number of receivers and different probe fixing. (Nortek, 2021, pp. 26, 41)


(a) Nortek Vector



(b) Nortek Vectrino

Figure 3.1.: Nortek ADVs (Nortek, 2021, p. 1).

3.2. Basic Principles of Acoustic Doppler Velocimetry

To explain the measurement principle of ADVs, one must first understand the basic physical principles of sound waves and their propagation. Therefore, this section starts with the underlying physical principles and ultimately explains the working principle of ADV measuring.

3.2.1. Sound Propagation

Literature defines sound as the propagation of a local change in pressure in gases, liquids and solids. Sound propagates in longitudinal mechanical waves. Transversal mechanical waves are also possible in solids or at boundary layers between gases and liquids. (Hering et al., 2016, pp. 420, 554)

Several parameters specify sound waves: amplitude, wavelength, period, frequency and wave speed (Harten, 2021, p. 146). While the amplitude gives the maximum deflection of the wave, the wavelength is the length after which the wave pattern repeats, and the period is the time needed for one repetition. The inverse of the period is the frequency. It shows the number of oscillations per time unit. (Wolfson, 2020, p. 267)

The speed of sound depends on the elasticity for solids, the compressibility for gases and liquids (also bulk modulus), and the density of the medium which the sound passes through. In this thesis, only the speed of sound in fluids is relevant, which calculates with

$$c_{\mathrm{f}} = \sqrt{\frac{K}{\rho}} \tag{3.1}$$

c_f – speed of sound in fluids $[\frac{\mathrm{m}}{\mathrm{s}}]$,
K – bulk modulus $[\frac{\mathrm{N}}{\mathrm{m}^2}]$,
ρ – density $[\frac{\mathrm{kg}}{\mathrm{m}^3}]$. (Harten, 2021, pp. 153–154)

Sound propagation obeys all observations that also prevail for light waves, such as reflection, refraction, and interference, and thus the superposition principle (Wolfson, 2020, pp. 273–281).

3.2.2. Doppler Effect

A point source of sound produces mechanical waves which propagate uniformly around the point source (see figure 3.2a). If the sound source moves, the propagation shifts from uniform to non-uniform. A stationary sound source and a moving observer of sound (e.g. a sensor or the human ear), or both of them moving at different speeds or in different directions, have the same effect. Figure 3.2b shows this shift for two observers (A and B) for a moving the sound source. Dependent on the direction of movement of the sound source, the sound wave apices appear with decreased or increased spacing. Consequently, the wavelengths decrease or increase, respectively, which affects frequency as wavelength and frequency behave inversely proportional. This effect is called the Doppler effect, or Doppler shift. (Harten, 2021, pp. 57–58, Wolfson, 2020, pp. 282–283)

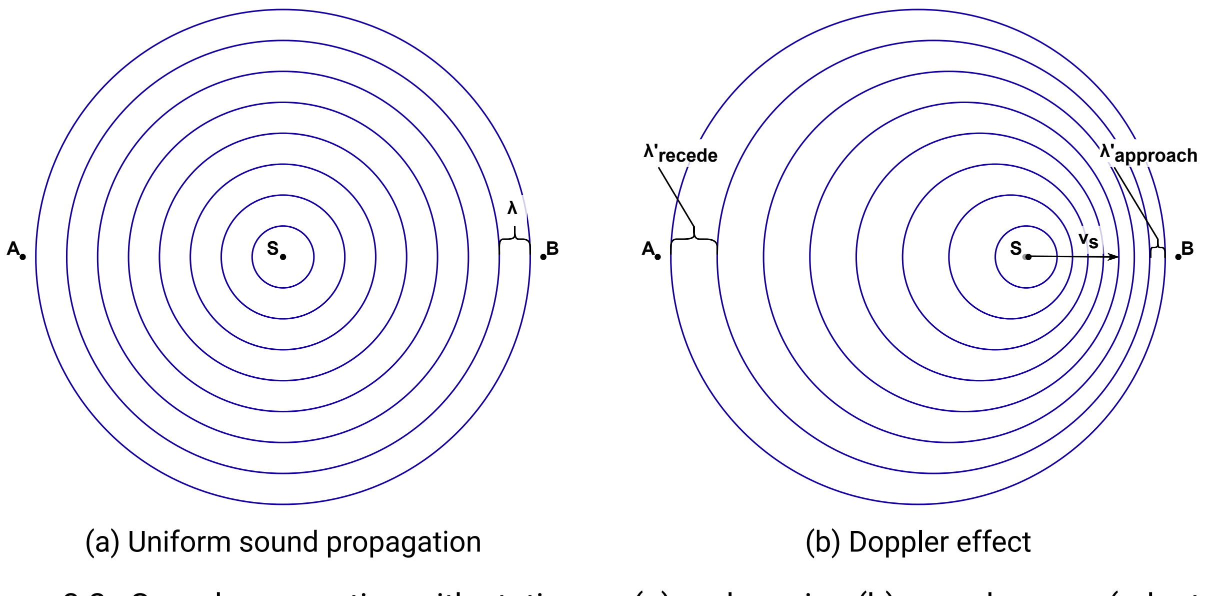


Figure 3.2.: Sound propagation with stationary (a) and moving (b) sound source (adapted from Wolfson, 2020, p. 282).

3.2.3. Working Principle of the Acoustic Doppler Velocimeter

The ADV device is installed in the water body and put into operation. First, the transmitter produces a pair of sound pulses of a preset frequency into the water. The sound pulses propagate per the laws of sound propagation. Because the sound waves reflect from particles suspended in the water in the direction of the instrument, they can then be picked up by the receivers of the ADV device. As the suspended particles in the water move along pathlines at flow velocity (see subsection 2.2.1), the reflection of the sound signals is also affected by the current flow velocity, which manifests itself in the form of a change in frequency - the Doppler effect. By detecting the phase difference of the two sound echos, the current velocity can be output using

$$v = \frac{\Delta\varphi c}{4\pi f_\mathrm{s} \Delta t} \tag{3.2}$$

v – velocity [$\frac{\mathrm{m}}{\mathrm{s}}$],
$\Delta\varphi$ – phase difference [-],
c – speed of sound [$\frac{\mathrm{m}}{\mathrm{s}}$],
f_s – transmitted frequency (sampling frequency) [Hz],
Δt – time difference between pulses [s]. (Nortek, 2021, pp. 12–13)

As the instantaneous velocity is measured by the degree of similarity or difference between the two sound pulses, the measurement technique is also known as the pulse-coherent method. The time between the two pulses introduces a small lag to each measurement which is important to consider for the range of velocities to be quantified. (Nortek, 2021, pp. 12–13)

The Nortek ADV devices transmit from the centre to a space in the water body called the sampling volume. This volume is defined at a known distance from the centre transducer, the transmit length and the receivers' intersection zone. The Nortek devices can measure in Cartesian coordinates or by beam orientation. The beam coordinates are the ”rawest” format which means that they are best when wanting to perform data analysis operations to decrease uncertainty. Further pointers on the uncertainty in measurement of ADV measurements will be brought up in the following section 3.3. (Nortek, 2021, pp. 13–15)

3.3. Uncertainty in Measurement

It is indispensable for every experiment that relies on measuring quantities to discuss measurement errors and uncertainty in measurements. The following subsections introduce a standard recommendation for evaluating statistical and non-statistical parameters that influence measurements - the Guide to the Expression of Uncertainty in Measurement (GUM). The GUM defines two types of evaluation and two types of errors, which are identified and indicated in detail for ADV measurements after this.

3.3.1. Guide to the Expression of Uncertainty in Measurement

To facilitate standardisation of the expression of uncertainty in measurement, the Joint Committee for Guides in Metrology (JCGM) regularly publishes and revises an international standard called the Guide to the Expression of Uncertainty in Measurement. Overall, the guide offers best-practice advice for measurements and a discussion of uncertainty, its components, possible corrections and practical tips. Ultimately, it recommends a procedure for the evaluation of uncertainty in measurement. (JCGM, 2008b)

The GUM differentiates between two types of uncertainty assessment methods: statistical analysis of measurements (type A) and methods relying on scientific judgement based on peripheral information on measurements (type B). Type A evaluation includes calculating statistical parameters, namely the arithmetic mean, variance, covariance or correlation for multi-measurand measurements and skewness and kurtosis, the third and fourth statistical moments. With the latter two parameters being taken on by the first supplement to the GUM, which provides a general numerical approach to GUM principles (JCGM, 2008a, p. 29). As variance can be hard to interpret in practice, the guide also allows the specification of the standard deviation. Type B evaluation is linked to a concept which is often called subjective probability as it not only takes the concerned measurement into account but also includes data from previous measurements, general knowledge concerning the measurement, boundary conditions, equipment and materials, expert and manufacturer recommendations, calibration procedures, certificates and reference data. Type A and type B evaluations can be combined to analyse data. (JCGM, 2008b, pp. 22–23)

The GUM defines uncertainty in measurement as follows:

> ”**uncertainty (of measurement)**
> parameter, associated with the result of a measurement, that characterizes the dispersion of the values that could reasonably be attributed to the measurand”
> - JCGM, 2008b, p. 14

Consequently, the uncertainty term and the traditional error term must be explicitly distinguished: Errors can be of random or systematic nature. While random errors result from unpredictable random effects, systematic errors result from recognised influence quantities. Both error types cause effects on the observation of the intended measurand, which in turn affects the uncertainty in said measurement. It is impossible to compensate for random effects, but their impact on uncertainty in measurement can be minimised by increasing the number of observations. Systematic errors can often be quantified, corrected or compensated. Even after correcting effects caused by errors, the measurement result remains an estimate, and a discussion of uncertainty remains essential. (JCGM, 2008b, p. 17)

Type A Evaluation

Here, type A evaluation parameters are defined with their equations specified in the GUM. However, the general terms are replaced with the notation for one velocity component (u, v and w) specified in subsection 2.2. The equations apply to the other velocity components, respectively. Also, alternative notations are included to ensure clear identification of parameters:

$$\mu_\mathrm{u} = \bar{u} = \frac{1}{n} \cdot \sum_{k=1}^{n} u_k \tag{3.3}$$

$$\sigma_\mathrm{u}^2 = \frac{1}{n-1} \cdot \sum_{k=1}^{n} (u_k - \bar{u})^2 \tag{3.4}$$

$$\sigma_\mathrm{u} = u_\mathrm{rms} = \sqrt{\frac{1}{n-1} \cdot \sum_{k=1}^{n} (u_k - \bar{u})^2} \tag{3.5}$$

$\mu_\mathrm{u}, \bar{u}$	– mean $[\frac{\mathrm{m}}{\mathrm{s}}]$,
u_k	– observed value $[\frac{\mathrm{m}}{\mathrm{s}}]$,
k	– control variable [-],
n	– number of observations,
σ_u^2	– variance $[\frac{\mathrm{m}^2}{\mathrm{s}^2}]$,
$\sigma_\mathrm{u}, u_\mathrm{rms}$	– standard deviation $[\frac{\mathrm{m}}{\mathrm{s}}]$. (JCGM, 2008b, p. 22)

For measurements that determine more than one measurand at once, the GUM recommends giving the covariance matrix elements or the elements of the correlation coefficient matrix. This applies in the case of flow velocity measurements with ADV devices as the devices record data for all three velocity components. The covariance of velocity components u and v - and other combinations, respectively - calculates as follows:

$$\overline{u'v'} = \frac{1}{n-1}\sum_{k=1}^{n}(u_k - \bar{u})(v_k - \bar{v}) \tag{3.6}$$

$\overline{u'v'}$ – covariance of velocity components u and v [$\frac{\mathrm{m}^2}{\mathrm{s}^2}$].

The correlation coefficient matrix is the normalised covariance matrix. Normalisation is performed by dividing by the multiplication of the standard deviations of the respective velocity components. (JCGM, 2008b, pp. 4, 12, 42) Further, the GUM indictates that skewness and kurtosis can be provided as supplementary information to analyse data regarding its distribution (JCGM, 2008a, p. 29). Skewness, the third statistical moment is calculated using the following formula with

$$\gamma_\mathrm{m} = \frac{\frac{1}{n-1}\cdot\sum_{k=1}^{n}(u_k - \bar{u})^3}{\sqrt{\frac{1}{n-1}\cdot\sum_{k=1}^{n}(u_k - \bar{u})^2}} \tag{3.7}$$

and kurtosis, the fourth statistical moment follows as

$$\omega_\mathrm{m} = \frac{\frac{1}{n-1}\cdot\sum_{k=1}^{n}(u_k - \bar{u})^4}{\left(\frac{1}{n-1}\cdot\sum_{k=1}^{n}(u_k - \bar{u})^2\right)^2} \tag{3.8}$$

γ_m – skewness [-],
ω_m – kurtosis [-] (Agarwal et al., 2021, p. 5).

Type B Evaluation

As type B evaluation does not follow a standardised procedure or a set of equations, it is nearly impossible to give an exhaustive summary of all type B evaluations possible for ADV measurements. While the ADV itself gives out a few parameters to rate the quality of any ADV observation, technical literature deems those as not fit to evaluate the quality of measurements (see section 4.1). They do not give information on the uncertainty in measurement involved. However, many type B evaluation strategies are implied by data correction methods as the approaches generally demarcate 'good' and 'bad' data (see sections 4.3 and 4.2). As stated by the GUM, uncertainties in correction might lead to bias and thus can introduce further uncertainty to measurements (JCGM, 2008b, p. 17).

A standard method for velocity measurements apart from the statistical evaluation is to inspect ADV measurements using turbulence spectra. By assuming that the turbulence model assumptions hold, it is expected that the slope of the spectra follows the theoretical Kolmogorov turbulence spectrum. The $-\frac{5}{3}$ slope in the inertial subrange of the spectrum is especially useful for comparison (see subsection 2.2.3). This approach is often used as authors assume the type A evaluation parameters to remain the same when correcting specific errors. Those errors have zero means, so statistical parameters sometimes cannot indicate the correction method's success or failure. However, there have been no efforts to formalise an evaluation using the Kolmogorov dissipation range. The only broadly recognised criterion is the $-\frac{5}{3}$ slope in the inertial subrange.

3.3.2. Random Effects

As previously mentioned, random effects are caused by unpredictable, random occurrences that influence the observation of the measurand. This type of effect is essential for ADV measurements, as random effects are inherent to flow velocity measurements: variation in flow velocities are caused by the turbulent nature of flows (see subsection 2.2.3), the forming of air bubbles (Mori et al., 2007) and swirls of dispersed sediments (Chanson et al., 2008), interference of fauna (e.g. fish movement) or nautic traffic (Chanson et al., 2007) when measuring in natural systems. The only way to resolve said variation is to sample at an adequate sampling rate to reach high enough timely resolution to display the measurand in detail. Another factor is waves requiring adequate sampling to find periodicity and to capture the sea state statistically. Then, wave fluctuations can be considered for static ADV measurements (C. J. Huang et al., 2018). (Nortek, 2021, pp. 20, 34–35, 58) A procedure to find an optimal sampling time to reduce the influences of random effects while still considering efficiency in time and resources is assayed in chapter 7.

3.3.3. Systematic Effects

In this subsection, several systematic errors that lead to systematic effects in ADV measurements are highlighted. Most of these errors can be avoided by considerate preparation and thoughtfully tuning the settings of the ADV. Many can also be prevented by preliminary tests or cross-examinations. This includes visually inspecting the device before use, pinging in still water and test measuring to chose adequate velocity ranges.

Data analysis methods which attempt to reduce uncertainties in measurement caused by systematic effects are presented in chapter 4 some of which are implemented in Python and tried for hydraulic data at hand in chapter 5.

Calibration

ADV measurement devices are factory-calibrated and remain accurate unless there are mechanical impacts that physically harm the probe heads. Calibrated ADVs have a bias of less than 1 % of the measured value. However, calibration might be needed for auxiliary sensors such as temperature, compass, pressure or tilt sensors. (Nortek, 2021, pp. 13, 23, 85)

Signal Strength

To confidently measure velocity with ADVs, the reflection of the emitted signal needs to surpass a certain level, which is quantified by a parameter called signal strength. It is measured using the original signal amplitude in a unit called counts, which is connected to the decibel term and describes the degree of required signal amplification by the device. (Nortek, 2021, pp. 19–20) A low number of suspended scatterers in the fluid usually leads to decreased reflection and thus to low signal strength. (Nortek, 2021, p. 113) The type of suspended particles also affects signal strength. (V. I. Nikora & Goring, 1998, p. 3)

Pulse-to-pulse Interference

The pair of short acoustic pulses emitted by an ADV can be exposed to circumstances that interfere with the expected reflection from particles. This effect is called pulse-to-pulse interference. (Nortek, 2021, p. 118) It occurs when measurements take place in proximity to solid boundaries or the water surface: the first pulse hits the boundary, is reflected, and arrives at the sampling volume while the second pulse simultaneously passes the sampling volume. The location where pulse-to-pulse interference occurs is called a weak spot. (Nortek, 2021, p. 55) The consequence of weak spots for velocity data from ADV measurements is an underestimation of flow velocities. (Precht et al., 2006, p. 10)

Connection Errors

Errors caused by an interrupted connection between the probe and the computer, either provoked by hardware or software, are called communication errors. They might occur due to cable damage, overly long cables or momentary loss of power. Depending on the specific issue, the sample might miss data points, or the device automatically adjusts the sampling rate. (Nortek, 2004, pp. 34–35, 106)

Noise Floor

The noise floor is specific for a device and describes the measured velocity when theoretically no velocity measurement is possible. The device will measure nothing but noise. The noise floor can be identified by, e.g. pinging in the air. The device will then give a number of counts which can be validated with reference values from the manufacturer. All sensor heads count similar noise floors. (Nortek, 2021, p. 71)

Phase Wrapping

Phase wrapping (also Doppler aliasing) results from pulse-coherent systems not identifying the Doppler phase shift distinctly. The Doppler phase shift is calculated using the covariance method with the arctangent, limiting resulting angles between $-\pi$ and π. (Rusello, 2009, p. 5) Yet, if the modulus of the Doppler shift is greater than π, the possibility to unambiguously determine the Doppler phase shift disappears, and velocity measurements show sudden, unexpected changes in magnitude, most of the time accompanied by a change in sign. (Nortek, 2021, p. 118)

Acoustic Streaming

Secondary flow caused by acoustic pulses generated by the device is called acoustic streaming. The induced streaming has a rather small effect on velocity measurements as it only ranges up to velocities of $3\,\frac{\mathrm{cm}}{\mathrm{s}}$. (Nortek, 2021, p. 56) Therefore, acoustic streaming is particularly important for measurements of small velocities as the possible errors might be relatively large compared to the measured values.

Analytical solutions can approximate acoustic streaming. They prove a dependency of the magnitude of acoustic streaming effects on the transmitted sound amplitude. Experiments with ADVs show increasing secondary flow in close proximity to the device and a beginning decline of the effects between $3\,\mathrm{cm}$ to $9\,\mathrm{cm}$. Also, flow perpendicular to the ADV measurement axis reduces but does not prevent acoustic streaming effects. (Poindexter et al., 2011, pp. 2, 8–11)

Probe Head Vibration

Even small vibrations of the ADV mount can generate large spikes in velocity data. Vibrations can be caused by wave impacts, high flow rates, unstable mounting of the device or vibrations from the laboratory, e.g. generated by nearby equipment. (Nortek, 2021, p. 63)

4. Data Analysis Methods for Acoustic Doppler Velocimeter Data

This section points out existing data analysis methods for ADV data. Subsection 4.1 addresses basic cutoff filters that use parameters directly output by the ADV device. The subsequent two subsections divide data analysis methods by two different types of erroneous data. While subsection 4.2 discusses analysis methods for detecting outliers in hydraulic data, i.e. despiking, subsection 4.3 picks up data analysis methods for reducing error effects caused by noise, i.e. denoising. However, it needs to be pointed out that defining the scopes of despiking and denoising analysis methods and distinguishing them from each other is quite tedious and not always possible. Some methods are used for both purposes or adapted to the other. Lastly, subsection 4.4 outlines options to remove or replace erroneous data points and indicates the advantages and disadvantages of either approach.

4.1. Basic Cutoff Filters

Basic cutoff filters reject measurements which do not fit a chosen quality cutoff value. ADVs give out two parameters to which basic cutoff filters can be applied: the signal-to-noise ratio (SNR) and the correlation score (COR). If the filters detect inadequate data points, the user either deletes them without replacement or chooses a data replacement method to find an appropriate replacement for the data points (see section 4.4). Several authors (Köse, 2013, p. 2, Agarwal et al., 2021, p. 2) also refer to these methods as spike detection filters. Other authors (Mori et al., 2007, p. 4) say that the parameters applied by these filters are not fit for processing ADV data as they show no apparent relation to erroneous data points. Generally, these basic filters are not very present in the relevant literature, and publications seldom pick them up. The occasional use indicates that they only offer limited means to process ADV data and must therefore only be applied as fast preliminary tools to get an overview of the recorded data and its quality.

4.1.1. Signal-to-noise Ratio Filter

The SNR is a parameter given by the device for each probe head. Just as the original signal amplitude in the unit counts (see subsection 3.3.3), it assesses the signal strength of a measurement. The SNR uses the unit decibel and relates the signal amplitude to the total current amplitude, including the noise floor (see subsection 3.3.3):

$$\text{SNR} = 20log_{10}\frac{A_\text{s}}{A_\text{t}} \tag{4.1}$$

SNR – signal-to-noise ratio [dB],
A_s – signal amplitude [counts],
A_n – total amplitude [counts].

Nortek recommends an SNR above 15 dB for original data and above 5 dB for mean data. (Nortek, 2021, p. 20)

4.1.2. Correlation Score Filter

The device also gives out the correlation score for each probe head. It describes the degree of similarity between the two pulses transmitted by the device. Complete correlation gives a score of 100 %, and no correlation gives a score of 0 %. A high correlation of the two pulses indicates a valid phase shift but not necessarily an accurate velocity measurement. For correlation filters, many researchers use a threshold of 70 % (Köse, 2013, p. 3, Wahl, 2000, p. 6, Agarwal et al., 2021, p. 8). Nortek recommends filtering data by a quality cutoff value determined by visual prescreening the recorded data. (Nortek, 2021, p. 21)

For a dataset produced with jet streams, Islam and Zhu, 2013, p. 6 find that correlation filtering removes many good data points and does not improve data quality. The authors conclude that correlation filtering does not suit all flow and turbulence conditions.

4.2. Despiking

An often observed problem with ADV data is that the velocity time-series contains sudden spikes. The spikes are assumed not to be part of the measured random variable as the corresponding accelerations exceed physical upper limits (Wahl, 2000, pp. 6–7). This error is known as phase wrapping (see subsection 3.3.3). The presence of spikes in ADV time-series influences the calculation of flow and turbulence parameters as data outliers can significantly alter statistical parameters (see subsection 2.2.3).

The following subsections outline scientific endeavours to eliminate spike data from ADV velocity signals, their benefits, challenges, critical reviews and efforts to improve despiking. While they pick up the most popular approaches to ADV despiking and even some lesser-known ones, they are not exhaustive.

4.2.1. 3D Phase Space Thresholding

A broadly applied method to eliminate spikes in ADV time-series is the 3D phase space thresholding method by Goring and Nikora, 2002, which is discussed and adapted by Wahl, 2003 and Mori et al., 2007.

The method is composed of three ideas:

- The derivative of a signal amplifies its high-frequency components.
- The universal threshold from wavelet theory (see subsection 2.3.4) gives the expected maximum of a time-series.
- Data holding similar properties form a cluster, and data outside the cluster must be suspected to be outliers.

From these ideas, a procedure is derived: First, one forms the first and second derivatives of the measured velocities, then calculates the standard deviation and the expected universal threshold. Then, the velocities and the second derivatives are cross-correlated, and the maximum boundaries of an ellipse in 3D phase space are calculated. Lastly, data points outside the projected ellipse are detected and replaced. The algorithm iterates until the replacement of outliers induces no more effects on the calculation of standard deviations. The left side of figure 4.1 shows the forming of data clusters in 3D phase space and the difference between the ellipse demarcation of the original method by Goring and Nikora, 2002 and the adaptation by Wahl, 2003. (Goring & Nikora, 2002)

Many studies on hydraulic problems that use ADV data apply the 3D phase space thresholding method and regularly compare the method to other despiking methods. It is cited often and pops up in virtually every publication concerning ADV data analysis, but still, authors run into problems. For example, Chanson et al., 2008, p. 4 find the method inadequate for velocimetry data in a natural estuarine system. However, they also state general difficulties in selecting appropriate techniques for lack of independent representative data sets for comparing methods. Islam and Zhu, 2013, p. 6 only consider the 3D phase space thresholding method effective if the number of spikes is significantly smaller than the number of non-spiky data points ($<5\,\%$).

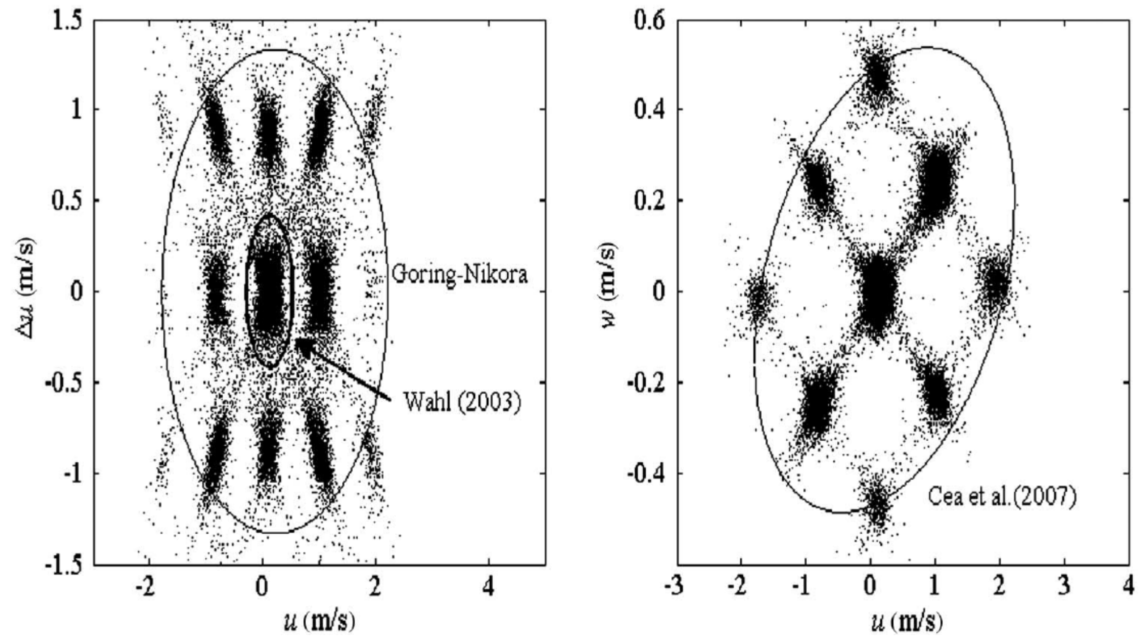


Figure 4.1.: Comparison of ellipse projection in 3D phase space between 3D phase space thresholding method and velocity correlation filter (Islam & Zhu, 2013, p. 3).

4.2.2. Velocity Correlation Filter

The velocity correlation filter developed by Cea et al., 2007 picks up several ideas of the 3D phase space thresholding method by Goring and Nikora, 2002. The major difference is that instead of cross-correlating the velocities and their second derivatives, it cross-correlates the velocity components against each other. This approach facilitates non-iterative filtering and replacement of erroneous data points as spike detection depends on the relation between the three-component measurements at one point rather than the relation between successive measurements (i.e. derivatives). However, this also means that applying the velocity correlation filter dismisses the advantages of idea one in list 4.2.1.

The authors test their method with ADV data recorded in a high turbulence fishway scale model containing many air bubbles and compare the despiking results to the 3D phase space thresholding method and its antecedent - the acceleration thresholding method by V. Nikora and Goring, 2000. They find that all methods work similarly well concerning the filtered signals and statistical parameters. Still, the amount of detected spikes varies with the choice of filter, which hints at differences in operation.

The right side of figure 4.1 shows how the velocity correlation filter pools ADV data into clusters and demarcates spiky from non-spiky measurement points. The method does not differ noticeably from the 3D phase space method displayed on the left side.

4.2.3. Wavelet Space Despiking

Razaz and Kawanisi, 2011 use wavelet theory (see subsection 2.3.4) for despiking ADV time-series. However, the authors do not apply discrete wavelet theory but take a slightly different approach with wavelet packet decomposition, where the composition includes both detailed and approximated coefficients. The method applies the universal threshold as the thresholding criterium and enhances the accuracy by introducing a robust scale estimator. This estimator is devised so that its influence function becomes unbound if the magnitude of a data point is relatively large, i.e., a spike. If the data point stays within the sample's morphology, the function is bound. The authors test several choices for this estimator and present a recommendation for future use; the new criterion is further called the shrinkage criterion.

The algorithm using the wavelet method requires the following steps:

1. Transform the time-series into a zero-mean sample.
2. Extract the wavelet packet basis using a spline filter with a single coefficient Fourier cosine series.
3. Set the shrinkage criterion.
4. Find spikes by detecting non-zero values in the inverse wavelet decomposition (Razaz & Kawanisi, 2011, p. 3).

The authors suggest autoregressive moving average (ARMA) modelling or Kalman filtering for data replacement after the despiking process.

4.2.4. Kernel Density Estimation Despiking

Islam and Zhu, 2013 devise a despiking algorithm based on a statistical method called kernel density estimation (KDE). The paper states the necessity for an improved despiking method as existing methods - essentially, the methods described in subsections 4.2.1 and 4.2.2 - prove inadequate for ADV measurements obtained in a wall jet in a flume. The authors discover that samples containing many spikes lead to the development of spike clusters that become part of the calculated ellipses and, therefore, are not detected as outliers. Indeed, figure 4.1 shows an example for which the demarcation technique of the 3D phase space thresholding method only works partially. Islam and Zhu, 2013, p. 6 argue, that for the showcased data set, all clusters within Goring and Nikora, 2002's ellipse, but the very central cluster are so-called spike clusters. According to the paper, Wahl, 2003's further development of the 3D phase space thresholding method successfully isolates said spike clusters. This can also be seen in figure 4.1. However, the statistics swayed by the high number of spikes still cannot produce satisfying filter results without manually manipulating the cutoff threshold. The proposed kernel density despiking method attempts to determine cutoff thresholds automatically by taking data and spike patterns into account.

KDE is a statistical technique relying on the fact that random variables are distributed in known probability distributions. Like other non-parametric techniques, KDE tries to identify a probability distribution from a sample. The procedure is comparable to constructing a histogram, but instead of forming a discrete distribution, it approximates a continuous distribution density function. This approximation is made by multiplying the bins of the histogram by a probability distribution function and summing up the resulting curves. For the KDE despiking, the method assumes the Gaussian probability distribution and uses the respective function (see equation 2.1) for multiplication; it uses a Gaussian kernel. Other kernels apply different probability distributions. For example, a Python package to perform kernel density estimations offers a tophat kernel, exponential kernel, linear kernel, cosine kernel, Epanechnikov kernel, and Gaussian kernel (Pedregosa et al., 2011). Another parameter to consider is the bandwidth: it determines the tradeoff between bias and variance of the continuous function. Therefore, mathematically, the kernel is a function dependent on the data set and the chosen bandwidth. (Haslwanter, 2021, p. 94)

The bivariate KDE equation used for KDE despiking goes as follows:

$$\hat{f}(x,y) = \frac{1}{2\pi N h_\mathrm{x} h_\mathrm{y}} \sum_{i=1}^{N} e^{-\frac{(x-x_i)^2}{2h_\mathrm{x}^2}-\frac{(y-y_i)^2}{2h_\mathrm{y}^2}} \tag{4.2}$$

$\hat{f}(x,y)$	– density estimation at location x and y,
x	– location of density estimation for x variable,
y	– location of density estimation for y variable,
N	– number of data points,
h_x	– bandwidth for x-axis,
h_y	– bandwidth for y-axis,
x_i	– realisation of x variable,
y_i	– realisation of y variable (Islam & Zhu, 2013, p. 3).

Using all this information, Islam and Zhu, 2013 devise a despiking algorithm with the following main points using a bivariate kernel designed by Botev et al., 2010:

1. Decide on one velocity component variable and devise its first derivative. These are the two variables used for the bivariate kernel density estimation.

2. Estimate the rotation angle of the principal axes of the grid with the classical least-squares approximation.

3. Transform and rescale variables to match the rotation angle and so that the values range from 0 to 1.
4. Use the density estimation equation 4.2. Locate the maxima of the density matrix and extract the corresponding density profiles for each variable.
5. Identify the size of the ellipse from the data and define the cutoff points using the slopes of normalised densities along the ellipse.
6. Calculate the ellipse according to the cutoff points and flag outliers.

Spike removal works per beam, so spikes detected in the v-velocity component are not eliminated from the u-velocity component and the w-velocity component and vice versa. Replacement of the removed data points remains elective. However, to calculate turbulence spectra, it is necessary to replace missing data points to maintain statistical propriety. (Islam & Zhu, 2013, p. 5)

Figure 4.2 visualises steps 4 and 5 of the KDE despiking algorithm (see list 4.2.4). The data in the figure is the same as in figure 4.1, which makes it easy to identify the difference in ellipse demarcation between the dispiking techniques.

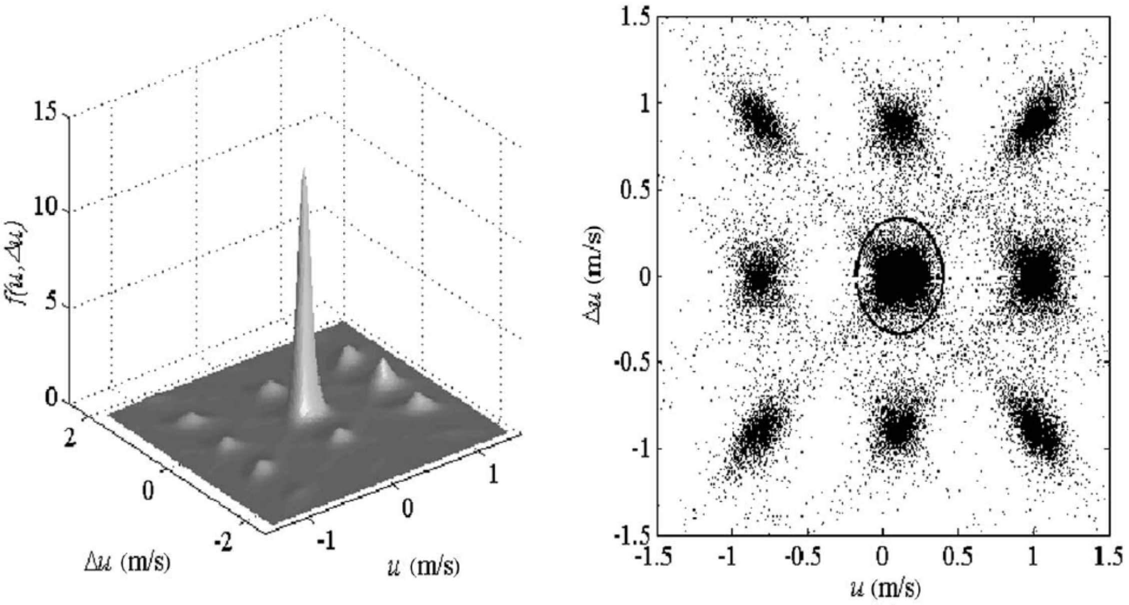


Figure 4.2.: Visualisation of the density map resulting from kernel density estmation (step 4) and demarcation of the ellipse to detect spikes (step 5) (Islam & Zhu, 2013, p. 4).

4.2.5. Autoregressive Moving Average Models

ARMA models are linear and discrete models that represent statistical processes. They use linear difference equations, thus represent linear statistic functions and can only approximate more complex systems. ARMA models operate using two separate models - an autoregressive (AR) model and a moving average (MA) model. Interestingly, the AR model corresponds to the design of digital IIR filters (see subsection 2.4.3), and the MA model corresponds to the design of digital FIR filters (see subsection 2.4.2). ARMA modelling also facilitates short-term forecasts, which is a common use of the method. (Alessio, 2016, pp. 471–484)

The motive of Dilling and MacVicar, 2017 is dissatisfaction with preceding ADV despiking methods. They criticise methods that remove non-spiky data points in time-series with few spikes, eliminate a high number of non-erroneous data points with high magnitude spikes, and replace spikes using simple replacement methods only. Therefore, the authors devise a use of the following ARMA equation and an application of the Bayesian information criterion (BIC) to determine model order:

$$u_t = \sum_{i=0}^{p} \phi_i u_{t-i} + \sum_{j=0}^{q} \theta_j \epsilon_{t-j} + \epsilon_t \tag{4.3}$$

$$BIC = (n-p-q) \cdot ln[\frac{n\sigma^2}{n-p-q}] + n(1 + ln\sqrt{2\pi}) + (p+q) \cdot ln[\frac{\sum_{t=1}^{n} u_t^2 - n\sigma^2}{p+q}] \tag{4.4}$$

u_t – velocity at time step,
t – time,
i – lag step for AR model,
p – total number of AR coefficients,
ϕ_i – AR coefficient at lag i,
j – lag step for MA model,
q – total number of MA coefficients,
θ_j – MA coefficient at lag j,
ϵ – residual (error) terms,
n – number of measurements,
σ – standard deviation (Dilling & MacVicar, 2017, pp. 2, 4).

The authors follow an algorithm as summarized:

1. Pre-screen data for extremely poor quality time-series.

2. Determine model order using the BIC.

3. Check the estimates of AR and MA coefficients, and update the model if they are off.
4. Assess whether the model residuals indicate a spike.
5. Calculate the difference between original and replacement data, normalise the values using the standard deviation (residual z-scores).
6. Determine actual spikes by evaluating the z-scores.
7. Replace spikes. (Dilling & MacVicar, 2017, pp. 3–5)

Dilling and MacVicar, 2017 find their method more precise than other prevalent methods, such as the 3D phase space thresholding method. An advantage is a progressive elimination of spikes by descending spike magnitude. This approach ensures detecting extreme values and less extreme spikes that might not represent fluctuations caused by turbulence but are indeed erroneous. (Dilling & MacVicar, 2017, p. 12)

The method is also applied to ADV data in a study by C. Huang et al., 2020. However, the authors use the method for denoising and compare it to a Kalman filtering denoising method (see subsection 4.3.7). They claim that an appropriate ARMA model delivers the white noise contaminating the signal in the form of its residuals and apply the ARMA model after despiking with the adapted 3D phase space thresholding approach (Mori et al., 2007).

When reading the publication, it becomes clear that this method requires a profound knowledge of the method and ADV data analysis as it contains many manual checks and adjustments. On the one hand, this may make the method more precise than others but, on the other hand, lead to complexity when processing large amounts of samples.

4.2.6. Singular Spectrum Analysis

Singular spectrum analysis (SSA) is a non-parametric spectral analysis method with the goal to constitute the original signal by linearly combining data-adaptive functions of time. Therefore, it does not use harmonic composition as classical spectral analysis methods do. Usually, the SSA method consists of four steps: embedding, decomposition, grouping and reconstruction.

Embedding means that a covariance matrix - containing the signal's desired and noisy part - is devised from the original signal. The decomposition step calculates the eigenvalues of the covariance matrix, giving out an eigenvalue spectrum. Every single eigenvalue represents a different variability in the original signal. Afterwards, the eigenvalues and eigenvectors are grouped by descending importance. Each group then forms a so-called trajectory matrix. A conversion of the trajectory matrices into joint univariate signals, which reconstructs the output signal, follows. (Alessio, 2016, p. 537)

Sharma et al., 2018 create a procedure to apply SSA to ADV recorded time-series. The authors follow the steps of conventional SSA closely. A significant difference is that after computing the eigenvectors of the covariance matrix, the method calculates the local mobility of each eigenvector. The local mobility further forms the basis for defining signal subspace. Then it removes eigenvectors that show higher mobility values than a threshold chosen from the morphology of the data. A sinusoidal signal of 1 Hz is chosen as the threshold in the publication because the authors identify this part of the spectrum as useful. The signal's reconstruction is then performed only with the eigenvectors according to the set threshold, and one gets the now despiked signal as the output.

The study finds that the SSA method produces cleaner signals than other methods with fewer remaining spikes while also being iteration-free. Interestingly, the slope in the inertial subrange matches Kolmogorov's $-\frac{5}{3}$ slope but at a significantly lower power level than the other reviewed methods. (Sharma et al., 2018, p. 5)

4.3. Denoising

Most of the denoising approaches for ADV devices aim to reduce Doppler noise, which is inherent to the measurement technique of acoustic Doppler velocimetry. While the approaches indeed employ the most different techniques and mathematical procedures, the nature of the present Doppler noise is almost universally defined as white noise. Whereas this thesis does not go into the specifics of white noise, its most important property for denoising strategies needs to be stated: white noise follows a normal distribution. The following subsections outline data analysis methods to reduce or eliminate noise from ADV measurements and give an overview of attempts to improve ADV measurements with denoising. This section does not provide an exhaustive list of all denoising methods but tries to pick up the most referenced ones regarding acoustic Doppler velocimetry.

4.3.1. Noise Floor Subtraction in Turbulent Spectra

An early consideration of noise in ADV measurements is by V. I. Nikora and Goring, 1998. The authors determine that Doppler noise is inherent to acoustic velocimetry. They name a finite residence time of scatterers in the sampling volume, small-scale turbulence and beam divergence as possible noise inducers and generally find that noise causes bias in statistical characteristics and turbulent spectra obtained from original ADV signals. The authors claim that noise in ADV measurements can be viewed as a constant fluctuation term that may be subtracted from the time-series. Therefore, they propose to measure a Doppler noise floor in still water to subsequently subtract the noise floor terms from the turbulence characteristics determined from the time-series measured in the experiment. In further discussion of the beforementioned method, Lemmin et al., 1999 highlight several flaws of the noise floor subtraction method in the time domain. The paper argues that Doppler noise is dependent on several circumstances, which sometimes will even be present in still waters. The discussion also picks up spectral analysis of ADV measurements and how spectral density often plateaus in high-frequency zones of spectrums.

As a continuation of the discussion, Dombroski and Crimaldi, 2007 try to utilize this plateau of density spectra combined with noise floor subtraction. For this, a high-frequency noise floor is subtracted from the turbulence spectra of the original signal. Although the filtered spectra show improvements compared to the original ADV signals, the authors generally dismiss noise filtering. They claim that noise filtering clouds errors, especially errors induced by measuring in proximity to walls, which leads to turbulence statistics appearing sound in cases they are indeed not. However, note that today's Nortek manual (Nortek, 2021) includes spacing from weak spots dependent on the set velocity range.

4.3.2. Digital Filtering

This subsection picks up the principal concepts of digital filters from section 2.4 and presents ADV analysis methods applying them to ADV data for denoising.

Gauss Lowpass Filter

Several authors report the application of Gauss lowpass filters to filter out Doppler noise. They all use a method devised by Lane et al., 1998, which uses the digital filter and the corresponding parameters designed by Biron et al., 1995. Note that Biron, 1997 correct a significant error in the Gauss function of the original publication.

The Gauss filter function used for this method is

$$w(t) = (2\pi\sigma^2)^{-\frac{1}{2}} \cdot e^{\left(\frac{-t^2}{2\sigma^2}\right)} \tag{4.5}$$

with sigma calculating as

$$\sigma = \left(\frac{ln(0.5)^{0.5}}{-2\pi^2 f_{50}^2}\right)^{\frac{1}{2}} \tag{4.6}$$

using the half-power frequency devised from the sampling frequency

$$f_{50} = \frac{f_s}{6} \tag{4.7}$$

$w(t)$	– Gauss filter function,
t	– time,
σ	– standard deviation of the filter,
f_{50}	– half-power frequency,
f_s	– sampling frequency. (Lane et al., 1998, p. 8)

The filter operates with a cutoff frequency at the Nyquist frequency, with the standard deviation of the filter as the only other interchangeable parameter. This parameter specifies the steepness of the transition band of the filter's frequency response. A higher standard deviation produces steeper frequency responses and smoother filtered signals; a smaller standard deviation produces wider transition bands and less attenuated filtered signals. However, the publications at hand specify a recommended standard deviation for applying the filter to ADV data that calculates as presented above.

The frequency response of the filter is written as

$$R(f) = e^{-2\pi^2\sigma^2 f^2} \tag{4.8}$$

$R(f)$	– Gauss filter frequency response,
f	– frequency,
σ	– standard deviation of the filter. (Biron et al., 1995, p. 6)

Lane et al., 1998 recognise the filter to help remove a high-frequency noise floor caused by aliasing (see subsection 2.3.2), which also finds support from other authors (Carbonneau and Bergeron, 2000, p. 4, Strom and Papanicolaou, 2007, p. 7) but does not lead to its implementation on a grand scale in future publications. All results are assessed with the approach to fit the power spectra of the velocity measurements to Kolmogorov's inertial subrange dissipation slope of $-\frac{5}{3}$.

Linear Minimum Mean Square Error

Hejazi et al., 2016 also assume that the noise in ADV velocity signals is random white noise. Using an FIR filter, the paper presents an algorithm applying linear minimum mean square error (LMMSE) estimation:

$$\sum_{k=-\infty}^{\infty} w(|n-k|)R_{\text{xx}}(k) = w_n * R_{\text{xx}}(n) = R_{\text{vv}}(n) \tag{4.9}$$

with a frequency response of

$$H(f) = \frac{P_{\text{vv}}(f)}{P_{\text{xx}}(f)} = \frac{P_{\text{vv}}(f)}{P_{\text{vv}}(f) + P_{\text{nn}}(f)} = \frac{\frac{P_{\text{vv}}(f)}{P_{\text{nn}}(f)}}{\frac{P_{\text{vv}}(f)}{P_{\text{nn}}(f)} + 1} \tag{4.10}$$

w_n	– filter coefficients,
n	– data points,
R_{vv}	– autocorrelation matrix of the true velocity,
R_{xx}	– autocorrelation matrix of the measured velocity,
R_{nn}	– autocorrelation matrix of the noise,
$H(f)$	– LMMSE filter frequency response,
f	– frequency,
P_{vv}	– Fourier transform of the autocorrelation matrix of the true velocity,
P_{xx}	– Fourier transform of the autocorrelation matrix of the measured velocity,
P_{nn}	– Fourier transform of the autocorrelation matrix of the noise (Hejazi et al., 2016, p. 5).

The autocorrelation matrices of the three components (true velocity, measured velocity and noise) are used in their Fourier transform for the filter equation as a result of using a Fourier transform method for approximating the IIR filter form with the above-defined FIR filter.

To assess the noise elimination method by LMMSE estimation, the authors use two different methods - an autoregressive model and a numerical 3D model. Alongside a smooth filtered signal and a significant improvement of SNR, both assessment methods show that the LMMSE is successful not only in denoising but also in despiking. Still, the paper only does a statistical assessment of the filtered signals and does not assess within the scopes of a turbulence model. (Hejazi et al., 2016, p. 8)

Lowpass Butterworth Filter

C. J. Huang et al., 2018 use a lowpass Butterworth filter to remove Doppler noise and high-frequency fluctuations from ADV measurements. The authors chose a first order Butterworth filter for denoising ADV data due to the filter's common noise-reducing efforts in turbulence spectra aside from ADV applications, such as in Roget et al., 2006. However, the publication does not explicitly specify filter characteristics but does discuss the identification of the Butterworth cutoff frequency by assessing wavenumber spectra. The authors expect that the spectra of the measured velocities deviate significantly from the theoretical spectra. Therefore, they approximate the spectra analytically and subsequently fit the measured ADV shear spectra to the theoretical spectra by calculating velocity shear with Taylor's frozen turbulence hypothesis and transforming it into wavenumber spectra. The wavenumber at which the beforementioned deviation occurs then dictates the cutoff frequency. (C. J. Huang et al., 2018, pp. 2–3)

Furthermore, the authors combine the Butterworth filter with another method called empirical mode decomposition (EMD), which this thesis does not pick up, as it is only described vaguely in the publication. C. J. Huang et al., 2018 find that combining the two methods is indeed constructive. The combination is indispensable for low turbulence conditions to produce good results, but only applying the Butterworth filter is also adequate for higher turbulence. The authors assess with the classical Kolmogorov approach in the inerial subrange and a newly developed assessment method. (C. J. Huang et al., 2018, p. 7)

4.3.3. Bifrequency Doppler Noise Repression

The bifrequency Doppler noise repression method by Hurther and Lemmin, 2008 applies cross-correlation of two independent velocity samples measured at different sampling frequencies in the same sampling volume. It requires double measurement at all measurement points. This practice seems inefficient, especially because many authors recommend sampling at the highest sampling frequency possible to reduce aliasing (see 2.3.2). Therefore, this method is not further discussed and is only mentioned for completeness.

4.3.4. Polynomial Least-Squares Regression

In Richard et al., 2013, the authors propose to estimate Doppler noise in ADV signals with polynomial least-squares regression. The basic assumption of their method is that the noise induces variance in the velocity signal's frequency domain and that the errors of adjacent measurements are not correlated.

Two curves are observed in the turbulent spectra: the measured spectra of the original signals and a curve fitted to the measured spectra by least-squares approximation. The differences between the curves are calculated and minimised as part of a regression function dependent on N (noise in the spectra) and K (constant in the spectra). This process gives a distinct linear system which can be solved numerically.

Although the paper finds good ways to determine noise contamination in ADV velocity signals, it does not propose appropriate methods to eliminate noise based on polynomial least-squares regression. (Richard et al., 2013)

4.3.5. Proper Orthogonal Decomposition

Proper orthogonal decomposition (POD) is a numerical data analysis method which simplifies complex contexts by reducing model order. The procedure realises the simplification by identifying the physical field's principal components from the data with its eigenvalues and eigenvectors. The resulting spatial functions are called modes. The modes and their combination can represent flow structures. Lastly, a reconstruction of the signal only recognises selected modes which alters the signal accordingly. (Weiss, 2019, pp. 1, 4)

Durgesh et al., 2014 apply POD to decompose ADV data. The procedure neglects high-order modes when recomposing the time-series, which is called a low-order reconstruction. The elimination of high order modes is possible because present noise is expected to be white noise and, therefore, connected to high-energy. Neglecting these modes in reconstruction gives a signal only containing desired components of the signal. The method is similar to the SSA method proposed by Sharma et al., 2018 for despiking (see subsection 4.2.6).

Although the method is straightforward, the authors face difficulties when choosing an appropriate number of modes for reconstruction. For this, they test two approaches. The first approach assumes that the spectra follow Kolmogorov's $-\frac{5}{3}$ slope in the intertial subrange and adds modes until the spectrum of the reconstructed signal fits said slope. The procedure accomplishes the slope with 359 modes. The second approach focuses on the energy levels of the signal. It also requires knowledge of the energy level of the noise contaminating the signal. The noise energy is subtracted from the total energy, and low-order modes are added until the sum of the mode energy levels accounts for the expected, clean signal energy level. Both approaches are effective, but many a priori assumptions are a general disadvantage of either approach. When comparing spectra from Gauss filtering with spectra from the POD method, the authors find that the POD method improves results, especially in high-frequency sections of the spectra. (Durgesh et al., 2014, pp. 8–10)

4.3.6. Noise Auto-correlation

Noise auto-correlation (NAC) is an approach also published in Durgesh et al., 2014. It exploits that instrument noise is identified as white noise and uses the superposition principle to separate the auto-correlations and cross-correlations of noise and true signal in every velocity component. Further, the authors assume the energy contribution of the white noise in the spectra, and by Fourier transforming the auto-correlation of the true signal, they determine the respective spectra. Therefore, the method highly relies on estimating the energy level of the noise, which can be a disadvantage for inexperienced researchers. Also, the method only gives noise-corrected frequency spectra and does not supply noise-corrected time-series. Still, it gives adequate spectra when comparing the spectra to spectra from other methods, like the POD or the Gauss filter. (Durgesh et al., 2014)

4.3.7. Kalman Filter

The Kalman filter is not a classical digital filter. Although it resembles an IIR filter in the time domain, it is a recursive filter for time-varying linear systems with forecasting potentials. Therefore it does not comply with the conditional properties of the classical digital filters presented in subsection 2.4.

C. Huang et al., 2020 use the filter for denoising purposes for laboratory and oceanic field data and compare the results from the Kalman filter to an ARMA model approach. The authors only supply basic information on their procedure and refer to a used Matlab package. They do not specify the chosen parameters to operate the package and the method. They report that both the ARMA model approach and the Kalman filter approach are useful to eliminate noise from already despiked signals and prove the output signals' quality improvements with the Kolmogorov slope approach in the inertial subrange. (C. Huang et al., 2020)

4.4. Data Replacement

While denoising efforts usually work with attenuation or amplification, operations applied to all data points, despiking methods identify single erroneous data points. For the time being, this identification of spikes does not alter the other data points of the set, and one must decide how to proceed further. There are two possible options available: eliminating spikes or replacing the spikes by determining an appropriate value for the concerned data point.

Elimination of spikes is the easiest solution to deal with identified spikes. Provided that data sets are large, elimination does not alter turbulence statistics compared to an appropriate replacement method. This course of action is valid for only obtaining a mean velocity or turbulent kinetic energy value. (Cea et al., 2007, p. 2)

Though, when working with methods that analyse in the frequency domain, like the Fourier analysis, or that rely on order and succession of values, such as forming derivatives, solely deleting data points causes bias by suddenly allocing the values of the signal to different time steps. This misalignment of data significantly impacts turbulence spectra, insinuating shifts in periodic phenomena. Therefore, proper spike replacement is indispensable for spectral analysis. (Islam and Zhu, 2013, p. 5, Razaz and Kawanisi, 2011, p. 2) Still, spike replacement can also introduce additional uncertainty to samples by adding artificial frequency components to spectra. Authors surmise the highest impact of replacement efforts on uncertainty in the spectra's high-frequency parts. (Doroudian et al., 2010, p. 9)

Several rather basic data replacement methods exist. They can be divided into three categories:

– Extrapolation: last valid data point, last two valid data points.

– Interpolation: linear between endpoints, polynomial, cubic spine.

– Statistical time-series parameters: mean, median.

Although authors attempt to evaluate the effects of different replacement methods on their data, there exists no consensus for a genuine recommendation. As a result, the replacement process is perceived as arbitrary but not satisfactory at the same time. (Wahl, 2003, p. 1, Hejazi et al., 2016, p. 2, Goring and Nikora, 2002, pp. 4, 9, Doroudian et al., 2010, p. 2, Cea et al., 2007, pp. 2, 5, Chanson et al., 2008, p. 5, Jesson et al., 2013, p. 3)

This discontentment with replacement methods lead to the publication by Razaz and Kawanisi, 2011 which uses time-series modelling and Kalman filtering for the prediction of values and the paper by Dilling and MacVicar, 2017, which applies autoregressive moving average (ARMA) models for replacing spike data. Both papers evaluate that their replacement strategies work reasonably well and are superior to the basic replacement methods.

4.5. Available Software

The most common free data analysis tool for ADV data is a software called WinADV. It was published in 1996 by the U.S. Bureau of Reclamation, Water Resources Research Laboratory. It offers post-processing methods for ADV data recorded with SinTek and Nortek ADV devices, such as visualisation, filtering, data flagging and statistical analysis. WinADV also offers easy visualisation of data and facilitates manual screening. The software comes with a user interface which makes it accessible to users without programming proficiency. (Wahl, 2000)

Although the software WinADV is quite old, it still provides the most needed tools when working with ADV data. It has considerable advantages over more flexible solutions like e.g. Microsoft Excel, which fails to process large data sets reasonably. Scripts in modern programming languages, such as Python or Matlab, require previous knowledge and, as in the case of Matlab, sometimes are costly. WinADV is also still recommended by expert panels, e.g. the German Federal Waterways Engineering and Research Institute (Sokoray-Varga B. & Höger V., 2014) and the realised methods are devised from recommendations by manufacturers (Nortek, 2021). Consequently, many researchers rely on the software and continuously use it.

In 2021 a research group published a Python framework for statistical analysis of turbulence in geophysical flows. The framework is called Python Statistical Analysis of Turbulence (P-SAT), written in Python 3+ and is provided on the platform Github under an open-source licence (Agarwal et al., 2021). To test and explain the framework's functionality, the researchers provide an accompanying publication, which includes the analysis of ADV data. The authors transfer several methods included in WinADV to an up-to-date language granting users more flexibility and adaptation possibilities. Beyond basic cutoff filters, they provide a Python implementation for the acceleration thresholding method (the 3D phase space thresholding method's antecedent). A disadvantage is that the framework does not include a graphical user interface (GUI) at present. (Agarwal et al., 2021)

Furthermore, several software solutions are published as Matlab code on Matlab's file exchange platform. This database includes the despiking methods by Mori et al., 2007 and Islam and Zhu, 2013. B. MacVicar et al., 2014 provide a Matlab toolbox which contains multiple algorithms to process and analyse velocity measurements.

5. Implementation of Data Analysis Methods in Python

The previous chapter gives an overview of available approaches to analysing and manipulating ADV data to improve data quality and decrease the uncertainty in measurement. Although respective publications suggest that the authors use computational support to operate their considerations, the technicalities of their implementations often remain unclear. Transparent and practical solutions for efficient ADV data handling are rare.

Ideally, a routine for laboratory work with the ADV needs to include timesaving ways to process the measured samples. This solution needs to be open-source and free, adaptable to different types of experiments and purposes. It must allow for processing multiple samples simultaneously while being computationally frugal and easy to handle even by programming beginners. Apart from practical requirements, it needs to reliably produce sound results in terms of the uncertainty in measurement.

Currently, only Agarwal et al., 2021's package offers an ADV data processing solution adequate by these standards. However, when preliminarily testing the package, the solution often fails to detect all spikes in a sample. As a result of this lack of practical processing solutions, this thesis attempts to implement other data analysis methods in a modern, open-source programming language: the kernel density estimation despiking method by Islam and Zhu, 2013 for despiking and a Gauss filter as described in Lane et al., 1998, and Biron et al., 1995 and a Butterworth filter similar to the one used in C. J. Huang et al., 2018 for denoising. These methods are chosen from all available methods in chapter 4 because the respective publications show successful application to turbulent ADV data. Another reason to select KDE despiking and Butterworth lowpass filtering is that they appear in recent publications. As an alternative to the Butterworth filter, the Gauss filtering approach is selected to represent a rather conventional approach to denoising that is not picked up often in newer literature.

While the first section 5.1 of this chapter gives insight into the used system, programming language, environment and libraries, section 5.2 describes little tools used for file management. Section 5.5 describes the utilisation of the P-SAT framework in this thesis. Section 5.3, section 5.4 and section 5.6 describe the implementation of the beforementioned data analysis methods in Python. As a data replacement solution, linear interpolation is implemented and outlined in section 5.7.

5.1. Hardware and Libraries

All programming and processing are done on a system with an Intel(R) Core(TM) i5-4590 CPU @ 3.30 GHz, 8 GB RAM, and Windows 10 Pro N on 64-Bit. Python (version 3.8.5) (Python Software Foundation, 2020) is utilised within the Spyder IDE (version 4.1.5) as a part of the Anaconda Distribution in (version 4.10.3) (Anaconda, 2021). The implemented Python scripts require the following external libraries for successful execution:

- NumPy: Support library for array and matrice operations and high-level mathematical functions (Harris et al., 2020).
- SciPy: Scientific computing library that, among other things, includes signal processing and transformation algorithms (Virtanen et al., 2020).
- pandas: Support library for multi-dimensional data structures and data operators (The pandas development team, 2022).
- matplotlib: Plotting library for 2D data visualisation (Hunter, 2007).
- seaborn: Library based on Matplotlib that offers even more visualisation options (Waskom, 2021).
- statsmodels: Package derived from NumPy, SciPy and Matplotlib, which offers advanced functions for statistical testing (Seabold & Perktold, 2010).

These specifications also apply to all Python implementations in chapters 6 and 7.

5.2. File Management Tools

To comfortably work with the data recorded by ADVs, the original format of the files needs to be converted from .dat to .csv. The script ”Make_csv_from_dat” (see source code A.1) performs this conversion. Further, defining respective methods to reliably and consistently import data from external files into Python scripts is necessary. For the scripts used in this thesis, there are three possible input formats:

- Original data .csv files denoted as input_file_state = 1.
- Files output by the P-SAT framework denoted as input_file_state = 2.

– Files output by KDE despiking, Butterworth filtering or Gauss filtering, denoted as input_file_state = 3.

The input file state is specified by the user. By executing the script, the method ”get_data” (see source code A.5) extracts the data according to the user-specified input file state. An if-statement ensures that the data is read with the according methods ”get_raw_data” (see source code A.2), ”get_PSAT_data” (see source code A.4) and ”get_KDE_G_B_data” (see source code A.3). The method ”get_data” gives out all velocity components, calculates the velocity magnitude and provides the time-steps for the time-series.

For using the P-SAT framework, it is necessary to convert the files so that the framework can reliably read them. The files need to have the format .dat and have the data sorted in a specific way. The conversion script ”Make_dat_for_P-SAT_from_dat” (see source code A.7) converts the recorded .dat files to .dat files that can be input to the P-SAT framework. Furthermore, the P-SAT framework requires itemising file names in an additional .txt file, which is quite time-consuming if the filenames are manually entered. This is why the framework is not very practical for batch processing a large number of files. This problem is solved by employing a script (see source code A.8) that extracts the filenames for the .txt files in the necessary format using the method ”get_filenames” (see source code A.9). The list of filenames can be copied directly from the console to the .txt file.

Sometimes, it is practical to sort files by their original filing system. In this thesis, this filing system corresponds to the turbulence conditions, i.e. cylinder diameter constructed in the flume. The initial sorting of files is required when analysing according to turbulence conditions (see chapter 6 for more information on the data at hand) - a ”File_mover” script (see source code A.6) is utilised. The script reads the filenames from the original filing system by employing the beforementioned method ”get_filenames” (see source code A.9) and then sorts the mixed files accordingly to the user-specified directory.

5.3. Gauss Filter

Original ADV data is loaded to the script using the methods explained in the previous section 5.2. Only one other parameter needs to be specified by the user: the sampling frequency used for measuring the data.

To apply the Gauss filter by Lane et al., 1998 and Biron et al., 1995 to one or several ADV measurements with Python, further specifications, namely sigma (see equation 4.6) and the impulse response function (see equation 4.8), need to be calculated. The method ”calculate_-sigma” (see source code A.10) is implemented. An x-axis according to the sampling frequency is devised using the implemented method ”get_x_f” (see source code A.11) to get the frequency response function. Sigma and the x-axis in the frequency domain are then input to a method called ”calculate_R_f” (see source code A.12), which calculates the filter’s needed response for the set sampling frequency. Figure 5.1 shows the frequency response for a sampling frequency of 25 Hz.

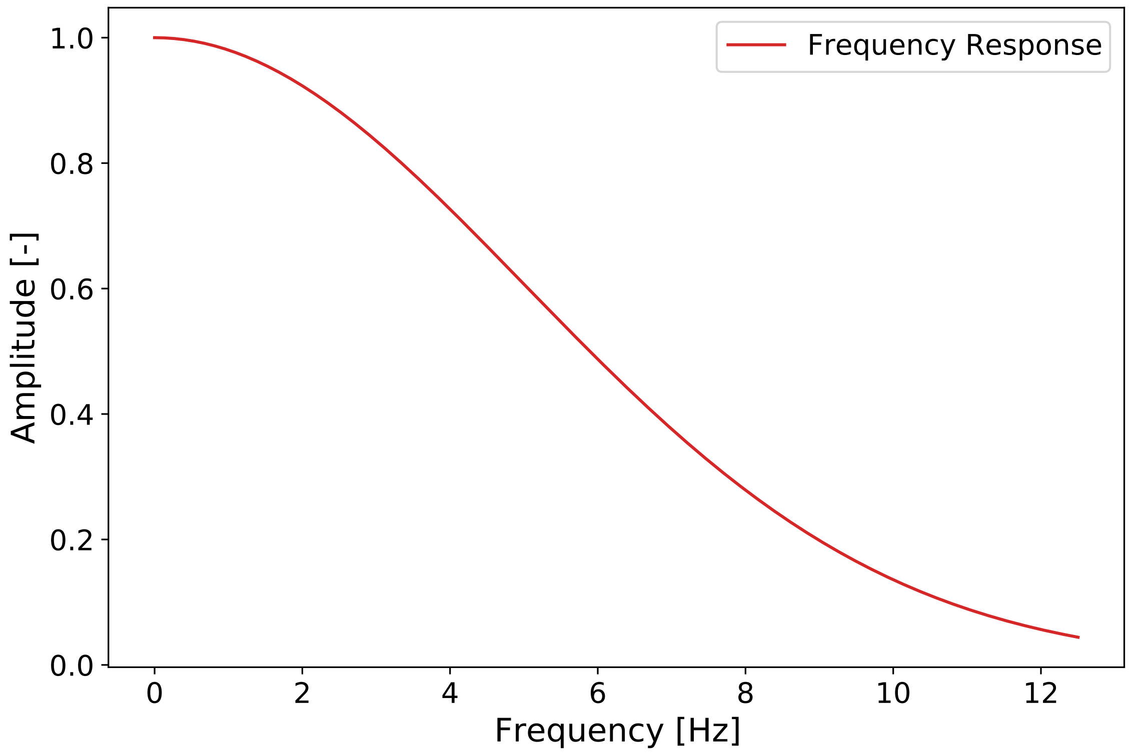


Figure 5.1.: Gauss filter frequency response with a sampling frequency of 25 Hz.

Then, the method ”gauss_filter” (see source code A.13) fast Fourier transforms (see subsection 2.3.4) every velocity component and multiplies the resulting frequency components by the filter frequency response. This operation is possible as a convolution in the time domain corresponds to multiplication in the frequency domain, respectively (see section 2.4). Then, the inverse fast Fourier transform transforms the results to the time domain. This process disregards complex values by only considering absolute values. The method gives a list which contains all filtered velocity components and is output as a .csv file to the same directory.

The script is showcased in the annex (see source code A.14). The whole process iterates all filenames in the specified directory to process several files with only one script execution. Applying a try-except statement within the iteration loop ensures that iteration continues even if an error occurs. The console gives out an indication of successful and unsuccessful filter applications. Wrong formatting of input files is the most likely exception to occur. The average runtime to filter all 160 samples amounts to ($7.17\,\mathrm{s}$).

The filter application can also be implemented using the Gauss filter function (see equation 4.5) and convoluting the signal. However, applying the frequency response function in the frequency domain seemed more straightforward.

5.4. Butterworth Filter

Again, the data input methods outlined in section 5.2 are used to process the measured ADV files with a Butterworth filter. Further, the user must specify the filter order and a cutoff frequency. For a first order filter, four different settings for the cutoff frequency are tested ($1\,\mathrm{Hz}$, $1.5\,\mathrm{Hz}$, $2\,\mathrm{Hz}$ and $2.5\,\mathrm{Hz}$), for a third order filter, only one setting ($\frac{fs}{2.93}$) is tested. The settings are determined from the review of technical literature. (C. J. Huang et al., 2018, B. J. MacVicar et al., 2007). The procedure to determine the optimal cutoff frequency described in subsection 4.3.2 is not applied as it is deemed detrimental to the comparability of results to filter each original data sample using different filter settings. Still, when only processing one file, the method might generate more precise results.

As the Butterworth filter is a popular filter for lowpass filtering, there are several options for its implementation in Python. This thesis uses the functions ”butter” and ”lfilter” from the SciPy signal package. Furthermore, two new methods are defined for this script. The method ”normalize_cutoff” (see source code A.15) assigns a value between 0 and 1 for the cutoff frequency set by the user. Scaling the values is necessary as the ”butter” function inquires for this range of values, with 1 signifying a cutoff frequency at the Nyquist frequency and 0 signifying a cutoff frequency at $0\,\mathrm{Hz}$. The method ”butterworth_filter” (see source code A.16) applies the actual filter operation: The IIR filter coefficients (see subsection 2.4.3) are identified with the ”butter” function by specifying the normalised cutoff frequency and the filter characteristics ”btype=’low’” for the lowpass filter and ”analog=False” for the digital filter.

The result is the filter frequency response, exemplarily showcased in figure 5.2 with an first order Butterworth filter and a cutoff frequency of $1\,\mathrm{Hz}$. Then each velocity component array is filtered using the function ”lfilter” with the beforehand calculated coefficients using the convolution operation (see subsection 2.3.3). The script ”Butterworth” (see source code A.17) iterates all files in the specified directory, applies the filter accordingly and gives out a file containing the filtered time-series in the directory of the original files. The console alerts the user to arising exceptions.

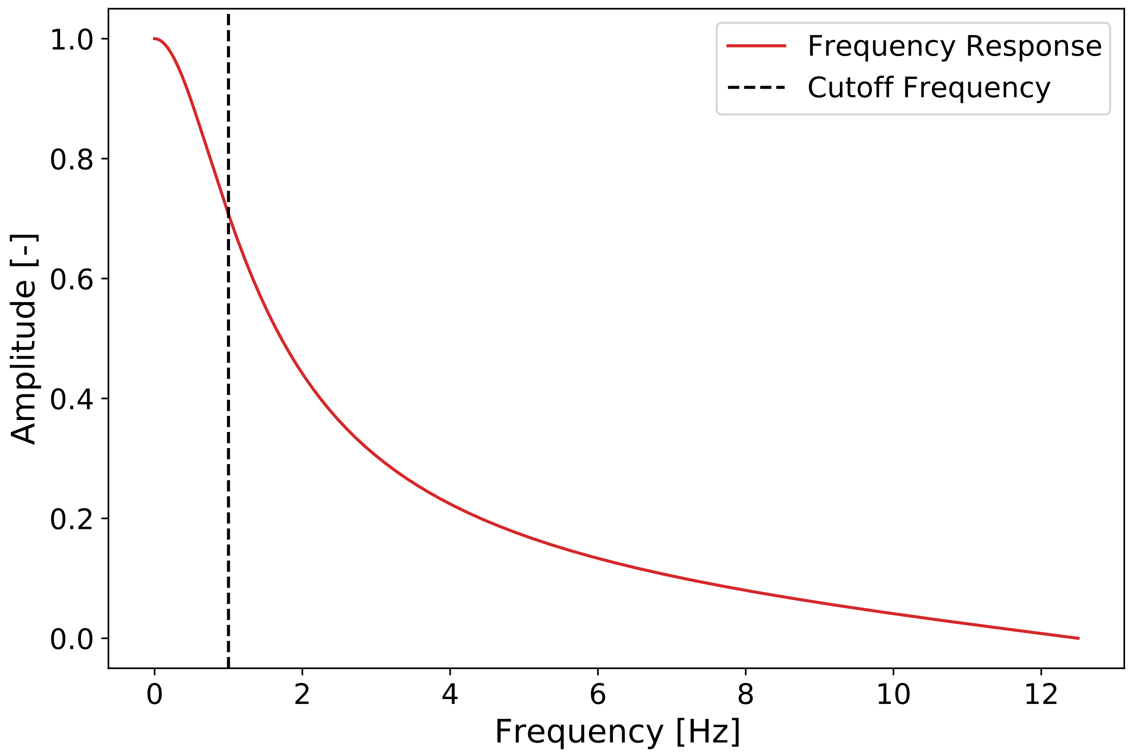


Figure 5.2.: First order Butterworth filter frequency response with a sampling frequency of $25\,\mathrm{Hz}$ and a cutoff frequency of $1\,\mathrm{Hz}$.

The average run time of the script to process 160 samples with the Butterworth filter is $6.90\,\mathrm{s}$. The course of the obtained spectra after Butterworth filtering corresponds to the spectra presented in C. J. Huang et al., 2018, p. 7, which gives confidence that the Butterworth filter works as intended.

5.5. Python Statistical Analysis of Turbulence (P-SAT) Framework

The PSAT package by Agarwal et al., 2021 is used to filter according to the acceleration thresholding method by V. I. Nikora and Goring, 1998. The mechanisms of this framework are not further explained in this thesis because the method is mainly applied here to test it and to compare the results from the KDE despiking method. The implementation authors give an example of use in their publication to learn how to use the framework (Agarwal et al., 2021). The PSAT package includes an implementation of linear interpolation between endpoints of a detected spike as the data replacement method. The iterative nature of the method (see subsection 4.2.1) leads to a high average run time of the script to process 160 samples that amounts to 260.52 s.

5.6. Kernel Density Estimation Despiking

For this section and the source codes in A.19, A.20, A.21, A.22, A.24, please note that adapting KDE despiking in Python, although at times undergoing vigorous modification, is based on the source codes by Islam and Zhu, 2013 and Botev et al., 2010 and the ParaMonte package by Shahmoradi et al., 2020. Before describing the efforts to implement KDE despiking in Python, relevant copyright notices are acknowledged in A.18.

Just as for all analysis methods, ADV data is read using the tools presented in section 5.2. To prepare the data further and facilitate the input into the kernel density function (see equation 4.2), the method ”get_kde_input” (see source code A.19) is introduced to apply steps 1-3 from the algorithm by Islam and Zhu, 2013. It includes forming the backward and forward derivatives and finally the central derivative, estimating the axis rotation and transforming the data according to the axis rotation.

The method ”get_kde_output” (see source code A.20) applies the kernel density function using Botev et al., 2010’s algorithm. First, it determines the minima and maxima of the velocity and the first derivative. By taking these values into account, the data is scaled from 0 (minima) to 1 (maxima). Then, the data is binned into 256 bins with the NumPy ”histogram2d” function. The number of bins is adopted from Islam and Zhu, 2013, in which the authors tested several settings and found 256 bins to work best. Afterwards, the method smooths the bins using the Gaussian kernel with a standard deviation corresponding to the user-specified bandwidth (in this case, hy=hx=0.01, adopted from suggestions in Islam and Zhu, 2013). The density profiles for the velocity and its derivative and the grid axes are output. Both are input to the following method, ”get_spike_Id” (see source code A.21), where they are used to identify the ellipses to demarcate the data clusters using the user-specified cutoff points.

The method ”KDE_despike” (see source code A.22) applies this procedure to all velocity components of the input ADV data and gives out a spike identifier list for each velocity component. This list can either be used to remove the data points or replace the data points with an appropriate data replacement strategy. The approach to implementing a data replacement method for this thesis is picked up in section 5.7.

The script to despike ADV data with the user input parameters can be found in the annex (see script A.24). The average run time of the script to process 160 samples with KDE despiking is 32.94 s. Exceptions which occurred while testing the code came up for selecting the wrong

”input_file_state” for samples with not enough measurement points (under 180) and samples without spikes. To use KDE despiking, the user needs to decide on an appropriate bandwidth, grid size and cutoff threshold.

5.7. Data Replacement

As discussed in section 4.4, spike replacement is essential if wanting to analyse velocity spectra after despiking. The P-SAT framework in Python uses a linear interpolation algorithm for replacing identified spikes and so does the Matlab code by Islam and Zhu, 2013. For implementing Islam and Zhu, 2013’s method in Python, neither implementation seems fitting. While the P-SAT interpolation method does not consider removing spikes per ADV beam, adapting the Python code just so that it can apply the Matlab implementation of the replacement strategy is unreasonably complicated.

Linear interpolation is still chosen as the replacement strategy to preserve comparability in this aspect with the P-SAT results. However, a new method is devised in Python, which caters to the previously explained implementation of KDE despiking in Python (see section 5.6). The method ”replace_spike_lin_interpolation” is showcased in the annex (see source code A.23). It uses the spike identifier list created by the method ”KDE_despike” (see source code A.22) as input. The values identified as spikes are replaced by linearly interpolating between the last and the next good values, i.e. values that are not spikes. Thus, neighbouring spikes are replaced with the same values. One fatal exception for replacing spikes using interpolation is if the signal’s first or last data points are identified as spikes. In these cases, the mean of the signal is used as it is simply not possible to perform the interpolation.

6. Evaluation of Data Analysis Methods in Python

In this chapter, the implementation of data analysis methods in Python (see chapter 5) is used to process ADV data recorded with a Nortek Vectrino device by Daniel Weidler in the hydraulic laboratory at TU Darmstadt in January 2021. Daniel Weidler recorded the ADV data while first and foremost measuring pressures with three pressure sensors mounted on a model fish and did not process or further utilise the ADV data. The sampling rate of the ADV was set to $25\,\mathrm{Hz}$, the nominal velocity range was set to $4\,\mathrm{m\,s}$, the transmit length was $1.8\,\mathrm{mm}$, and the sampling volume was set to $7.0\,\mathrm{mm}$. As Daniel Weidler's thesis investigates the pressure patterns resulting from vortex shedding in a flume, the velocity measurements correspond to four types of present turbulence: type 1 represents velocities without a cylinder in the flume, type 2 represents velocities in the wake of a DN50 cylinder, type 3 represents velocities in the wake of a DN90 cylinder, and type 4 represents velocities in the wake of a DN160 cylinder. (Weidler, 2021, pp. 43–44) In total, 160 samples for four different turbulence conditions (type 1 = 20 samples, type 2 = 40 samples, type 3 = 48 samples, type 4 = 52 samples) are used in this thesis. Details on the measuring grid used by Daniel Weidler are not given, as the samples are only evaluated independently of their relative position in the grid.

Evaluation methods for the uncertainty in measurement per the GUM (see subsection 3.3.1) are implemented to evaluate the effects of the processing methods. First, visual inspection of time-series in the time domain and the analysis of turbulence spectra realise Type B evaluation by comparing original samples to filtered samples (see section 6.1). Then, section 6.2 highlights Type A evaluation parameters for original and filtered samples. Criteria are defined and categories are devised to quantify effects on evaluation parameters caused by data analysis efforts. Lastly, section 6.3 draws conclusions from the evaluations.

6.1. Visual Inspection

Although visual inspection of time-series and spectra are not as compelling as quantitative statements, they can still serve to give general ideas on the functioning of data analysis efforts.

For example, by examining time-series, it becomes apparent whether despiking efforts were effective with a single glance: if spikes remain in the time-series, despiking was unsuccessful. For spectra, one can compare the course of the energy dissipation function to Kolmogorov's spectrum of turbulent flow (see figure 2.3). By plotting the $-\frac{5}{3}$ slope of energy dissipation in the inertial subrange, the log-log plot can indicate whether the examined sample follows the basic structure of the theoretical spectrum. While a similar course signifies good data in terms of Kolmogorov's law, a strongly deviating course indicates high uncertainty in the measurement (see subsection 2.2.3).

A Python script (see source code A.25) visualises samples in the time domain. The script gives out a visualisation of all velocity components (green, red, yellow) and the velocity magnitude (blue) of the processed sample in comparison to each time-series of the original sample (brown). The figures are then sorted according to the turbulence conditions of the samples (see source code A.6) and screened manually.

For constructing the energy dissipation spectra, Welch's method is employed in a Python script (see source code A.26). The method is a popular approach to spectral density estimation and delivers an adequate approximation for power spectra in an efficient way. It is chosen in this thesis as well-established implementations in Python already exist, which only need to be adapted to the problem at hand. Here, the script gives out the spectra of the original velocity components (dotted) and the filtered velocity components (line) spectra in one figure. The grid in the background shows the $-\frac{5}{3}$ slope that the spectra are compared against. The figures are then sorted according to the turbulence conditions of the samples (see source code A.6) and screened manually. To improve this evaluation approach and quantify data quality in terms of fit to Kolmogorov's spectrum, one could further implement automatical classification of spectra.

The two following subsections discuss findings and conspicuous features in the time-series and spectra produced by these scripts for the samples at hand. However, as discussing the visual evaluation of all 160 samples in detail goes beyond the scope of this thesis, one type 4 measurement (3_2_V20210111151049) is chosen, and general statements that can be made for all samples are shown for this measurement. All figures for time-series and spectra for this measurement can be found in the annex (see figures A.1 to A.24). Figures for this measurement relevant to explanations are displayed in the text, but only the time-series of the velocity magnitudes are shown. For the time-series of all velocity components, see figures A.9 to A.20 in the annex. Figures concerning this sample do not further reference the sample identifier. Sporadically, visualisations for other samples are picked up to highlight specific findings, mostly regarding data analysis effects on other turbulence conditions. These figures specify the respective sample identifiers. To see the figures and data for all 160 samples, view the supplementary material to this thesis.

6.1.1. Python Statistical Analysis of Turbulence (P-SAT) Framework

The P-SAT framework is mostly successful in despiking and somewhat smoothing the velocity signal in the time domain. Still, the method struggles to eliminate all spikes reliably, as shown in the despiked time-series in figure 6.1 at around $t = 12\,\mathrm{s}$. At least one noticable spike remains in 121 of the 160 samples. Many references report that this problem occurs with a high number of spikes or at high turbulence levels. The evaluation at hand does not verify this statement: neither samples with a low number of spikes nor samples at comparatively low turbulence levels (type 1 samples) are filtered perfectly. This issue leads to the assumption that the P-SAT framework (and, therefore, the acceleration thresholding method) cannot detect particular spikes within the projected ellipse's boundaries. The problem can be connected to thresholding or an error type that produces spikes and opposes the applied data clustering. Apart from that, it can be an issue with the code, or it occurs due to incorrectly using the framework.

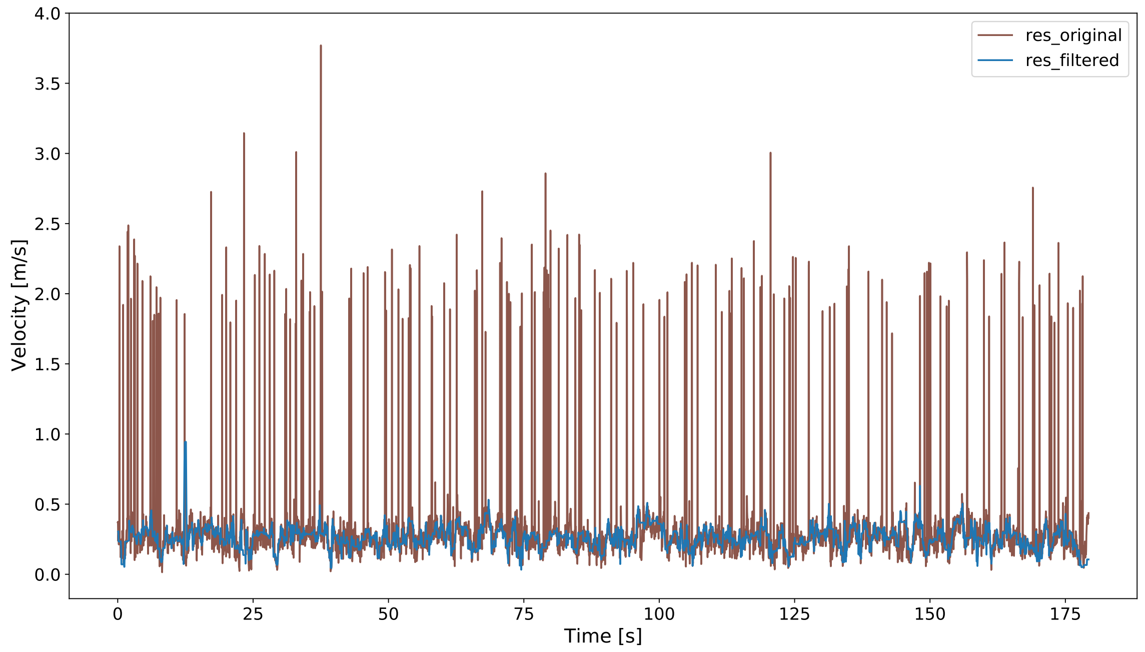


Figure 6.1.: Exemplary visualisation of P-SAT filtered time-series against original time-series.

Overall, the spectra of the velocity components (see 6.3a) approach Kolmogorov's $-\frac{5}{3}$ slope in the inertial subrange, and the effect of the data analysis method is undoubtedly apparent when comparing filtered spectra to the spectra of the original data. The filtered spectra follow a relatively horizontal course in the higher frequency part of the spectra ($>7\,\mathrm{Hz}$) that resembles the course of the original spectra but at a lower energy level. Comparing the spectra to the theoretical energy dissipation spectrum for turbulent flow, one finds a lack of the sudden plummeting of the spectra at high frequencies. Overall, all velocity components follow their courses at the same energy levels.

While the spectra for type 3 and type 4 turbulence conditions accord well to the theoretical turbulence spectrum, the spectra of type 1 and type 2 turbulence conditions show prominent upward arching in the central part of the spectra, where the course should be rather straight-lined. Figure A.25 in the annex shows this effect displaying exemplary spectra for a type 2 sample.

6.1.2. Kernel Density Estimation Despiking

The implemented KDE despiking Python script works reliably for despiking the samples. Regardless of turbulence conditions and the present numbers of spikes, not a single spike stays undetected. The method does not have a smoothing effect on the signal; instead, only from screening the time-series (see figure 6.2), it seems like the overall variance is maintained.

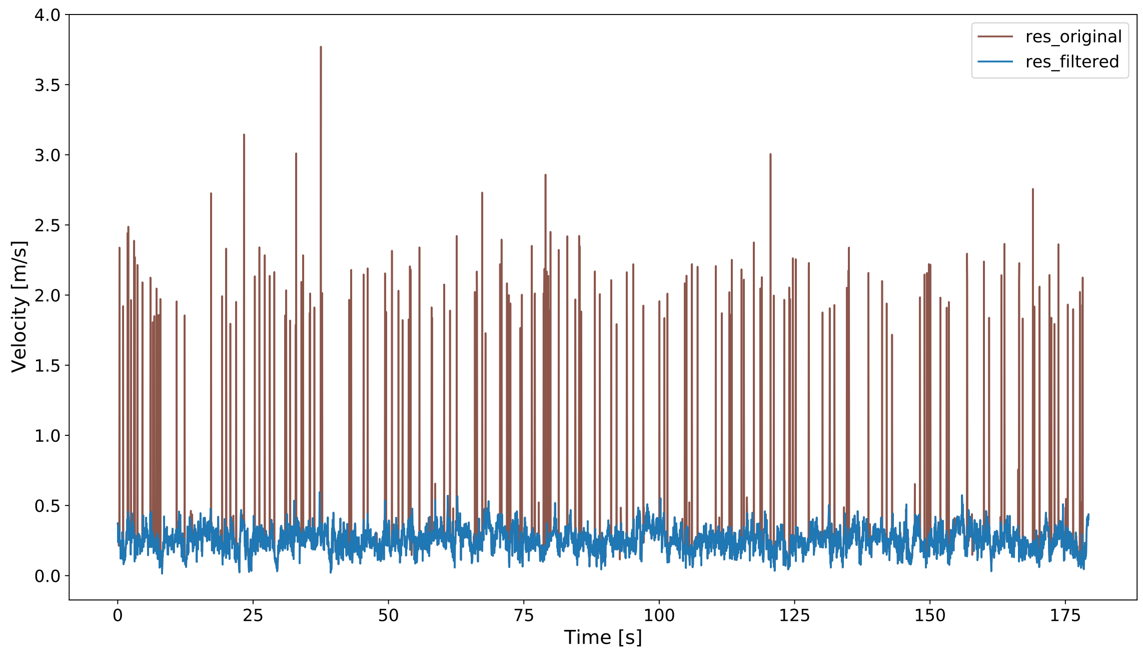
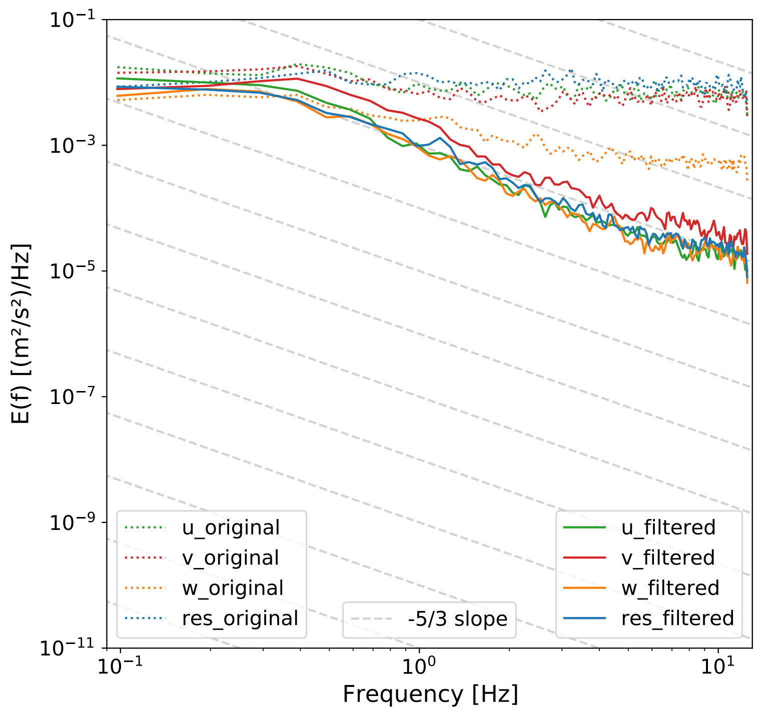
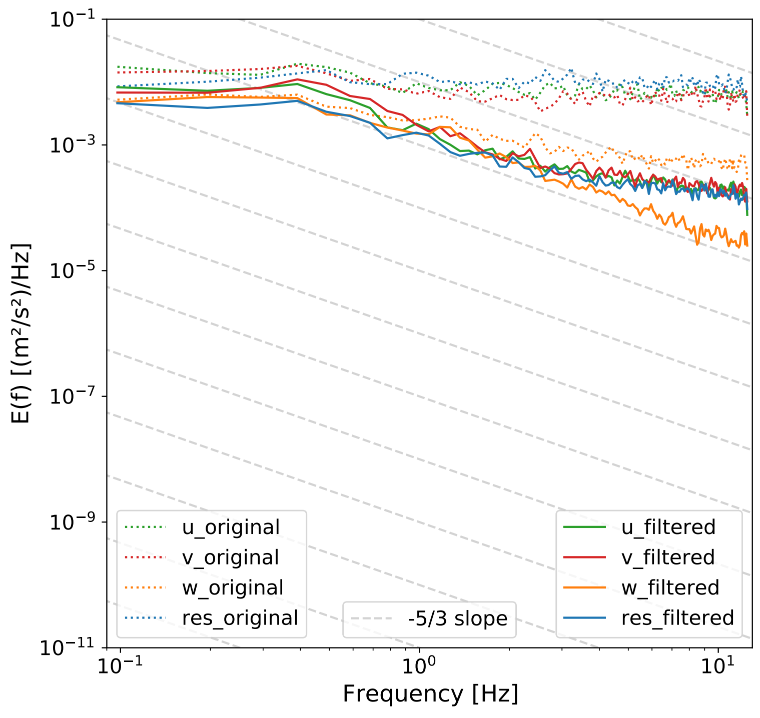


Figure 6.2.: Exemplary visualisation of KDE despiked time-series against original time-series.

The spectra for the KDE despiking method generally show an approach to Kolmogorov's $-\frac{5}{3}$ slope compared to the original data spectra. For the type 1 and type 2 samples, the gradient of the central part of the spectra is too flat (see figure A.26 in the annex). However, the gradient



(a) Spectra from P-SAT filtered time-series.



(b) Spectra from KDE despiked time-series.

Figure 6.3.: Exemplary visualisation of spectra from despiked time-series.

approximates Kolmogorov's $-\frac{5}{3}$ slope for type 3 and 4 samples. Just as for the P-SAT spectra, the KDE despiked spectra follow a somewhat horizontal course in the higher frequency part of the spectra ($>7\,\mathrm{Hz}$) that resembles the course of the original spectra at a lower energy level. It must be noted that the velocity components appear at the same energy levels in the lower frequency parts of the spectra, but the w-velocity spectra separate from the other component spectra between $2\,\mathrm{Hz}$ and $4\,\mathrm{Hz}$ to follow a course at a lower energy level. Figure 6.3b shows the spectra for the exemplary sample.

6.1.3. Butterworth Filtering

The applied Butterworth lowpass filtering script attenuates all present frequencies with a significantly increasing attenuation for frequencies beyond the set cutoff frequency, making the signal appear smoother. The attenuation is the strongest for the lowest chosen cutoff frequency. Some spikes are only slightly attenuated, and others are strongly attenuated (and do no longer appear as spikes). This outcome indicates that spikes are present in all frequencies, which does not advocate using lowpass filtering exclusively for ADV data processing.

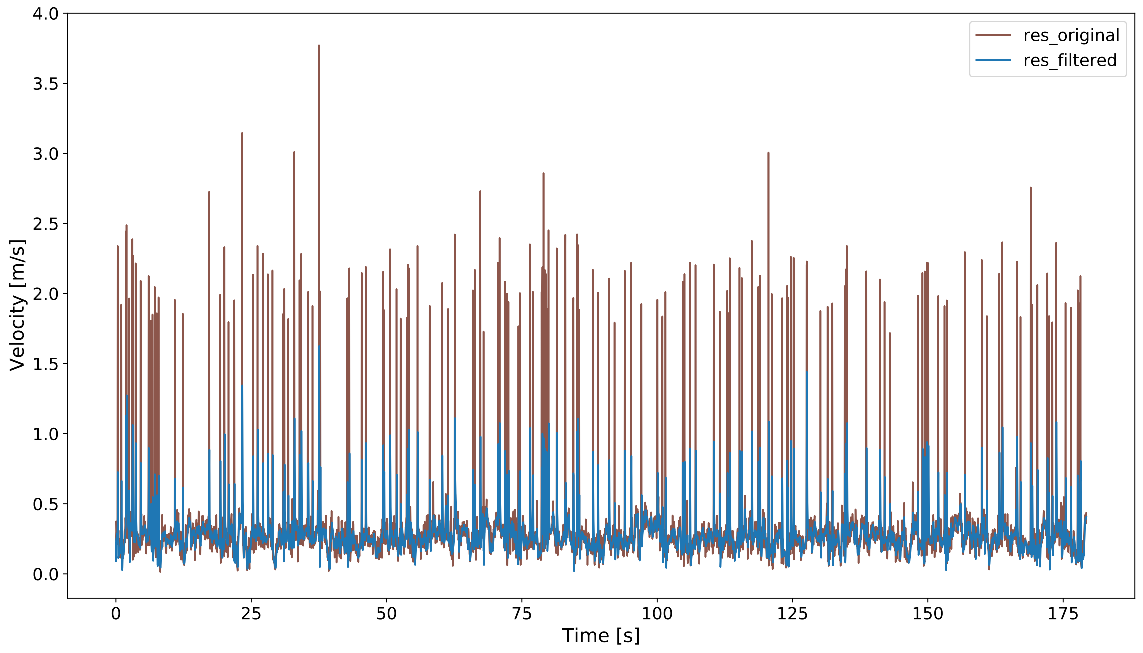


Figure 6.4.: Exemplary visualisation of Butterworth filtered ($2.5\,\mathrm{Hz}$, order=1) time-series against original time-series.

These statements are valid for all turbulence conditions and number of spikes for the samples at hand. The exemplary time-series that undergoes Butterworth filtering with a cutoff frequency of 2.5 Hz and an order of 1 is shown in figure 6.4. The time-series filtered by the other settings of the Butterworth filter can be viewed in the annex (see figures A.1 to A.4).

While all cutoff frequencies up to 2.5 Hz provide plausible results, filtering with the cutoff frequency of $\frac{f_s}{2.93}$ and filter order 3, as suggested in B. J. MacVicar et al., 2007 and Roy et al., 1997, yields no discernible effect on the time-series. The same can be stated for the spectra obtained with this setting. As this approach is generally unsuccessful the respective time-series and spectra can only be found in the annex (see figures A.4 and A.24).

However, the spectra obtained with the cutoff frequencies up to 2.5 Hz provide spectra that show a good fit to the theoretical energy dissipation spectrum for turbulent flows (see figures A.21 to A.23 and 6.6a). All cutoff frequencies show similar results, but a slightly increasing upward arch in the central part of the spectra can be identified with increasing cutoff frequencies. For all turbulence conditions and samples, the w-velocity component spectra linger at slightly lower energy levels than the spectra of the other components.

The w-velocity component spectra's course improves for higher cutoff frequencies, the best spectra are therefore produced by the first order Butterworth filter with a cutoff frequency of 2.5 Hz displayed in 6.6a. In the high frequency part of the spectra, the component spectra plummet while the spectra of the velocity magnitude maintain a horizontal course.

6.1.4. Gauss Filtering

The Gauss filtering script causes a substantial attenuation of the signal compared to the Butterworth filter (see figure 6.5). Interestingly, there is a sign change of negative spikes (see figure A.12 in the annex). As with the Butterworth filter, some spikes disappear entirely, and others are only attenuated, which again points to spikes occurring regardless of frequency.

Gauss filtering improves the filtered turbulence spectra compared to the original data spectra. Although an approach to the course of the theoretical energy dissipation spectrum can be seen, the spectra do not distinctly display the desired slopes' features (see figure 6.6b). Especially the straight-lined central part of the theoretical spectrum is missing - this part of the spectra is somewhat arched upward after Gauss filtering.

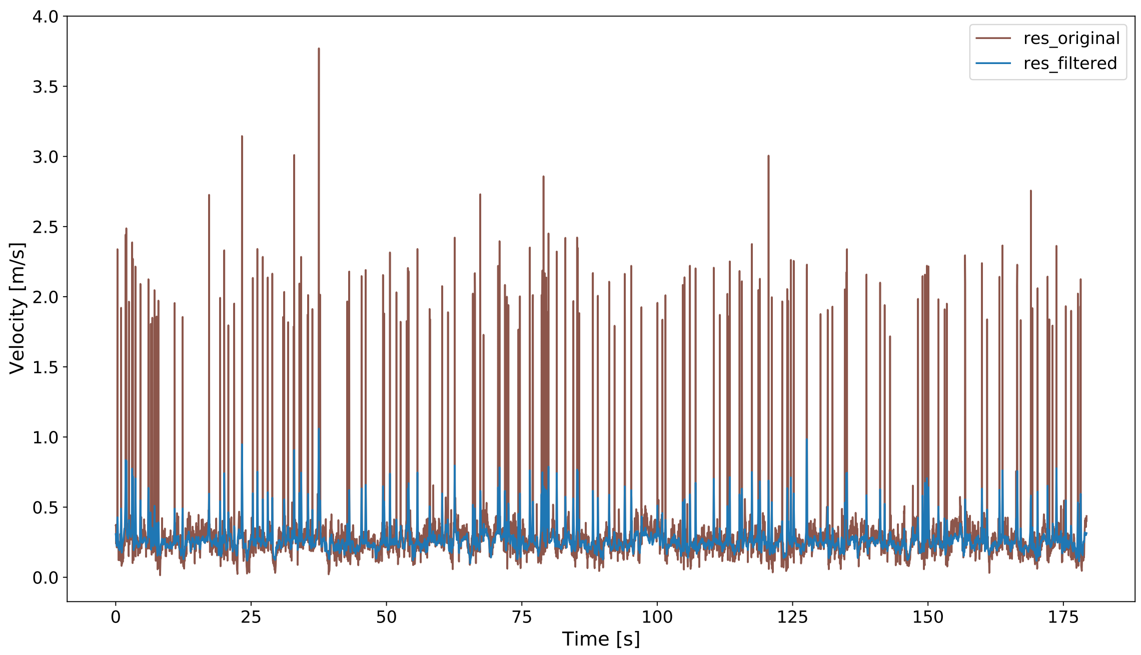
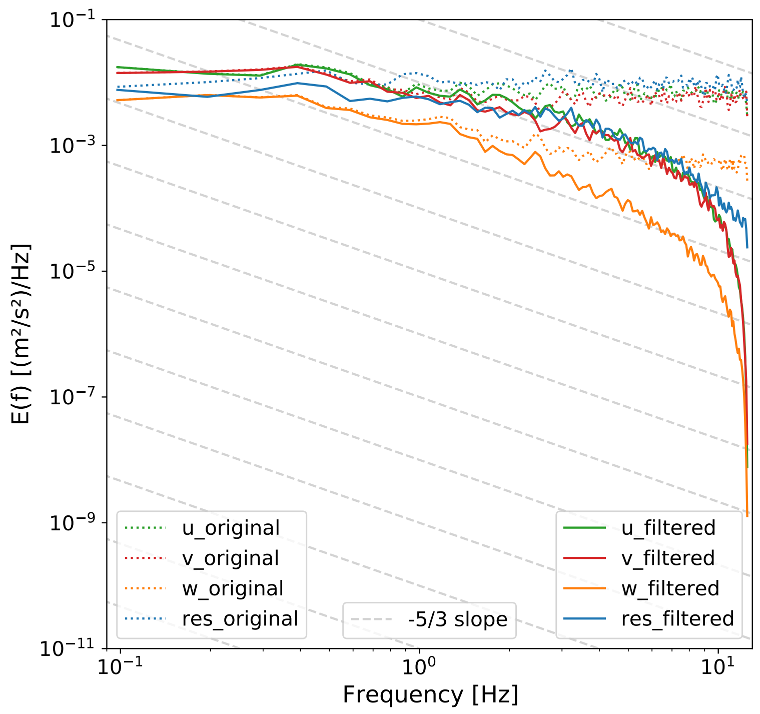


Figure 6.5.: Exemplary visualisation of Gauss filtered time-series against original time-series.

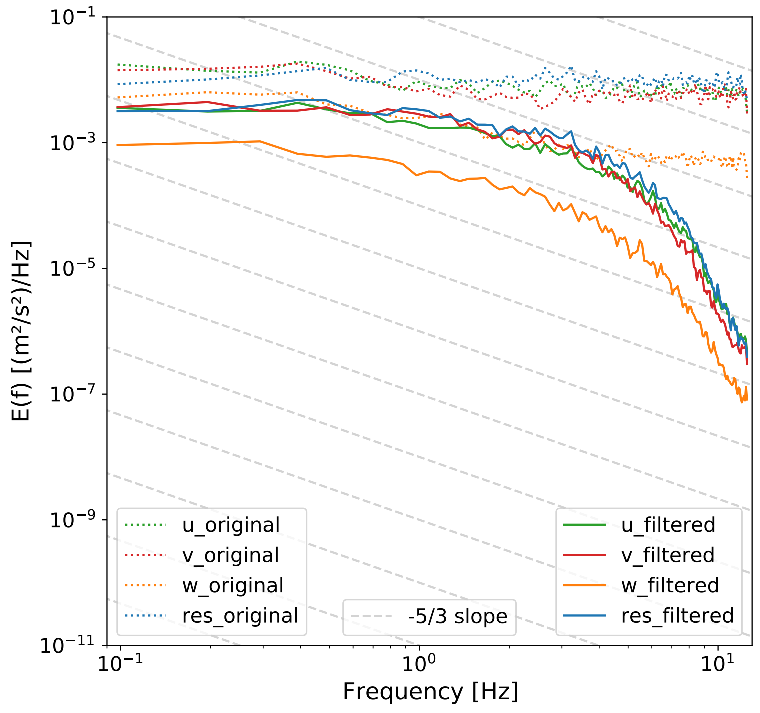
6.1.5. Combination of Methods

The screening of spectra shows that no method provides satisfactory results in all aspects. While, on the one hand, digital filtering generates the best results concerning turbulence spectra, the time-series are still contaminated with spikes. On the other hand, the despiking methods despike more or less reliably, but they have clear disadvantages in terms of spectral characteristics. Therefore, the combination of previously semi-productive methods is examined additionally. It must be noted that despiking efforts are always performed before applying denoising filters. This order is vital as the attenuation of spikes might disrupt data morphology, consequently leading to unsatisfactory despiking: by attenuating spikes before despiking, the data clusters, which pose a fundamental concept for despiking methods, change, which in turn affects cutoff or demarcation processes.

Both despiking methods are combined with the most promising filtering methods - the first order Butterworth filter with a cutoff frequency of 2.5 Hz and the Gauss filter. As expected, the time-series smooth out by lowpass filtering after despiking. Generally, the Gauss filter attenuates the signal more than the Butterworth filter. Because the P-SAT framework already smooths the signal somewhat, the resulting signal from combining P-SAT and Gauss filtering seems too strongly attenuated. Inspection of the time-series shows that the time-series of several samples miss some distinguishable features.



(a) Spectra from Butterworth filtered ($2.5\,\mathrm{Hz}$, order=1) time-series.



(b) Spectra from Gauss filtered time-series.

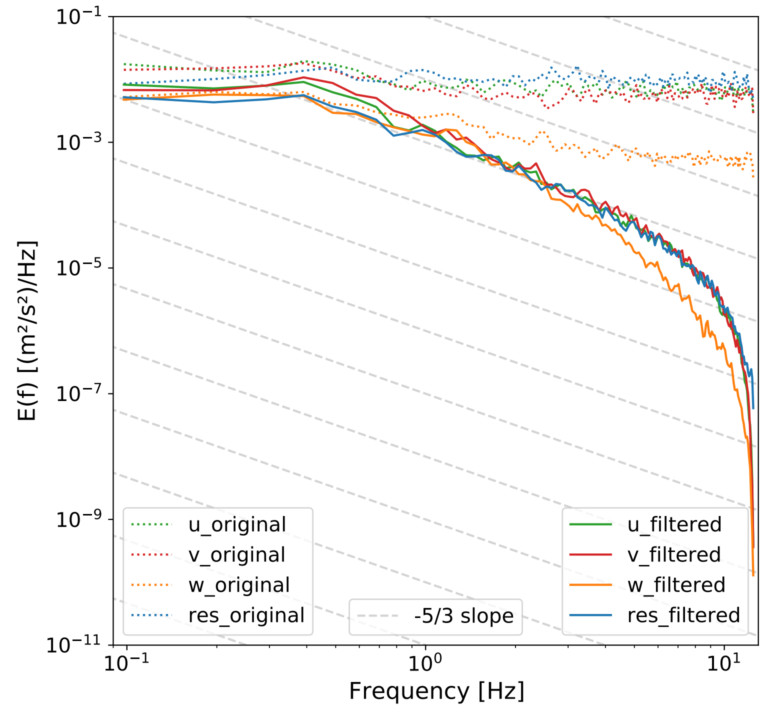
Figure 6.6.: Exemplary visualisation of spectra from denoised time-series.

Moreover, the additional use of denoising methods after P-SAT despiking does not substantially affect the spikes remaining in the signal. Sporadically, the remaining spikes are attenuated slightly. Sometimes, they are attenuated so strongly that they thoroughly blend with the signal. Again, this demonstrates that spikes are present at all frequencies. Correspondingly, it can be deduced that denoising efforts cannot compensate for shortfalls in despiking. The time-series of the combinations are showcased in the annex (see figures A.5 to A.8 and figures A.17 to A.20).

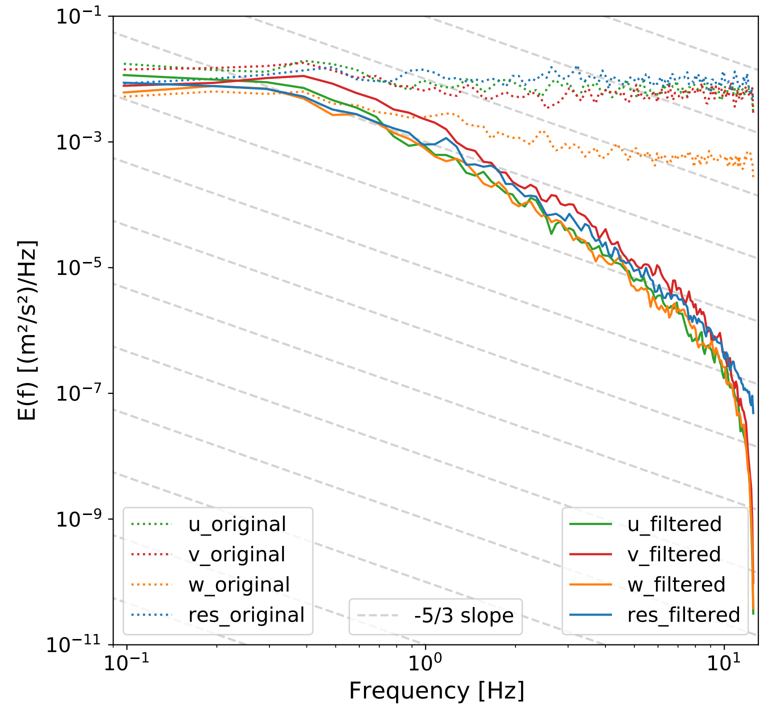
Figure 6.7 shows the spectra obtained from combining both despiking methods with the best Butterworth filter option with cutoff frequency at $2.5\,\mathrm{Hz}$ and order 1. The spectra of the combined use of KDE despiking and the Butterworth filter, displayed in figure 6.7a, show significant improvement compared to only the Butterworth filter (see figure 6.6a) and only KDE despiking (see figure 6.3b): the central part of the spectra distinctly follows Kolmogorov's $-\frac{5}{3}$ slope and the high turbulence part of the spectra plummets instead of remaining horizontal. Also, it can be observed that the course of the w-velocity component spectra aligns with the other components' spectra regarding energy levels. The spectra' starting energy level remains nearly unchanged compared to the original spectra - the level is only slightly reduced.

Applying the P-SAT framework in combination with the above-specified Butterworth filter produces the spectra in figure 6.7b. The spectra show improvement in the high-frequency parts, but the slope in the central part of the spectra is also disadvantageously altered. Compared to figure 6.3a, the slope is significantly steeper, especially for frequencies beyond $1.5\,\mathrm{Hz}$.

Figure 6.8 shows the spectra obtained from both despiking methods combined with the Gauss filter. Both combinations produce satisfying spectra. The high-frequency part of the spectra does not plummet as sharply as with the Butterworth filter, but the central parts of the spectra show good accordance with the $-\frac{5}{3}$ slope. The stronger attenuation of the Gauss filter shows in the decreased energy levels of the spectra, which are already apparent at the lowest frequencies.

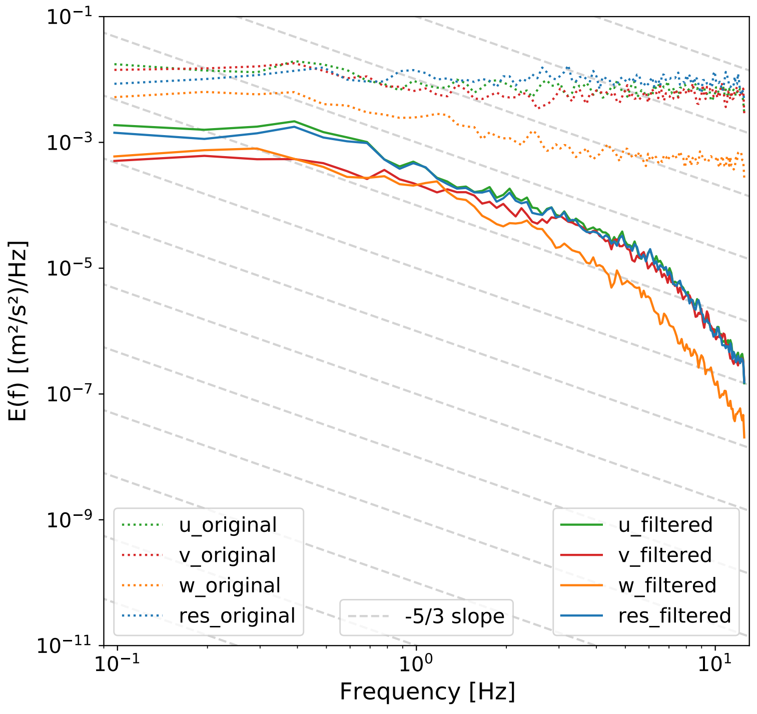


(a) Spectra from KDE despiked and Butterworth filtered time-series.

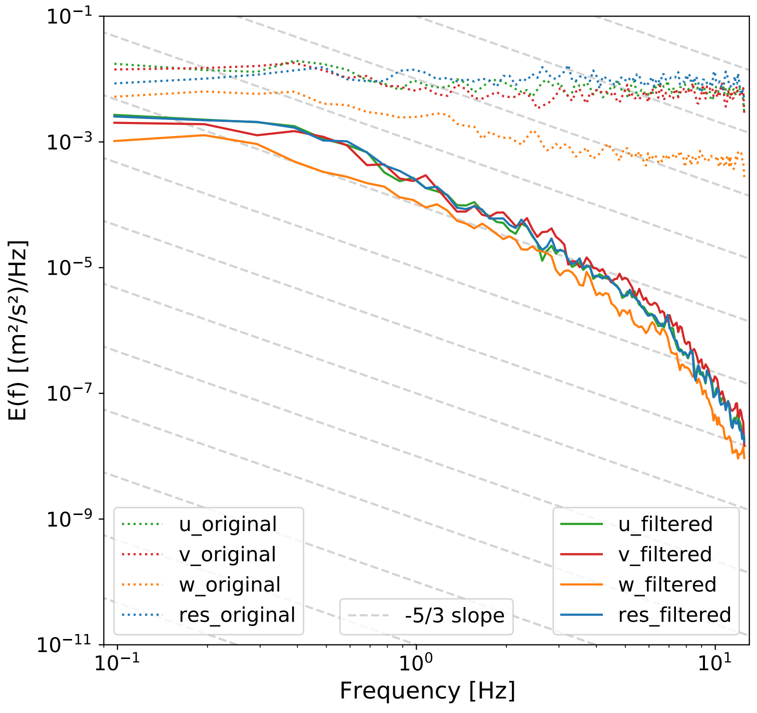


(b) Spectra from P-SAT filtered and Butterworth filtered time-series.

Figure 6.7.: Exemplary visualisation of spectra from time-series obtained by combination of despiking methods and Butterworth filtering ($2.5\,\mathrm{Hz}$, order=1).



(a) Spectra from KDE despiked and Gauss filtered time-series.



(b) Spectra from P-SAT filtered and Gauss filtered time-series.

Figure 6.8.: Exemplary visualisation of spectra from time-series obtained by combination of despiking methods and Gauss filtering.

6.2. Statistical Parameters

The GUM asks for several parameters to be surveyed regarding type A evaluation (see subsection 3.3.1). Python scripts can be used for type A evaluation to efficiently give out needed parameters for each input sample, e.g., using the output parameters for reporting. This application is easily realised using a relatively straightforward script (see source code A.27). The implementation determines the mean, the variance, the standard deviation, the global maxima and minima, the skewness and the kurtosis for all velocity components and the velocity magnitude. Furthermore, the covariance and correlation of all velocity components and the turbulent kinetic energy are computed. The script runs on average 11.82 s to calculate all parameters for the 160 original samples.

Here, the calculation of parameters according to type A evaluation is consulted to determine the effects of implemented data analysis methods. For this, it is necessary to establish criteria to quantify the change in parameters. The following subsections explain four different parameters and how they can be used to determine the improvement or degradation of samples due to the ADV data analysis methods implemented in Python (see chapter 5). For every data analysis method, the parameters are evaluated in comparison to the original samples or the normal distribution and then are allocated to one of four categories. By categorising the parameters, it can be determined what share of the 160 samples changes in which manner and at what magnitude. The percentage of samples per category for each analysis method is visualised in a heatmap for every velocity component to grasp the parameter change at a glance. It is not productive to evaluate the parameters of the velocity magnitude, as spikes turn positive as a consequence of squaring component values. Thus, it can no longer be assumed that the mean of the spikes is zero, which might not be especially relevant for samples containing a small number of spikes but does indeed influence samples with many spikes.

A script (see source code A.35 for one examplary parameter) is executed in Python to create the heatmaps. The script employs the four methods ”get_filenames”, ”get_parameters”, ”count_-categories” and ”sort_category_by_method”, which remain the same for every parameter. The method ”get_filenames” (see source code A.9) identifies the basenames of the files in the user-specified directory. Then, the method ”get_parameters” (see source code A.28) combines the basenames with the file name endings in the directory, extracts the parameters from the parameter files created with the ”Parameter_Calculation” script (see source code A.27) and creates a data frame for all parameters for one sample. This method needs to be changed when changing the methods one wants to compare or when changing the file denotation. In the next step, a categorisation method is employed. The methods to categorise the parameters are different for each parameter and picked up separately in the following subsections. After categorisation, the category data must be sorted by method (see source code A.30) to be able to visualise the data with the heatmap function from the seaborn library. All in all, the user needs to specify a directory containing all parameter files, the number of methods to compare, the number of velocity components, the number of categories, the number of samples to compare and needs to name the methods in their order in the parameter data frame, as well as the data frame indices of the parameter to compare (see table A.1 in the annex). The script gives out the heatmap figures in the directory in which the script is executed.

6.2.1. Mean

The mean is an essential statistical parameter to evaluate because it is the parameter used for most ADV data applications. Even after applying data analysis methods such as despiking and denoising, no change in the mean should occur when comparing the means of filtered and original means. This indicator is a deduction from the error effects (spikes and white noise) being expected to exhibit a mean of zero.

Therefore, this analysis determines relative positive and negative changes between original and filtered samples. While a positive relative change expresses that the data analysis operation increases the filtered mean compared to the original mean, a negative relative change attests to a decrease in mean. The calculation is displayed in equation 6.1.

$$\Delta\mu = \frac{(\mu_{\text{filtered}} - \mu_{\text{original}})}{\mu_{\text{original}}} \cdot 100 \tag{6.1}$$

$\Delta\mu$	– relative change in means [%],
μ_{filtered}	– mean of the filtered sample [$\frac{\text{m}}{\text{s}}$],
μ_{original}	– mean of the original sample [$\frac{\text{m}}{\text{s}}$].

The relative change in means is then sorted into established categories using the method ”categorise_means” (see source code A.31). The established categories are:

- Category 1: $\Delta\mu \geq -1\,\%$ and $\Delta\mu \leq 1\,\%$,
- Category 2: $\Delta\mu \geq -5\,\%$ and $\Delta\mu \leq 5\,\%$,
- Category 3: $\Delta\mu \geq -10\,\%$ and $\Delta\mu \leq 10\,\%$,
- Category 4: $\Delta\mu < -10\,\%$ and $\Delta\mu > 10\,\%$.

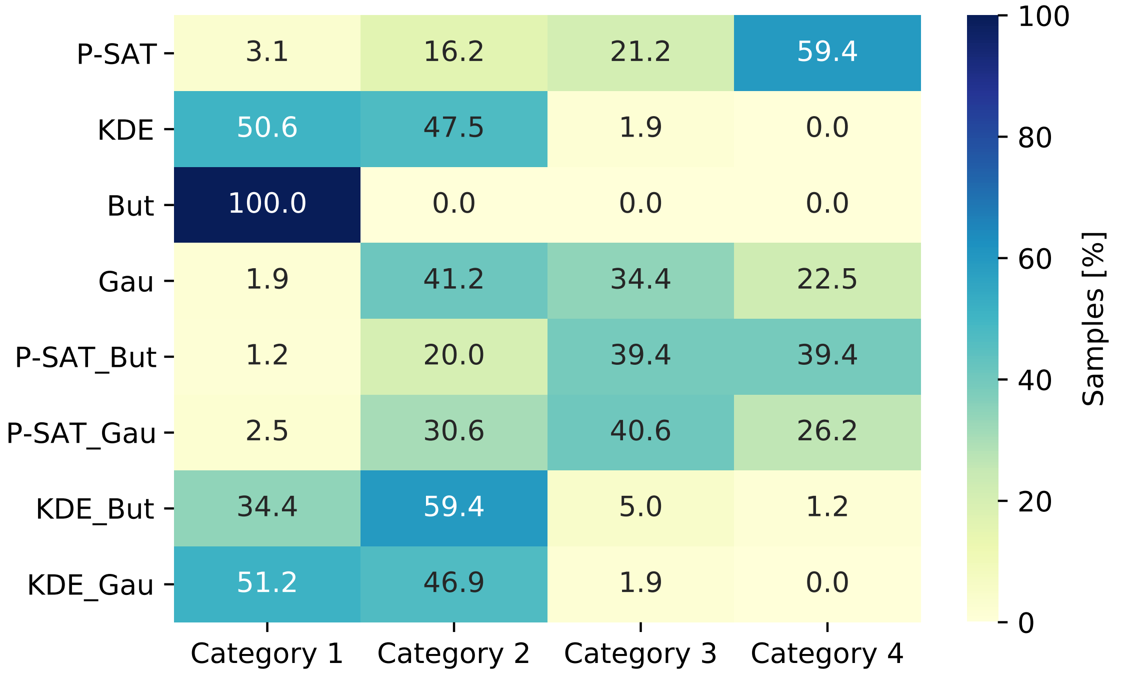


Figure 6.9.: Heatmap of the categorisation of the relative change in means in the u-velocity component for all implemented data analysis methods (n=160 samples).

While categories 1 and 2 correspond to values often chosen in statistical contexts, the boundary for category three is selected because a relative change over 10 % already seemed unacceptable to attest to mediocrity. When evaluating the created heatmap for the u-velocity component, the options KDE despiking, Butterworth, and both KDE despiking combinations are the methods inducing the slightest relative change in the mean. Gauss filtering and both P-SAT despiking combinations work reasonably well. The P-SAT framework changes the mean of the u-velocity component the most, with 95 samples (59 % of all samples) in category 4.

The heatmaps for the two horizontal velocity components are showcased in the annex (see figures A.27 and A.28). The v-velocity and w-velocity components are susceptible to small absolute mean changes because the values themselves are small. Even the slightest absolute changes cause significant relative changes. The Butterworth filter still manages to perform well for these velocity components. The second and third best methods are KDE despiking and KDE despiking combined with Gauss filtering. Also, it must be noted that the v-velocity and w-velocity components in the samples at hand are at high risk for high uncertainty in measurement because some absolute values are smaller than $10\,\frac{\mathrm{mm}}{\mathrm{s}}$, which is the given accuracy of the measurement device. All methods have the same effect on the mean change disregarding the turbulence conditions, which are evaluated by comparing the distributions of shares in the categories (the respective heatmaps are included in the supplementary materials).

6.2.2. Variance

The primary goal of the data analysis methods applied in this thesis is to eliminate known error terms from samples. These processes also induce a decrease in signal variance. A relative change approach, just as for the change in the mean, is selected to quantify this decrease. The respective calculation goes as follows:

$$\Delta\sigma^2 = \frac{\left(\sigma^2_{\text{filtered}} - \sigma^2_{\text{original}}\right)}{\sigma^2_{\text{original}}} \cdot 100 \tag{6.2}$$

$\Delta\sigma^2$	– relative change in variance [%],
$\sigma^2_{\text{filtered}}$	– variance of the filtered sample [$\frac{\text{m}^2}{\text{s}^2}$],
$\sigma^2_{\text{original}}$	– variance of the original sample [$\frac{\text{m}^2}{\text{s}^2}$].

On the one hand, a decrease of variance in the signal indicates a reduction of erroneous data; on the other hand, the lowering of signal variance lets the signal appear smoother. The smoothing makes it easier to identify primary patterns in the data and gather periodicity. However, if the decrease in variance is too high, it can also make essential features of the signal indistinct.

Four categories are established to categorise the decrease in the variance. They are following the findings by Cea et al., 2007, p. 8, a paper that investigates the effects of the acceleration thresholding method (the method implemented in the P-SAT framework) on signal variance. They find a reduction of the u-velocity and one of the horizontal velocity components of around $30\,\%$, while the variance of the remaining horizontal velocity component only changes slightly. This information is used to section the categories so that they allow comparability with this study (see respective method ”categorise_variance” in source code A.32):

- Category 1: $\Delta\sigma^2 \leq -50\,\%$,
- Category 2: $\Delta\sigma^2 \leq -25\,\%$ and $\Delta\sigma^2 > -50\,\%$,
- Category 3: $\Delta\sigma^2 \leq -10\,\%$ and $\Delta\sigma^2 > -25\,\%$,
- Category 4: $\Delta\sigma^2 > -10\,\%$.

The analysis of the statical parameters of the samples at hand (see figure 6.10) shows that the reduction of the variance caused by the acceleration thresholding method is generally in the same range of decrease as in the study by Cea et al., 2007. All other methods reduce the variance by more than $50\,\%$ (see figures A.29 and A.30 in the annex). For the horizontal velocity componentes, a slightly decrease can be attested for the w-velocity component for the P-SAT framework, KDE despiking and the Butterworth filter. Nevertheless, most sample variances still decrease significantly and not only slightly, as proposed by Cea et al., 2007. However, it also remains in question what kind of change the authors deem as ”slight change” as no values are mentioned in this regard. All methods have the same effect on the variance change disregarding of turbulence conditions (the heatmaps by turbulence conditions are included in the supplementary materials).

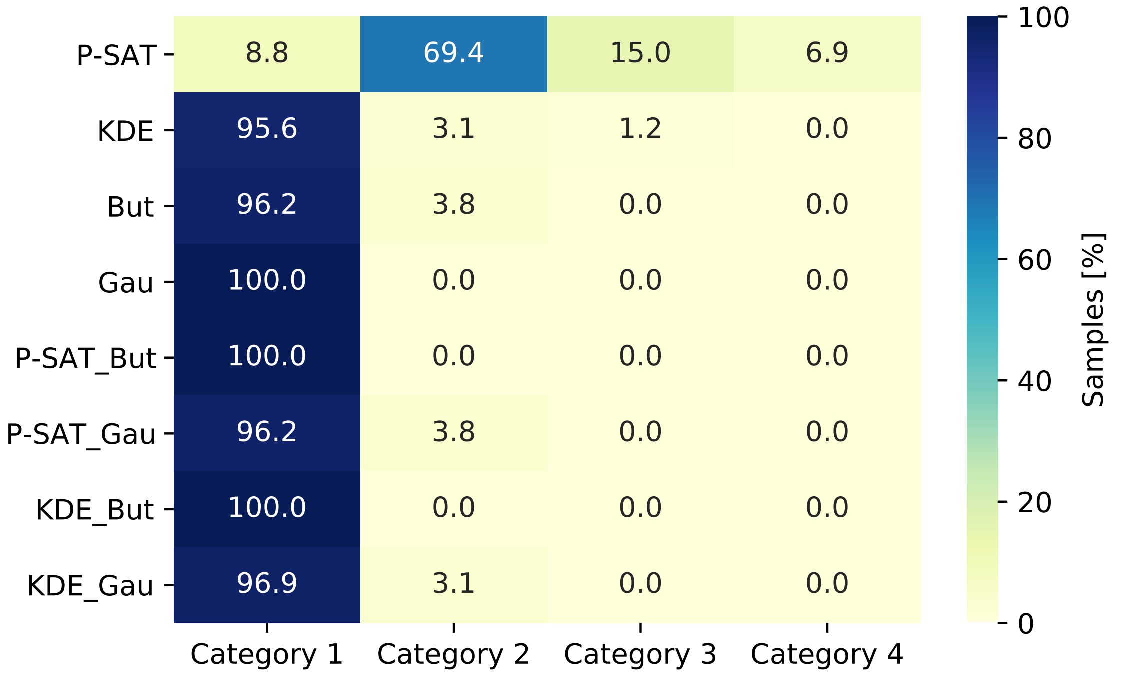


Figure 6.10.: Heatmap of the categorisation of the relative change in variance in the u-velocity component for all implemented data analysis methods (n=160 samples).

6.2.3. Skewness

The skewness is a measure to evaluate if fluctuations in the signal are symmetric in relation to the mean of the signal or if they deviate to either side of the mean, i.e. are skewed positively (distribution leaning to the left side) or negatively (distribution leaning to the right side) (Cea et al., 2007, p. 9). The evaluation of the samples regarding the skewness does not consider parameter change. Instead, the skewness of the original and filtered samples is compared to the skewness of the normal distribution, which is symmetric. Then the effect of each data analysis method is evaluated by assessing the number of samples that move to a category attesting less deviation from the normal distribution compared to the original samples. This approach is possible as turbulent velocities are considered to be nearly normally distributed (see subsection 2.2.3). The skewness of the normal distribution is zero ($\gamma_m = 0$), and the categories are defined as follows:

- Category 1: $\gamma_m \geq -0.1$ and $\gamma_m \leq 0.1$,
- Category 2: $\gamma_m \geq -0.5$ and $\gamma_m \leq 0.5$,
- Category 3: $\gamma_m \geq -2$ and $\gamma_m \leq 2$,
- Category 4: $\gamma_m < -2$ and $\gamma_m > 2$.

The sectioning of categories is deduced from values from the references Bailly and Comte-Bellot, 2015, p. 11 and Cea et al., 2007, p. 9 that both report skewness for filtered velocity signals. These values coincide with references on basic statistics that call distributions to substantially deviate from the normal distribution for values $\gamma_m < -2$ and $\gamma_m > 2$ (Schiefer & Schiefer, 2021, p. 33). The method ”categorise_skew” according to categorise the samples by skewness can be viewed in the annex (see source code A.33).

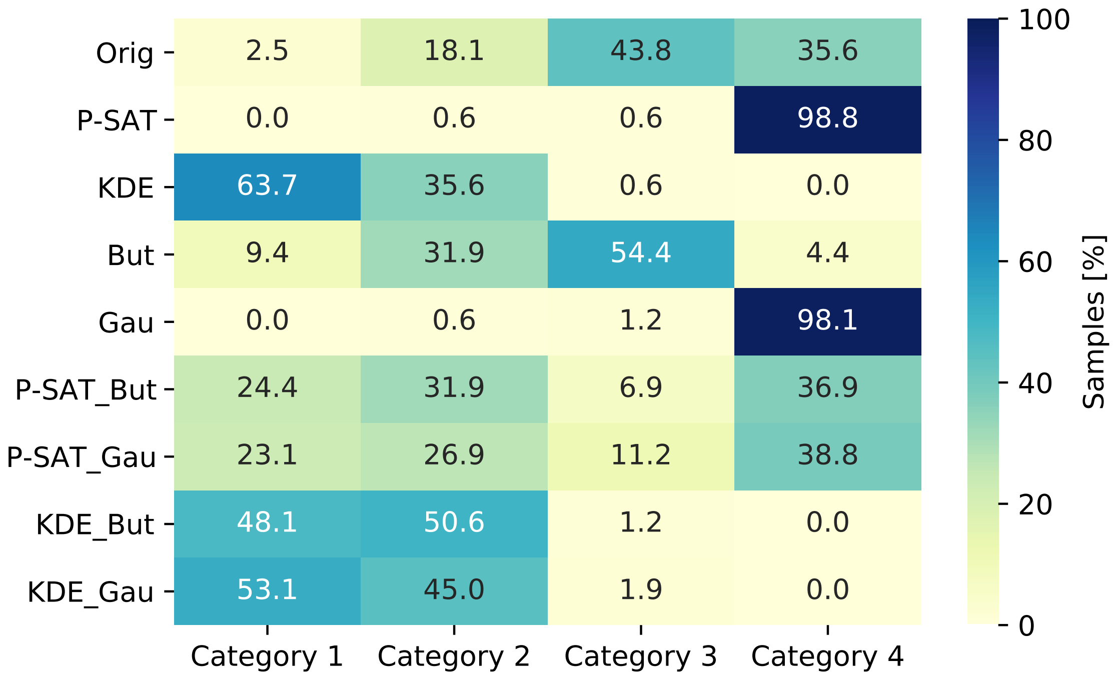


Figure 6.11.: Heatmap of the categorisation of the skewness in the u-velocity component for all implemented data analysis methods in comparison to the categorisation of the original samples (n=160 samples).

Analysing the heatmap of samples sorted by skewness categories for the u-velocity component (see figure 6.11), it becomes apparent that only the method KDE and both KDE combinations lead to most samples showing a fairly symmetrical distribution. The P-SAT framework and the Gauss filter even make the distributions more skewed than they were in the original samples. Especially from the P-SAT framework one would expect better results regarding the skewness.

The same can be stated for the v-velocity component (see figure A.31 in the annex) and for the w-velocity component (see figure A.32 in the annex). Interestingly, the P-SAT method performs better for the w-velocity component, but the original samples are also less skewed for this component. Overall, the results for the acceleration thresholding method (P-SAT framework) do not agree with the findings of Cea et al., 2007. They determine the effect of the acceleration thresholding method on the skewness of turbulent velocity samples and find an improvement in the skewness parameter when comparing filtered to original samples.

All methods have the same effect on the skewness disregarding the turbulence conditions (the heatmaps by turbulence conditions are included in the supplementary materials).

6.2.4. Kurtosis

The kurtosis of a signal is the parameter that evaluates the magnitude of present fluctuations. Therefore, a signal that contains spikes shows a significantly higher kurtosis than a non-spiky signal. The categorisation of the kurtosis parameters follows the same logic as the one for skewness in the previous subsection 6.2.3. The kurtosis of the normal distribution is three ($\omega_m = 3$), which is denoted as mesokurtic. A distribution with smaller values for the kurtosis is called leptokurtic, and a distribution with smaller values for the kurtosis is called platykurtic. Leptocurtic distribution does not exist with the samples at hand, as they are rather contaminated with spikes than extenuated values. Accordingly, the categories are defined as follows:

- Category 1: $\omega_m \geq 2$ and $\omega_m \leq 4$,
- Category 2: $\omega_m > 4$ and $\omega_m \leq 10$,
- Category 3: $\omega_m > 10$ and $\omega_m \leq 50$,
- Category 4: $\omega_m > 50$.

The sectioning of categories is again chosen based on Schiefer and Schiefer, 2021, p. 33 deeming a distribution significantly different from the normal distribution with the kurtosis of $\omega_m < 1$ and $\omega_m > 5$. The values in Bailly and Comte-Bellot, 2015, p. 11 and Cea et al., 2007, p. 9 confirm this order of magnitude for turbulent velocity distributions. The Python implemented method ”categorise_kurt” to categorise the samples by kurtosis can be viewed in the annex (see source code A.34).

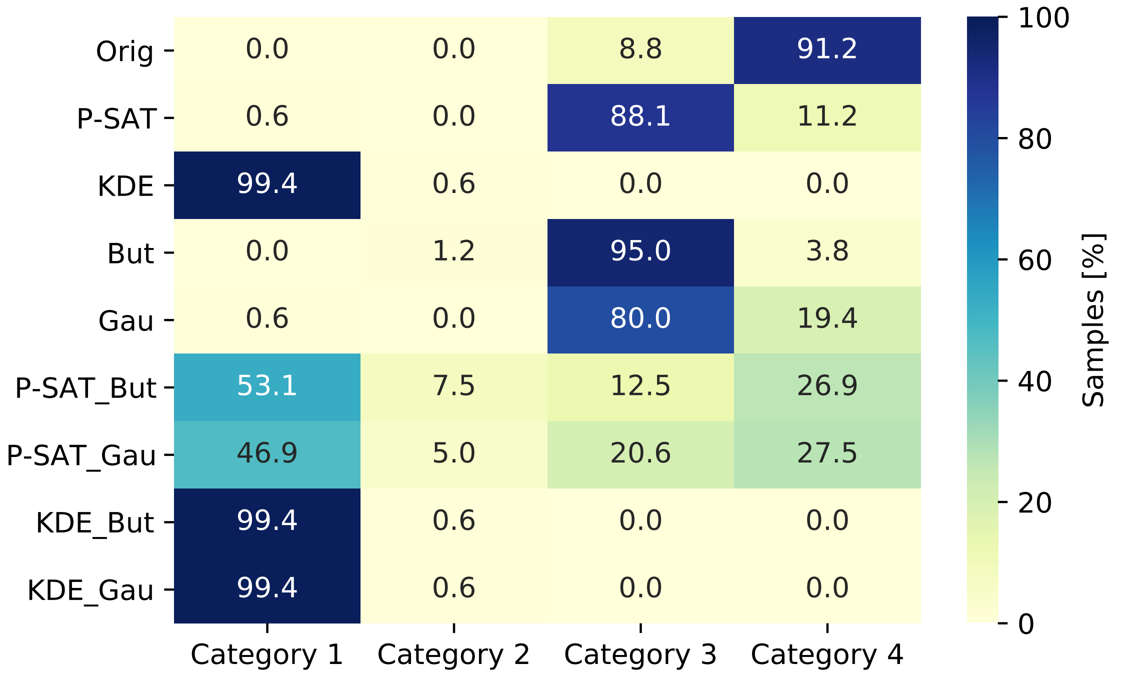


Figure 6.12.: Heatmap of the categorisation of the kurtosis in the u-velocity component for all implemented data analysis methods in comparison to the categorisation of the original samples (n=160 samples).

The heatmap for the kurtosis of the u-velocity component (see figure 6.12) shows that all data analysis methods improve a large share of samples in terms of the kurtosis. This effect is not surprising, as they all attenuate the signal. The methods showing the most improvement and the best mesokurtic distribution in samples are the KDE despiking method and its combinations. The P-SAT combinations also work well, but around a quarter of the samples stay in the extremely platykurtic realm. Again, the findings concerning the acceleration thresholding method (implemented by the P-SAT framework) do not agree with respective references.

The same can be stated for the heatmap of the v-velocity component (see figure A.33 in the annex). Just as for the skewness, the results are improved for the w-velocity component (see figure A.34 in the annex) because the original samples already show a more favourable distribution in the original samples.

All methods have the same effect on the kurtosis disregarding the turbulence conditions (the heatmaps by turbulence conditions are included in the supplementary materials).

6.3. Discussion of Results

Firstly, the evaluation procedure itself needs to be discussed. The procedure is devised by taking into account recommendations by the GUM and is split into three separate approaches: visual inspection of time-series (type B), visual inspection of turbulence spectra (type B) and evaluation of statistical parameters (type A). Generally, all of these approaches are useful somehow, but one must be careful of their scopes.

Their constraints become evident when evaluating the filtered sample's output by the P-SAT framework: The spectra produced by this method are fine, but when inspecting the time-series it is evident that single spikes remain in the time-series. Based on the respectable spectra, one could overlook the remaining spikes and not give them much weight when reviewing the big picture. However, the method additionally manipulates the data morphology in a disadvantageous way, which becomes clear when reviewing the skewness parameter, for which the results worsen by applying the P-SAT framework. One would never identify this flaw in the statistical distribution by merely evaluating the time-series, which shows that type A evaluation is obligatory when evaluating data, just as recommended by the GUM. All in all, this leads to three principle results that are deduced for evaluation strategies:

- Visual inspection is mainly helpful for evaluating spectra and only evaluating time-series provisionally.
- Assessing statistical parameters is essential to evaluating measurements and cannot be replaced by merely visually inspecting the time-series or visually inspecting spectra.
- Only the combination of statistical evaluation and inspection of spectra reliably evaluates the effectiveness of data analysis methods for turbulent velocity data.

Concerning the effectiveness of the data analysis methods implemented in Python, several conclusions are drawn from the findings presented in the previous subsections:

- Original ADV data must not be further used without adequately evaluating the data and potentially processing the data accordingly.
- Either the P-SAT framework does not work as intended by the authors, or it is operated incorrectly in this thesis.
- Digital noise filtering efforts effectively attenuate signals, and their application manipulates the spectra positively, but they must only be used for already despiked data.
- Choosing adequate filter characteristics and parameters is essential for filtering to yield satisfactory results.

- The best settings for the Butterworth filter for the data at hand are a cutoff frequency of 2.5 Hz and an order of 1.
- Data analysis methods can differ for studied turbulence conditions.
- The KDE despiking method in combination with Butterworth lowpass filtering produces the best results regarding spectra, time-series and statistical properties. It also filters produces reliable results for every studied turbulence condition.

The outlook of this thesis (see chapter 8) discusses the consequences of these results for future work with turbulent flow ADV data while also taking the results from the following chapter (see chapter 7) on the optimal sampling time for ADV measurements under consideration. It also gives insight into the general practicalities of the implementation in Python and transferability to other measurement devices and experimental set-ups.

7. Optimal Sampling Time

The sampling time is a rather definitive decision when undertaking a hydraulic study. The researcher intends to choose a sampling time that delivers statistically significant measurements and aims for the shortest sampling time possible to save project time and optimise laboratory and resource access. Although there are general recommendations for sampling times, one could save time by determining optimal sampling times for standard measurement devices or similar research questions. Determining optimal sampling times could minimise laboratory work while affirming the statistical significance of measurements.

In literature, one finds several different recommendations regarding sampling times: Adam and Lehmann, 2011, p. 97 mention $3\,\text{min}$ to $5\,\text{min}$ for turbulent flow to have high enough resolution for fluctuations but do not specify sampling frequencies. Chanson et al., 2007, p. 4 recommend at least 5000 measurement points for first and second statistical moments and a range of 25000-50000 measurement points for third and fourth statistical moments. $40\,\text{s}$ is the sampling time recommended by US agencies and institutions (Turnipseed & Sauer, 2010, p. 58). However, the manufacturer SonTek emphasises that higher sampling times $t_s > 40\,\text{s}$ might be needed for turbulent flow or low flow velocities but does not give tangible recommendations (SonTek, 2011, p. 126). An extensive study on sampling times by Díaz Lozada et al., 2021 introduces a new method called dynamic selection of exposure time (SET). After measuring increments of $5\,\text{s}$ to $10\,\text{s}$, the data is statistically evaluated (by parameters and distribution) using real-time bootstrapping (a statistical resampling technique) and compared to user-set confidence margins. If the user-set requirements regarding uncertainties in measurement are met, the recording of data is stopped. The authors compare the time needed for data collection across entire cross-sections and could decrease the sampling times between $36\,\%$ and $49\,\%$ compared to a fixed sampling time of $40\,\text{s}$. (Díaz Lozada et al., 2021, pp. 3, 9) When reviewing the literature on sampling time, it stands out that recommendations are either ambiguous or associated with a great deal of effort.

This chapter aims to illustrate the process of finding a procedure to determine an optimal sampling time for hydraulic measurements via hypothesis testing. As this thesis has a time limit, the procedure is developed for only one measurement device (Nortek Vectrino) with one setting ($25\,\text{Hz}$) at six measurement points. Firstly, the general approach to the problem will be explained. Secondly, a detailed protocol for implementing this general approach in

Python is given. This subsection also gives further insight into technicalities and decision making regarding resampling and input parameters. At the end of the section, the results are critically evaluated. A discussion gives recommendations and comments on the transferability of the method to different research settings and measurement devices.

7.1. General Approach

Six long-duration ADV measurements were undertaken in the hydraulic laboratory at TU Darmstadt by Katharina Bensing to analyse optimal sampling times. Each measurement was carried out with the Vectrino device by Nortek at a sampling rate of $25\,\mathrm{Hz}$ for about $30\,\mathrm{min}$ (around 45000 measurement points) at two different horizontal positions in a flume (centre and wall) and three vertical positions over the flume's bottom ($0.1\,\mathrm{m}$, $0.4\,\mathrm{m}$ and $0.6\,\mathrm{m}$) per horizontal position. The original data from the ADV is not filtered or otherwise processed as it is presumed that these techniques alter the samples' statistical parameters, which would distort the analysis on optimal sampling times.

It is presumed that by prolonging the sampling time, statistical parameters converge toward a constant value as per the LLN (see subsection 2.1.1). Although this can be shown quite easily by plotting sample means for increasing sampling times, the question remains when a researcher can be sure enough to measure a significant sample meeting expected quality parameters (see subsection 2.1.2). Nonetheless, plotting the sample means can help determine the degree of quality the researcher is looking for. By plotting the sample means, the researcher can gain insight concerning the dispersion of their data and can evaluate what defines the quality of statistical parameters of a sample in the context of the research question.

A comparison of means is performed to quantify at what sampling time the statistical parameters of a sample are generally robust enough to meet the desired quality: the means of samples with shorter sampling times are compared to the mean of a sample for which one can be sure that the dispersion is small enough (subsequently called the base sample). Typically, the comparison of means is carried out with a t-test (see subsection 2.1.3) or a Mann–Whitney-U test. However, as these test for differences in means, the TOST is used in this paper instead to test for equivalence in means (see subsection 2.1.4). A margin of equivalence for the TOST can be devised by assessing the plot of sample means.

To ensure that the test results are not random but significant and reliable, the TOST is carried out for each increment (time steps of $20\,\mathrm{s}$) of many discriminable samples (n = 2000). The TOST compares against ten different base samples. By counting the number of acceptance of the null hypothesis and calculating the mean number of acceptance across all combinations of samples, an indicator of performance to produce satisfying samples is given for each sampling time. This procedure is performed for each of the six long-duration measurements. The results are expected to be similar for each measurement for the same test setup (i.e. the same significance level and the same margin of equivalence). Still, there might be slight differences in the original data of the measurements in proximity to the wall and the bottom of the flume because they contain more measurement errors (see subsection 3.3.3) than the measurements appropriately spaced from the bottom and the wall.

Although this allows a statement on whether a shorter sampling time generally produces samples of an acceptable quality concerning their statistical parameters, it does not eliminate errors. It excludes the possibility that a sample might not be of sound quality on occasion. There still must be a consideration of the margin of errors in hypothesis testing and the sampling itself.

7.2. Requirements and Input Parameters

Before starting the actual implementation of the TOST in Python, the requirements to perform a t-test were considered (see subsection 2.1.3) and transferred to the problem at hand: Here, each long-duration ADV measurement at full measuring time, i.e. around $30\,\mathrm{min}$, represents one population. Each population comprises discrete, independent and interval-scaled measurements. As discriminable samples are resampled from all data points, the independence of samples is ensured by choosing a maximum sample size of 30000 measurements, i.e. a sampling time of $20\,\mathrm{min}$. Normal distribution of the examined characteristic (see subsection 6.2.2) is assumed by choosing a minimum sample size of 500 measurements, i.e. a sampling time of $20\,\mathrm{s}$. The compared samples vary in size; thus, variance homogeneity is not given and the Welch test must be used.

The needed parameters to carry out a t-test (in this case, a TOST) are defined beforehand. The null hypothesis states that the velocity means of the two samples do not pass as equivalent regarding to the chosen equivalence margin. The alternate hypothesis states that the velocity means of the two samples pass as equivalent under consideration of the chosen equivalence margin (see figure 2.2). The standard significance level of $\alpha = 0.05$ is chosen. Hence, for each one-tailed test in the TOST, the significance level is $\alpha = 0.025$. After calculating the test statistics, the TOST factors in the greater p-value of the two one-tailed tests.

7.3. Comparison of Means

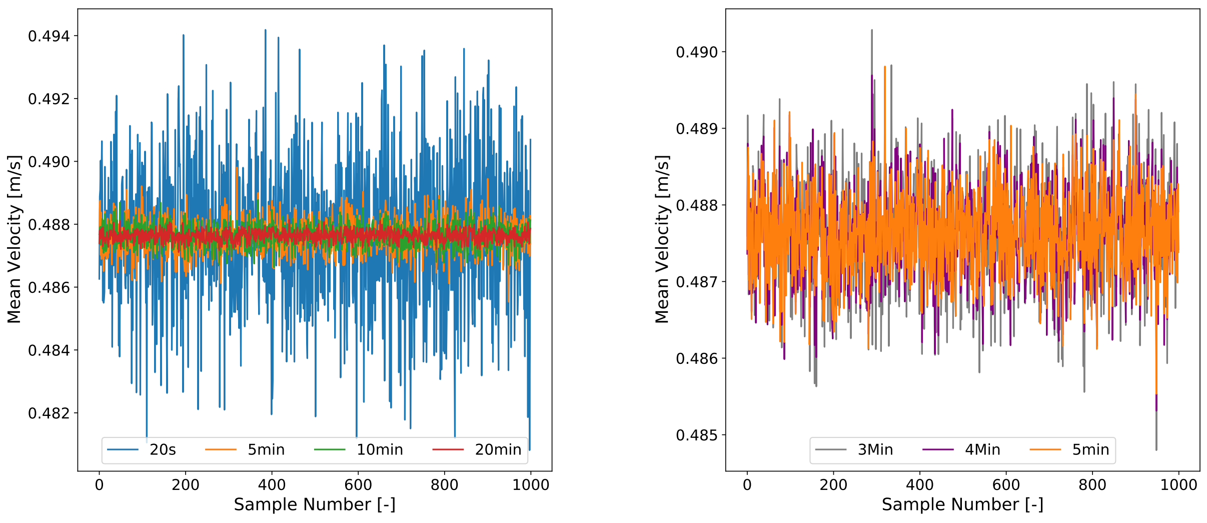
After the prior considerations, the comparison of sample means for varying sampling times is implemented in Python to show convergence toward a constant value when prolonging sampling times and gain some insight into the data at hand for choosing an appropriate margin of equivalence δ.

As a first step, the original data files are converted to comma-separated values (.csv files). The method ”get_population_df” (see source code A.36) is written to read the created .csv files by inputting their file paths, strip unnecessary data, calculate the velocity magnitude using the three recorded velocity vectors and generate a data frame containing an index and the velocity magnitude. The output data frame serves as the population for further statistical operations.

Then, the method ”get_samples” (see source code A.37) is created to get a set number of random but reproducible samples from the population data frame with the desired minimum number of measurements. Reproducibility of samples is ensured by using the option ”random_state” within the ”sample” function. The method ”extend_samples” (see source code A.38) is used to extend the previously created samples in set increments by a set number of measurements for an established number of steps. This method creates a nested list ”list_of_samples” specifying the steps and all the created samples for each step. A similar method, ”calculate_means” (see source code A.39), calculates the means of all the samples in the nested list and creates a new nested list, ”list_of_means”, which includes the steps and all the sample means for each step. This nested list can now be used to plot sample means for different sampling times.

For this analysis, the script (A.45) is executed with the input parameters as specified. Using the ”list_of_means”, it is possible to plot the mean of a specific sample for each step, i.e. each sampling time. The following figures show the comparison of means for the long-duration measurement ”0.4_center” (the measurement expected to be the most accurate). While the first figure 7.1a shows an overall comparison of means for varying sampling times, the next figure 7.1b only shows the sampling times given as usual recommendations for measurement devices, namely sampling times between $3\,\text{min}$ and $5\,\text{min}$. The comparison of means for all six long-duration ADV measurements can be found in the annex (see figures A.35 to A.39).

From the overall comparison of means, it can be deduced that, as expected, the sample means converge towards a value with increasing sampling time. Then, the resolution for the plot is adjusted so that it only shows sample means for the sampling times of interest (see figure 7.1b). Nevertheless, merely by visually assessing this plot, it is impossible to grasp the degree of improvement of data between timesteps in a substantiated manner. Consequently, a basic comparison of means gives a general idea of how the accuracy of sample means improves when increasing sampling times on a large scale, but it is not fit to facilitate educated decisions when refining and optimising sampling times. As a result, the TOST is performed employing two different equivalence margins of $\delta = 0.004$ and $\delta = 0.006$, which were chosen based on the comparison of means.



(a) Comparison of means for sampling times up to $20\,\mathrm{min}$.

(b) Comparison of means for recommended sampling times.

Figure 7.1.: Comparison of means for the long-duration measurement in central horizontal position and vertical position of $0.4\,\mathrm{m}$ over bottom for sampling times up to $20\,\mathrm{min}$ and for recommended sampling times.

7.4. Testing for Equivalence of Means

The TOST is implemented in Python using an already existing method for calculating test statistics and p-values. The method "weightstats.ttost_ind" is part of the statsmodels package (Seabold & Perktold, 2010). It offers testing for equivalence of two independent samples with various settings and specifications.

Reading and processing the original data is executed the same way as for the comparison of means using the method "get_population_df" (see source code A.36). For the TOST, two samples are needed to compare their respective sample means. The base sample always contains the maximum number of measurements of 30000, i.e. $20\,\mathrm{min}$. It is created with the method "get_base_sample" (see source code A.40). Iterating the script makes it possible to compare the same samples to different base samples.

To get samples of varying sampling times, the previously explained methods "get_samples" (see source code A.37) and "extend_samples" (see source code A.38) are adopted. As processing the script is rather computationally intensive and the analysis is performed with a private computer of limited computational capacity, a workaround to get more discriminable samples without increasing the number of samples is implemented. This workaround is realised by shuffling the population and kept reproducible by once again defining a ”random_state=random_-state_population” within the ”sample” function. The method is called ”shuffle_population” (see source code A.41) and ensures that the first 100 samples in the data frame are different for each set ”random_state_population”. This solution is disadvantageous when performing the analysis with more than 100 samples because the user must alter the ”random_state_-population”, but it prevents the script from crashing, e.g. because the computational resources are depleted.

The methods ”calculate_pvalues” (see source code A.42) works analogously to the method ”calculate_means” (see source code A.39) in the comparison of means but uses the above mentioned method ”weightstats.ttost_ind” from the statsmodels package to calculate the p-value instead of the sample mean. The setting ”usevar=’unequal’” is specified to employ the Welch test for unequal sample sizes. The resulting p-values are sorted by step, i.e. by sampling time. For further analysis, it is preferred that the p-values are sorted by sample instead. This is realised with the method ”sort_pvalues_by_sample” (see source code A.43).

Lastly, the method ”count_accept_H0” (see source code A.44) counts all p-values for each sample at each sampling time that are greater than the set significance level , which represents the acceptance of the null hypothesis. An acceptance of the null hypothesis means that the two samples in the TOST are not equivalent at an equivalence margin.

For the analysis, the script ”TOST” (see source code A.46) is performed with the parameters as specified. Ten script iterations are undertaken with the indicated 20 random population states at 100 samples each and for two different equivalence margins. For each base sample, the average acceptance rate per time step in percent (number of H_0 acceptance per 100 discriminable samples per ”random_state_population”), which corresponds to 2000 samples in total, is calculated and then visualised. The visualisation script (see source code A.47) is separated from the analysis script for practical reasons. The visualised results for all long-duration ADV measurements (see figures A.42 - A.47), as well as the acceptance rate data can be found in the annex (see tables A.4 - A.11).

As expected, the acceptance rates of the null hypotheses plummet with increasing sampling times (see figure 7.2). Also, it is clear that increasing the equivalence margin makes more samples equivalent at shorter sampling times. The data is analysed for when the acceptance rate of the hypothesis falls below $5\,\%$, which means that only $5\,\%$ of the samples do not comply with the chosen equivalence margin for equivalence in means. For $\delta = 0.004$ the optimal sampling times correspond to $7\,\text{min}$ to $10\,\text{min}$, and for $\delta = 0.006$ the optimal sampling times correspond to $3\,\text{min}$ to $5\,\text{min}$ (see table A.14 in the annex).

This analysis also points out that the needed sampling times to produce significant samples are generally higher for the measurements recorded in proximity to the wall and the bottom of the flume than for the measurements away from the wall and further into the water body. Also, note that the equivalence margins are chosen very conservatively. This setting implies that for applications that concur with higher equivalence margins, the optimal sampling times are even lower than the ones presented for $\delta = 0.006$.

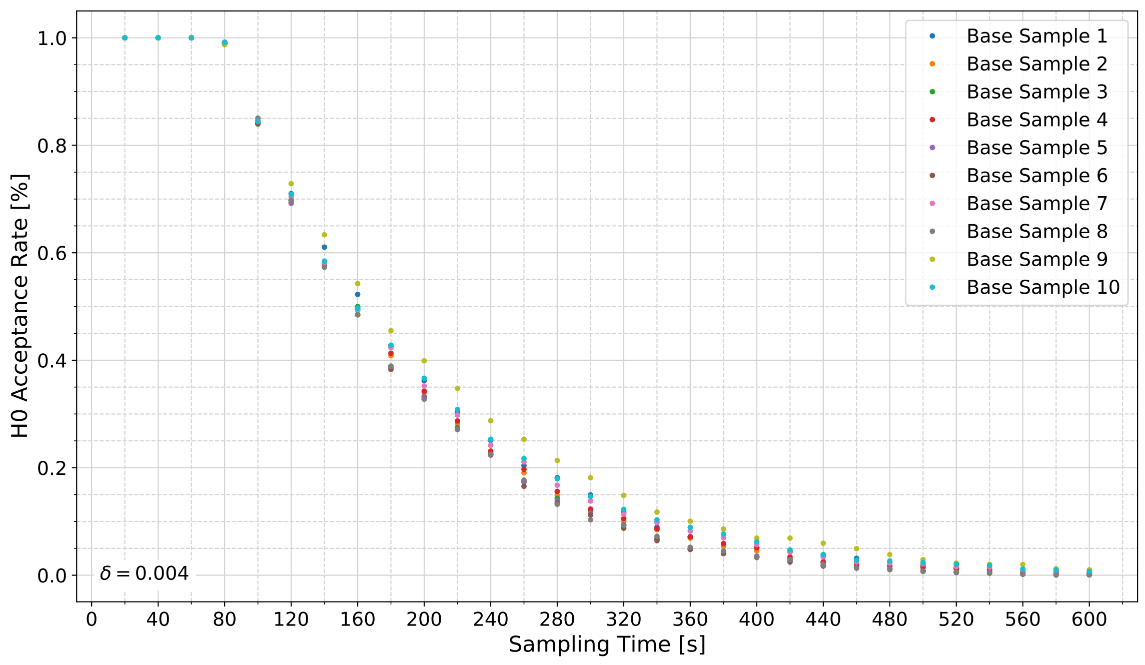
The following subsection 7.5 highlights the possible implications of optimising sampling times in hydraulic experiments and general laboratory practice.

7.5. Discussion of Results

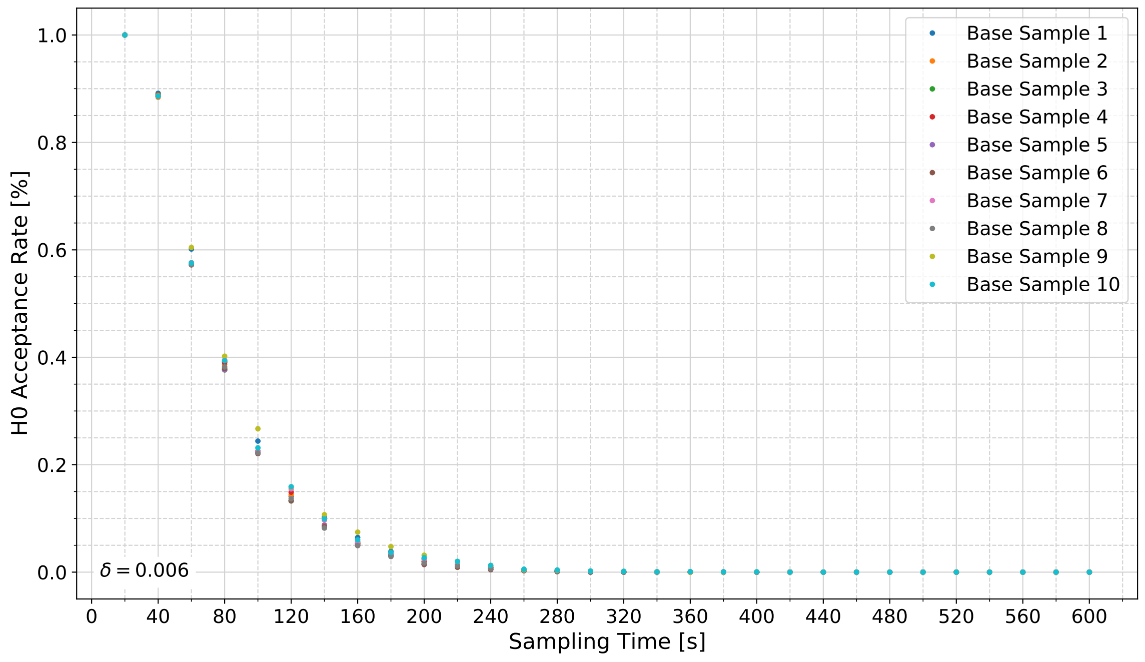
When attempting to find an optimal sampling time for ADV measurements, the primary goal is to save resources. So, every effort undertaken to find the optimal sampling time for a planned hydraulic experiment must be traded off against the potential saving of resources realised by optimisation. For example, shaving off several minutes of measuring time via optimisation to realise measurements in a large measurement grid with many measurement points might be worthwhile, while performing a time-consuming analysis on measurement times when only recording a few measurements can be a waste of resources. Apart from more resource efficiency, the optimisation approach also ensures a level of statistical significance which enables making educated decisions on sampling times and increases awareness of uncertainty in the measurements among measurement device users.

Several samples at different measurement points are analysed to develop the optimisation approach for hydraulic data measurements. The process is retaken for many different samples to ensure reproducibility of results for different cases and test the robustness of the approach. When applying the optimisation approach to an actual project, one would only record one long-duration measurement, ideally at the measuring point expected to show the highest uncertainty in measurement. Also, hypothesis testing would include much fewer samples to minimise computation times. Furthermore, the long-duration measurements were not processed before optimising when developing this approach. This is a quite bidirectional point to consider: on the one hand, one wants to produce significant samples independent of possible uncertainties but on the other hand, optimising the sampling time after processing promises to decrease the sampling time even further. However, after some thought and performing the statistical evaluation in chapter 6, it appears more logical to use filtered samples in future sampling time optimisations using hypothesis testing. By doing this, one can be sure that higher statistical moments agree with general recommendations and that the samples are not biased regarding the statistical distribution of the data.

Aside from the effectiveness of the sampling time optimisation approach, it is important to stress that the process needs to be adapted to every new experiment setup. Correspondingly, several circumstances allow an advantageous use of the approach, like a precise research question, details on the measurement campaign and knowledge of the device and its accuracy. Moreover, the researcher should evaluate the time needed to understand and execute the



(a) Null hypothesis acceptance rates for the long-duration measurement in central horizontal position and vertical position of $0.4\,\mathrm{m}$ over bottom at $\delta = 0.004$.



(b) Null hypothesis acceptance rates for the long-duration measurement in central horizontal position and vertical position of $0.4\,\mathrm{m}$ over bottom at $\delta = 0.006$.

Figure 7.2.: Comparison of null hypothesis acceptance rates for the long-duration measurement in central horizontal position and vertical position of $0.4\,\mathrm{m}$ over bottom at two equivalence levels.

method. A requirement that definitely must be met is the independence of measurements. All these considerations facilitate the choice of fitting input parameters for the optimisation process and help assess whether the optimisation is genuinely essential and profitable.

Several general statements can be deduced from the findings presented in the previous subsections:

- Optimising the sampling time regarding to statistical parameters is possible via hypothesis testing.
- Optimisation of the sampling time is only genuinely advantageous if high confidence in the data is needed or if the collection of data is extensive.
- For the data at hand, the sampling time recommendations in the literature of 3 min to 5 min correspond to very small equivalence margins.
- This approach only recognises equivalence in means. An optimisation that uses higher statistical moments could result in longer optimal sampling times.

8. Conclusion and Outlook

Initiated by a lack of practical, efficient, transparent and open-source tools to reduce uncertainty from ADV measurements, this thesis reviews ADV data analysis approaches. While many data analysis methods seem worthwhile, some apply complicated mathematical operations and theories and require good mathematical understanding and programming proficiency to transfer the abstract concepts into working Python code. For this thesis, three methods (one despiking method and two denoising methods) were implemented in the programming language Python. The processing of 160 samples of ADV measurements in turbulent flow shows that the overall most successful combination of data analysis methods implemented and evaluated here is KDE despiking in combination with Butterworth filtering. The implementations are now available for other velocity measurements using ADV devices.

A despiking method that was implemented in Python and published in 2021 - the P-SAT framework - could not be employed successfully. Either the method does not work as intended, or the code is used incorrectly. Although the results from the P-SAT framework are severely flawed in a statistical sense, the produced spectra show potential. Therefore, I deem it profitable to find the mistakes in use or in code that lead to the disadvantageous statistical parameters, to provide an alternative despiking method in the Python language for future projects with the ADV.

Furthermore, the calculation of statistical parameters for ADV measurements is implemented by creating a respective script in Python. This implementation facilitates calculating relevant statistical parameters for reporting in compliance with the GUM. Furthermore, it is found that data evaluation by statistical parameters is overall a fit approach to detect uncertainties in ADV data, but for turbulent flow, spectra should be evaluated additionally. In this thesis, spectra are only evaluated visually by manual screening. A Python implementation can be realised to improve this evaluation so that it automatically evaluates spectra regarding the degree to which they follow the theoretical Kolmogorov spectrum. For example, this implementation can be realised by comparing function properties or by utilising an image data processing approach. For more information on automated detection of the $-\frac{5}{3}$ slope in the inertial subrange see e.g. Ortiz-Suslow et al., 2020. All in all, processed data should always be evaluated to ensure that the methods work reliably and to work in agreement with good scientific practice.

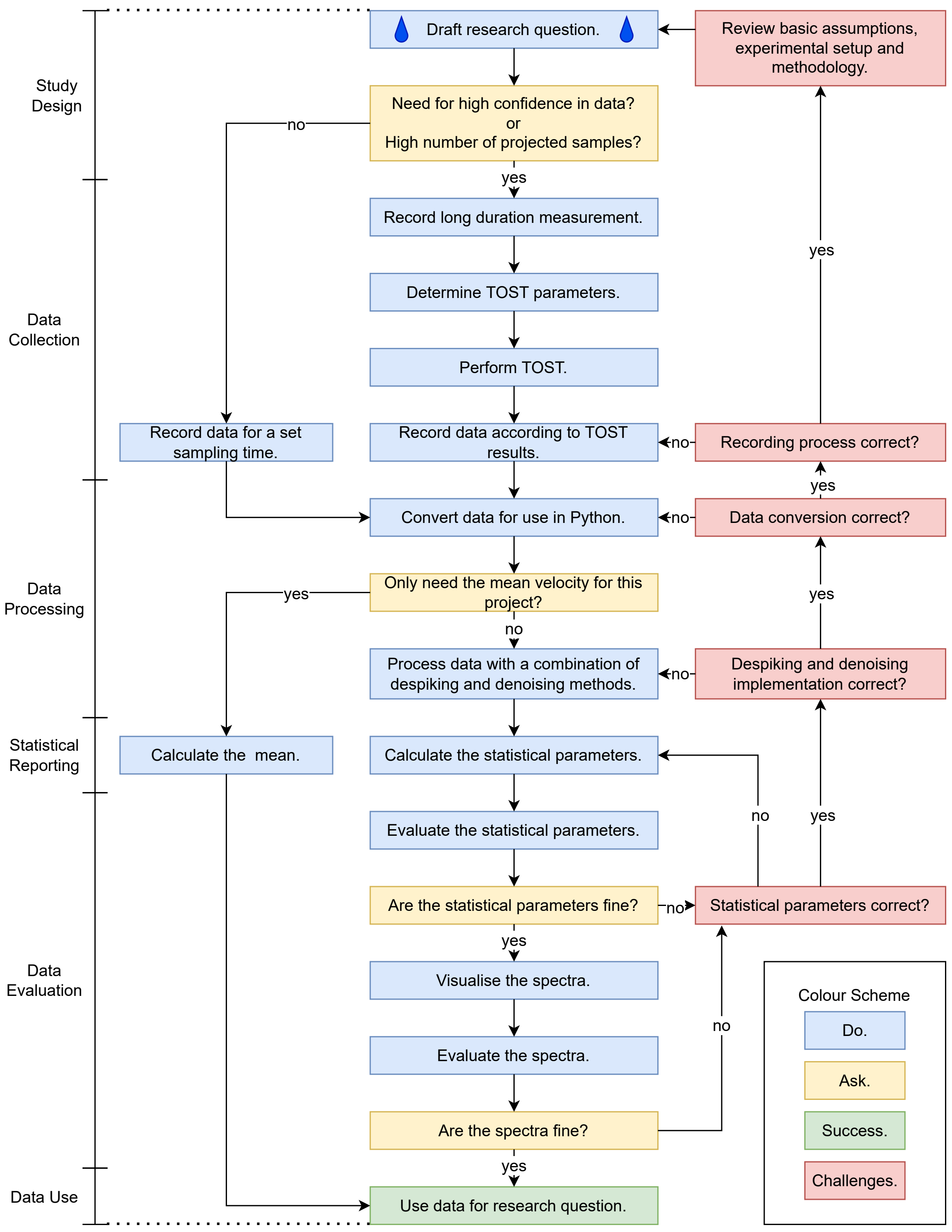
The TOST for equivalence in means is explored in this thesis to find an optimal sampling time for ADV measurements and improve the data collection process. Overall, the attempt is successful, and the method can be used to devise optimal sampling times regarding a priori set quality levels. When using the method, I recommend a minimum long time duration measurement of 30 min (for the Vectrino at a sampling frequency of 25 Hz, i.e. 45000 measurement points). Computing the null hypothesis acceptance rates can get quite time-consuming due to the many iterations. Therefore, I recommend testing with ten different base samples ("iterations_of_script=10") and only two values for "random_state_population". By doing this, it can be ensured that the introduced redundancy can detect mistakes.

I suggest testing the TOST procedure for more flow conditions and original and filtered data to improve the method further. Also, at the moment, only the first statistical moment is considered, which in the future can be extended by trying to equivalence test for goodness-of-fit with the χ-square-test (Staudte, 2020). Also, alternative tests (such as the Mann-Whitney-U test for the mean) can be implemented so that the test results can be tested for coherence. The method is also transferable to other devices.

An in-depth study on optimal sampling times might be worthwhile to give general recommendations for future projects. The study could examine long-duration measurements for various flow conditions using multiple devices frequently used in the hydraulic laboratory. Applying the TOST with several equivalence and confidence levels can produce data to assess the optimal sampling time for many settings.

Lastly, a flow chart (see figure 8.1) is proposed that incorporates the findings of this thesis into a systematic approach to working with data from ADV devices in the hydraulic laboratory. The left side of the flow chart divides the diagram into critical project phases concerning ADV data. The work flow starts in the rectangle with the two droplets. By following the arrows and answering the questions that arise along the way, essential steps to ADV data handling are respected. The colours in the flow chart represent the nature of the step that is represented by the respective rectangle. The right side of the flow chart integrates a fail safe into the procedure so that eventual problems are solved systematically. As the flow chart mainly includes actions and questions tangential to topics from this thesis, I am sure that there is still a lot to add to this systematic approach and therefore make no claim that the flow chart is exhaustive.

Altogether, this thesis shows that Python as a programming language is very well suited for hydraulic data analysis, even for users without a plethora of previous programming experience. This is due to much information and many learning platforms available for free. Considering this, I find it surprising to find only a few specialised, open-source tools for hydraulics or water engineering questions in this language. I believe it could be rewarding to work towards specialised open-source tools in the hydraulics and water engineering field.



Study Design
Data Collection
Data Processing
Statistical Reporting
Data Evaluation
Data Use
Draft research question.
Review basic assumptions, experimental setup and methodology.
Need for high confidence in data? or High number of projected samples?
no
yes
Record long duration measurement.
Determine TOST parameters.
Perform TOST.
Record data for a set sampling time.
Record data according to TOST results.
Recording process correct?
Convert data for use in Python.
Data conversion correct?
Only need the mean velocity for this project?
Process data with a combination of despiking and denoising methods.
Despiking and denoising implementation correct?
Calculate the mean.
Calculate the statistical parameters.
Evaluate the statistical parameters.
Are the statistical parameters fine?
Statistical parameters correct?
Visualise the spectra.
Evaluate the spectra.
Are the spectra fine?
Use data for research question.
Colour Scheme
Do.
Ask.
Success.
Challenges.

Figure 8.1.: Proposal for a systematic approach to working with data from ADV devices in the hydraulic laboratory.

References

Adam, B., & Lehmann, B. (2011). *Ethohydraulik*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-17210-6

Agarwal, M., Deshpande, V., Katoshevski, D., & Kumar, B. (2021). A novel Python module for statistical analysis of turbulence (P-SAT) in geophysical flows. *Scientific reports*, *11*(1), 3998. https://doi.org/10.1038/s41598-021-83212-1

Alessio, S. M. (2016). *Digital Signal Processing and Spectral Analysis for Scientists: Concepts and Applications* (1st ed. 2016). Springer. https://doi.org/10.1007/978-3-319-25468-5

Anaconda. (2021). Anaconda Software Distribution. Retrieved May 21, 2022, from https://anaconda.com/

Avila, K., Moxey, D., de Lozar, A., Avila, M., Barkley, D., & Hof, B. (2011). The onset of turbulence in pipe flow. *Science (New York, N.Y.)*, *333*(6039), 192–196. https://doi.org/10.1126/science.1203223

Bailly, C., & Comte-Bellot, G. (2015). *Turbulence*. Springer.

Biron, P. (1997). Errata. *Mathematical Geology*, *29*(5), 725. https://doi.org/10.1007/BF02769653

Biron, P., Roy, A. G., & Best, J. L. (1995). A scheme for resampling, filtering, and subsampling unevenly spaced laser Doppler anemometer data. *Mathematical Geology*, *27*(6), 731–748. https://doi.org/10.1007/BF02273535

Bollrich, G. (2013). *Technische Hydromechanik* (7. Auflage). Beuth Verlag GmbH.

Bonamente, M. (2013). *Statistics and Analysis of Scientific Data*. Springer New York. https://doi.org/10.1007/978-1-4614-7984-0

Botev, Z. I., Grotowski, J. F., & Kroese, D. P. (2010). Kernel density estimation via diffusion. *The Annals of Statistics*, *38*(5). https://doi.org/10.1214/10-AOS799

Carbonneau, P. E., & Bergeron, N. E. (2000). The effect of bedload transport on mean and turbulent flow properties. *Geomorphology*, *35*(3-4), 267–278. https://doi.org/10.1016/S0169-555X(00)00046-5

Cea, L., Puertas, J., & Pena, L. (2007). Velocity measurements on highly turbulent free surface flow using ADV. *Experiments in Fluids*, *42*(3), 333–348. https://doi.org/10.1007/s00348-006-0237-3

Chanson, H., Trevethan, M., & Koch, C. (2007). Discussion of “Turbulence Measurements with Acoustic Doppler Velocimeters” by Carlos M. García, Mariano I. Cantero, Yarko Niño, and Marcelo H. García. *Journal of Hydraulic Engineering*, *133*(11), 1283–1286. https://doi.org/10.1061/(ASCE)0733-9429(2007)133:11(1283)

Chanson, H., & Larrarte, F. (Eds.). (2008). *Acoustic Doppler Velocimetry in the Field and in Laboratory: Practical Experiences* (Vol. CH70/08).

Chanson, H., Trevethan, M., & Aoki, S.-i. (2008). Acoustic Doppler velocimetry (ADV) in small estuary: Field experience and signal post-processing. *Flow Measurement and Instrumentation*, *19*(5), 307–313. https://doi.org/10.1016/j.flowmeasinst.2008.03.003

Day, M. A. (1990). The no-slip condition of fluid dynamics. *Erkenntnis*, *33*(3), 285–296. https://doi.org/10.1007/BF00717588

Díaz Lozada, J. M., García, C. M., Scacchi, G., & Oberg, K. A. (2021). Dynamic Selection of Exposure Time for Turbulent Flow Measurements. *Journal of Hydraulic Engineering*, *147*(10), 04021035. https://doi.org/10.1061/(ASCE)HY.1943-7900.0001922

Dilling, S., & MacVicar, B. J. (2017). Cleaning high-frequency velocity profile data with autoregressive moving average (ARMA) models. *Flow Measurement and Instrumentation*, *54*, 68–81. https://doi.org/10.1016/j.flowmeasinst.2016.12.005

Dombroski, D. E., & Crimaldi, J. P. (2007). The accuracy of acoustic Doppler velocimetry measurements in turbulent boundary layer flows over a smooth bed. *Limnology and Oceanography: Methods*, *5*(1), 23–33. https://doi.org/10.4319/lom.2007.5.23

Donoho, D. L. (1995). De-noising by soft-thresholding. *IEEE Transactions on Information Theory*, *41*(3), 613–627. https://doi.org/10.1109/18.382009

Doroudian, B., Bagherimiyab, F., & Lemmin, U. (2010). Improving the accuracy of four-receiver acoustic Doppler velocimeter (ADV) measurements in turbulent boundary layer flows. *Limnology and Oceanography: Methods*, *8*(11), 575–591. https://doi.org/10.4319/lom.2010.8.0575

Durgesh, V., Thomson, J., Richmond, M. C., & Polagye, B. L. (2014). Noise correction of turbulent spectra obtained from acoustic doppler velocimeters. *Flow Measurement and Instrumentation*, *37*, 29–41. https://doi.org/10.1016/j.flowmeasinst.2014.03.001

Frost, I. (2017). *Statistische Testverfahren, Signifikanz und p-Werte: Allgemeine Prinzipien verstehen und Ergebnisse angemessen interpretieren*. Springer VS. https://doi.org/10.1007/978-3-658-16258-0

Gazi, O. (2018). *Understanding Digital Signal Processing* (Vol. 13). Springer Singapore. https://doi.org/10.1007/978-981-10-4962-0

Goring, D. G., & Nikora, V. I. (2002). Despiking Acoustic Doppler Velocimeter Data. *Journal of Hydraulic Engineering*, *128*(1), 117–126. https://doi.org/10.1061/(ASCE)0733-9429(2002)128:1(117)

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., Del Río, J. F., Wiebe, M., Peterson, P., …

Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Harten, U. (2021). *Physik*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-61698-7

Haslwanter, T. (2021). *Hands-on Signal Analysis with Python*. Springer International Publishing. https://doi.org/10.1007/978-3-030-57903-6

Hejazi, K., Falconer, R. A., & Seifi, E. (2016). Denoising and despiking ADV velocity and salinity concentration data in turbulent stratified flows. *Flow Measurement and Instrumentation*, *52*, 83–91. https://doi.org/10.1016/j.flowmeasinst.2016.09.010

Hering, E., Martin, R., & Stohrer, M. (2016). *Physik für Ingenieure*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-49355-7

Huang, C. J., Ma, H., Guo, J., Dai, D., & Qiao, F. (2018). Calculation of turbulent dissipation rate with acoustic Doppler velocimeter. *Limnology and Oceanography: Methods*, *16*(5), 265–272. https://doi.org/10.1002/lom3.10243

Huang, C., Qiao, F., & Ma, H. (2020). Noise reduction of acoustic Doppler velocimeter data based on Kalman filtering and autoregressive moving average models. *Acta Oceanologica Sinica*, *39*(12), 106–113. https://doi.org/10.1007/s13131-020-1641-x

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Hurther, D., & Lemmin, U. (2008). Improved Turbulence Profiling with Field-Adapted Acoustic Doppler Velocimeters Using a Bifrequency Doppler Noise Suppression Method. *Journal of Atmospheric and Oceanic Technology*, *25*(3), 452–463. https://doi.org/10.1175/2007JTECHO512.1

Islam, M. R., & Zhu, D. Z. (2013). Kernel Density–Based Algorithm for Despiking ADV Data. *Journal of Hydraulic Engineering*, *139*(7), 785–793. https://doi.org/10.1061/(ASCE)HY.1943-7900.0000734

JCGM. (2008a). Evaluation of measurement data - Supplement 1 to the “Guide to the expression of uncertainty in measurement” - Propagation of distributions using a Monte Carlo method (101:2008). Retrieved April 18, 2022, from https://www.bipm.org/documents/20126/2071204/JCGM_101_2008_E.pdf/325dcaad-c15a-407c-1105-8b7f322d651c

JCGM. (2008b). Evaluation of measurement data — Guide to the expression of uncertainty in measurement (100:2008). Retrieved December 2, 2021, from https://www.bipm.org/documents/20126/2071204/JCGM_100_2008_E.pdf/cb0ef43f-baa5-11cf-3f85-4dcd86f77bd6

Jesson, M., Sterling, M., & Bridgeman, J. (2013). Despiking velocity time-series—Optimisation through the combination of spike detection and replacement methods. *Flow Measurement and Instrumentation*, *30*, 45–51. https://doi.org/10.1016/j.flowmeasinst.2013.01.007

Juzek, T. S., & Kizach, J. (2019). How to Set Delta in the Two-One-Sided T-tests Procedure (TOST). *Journal of Research Design and Statistics in Linguistics and Communication Science, 5*(1-2), 153–169. https://doi.org/10.1558/jrds.39002

Kolmogorov, A. N. (1991). The local structure of turbulence in incompressible viscous fluid for very large Reynolds numbers. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, *434*(1890), 9–13. https://doi.org/10.1098/rspa.1991.0075

Köse, Ö. (2013). Various filtering algorithms used to eliminate outliers in velocity time series obtained by ADVs (acoustic Doppler velocimeter). *Arabian Journal of Geosciences*, *6*(7), 2691–2697. https://doi.org/10.1007/s12517-012-0523-8

Lane, S. N., Biron, P. M., Bradbrook, K. F., Butler, J. B., Chandler, J. H., Crowell, M. D., McLelland, S. J., Richards, K. S., & Roy, A. G. (1998). Three-dimensional measurement of river channel flow processes using acoustic doppler velocimetry. *Earth Surface Processes and Landforms*, *23*(13), 1247–1267. https://doi.org/10.1002/(SICI)1096-9837(199812)23:13{\textless}1247::AID-ESP930{\textgreater}3.0.CO;2-D

Lange, T., & Mosler, K. (2017). *Statistik kompakt*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-53467-0

Lemmin, U., Lhermitte, R., Nikora, V. I., & Goring, D. G. (1999). ADV Measurements of Turbulence: Can We Improve Their Interpretation? *Journal of Hydraulic Engineering*, *125*(9), 987–988. https://doi.org/10.1061/(ASCE)0733-9429(1999)125:9(987)

Li, Y., & Meneveau, C. (2005). Origin of non-Gaussian statistics in hydrodynamic turbulence. *Physical review letters*, *95*(16), 164502. https://doi.org/10.1103/PhysRevLett.95.164502

MacVicar, B. J., Beaulieu, E., Champagne, V., & Roy, A. G. (2007). Measuring water velocity in highly turbulent flows: field tests of an electromagnetic current meter (ECM) and an acoustic Doppler velocimeter (ADV). *Earth Surface Processes and Landforms*, *32*(9), 1412–1432. https://doi.org/10.1002/esp.1497

MacVicar, B., Dilling, S., & Lacey, J. (2014). Multi-instrument turbulence toolbox (MITT): Open-source MATLAB algorithms for the analysis of high-frequency flow velocity time series datasets. *Computers & Geosciences*, *73*, 88–98. https://doi.org/10.1016/j.cageo.2014.09.002

Meyer, M. (2021). *Signalverarbeitung*. Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-32801-6

Morgenschweis, G. (2018). *Hydrometrie: Theorie und Praxis der Durchflussmessung in offenen Gerinnen* (2. Auflage). Springer Vieweg. https://doi.org/10.1007/978-3-662-55314-5

Mori, N., Suzuki, T., & Kakuno, S. (2007). Noise of Acoustic Doppler Velocimeter Data in Bubbly Flows. *Journal of Engineering Mechanics*, *133*(1), 122–125. https://doi.org/10.1061/(ASCE)0733-9399(2007)133:1(122)

Mrtz. (2004). Aliasing. Retrieved May 26, 2022, from https://commons.wikimedia.org/wiki/File:Aliasing_mrtz.svg

Nikora, V., & Goring, D. (2000). Flow Turbulence over Fixed and Weakly Mobile Gravel Beds. *Journal of Hydraulic Engineering*, *126*(9), 679–690. https://doi.org/10.1061/(ASCE)0733-9429(2000)126:9(679)

Nikora, V. I., & Goring, D. G. (1998). ADV Measurements of Turbulence: Can We Improve Their Interpretation? *Journal of Hydraulic Engineering*, *124*(6), 630–634. https://doi.org/10.1061/(ASCE)0733-9429(1998)124:6(630)

Nortek. (2004). Vectrino Velocimeter User Guide.

Nortek. (2021). The comprehensive Manual for Velocimeters: Vector | Vectrino | Vectrino Profiler.

Oertel, H. (2017). *Prandtl - Führer durch die Strömungslehre*. Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-08627-5

Oertel, H., Böhle, M., & Reviol, T. (2015). *Strömungsmechanik*. Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-07786-0

Ortiz-Suslow, D. G., Wang, Q., Kalogiros, J., & Yamaguchi, R. (2020). A Method for Identifying Kolmogorov's Inertial Subrange in the Velocity Variance Spectrum. *Journal of Atmospheric and Oceanic Technology*, *37*(1), 85–102. https://doi.org/10.1175/JTECH-D-19-0028.1

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Müller, A., Nothman, J., Louppe, G., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. Retrieved March 28, 2022, from https://scikit-learn.org/stable/modules/density.html#kernel-density

Piaggio, G., Elbourne, D. R., Altman, D. G., Pocock, S. J., & Evans, S. J. W. (2006). Reporting of noninferiority and equivalence randomized trials: an extension of the CONSORT statement. *JAMA*, *295*(10), 1152–1160. https://doi.org/10.1001/jama.295.10.1152

Poindexter, C. M., Rusello, P. J., & Variano, E. A. (2011). Acoustic Doppler velocimeter-induced acoustic streaming and its implications for measurement. *Experiments in Fluids*, *50*(5), 1429–1442. https://doi.org/10.1007/s00348-010-1001-2

Precht, E., Janssen, F., & Huettel, M. (2006). Near-bottom performance of the Acoustic Doppler Velocimeter (ADV) – a comparative study. *Aquatic Ecology*, *40*(4), 481–492. https://doi.org/10.1007/s10452-004-8059-y

Puthusserypady, S. (2021). *Applied Signal Processing*. Now Publishers. https://doi.org/10.1561/9781680839791

Python Software Foundation. (2020). Python. Retrieved May 21, 2022, from https://www.python.org/

Razaz, M., & Kawanisi, K. (2011). Signal post-processing for acoustic velocimeters: detecting and replacing spikes. *Measurement Science and Technology*, *22*(12), 125404. https://doi.org/10.1088/0957-0233/22/12/125404

Reynolds, A. J. (1974). *Turbulent flows in engineering*. Wiley.

Richard, J.-B., Thomson, J., Polagye, B., & Bard, J. (2013). Method for identification of Doppler noise levels in turbulent flow measurements dedicated to tidal energy. *International Journal of Marine Energy*, *3-4*, 52–64. https://doi.org/10.1016/j.ijome.2013.11.005

Roget, E., Lozovatsky, I., Sanchez, X., & Figueroa, M. (2006). Microstructure measurements in natural waters: Methodology and applications. *Progress in Oceanography*, *70*(2-4), 126–148. https://doi.org/10.1016/j.pocean.2006.07.003

Rose, E. M., Mathew, T., Coss, D. A., Lohr, B., & Omland, K. E. (2018). A new statistical method to test equivalence: an application in male and female eastern bluebird song. *Animal Behaviour*, *145*, 77–85. https://doi.org/10.1016/j.anbehav.2018.09.004

Ross, S. M. (2010). *Introductory statistics* (3. ed.). Academic Press. http://www.sciencedirect.com

Roy, A. G., Biron, P. M., & Lapointe, M. F. (1997). Implications of low-pass filtering on power spectra and autocorrelation functions of turbulent velocity signals. *Mathematical Geology*, *29*(5), 653–668. https://doi.org/10.1007/BF02769649

Rusello, P. J. (2009). A Practical Primer for Pulse Coherent Instruments (Nortek, Ed.).

Schiefer, H., & Schiefer, F. (2021). *Statistics for Engineers*. Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-32397-4

Schlichting, H., & Gersten, K. (2006). *Grenzschicht-Theorie: Mit 22 Tabellen* (10., überarbeitete Auflage). Springer.

Schuirmann, D. J. (1987). A comparison of the two one-sided tests procedure and the power approach for assessing the equivalence of average bioavailability. *Journal of pharmacokinetics and biopharmaceutics*, *15*(6), 657–680. https://doi.org/10.1007/BF01068419

Seabold, S., & Perktold, J. (2010). statsmodels: Econometric and statistical modeling with python. *9th Python in Science Conference*.

Shahmoradi, A., Bagheri, F., & Osborne, J. A. e. (2020). Fast fully-reproducible serial/parallel Monte Carlo and MCMC simulations and visualizations via ParaMonte::Python library. *arXiv e-prints*, arXiv:2010.00724.

Sharma, A., Maddirala, A. K., & Kumar, B. (2018). Modified singular spectrum analysis for despiking acoustic Doppler velocimeter (ADV) data. *Measurement*, *117*, 339–346. https://doi.org/10.1016/j.measurement.2017.12.025

She, Z.-S. (1991). Intermittency and non-gaussian statistics in turbulence. *Fluid Dynamics Research*, *8*(1-4), 143–158. https://doi.org/10.1016/0169-5983(91)90039-L

Siebert, M., & Ellenberger, D. (2020). Validation of automatic passenger counting: introducing the t-test-induced equivalence test. *Transportation*, *47*(6), 3031–3045. https://doi.org/10.1007/s11116-019-09991-9

Skiadas, C. (Ed.). (2016). *The Foundations of Chaos Revisited: From Poincaré to Recent Advancements: The Kolmogorov Law of turbulence, What can rigorously be proved ? Part II*. Springer. https://doi.org/10.1007/978-3-319-29701-9

Sokoray-Varga B., & Höger V. (2014). *Messungen mit dem Vectrino. BAW Empfehlung. Ausgabe 2014*. BAW.

Song, H. (2018). *Engineering Fluid Mechanics*. Springer Singapore. https://doi.org/10.1007/978-981-13-0173-5

SonTek. (2011). RiverSurveyor S5/M9 system manual: Firmware Version 2.00 (SonTek, Ed.).

Spurk, J. H., & Aksel, N. (2020). *Fluid Mechanics*. Springer International Publishing. https://doi.org/10.1007/978-3-030-30259-7

Staudte, R. G. (2020). Evidence for goodness of fit in Karl Pearson chi-squared statistics. *Statistics*, *54*(6), 1287–1310. https://doi.org/10.1080/02331888.2020.1862115

Stein, J. Y. (2000). *Digital signal processing: A computer science perspective*. Wiley. http://swbplus.bsz-bw.de/bsz088772233cov.htm

Strom, K. B., & Papanicolaou, A. N. (2007). ADV Measurements around a Cluster Microform in a Shallow Mountain Stream. *Journal of Hydraulic Engineering*, *133*(12), 1379–1389. https://doi.org/10.1061/(ASCE)0733-9429(2007)133:12(1379)

Sundararajan, D. (2021). *Digital Signal Processing: An Introduction* (1st ed. 2021). Springer International Publishing; Imprint Springer. https://doi.org/10.1007/978-3-030-62368-5

Tennekes, H., & Lumley, J. L. (1972). *A first course in turbulence*. MIT Press. http://mitpress.mit.edu/books/first-course-turbulence

Teolis, A. (2017). *Computational Signal Processing with Wavelets*. Birkhäuser. https://doi.org/10.1007/978-3-319-65747-9

The pandas development team. (2022). Pandas. https://doi.org/10.5281/zenodo.3509134

Tsinober, A. (2009). *An informal conceptual introduction to turbulence: Second edition of "An informal introduction to turbulence"* (2. ed., Vol. 92). Springer. http://www.loc.gov/catdir/enhancements/fy1202/2009930633-d.html

Turnipseed, D. P., & Sauer, V. B. (2010). Discharge measurements at gaging stations (U.S. Geological Survey, Ed.).

Veloni, A. (2019). *Digital and Statistical Signal Processing*. Chapman and Hall/CRC. https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=5538941

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … van Mulbregt, P. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

von Kusserow, U. (2018). *Chaos, Turbulenzen und kosmische Selbstorganisationsprozesse*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-55895-9

Wahl, T. L. (2000). Analyzing ADV Data Using WinADV. In R. H. Hotchkiss & M. Glade (Eds.), *Building Partnerships* (pp. 1–10). American Society of Civil Engineers. https://doi.org/10.1061/40517(2000)300

Wahl, T. L. (2003). Discussion of "Despiking Acoustic Doppler Velocimeter Data" by Derek G. Goring and Vladimir I. Nikora. *Journal of Hydraulic Engineering*, *129*(6), 484–487. https://doi.org/10.1061/(ASCE)0733-9429(2003)129:6(484)

Waldi, R. (2019). *Statistische Datenanalyse*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-60645-2

Walker, E., & Nowacki, A. S. (2011). Understanding equivalence and noninferiority testing. *Journal of general internal medicine*, *26*(2), 192–196. https://doi.org/10.1007/s11606-010-1513-8

Waskom, M. (2021). seaborn: statistical data visualization. *Journal of Open Source Software*, *6*(60), 3021. https://doi.org/10.21105/joss.03021

Weidler, D. O. (2021). Drucksignaturen im Nachlauf von Zylindern unterschiedlichen Durchmessers.

Weiss, J. (2019). A Tutorial on the Proper Orthogonal Decomposition. https://doi.org/10.14279/depositonce-8512

Westlake, W. J. (1976). Symmetrical Confidence Intervals for Bioequivalence Trials. *Biometrics*, *32*(4), 741. https://doi.org/10.2307/2529259

Winder, S. (2002). *Analog and Digital Filter Design, 2nd Edition* (2nd edition). Newnes; Safari. https://learning.oreilly.com/library/view/-/9780750675475/?ar

Wolfson, R. (2020). *Essential university physics: Volume 1* (Fourth edition). Pearson Education. https://elibrary.pearson.de/book/99.150005/9781292350257

A. Annex

A.1. Figures

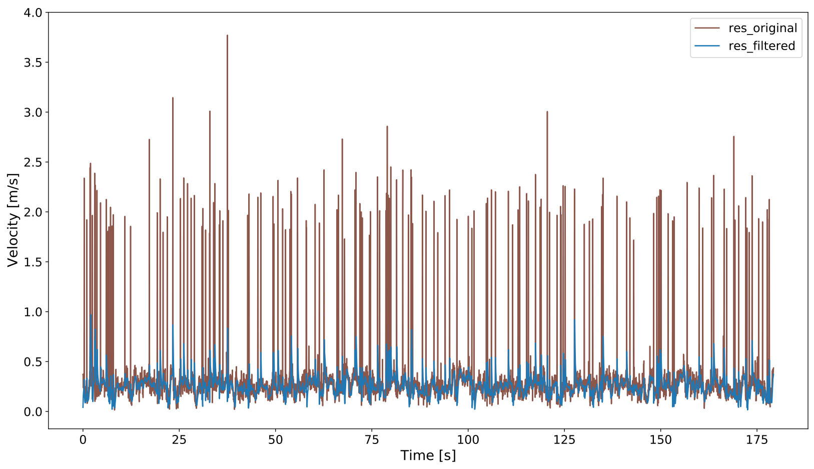


Figure A.1.: Exemplary visualisation of Butterworth filtered ($1\,\mathrm{Hz}$, order=1) time-series against original time-series.

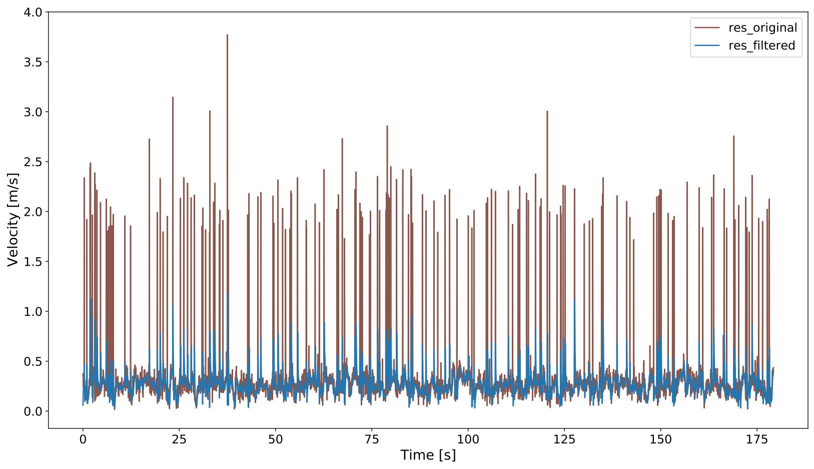


Figure A.2.: Exemplary visualisation of Butterworth filtered ($1.5\,\mathrm{Hz}$, order=1) time-series against original time-series.

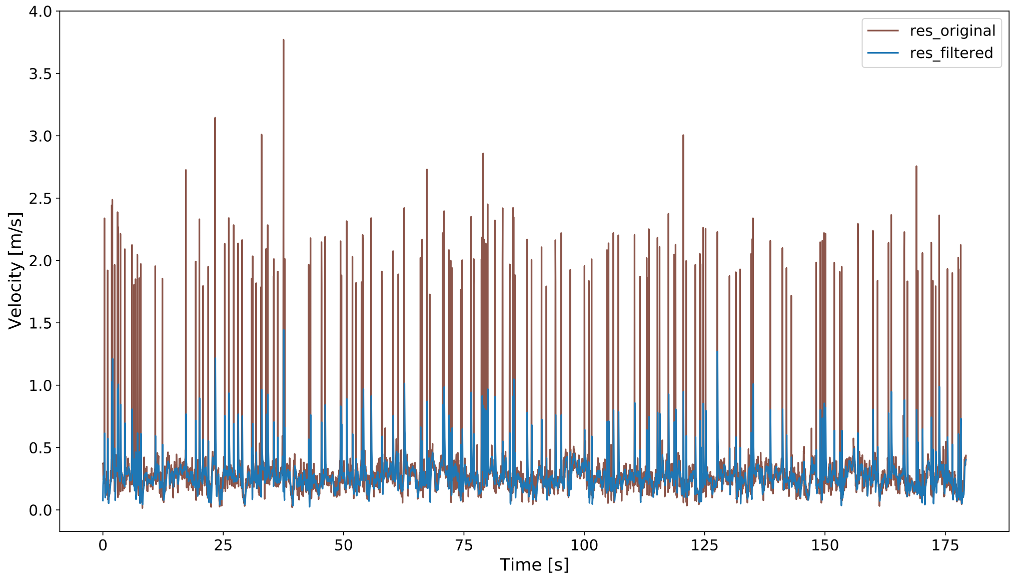


Figure A.3.: Exemplary visualisation of Butterworth filtered ($2\,\mathrm{Hz}$, order=1) time-series against original time-series.

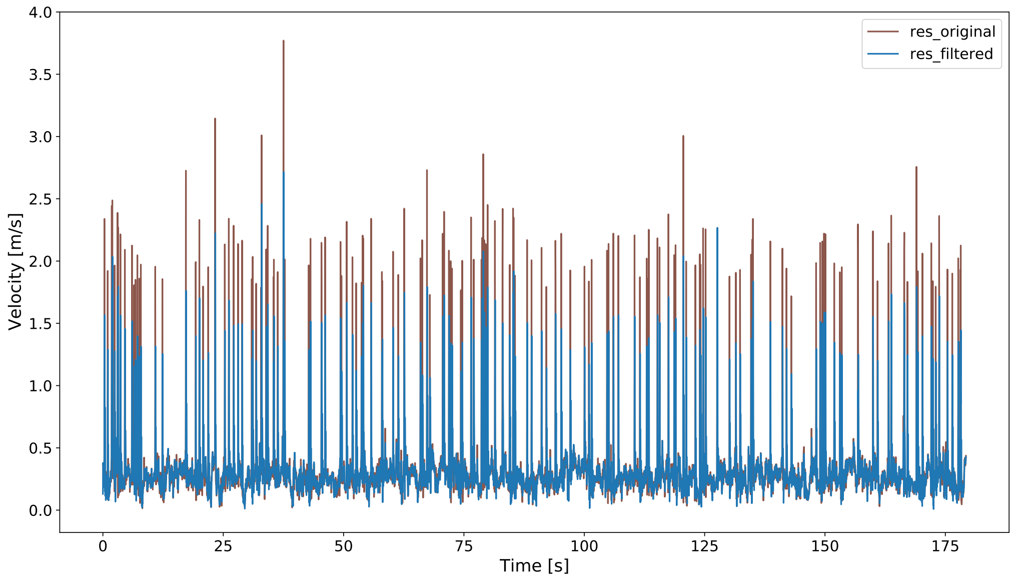


Figure A.4.: Exemplary visualisation of Butterworth filtered ($\frac{f_s}{2.93}$, order=3) time-series against original time-series.

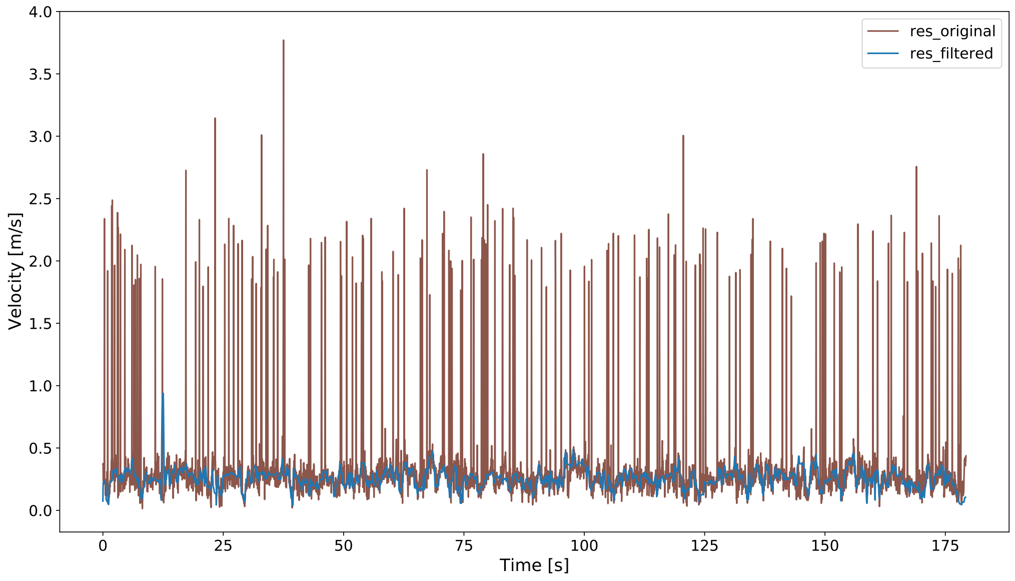


Figure A.5.: Exemplary visualisation of P-SAT and Butterworth filtered ($2.5\,\mathrm{Hz}$, order=1) time-series against original time-series.

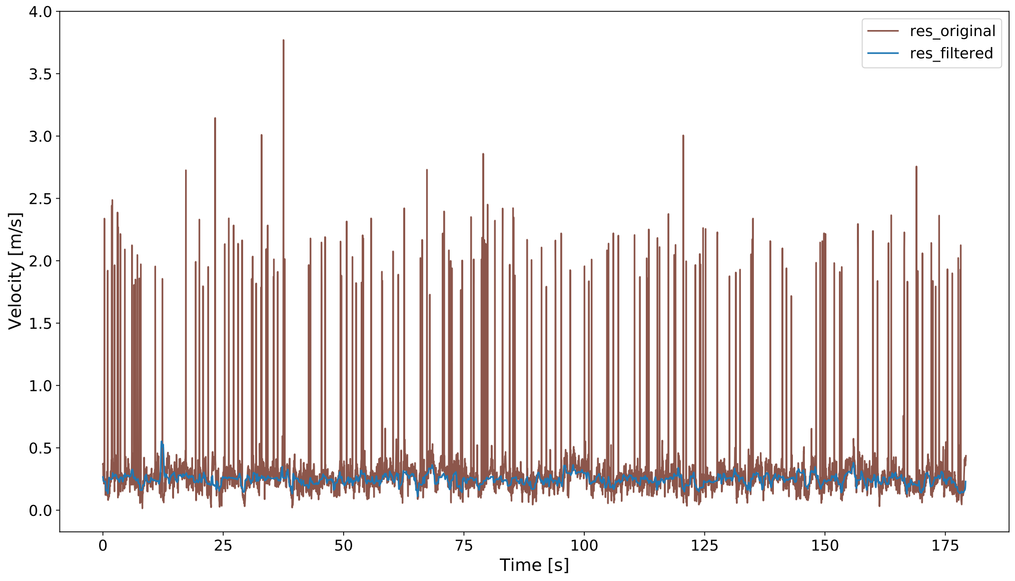


Figure A.6.: Exemplary visualisation of P-SAT and Gauss filtered time-series against original time-series.

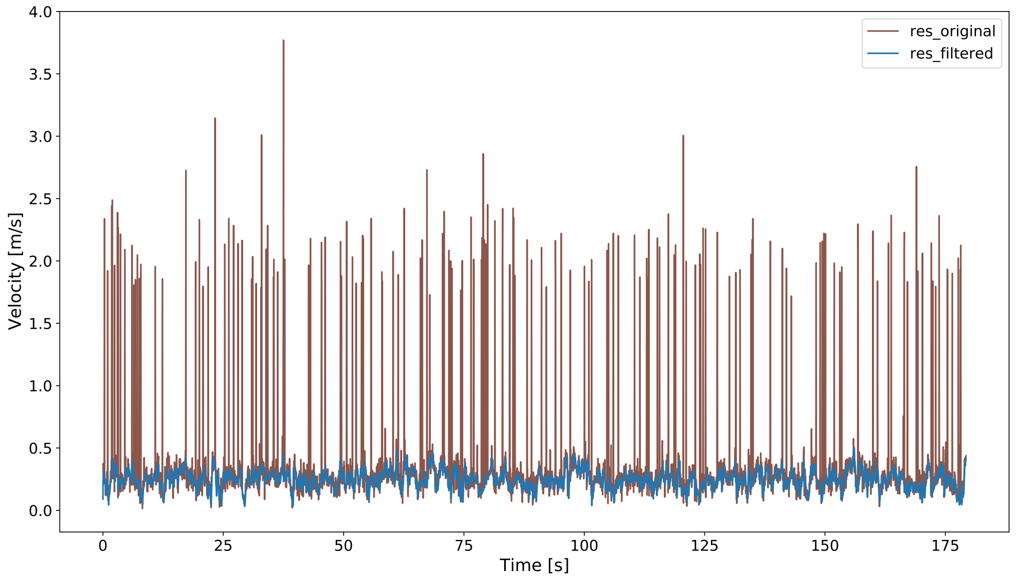


Figure A.7.: Exemplary visualisation of KDE despiked and Butterworth filtered ($2.5\,\mathrm{Hz}$, order=1) time-series against original time-series.

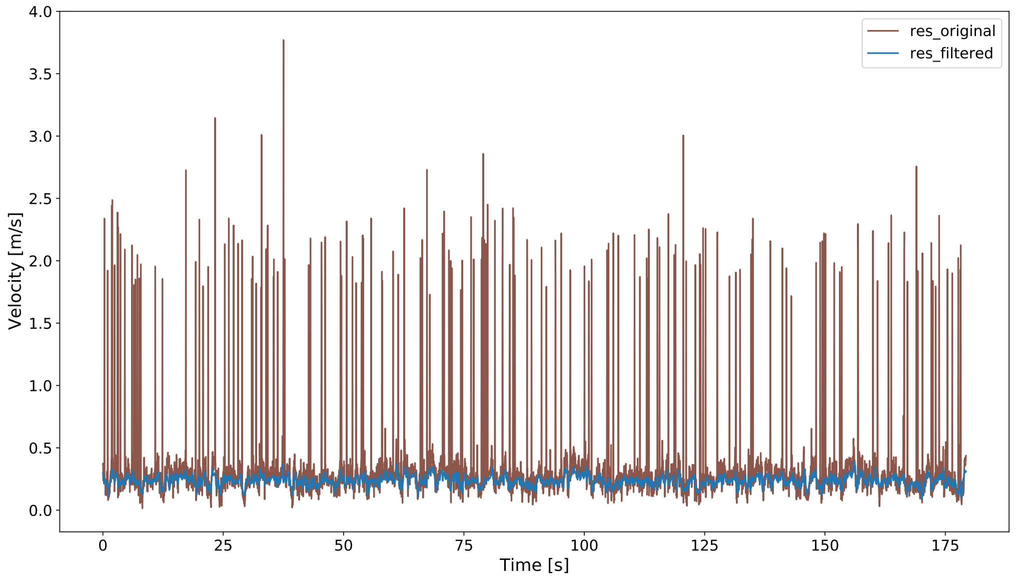


Figure A.8.: Exemplary visualisation of KDE despiked and Gauss time-series against original time-series.

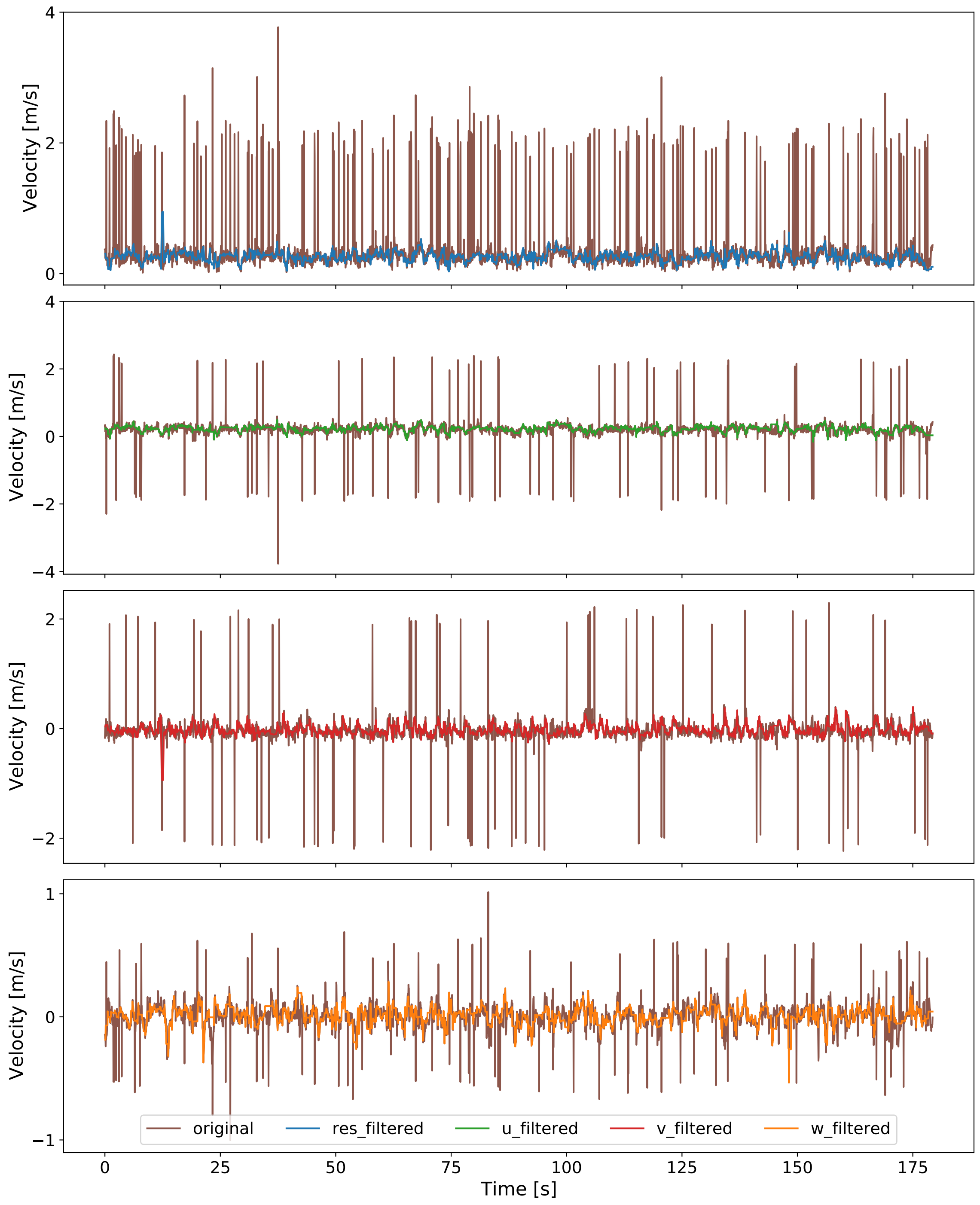


Figure A.9.: All components - exemplary visualisation of P-SAT filtered time-series against original time-series.

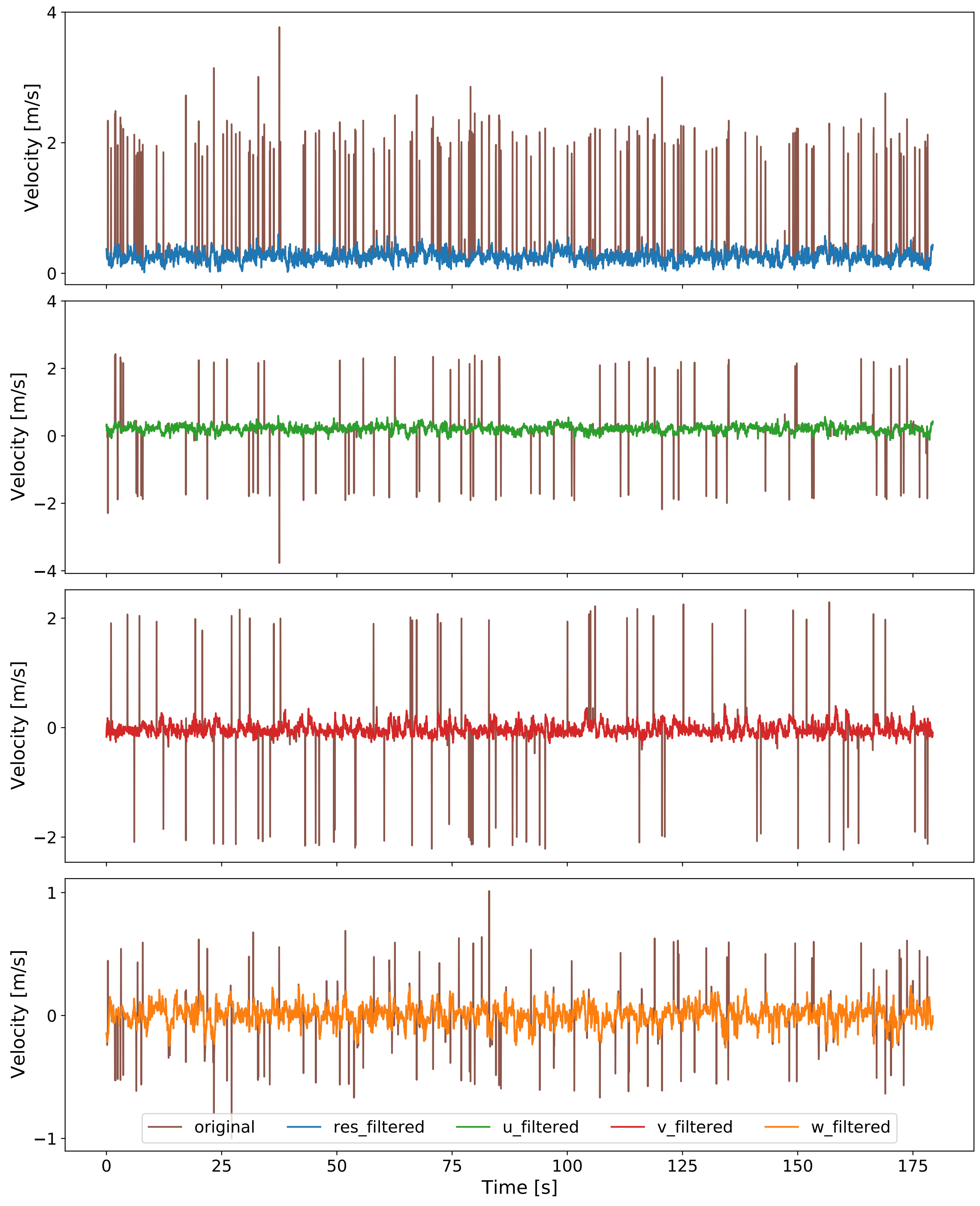


Figure A.10.: All components - exemplary visualisation of KDE despiked time-series against original time-series.

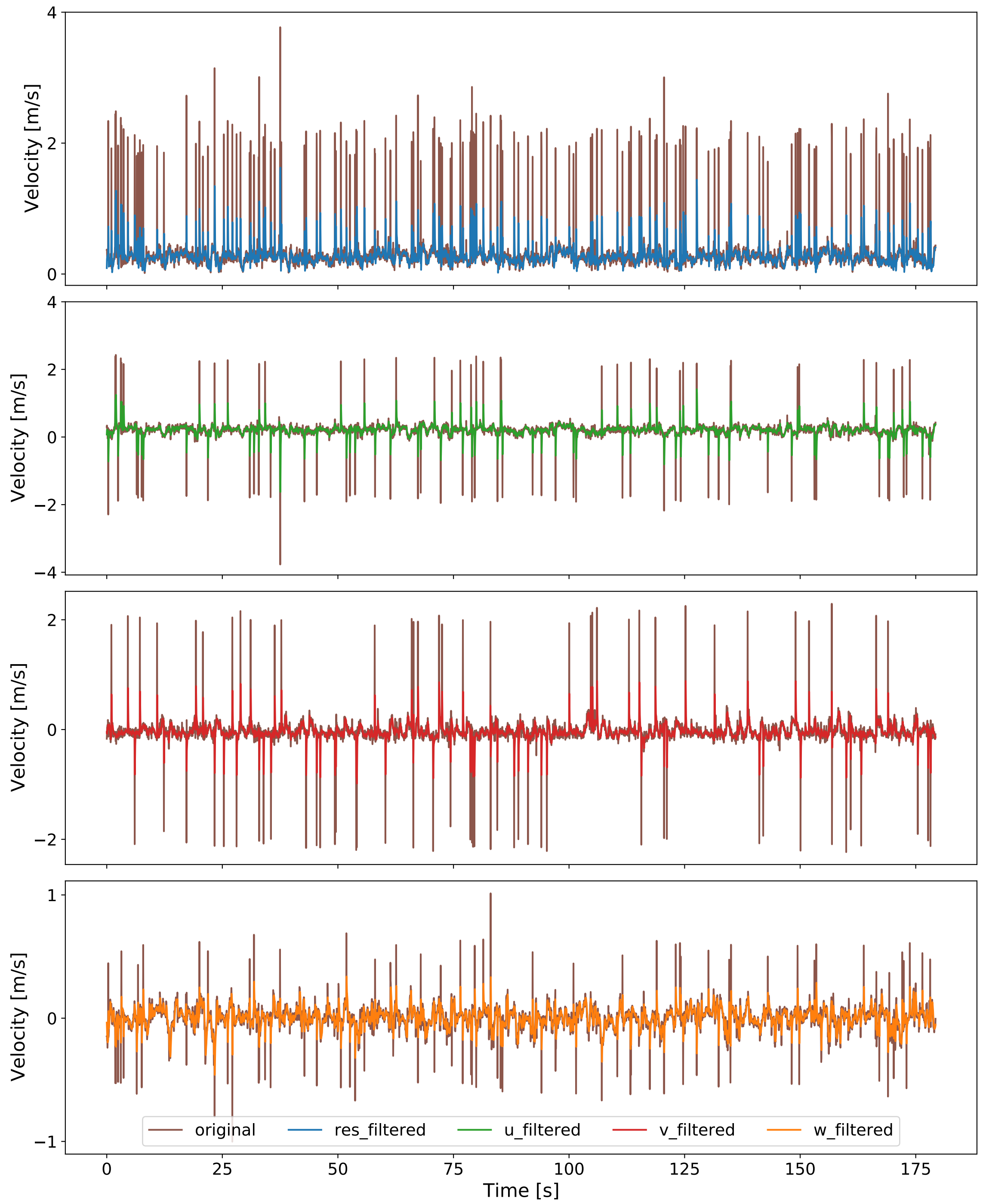


Figure A.11.: All components - exemplary visualisation of Butterworth filtered ($2.5\,\mathrm{Hz}$, order=1) time-series against original time-series.

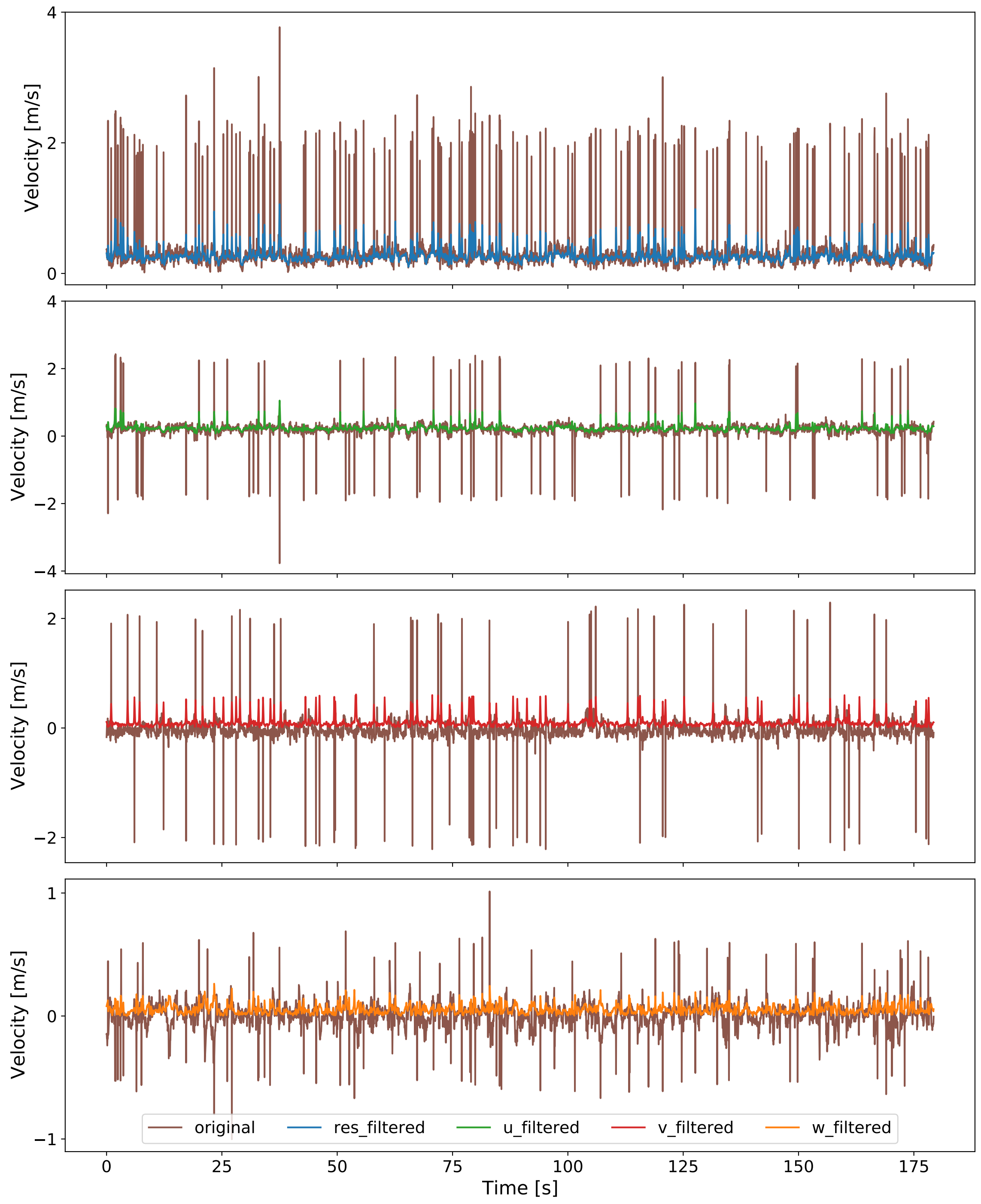


Figure A.12.: All components - exemplary visualisation of Gauss filtered time-series against original time-series.

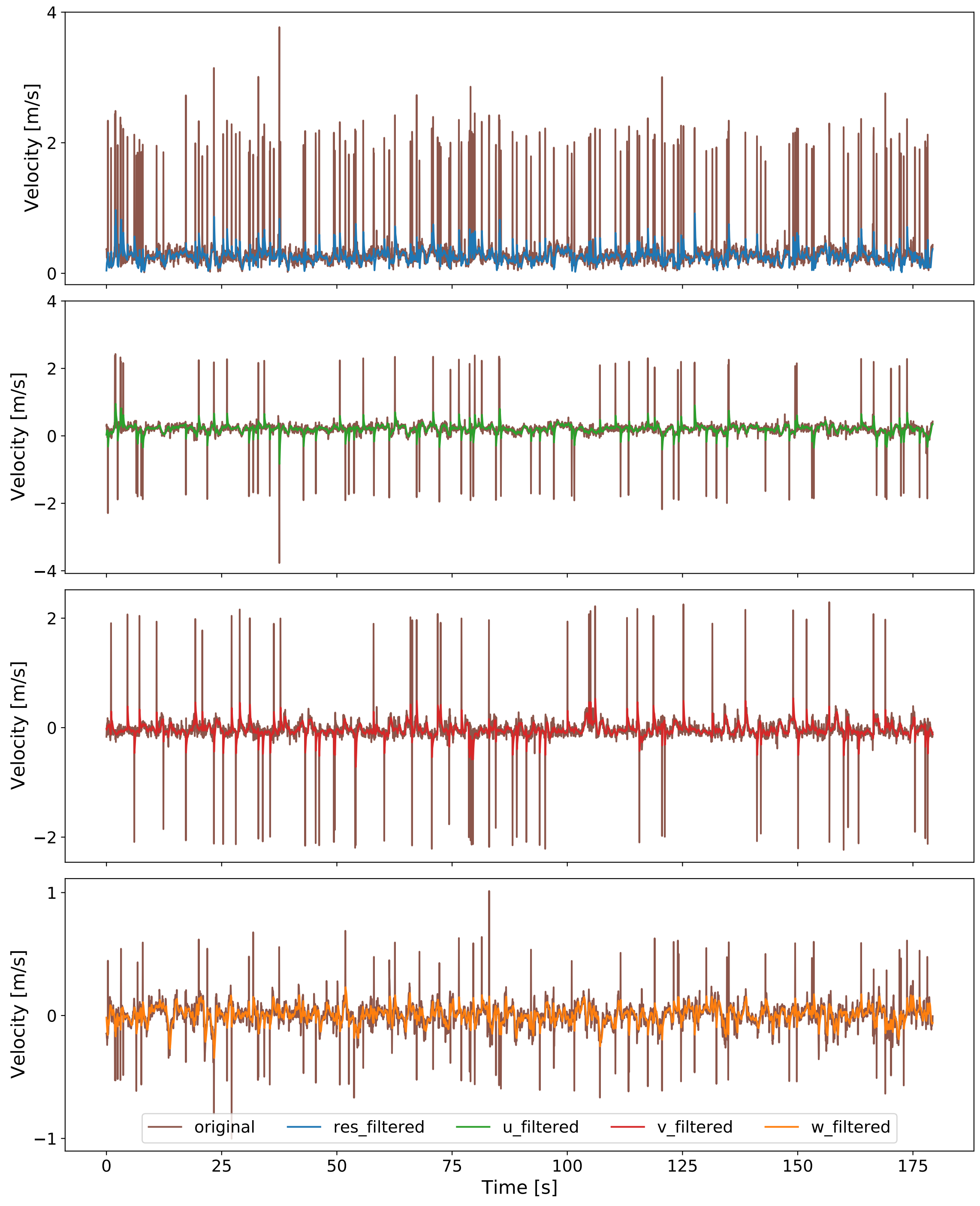


Figure A.13.: All components - exemplary visualisation of Butterworth filtered ($1\,\mathrm{Hz}$, order=1) time-series against original time-series.

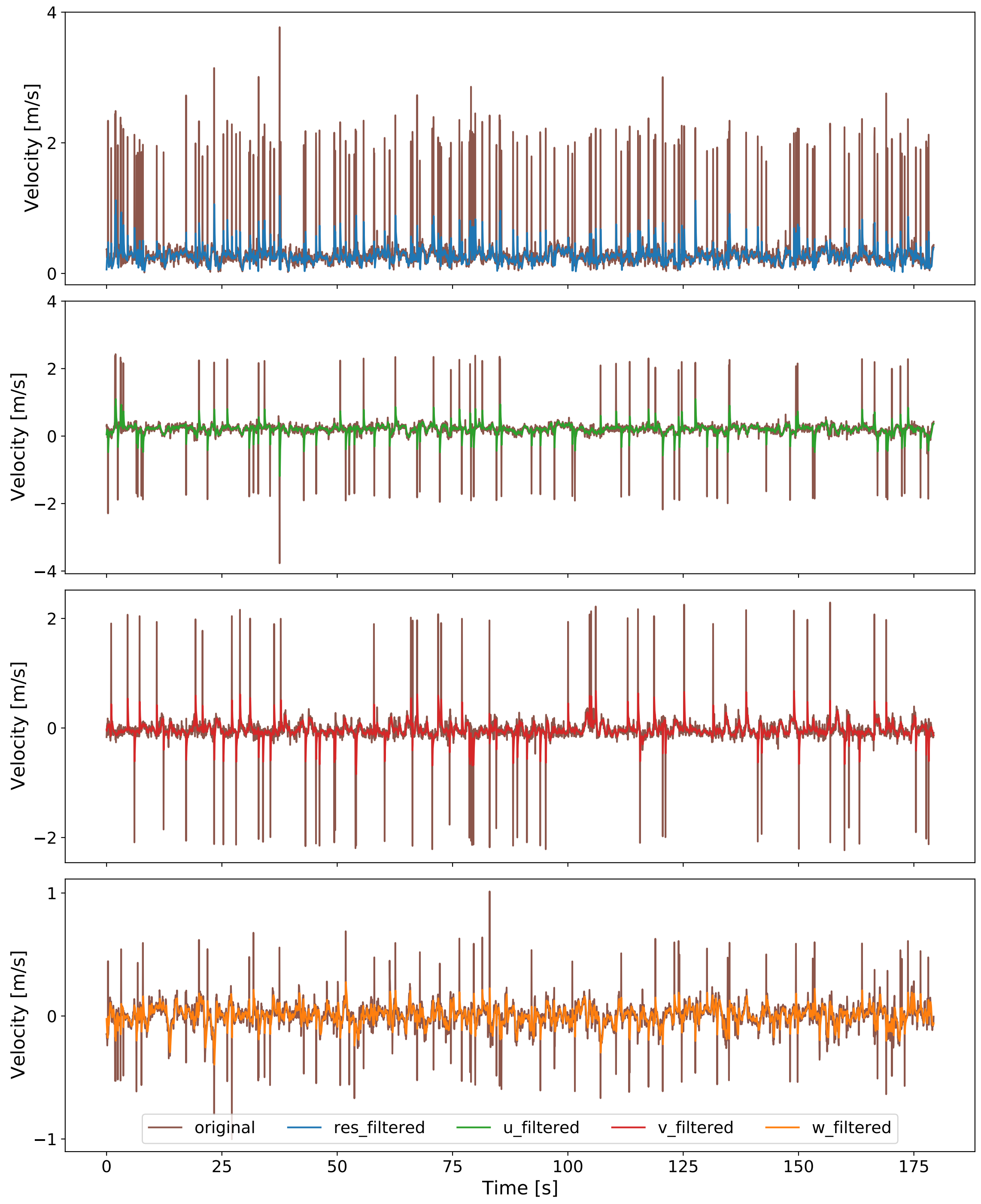


Figure A.14.: All components - exemplary visualisation of Butterworth filtered ($1.5\,\mathrm{Hz}$, order=1) time-series against original time-series.

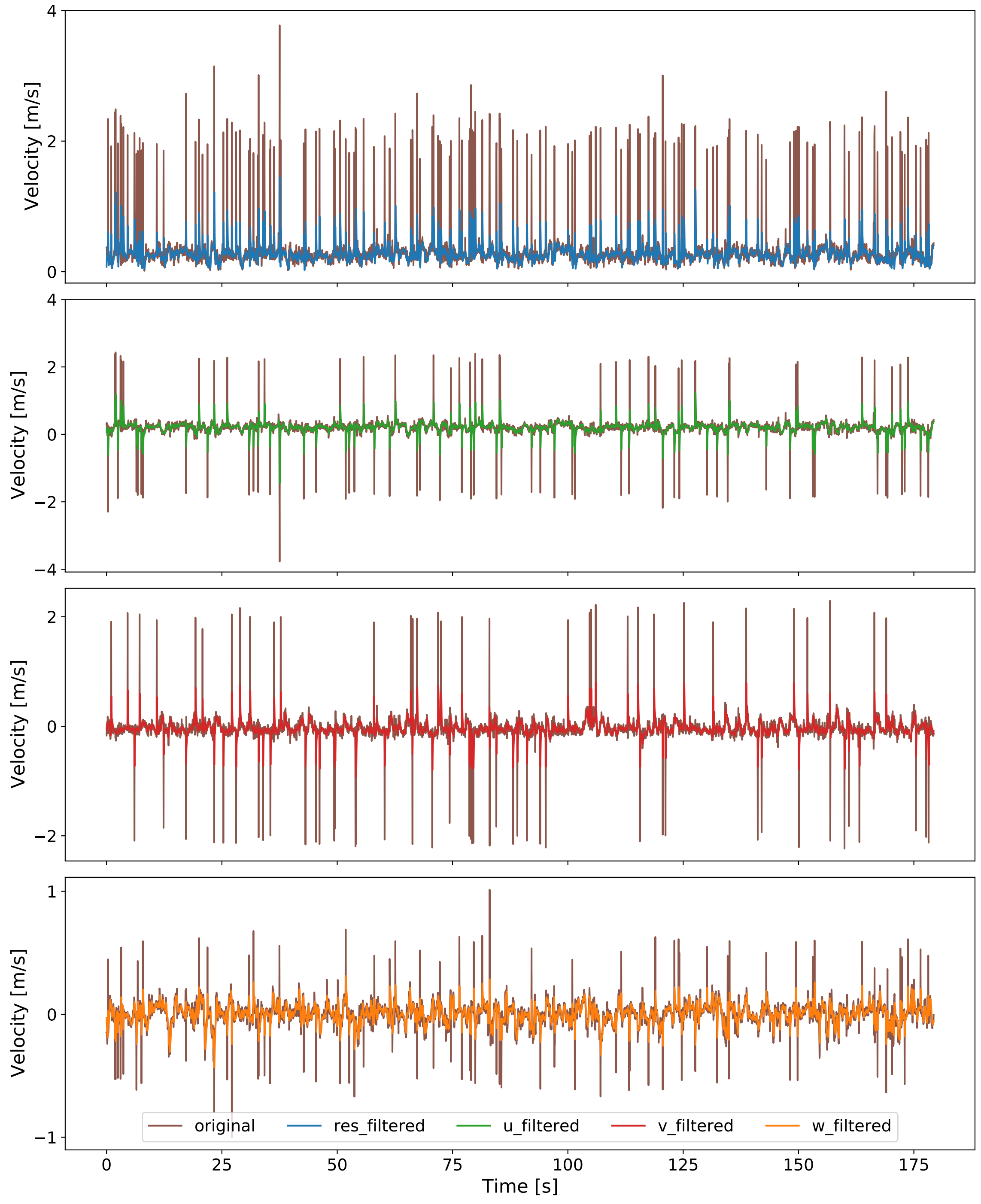


Figure A.15.: All components - exemplary visualisation of Butterworth filtered ($2\,\mathrm{Hz}$, order=1) time-series against original time-series.

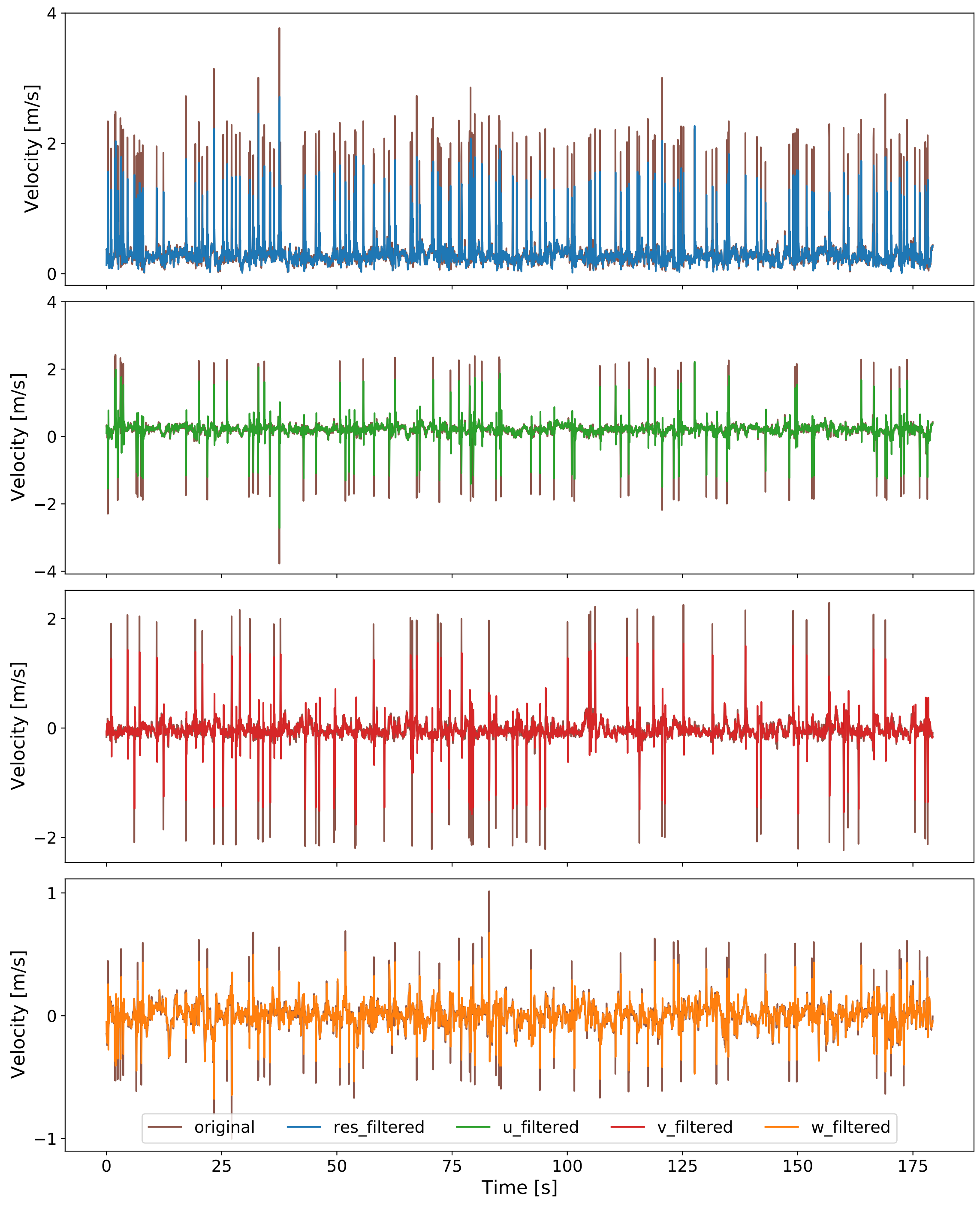


Figure A.16.: All components - exemplary visualisation of Butterworth filtered ($\frac{f_s}{2.93}$, order=3) time-series against original time-series.

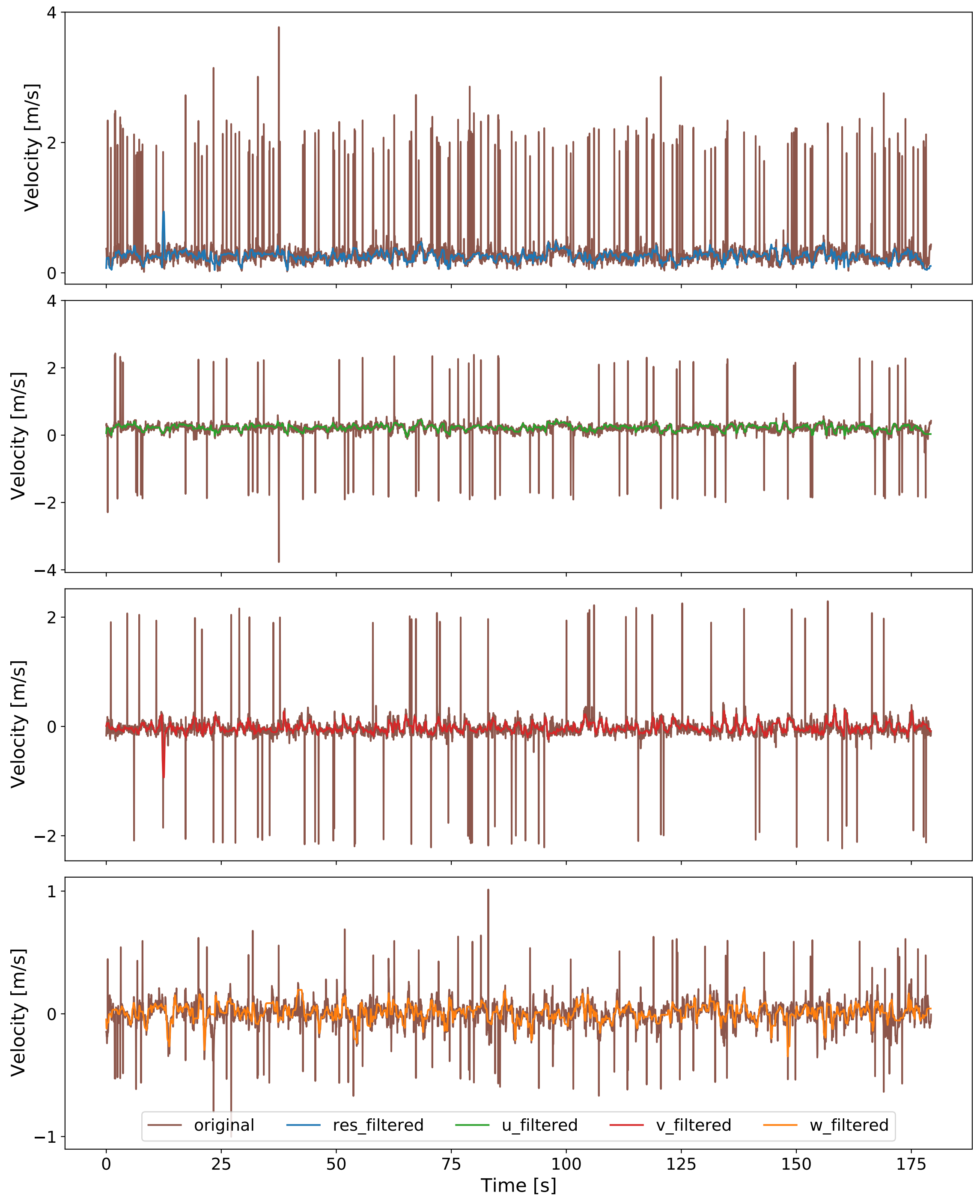


Figure A.17.: All components - exemplary visualisation of P-SAT and Butterworth filtered ($2.5\,\mathrm{Hz}$, order=1) time-series against original time-series.

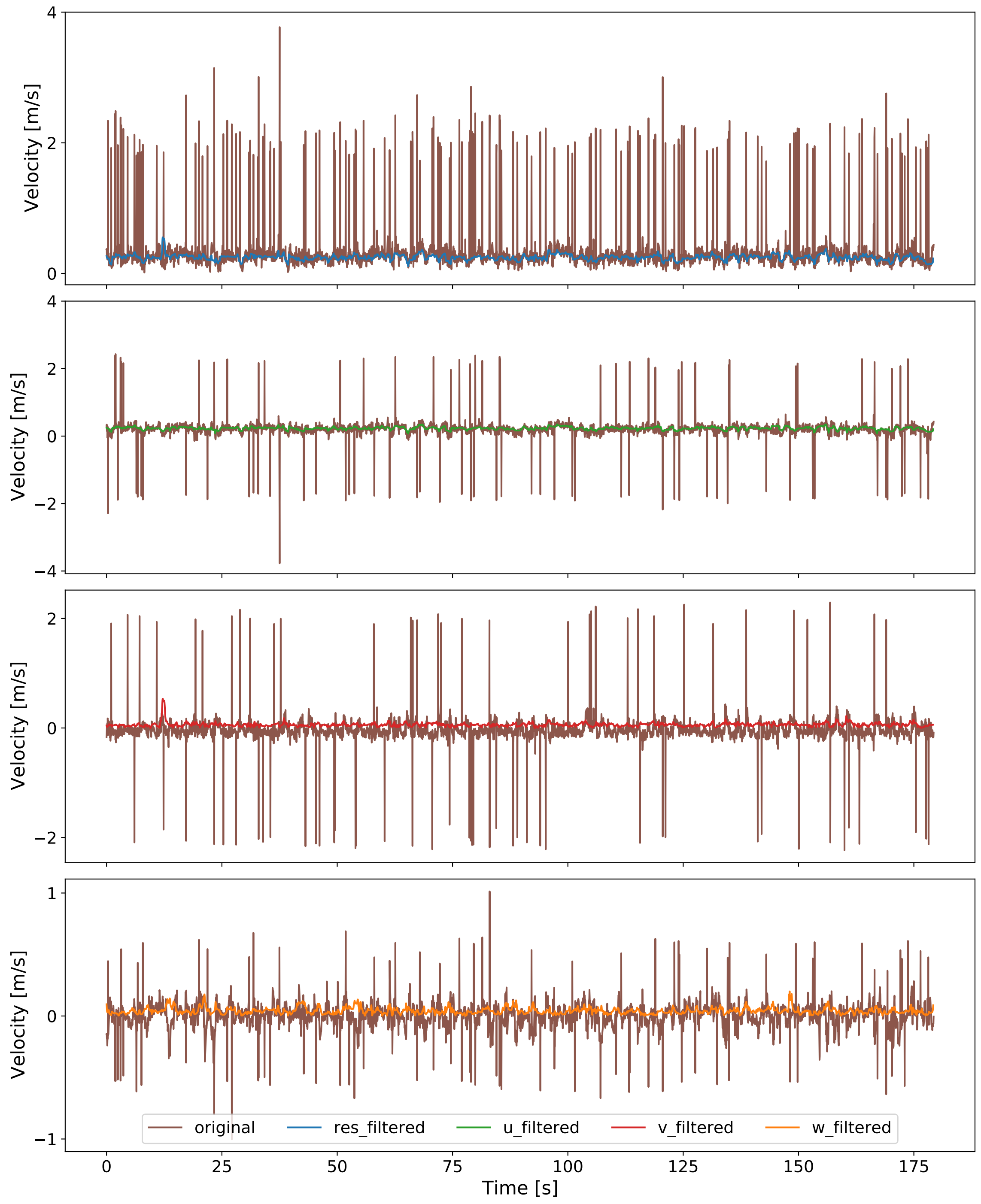


Figure A.18.: All components - exemplary visualisation of P-SAT and Gauss filtered time-series against original time-series.

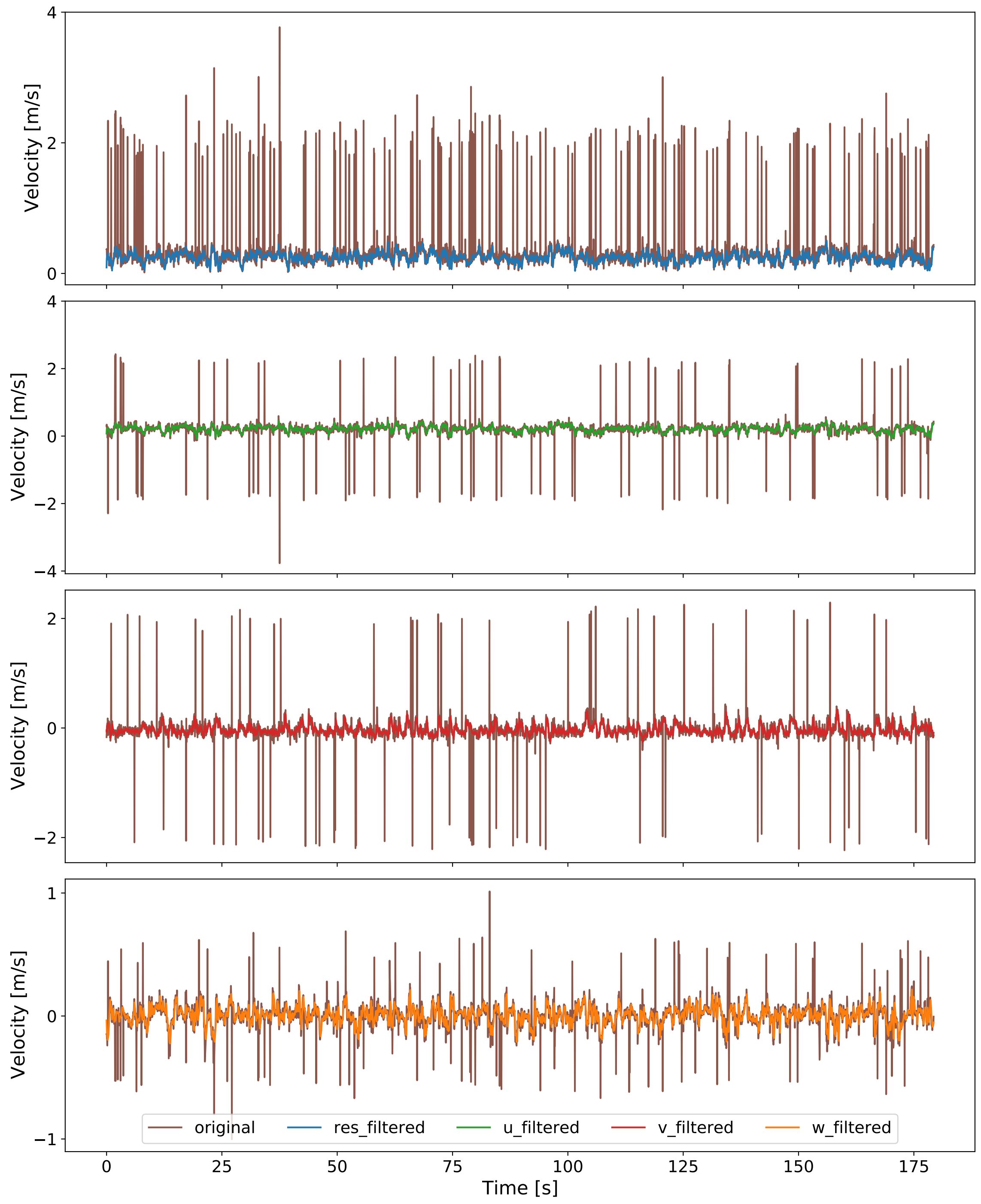


Figure A.19.: All components - exemplary visualisation of KDE despiked and Butterworth filtered ($2.5\,\mathrm{Hz}$, order=1) time-series against original time-series.

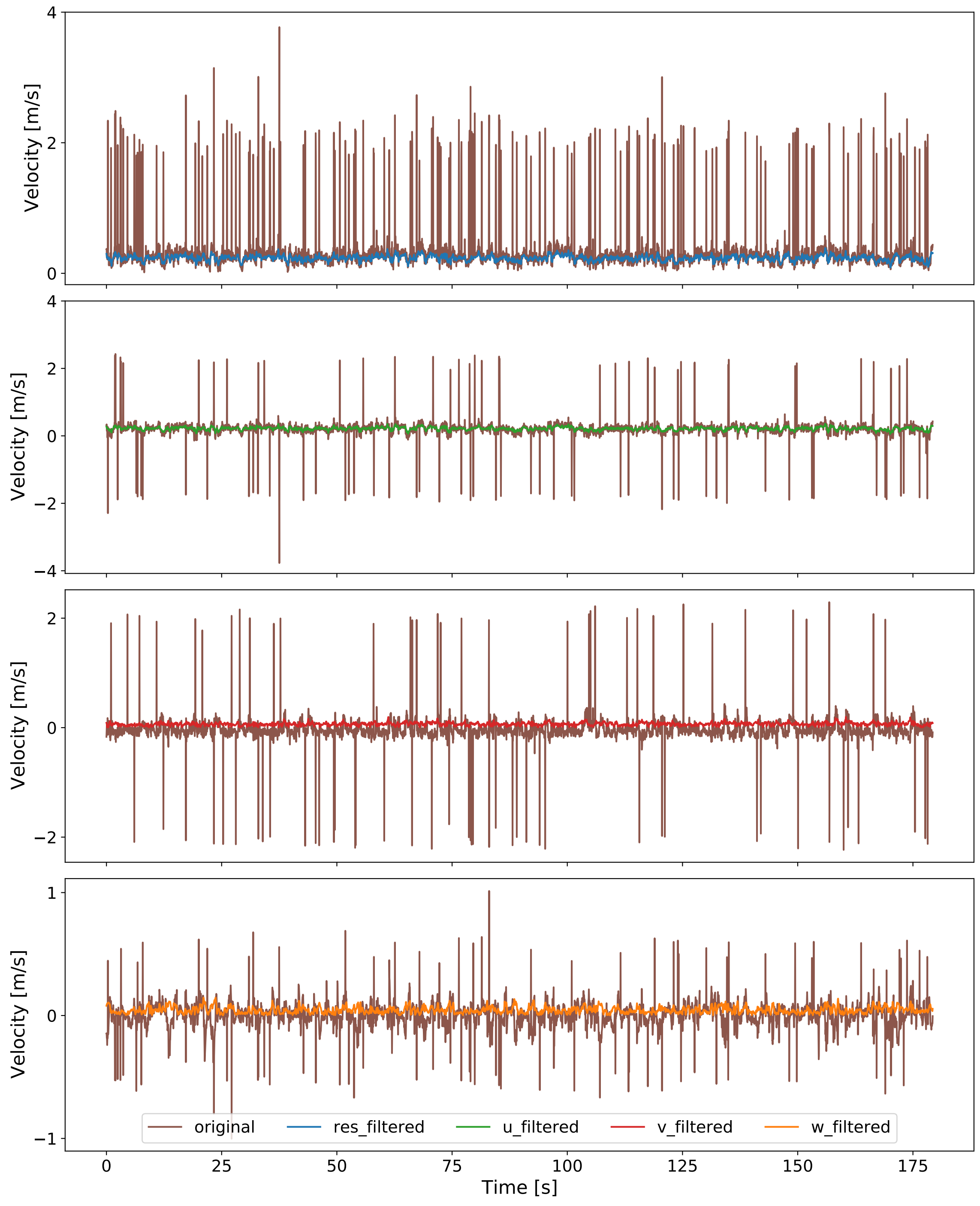


Figure A.20.: All components - exemplary visualisation of KDE despiked and Gauss time-series against original time-series.

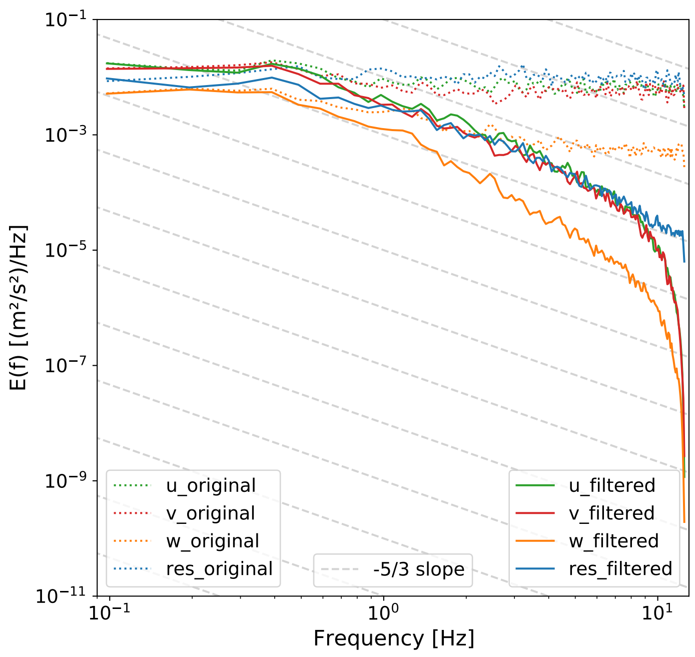


Figure A.21.: Spectra from Butterworth filtered ($1\,\mathrm{Hz}$, order=1) time-series.

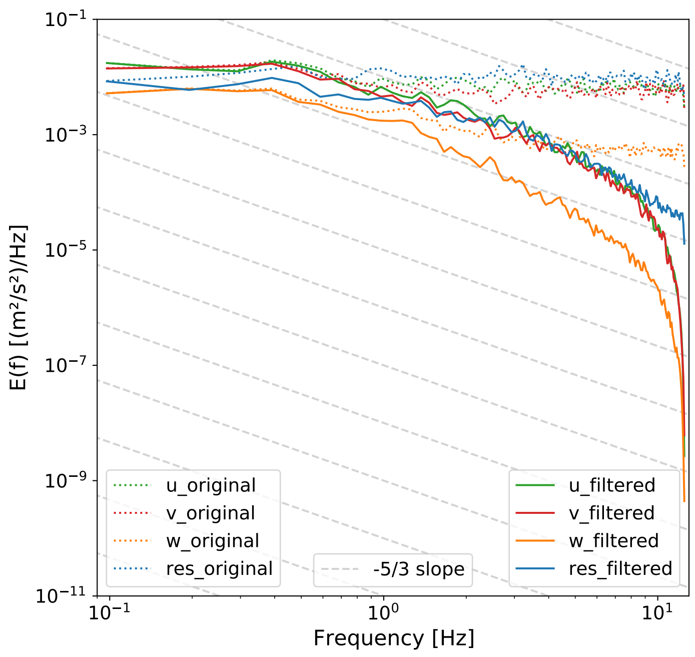


Figure A.22.: Spectra from Butterworth filtered ($1.5\,\mathrm{Hz}$, order=1) time-series.

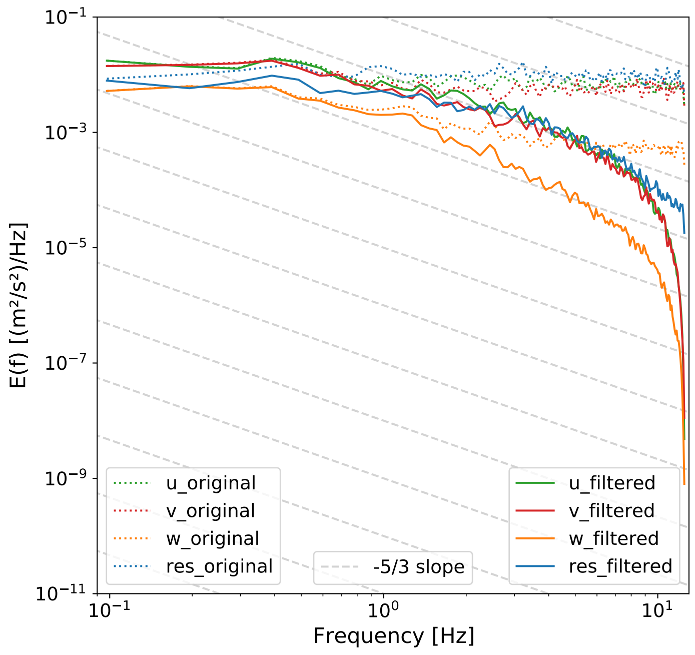


Figure A.23.: Spectra from Butterworth filtered ($2\,\mathrm{Hz}$, order=1) time-series.

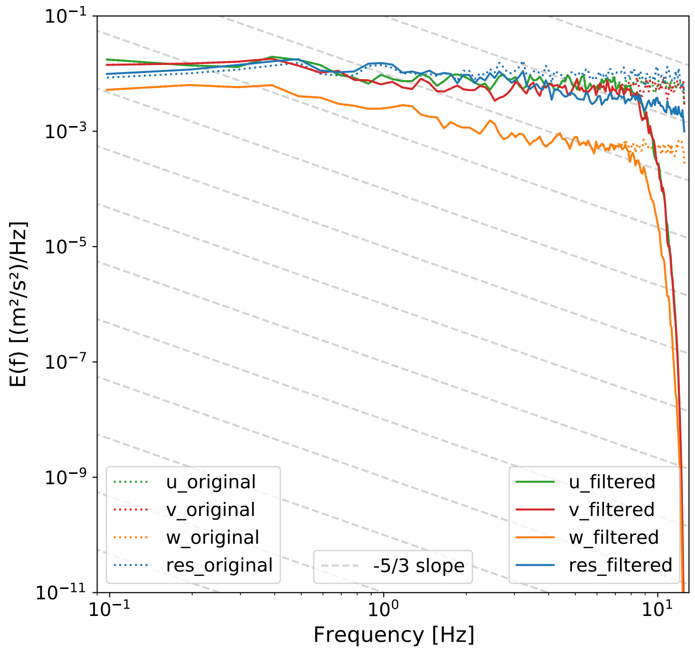


Figure A.24.: Spectra from Butterworth filtered ($\frac{f_s}{2.93}$, order=3) time-series.

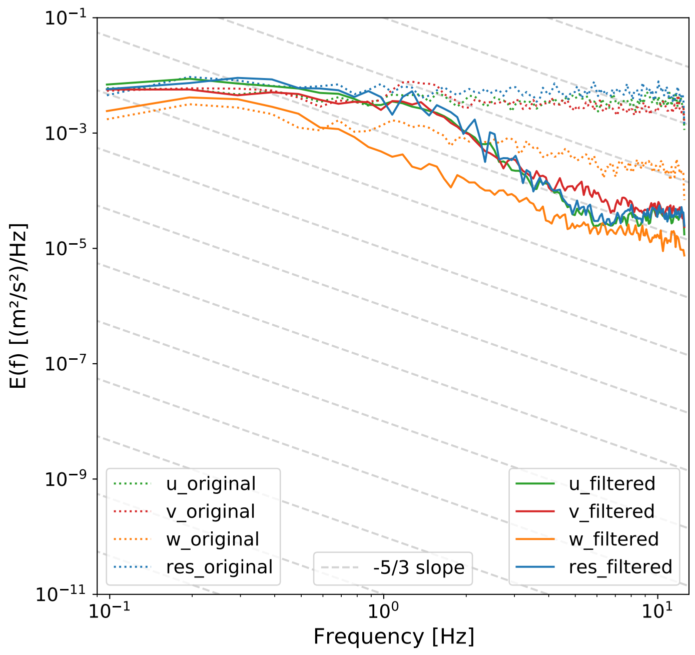


Figure A.25.: Spectra from P-SAT filtered time-series for type 2 turbulence conditions (9A_-3_V20210106125137).

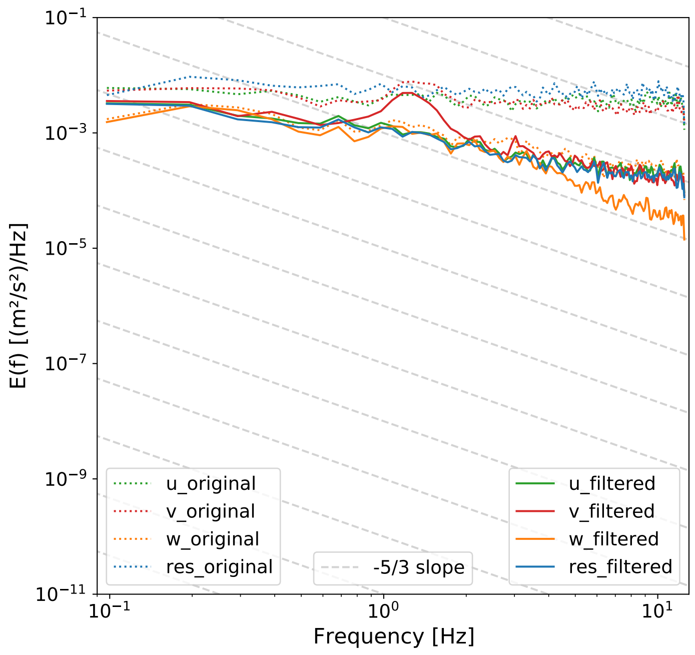
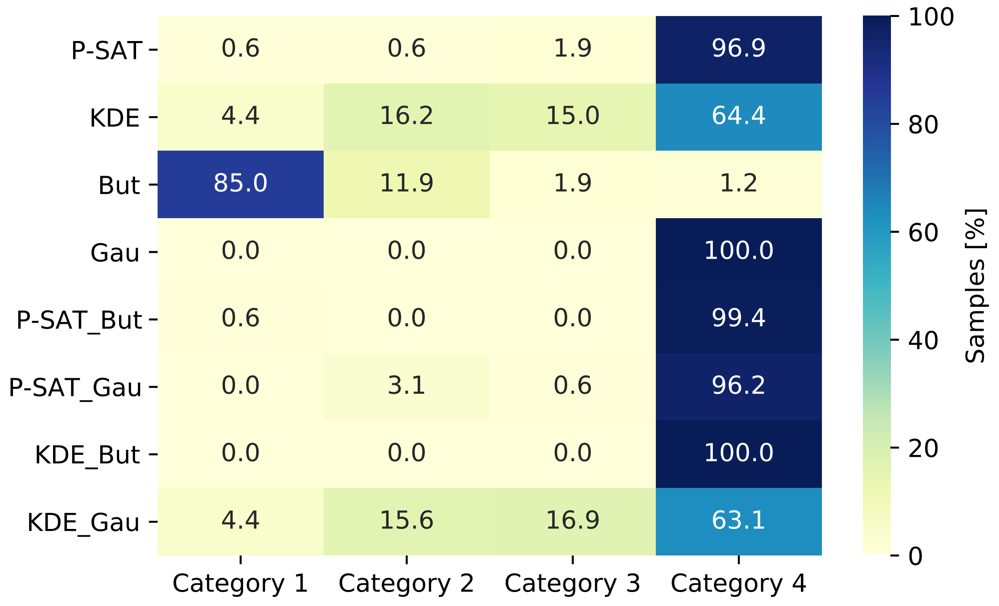
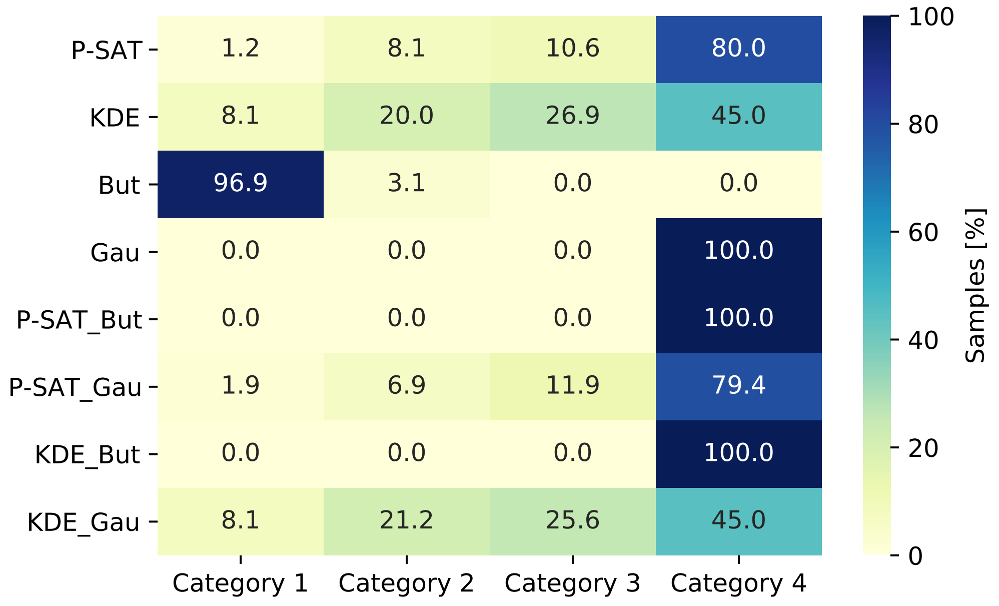


Figure A.26.: Spectra from KDE despiked time-series for type 2 turbulence conditions (9A_-3_V20210106125137).



	Category 1	Category 2	Category 3	Category 4
P-SAT	0.6	0.6	1.9	96.9
KDE	4.4	16.2	15.0	64.4
But	85.0	11.9	1.9	1.2
Gau	0.0	0.0	0.0	100.0
P-SAT_But	0.6	0.0	0.0	99.4
P-SAT_Gau	0.0	3.1	0.6	96.2
KDE_But	0.0	0.0	0.0	100.0
KDE_Gau	4.4	15.6	16.9	63.1

Figure A.27.: Heatmap of the categorisation of the relative change in means in the v-velocity component for all implemented data analysis methods (n=160 samples).



	Category 1	Category 2	Category 3	Category 4
P-SAT	1.2	8.1	10.6	80.0
KDE	8.1	20.0	26.9	45.0
But	96.9	3.1	0.0	0.0
Gau	0.0	0.0	0.0	100.0
P-SAT_But	0.0	0.0	0.0	100.0
P-SAT_Gau	1.9	6.9	11.9	79.4
KDE_But	0.0	0.0	0.0	100.0
KDE_Gau	8.1	21.2	25.6	45.0

Figure A.28.: Heatmap of the categorisation of the relative change in means in the w-velocity component for all implemented data analysis methods (n=160 samples).

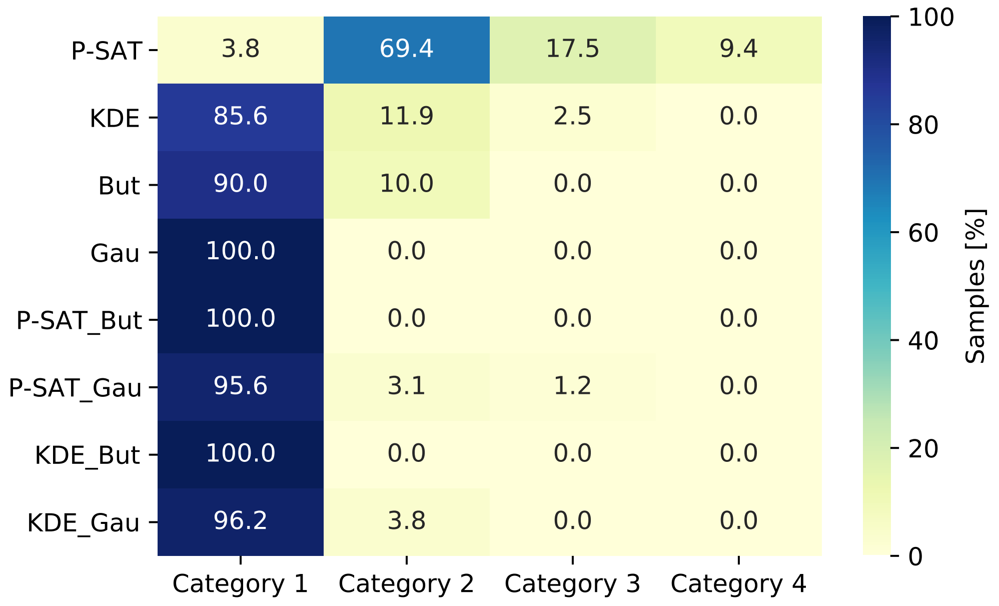


Figure A.29.: Heatmap of the categorisation of the relative change in variance in the v-velocity component for all implemented data analysis methods (n=160 samples).

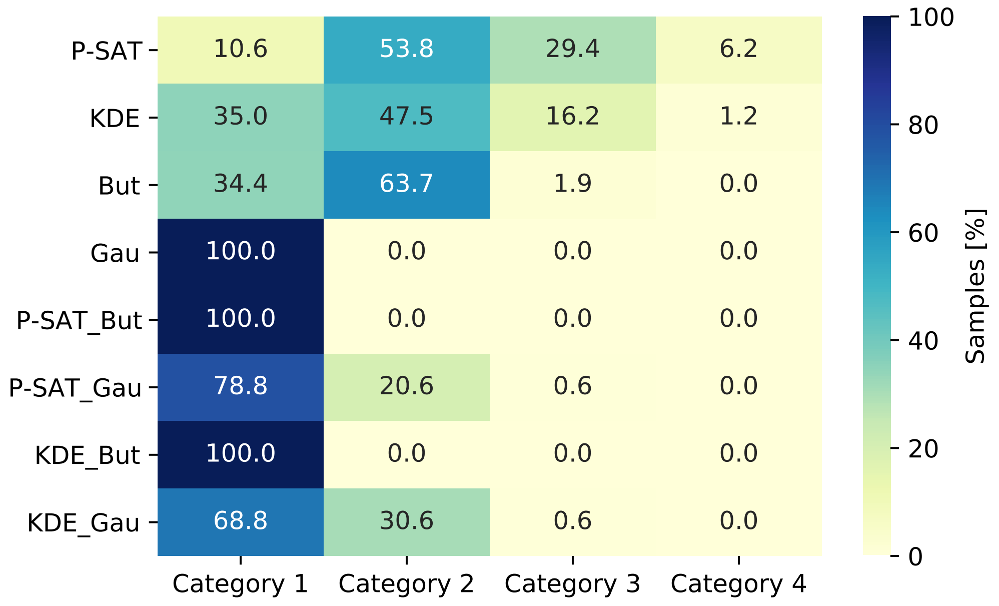
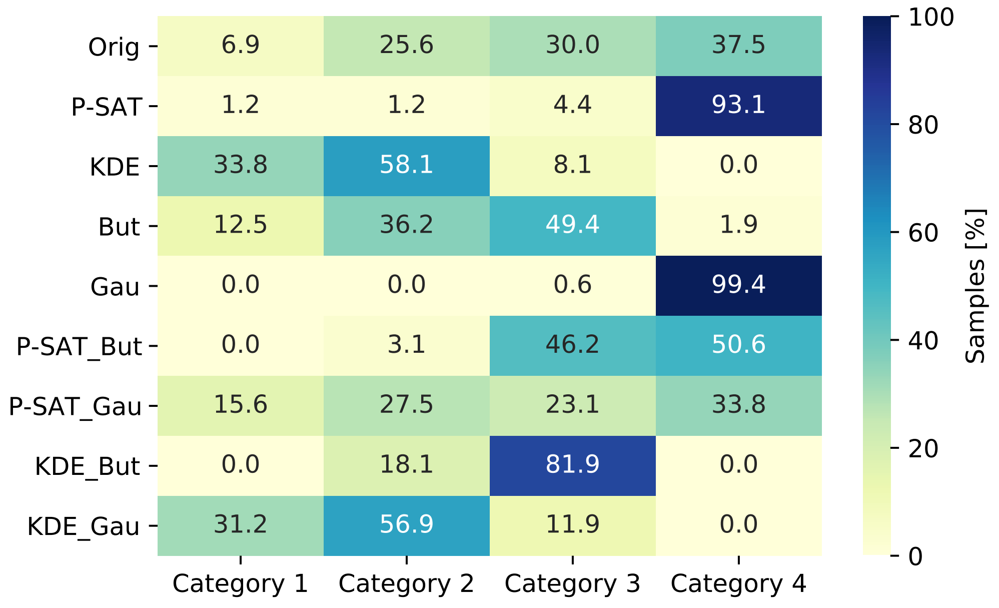
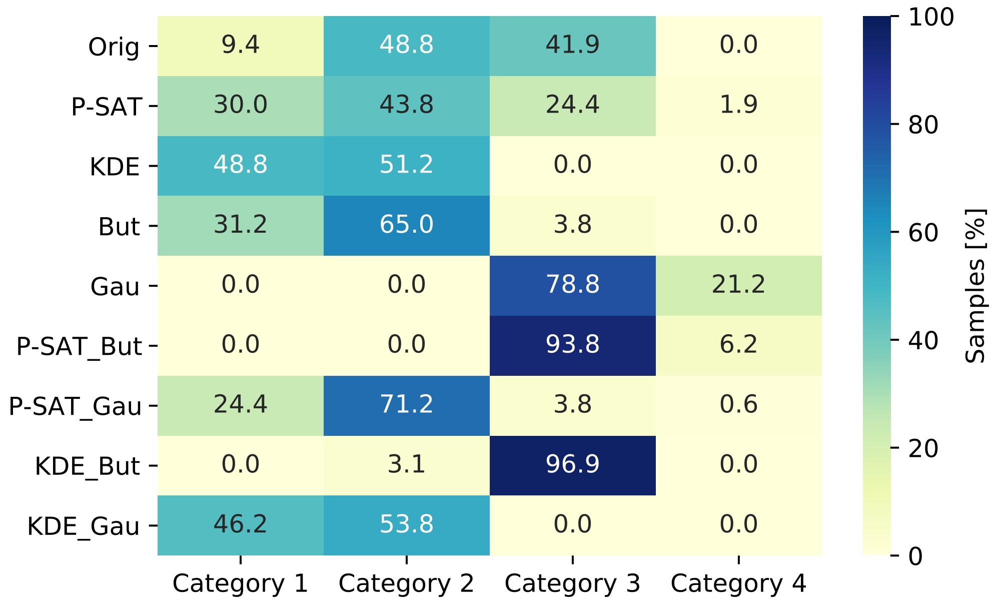


Figure A.30.: Heatmap of the categorisation of the relative change in variance in the w-velocity component for all implemented data analysis methods (n=160 samples).



	Category 1	Category 2	Category 3	Category 4
Orig	6.9	25.6	30.0	37.5
P-SAT	1.2	1.2	4.4	93.1
KDE	33.8	58.1	8.1	0.0
But	12.5	36.2	49.4	1.9
Gau	0.0	0.0	0.6	99.4
P-SAT_But	0.0	3.1	46.2	50.6
P-SAT_Gau	15.6	27.5	23.1	33.8
KDE_But	0.0	18.1	81.9	0.0
KDE_Gau	31.2	56.9	11.9	0.0

Figure A.31.: Heatmap of the categorisation of the skewness in the v-velocity component for all implemented data analysis methods in comparison to the categorisation of the original samples (n=160 samples).



	Category 1	Category 2	Category 3	Category 4
Orig	9.4	48.8	41.9	0.0
P-SAT	30.0	43.8	24.4	1.9
KDE	48.8	51.2	0.0	0.0
But	31.2	65.0	3.8	0.0
Gau	0.0	0.0	78.8	21.2
P-SAT_But	0.0	0.0	93.8	6.2
P-SAT_Gau	24.4	71.2	3.8	0.6
KDE_But	0.0	3.1	96.9	0.0
KDE_Gau	46.2	53.8	0.0	0.0

Figure A.32.: Heatmap of the categorisation of the skewness in the w-velocity component for all implemented data analysis methods in comparison to the categorisation of the original samples (n=160 samples).

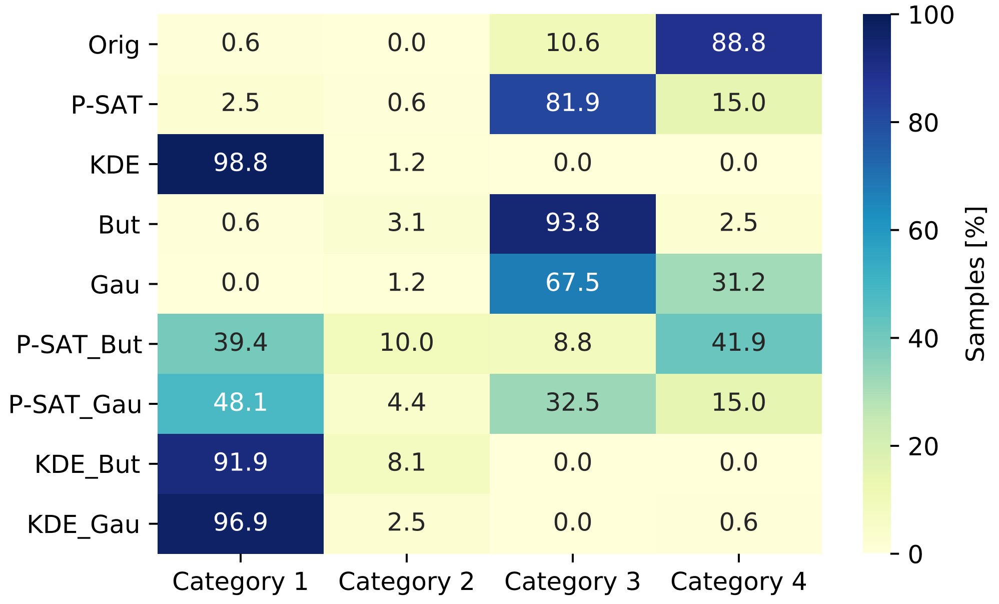


Figure A.33.: Heatmap of the categorisation of the kurtosis in the v-velocity component for all implemented data analysis methods in comparison to the categorisation of the original samples (n=160 samples).

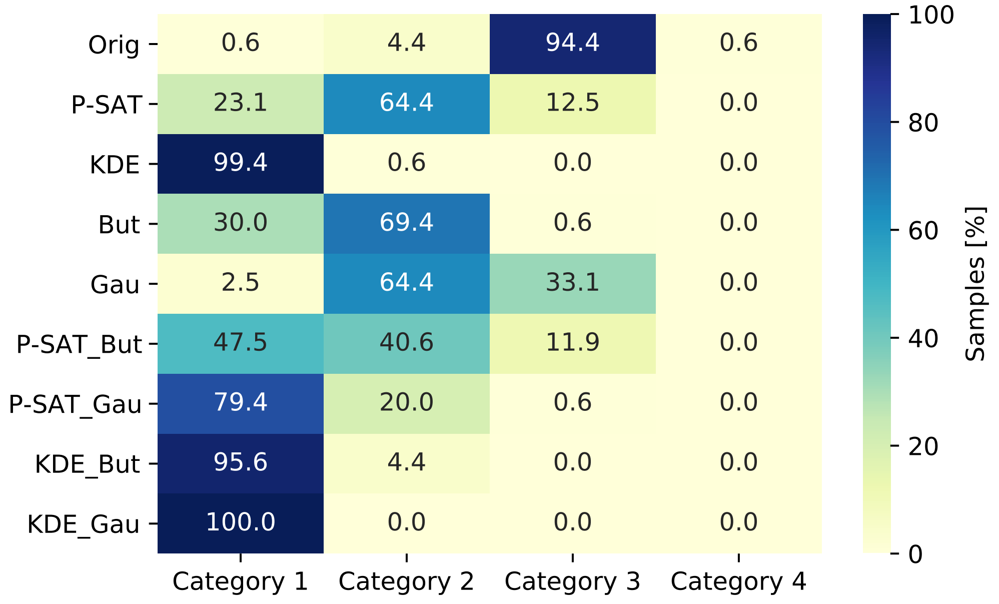
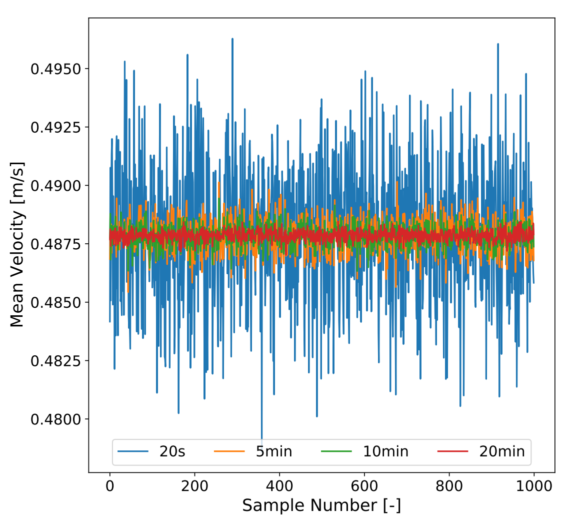
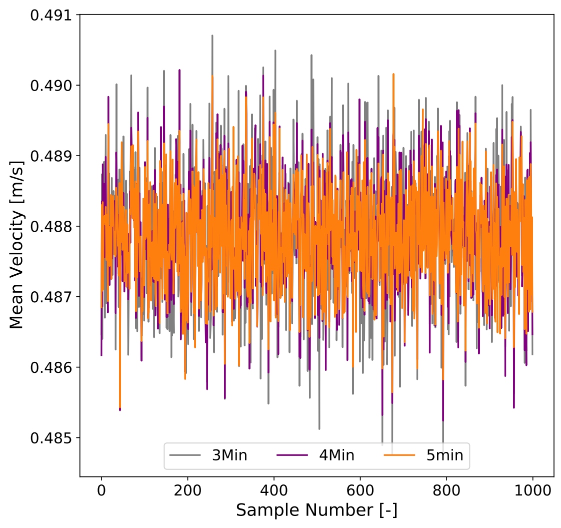


Figure A.34.: Heatmap of the categorisation of the kurtosis in the w-velocity component for all implemented data analysis methods in comparison to the categorisation of the original samples (n=160 samples).

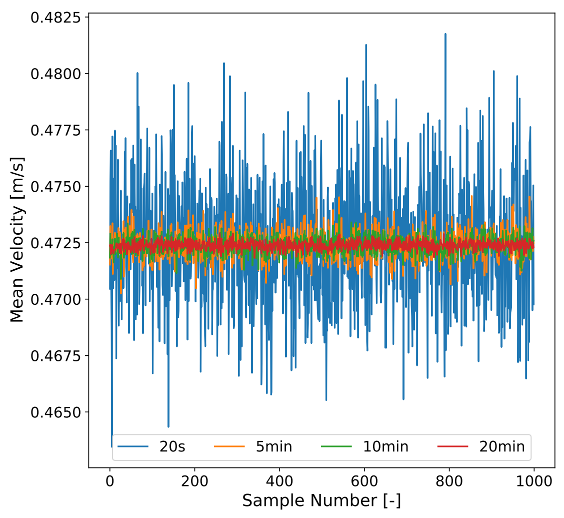


(a) Comparison of means for sampling times up to $20\,\mathrm{min}$.

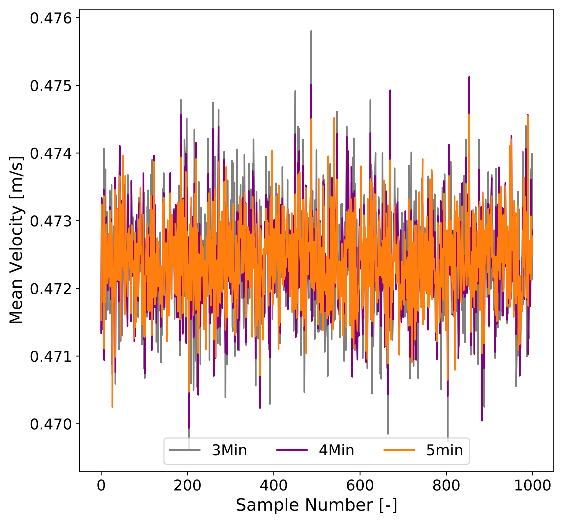


(b) Comparison of means for recommended sampling times.

Figure A.35.: Comparison of means for the long-duration measurement in wall proximity and vertical position of $0.1\,\mathrm{m}$ over bottom for sampling times up to $20\,\mathrm{min}$ and for recommended sampling times.

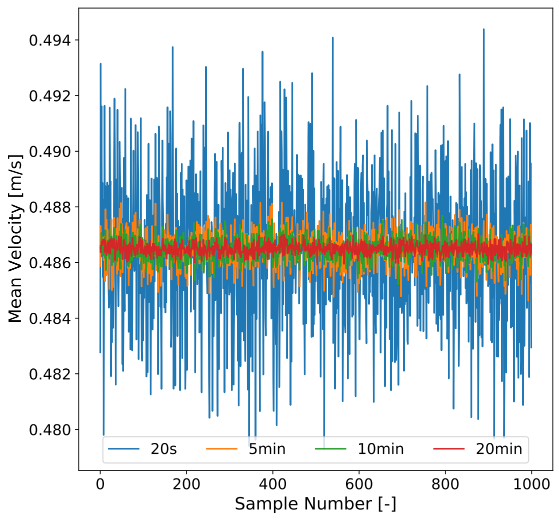


(a) Comparison of means for sampling times up to $20\,\mathrm{min}$.

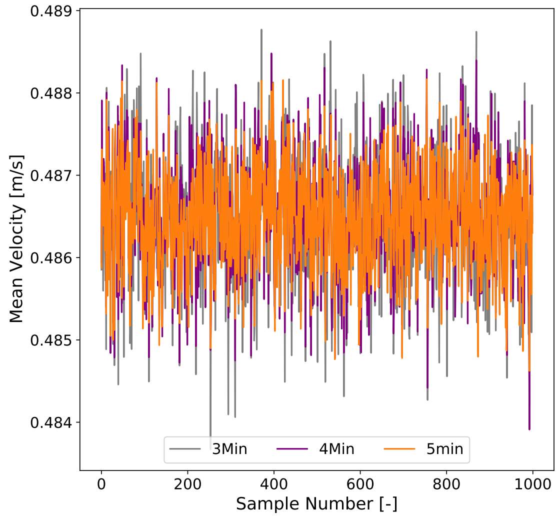


(b) Comparison of means for recommended sampling times.

Figure A.36.: Comparison of means for the long-duration measurement in central horizontal position and vertical position of $0.1\,\mathrm{m}$ over bottom for sampling times up to $20\,\mathrm{min}$ and for recommended sampling times.

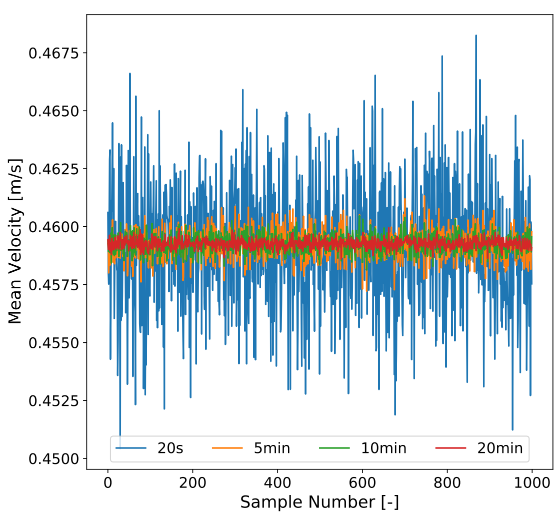


(a) Comparison of means for sampling times up to $20\,\mathrm{min}$.

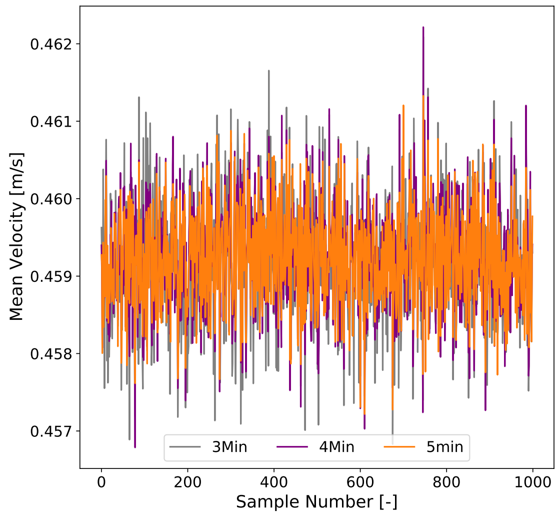


(b) Comparison of means for recommended sampling times.

Figure A.37.: Comparison of means for the long-duration measurement in wall proximity and vertical position of $0.4\,\mathrm{m}$ over bottom for sampling times up to $20\,\mathrm{min}$ and for recommended sampling times.

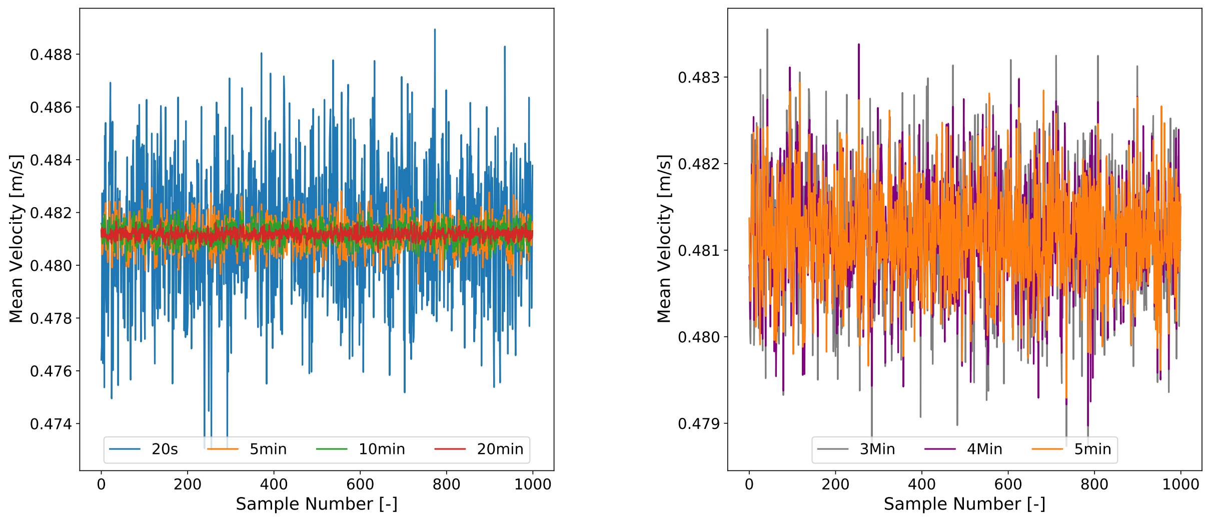


(a) Comparison of means for sampling times up to $20\,\mathrm{min}$.



(b) Comparison of means for recommended sampling times.

Figure A.38.: Comparison of means for the long-duration measurement in wall proximity and vertical position of $0.6\,\mathrm{m}$ over bottom for sampling times up to $20\,\mathrm{min}$ and for recommended sampling times.



(a) Comparison of means for sampling times up to $20\,\mathrm{min}$.

(b) Comparison of means for recommended sampling times.

Figure A.39.: Comparison of means for the long-duration measurement in central horizontal position and vertical position of $0.6\,\mathrm{m}$ over bottom for sampling times up to $20\,\mathrm{min}$ and for recommended sampling times.

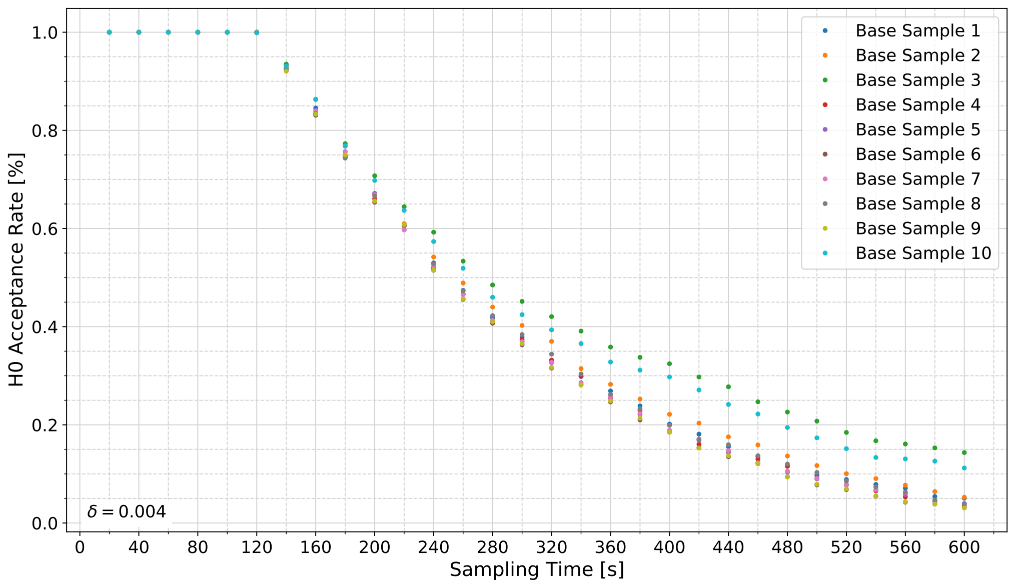


Figure A.40.: Null hypothesis acceptance rates for the long-duration measurement in wall proximity and vertical position of $0.1\,\mathrm{m}$ over bottom at $\delta = 0.004$.

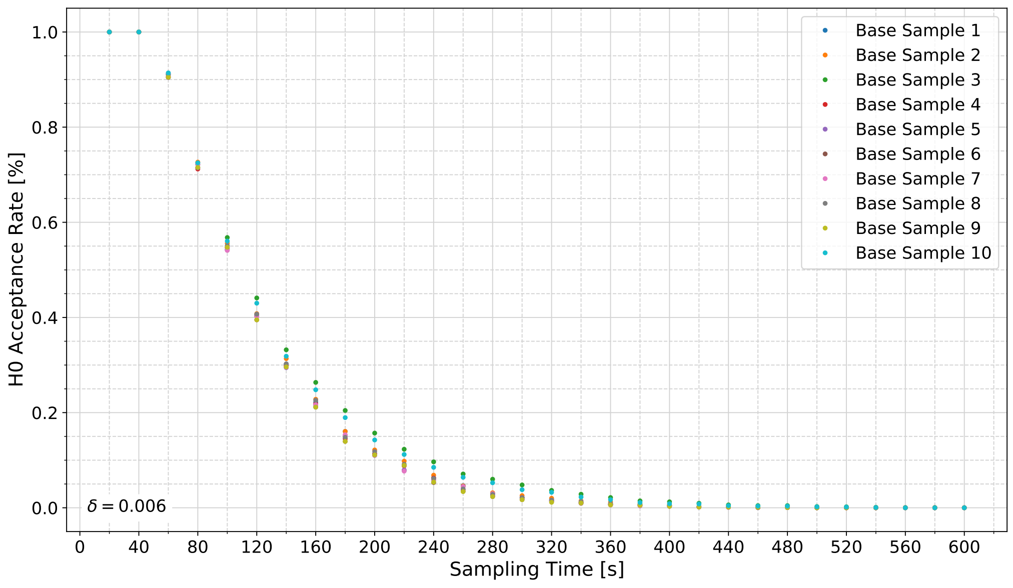


Figure A.41.: Null hypothesis acceptance rates for the long-duration measurement in wall proximity and vertical position of $0.1\,\mathrm{m}$ over bottom at $\delta = 0.006$.

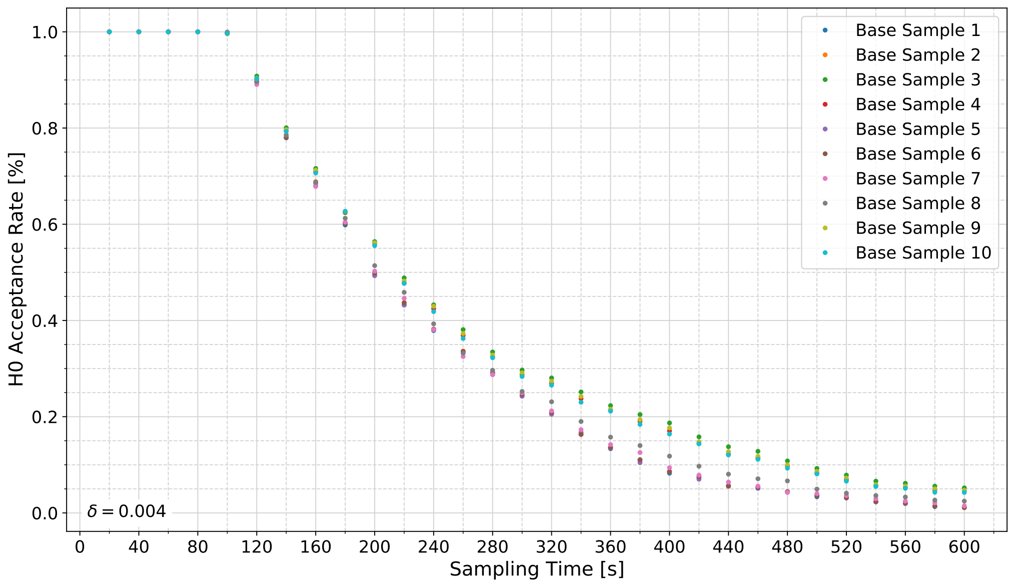


Figure A.42.: Null hypothesis acceptance rates for the long-duration measurement in central horizontal position and vertical position of $0.1\,\mathrm{m}$ over bottom at $\delta = 0.004$.

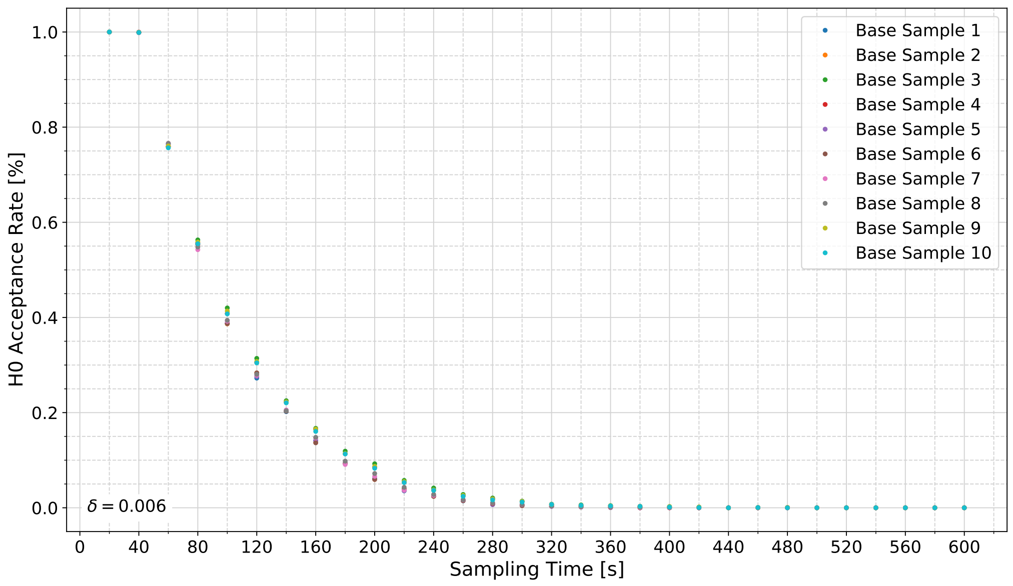


Figure A.43.: Null hypothesis acceptance rates for the long-duration measurement in central horizontal position and vertical position of $0.1\,\mathrm{m}$ over bottom at $\delta = 0.006$.

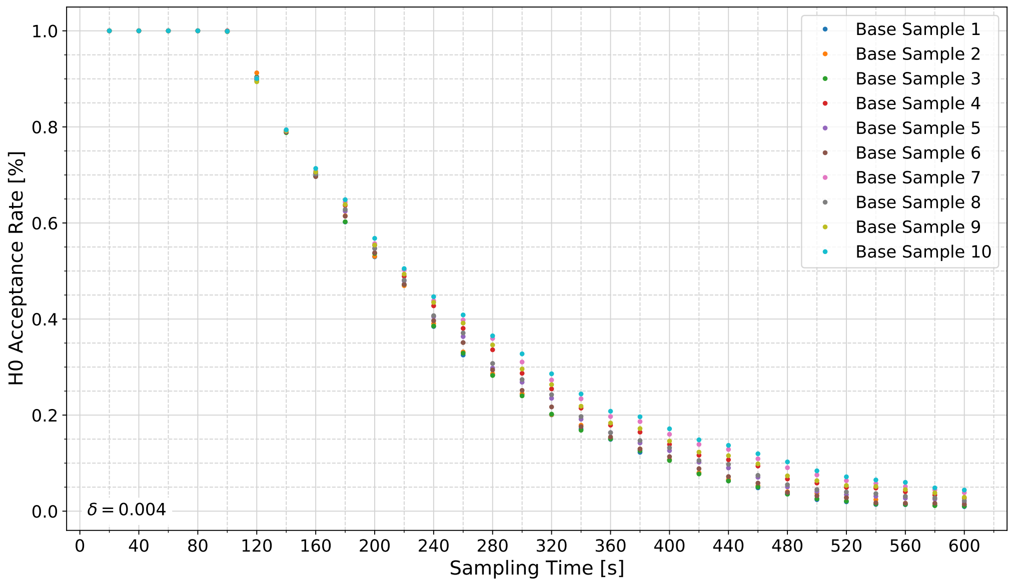


Figure A.44.: Null hypothesis acceptance rates for the long-duration measurement in wall proximity and vertical position of $0.4\,\mathrm{m}$ over bottom at $\delta = 0.004$.

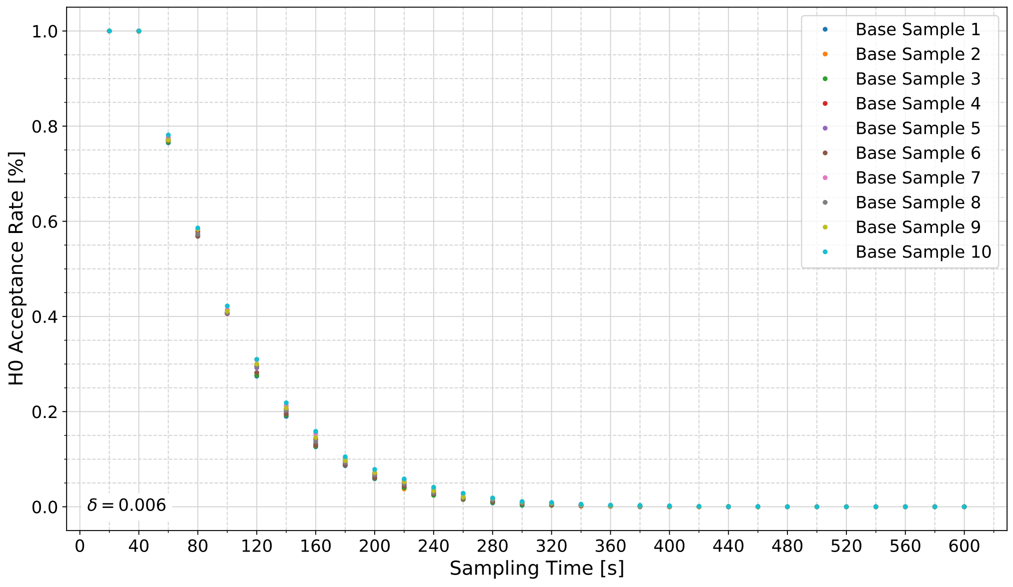


Figure A.45.: Null hypothesis acceptance rates for the long-duration measurement in wall proximity and vertical position of $0.4\,\mathrm{m}$ over bottom at $\delta = 0.006$.

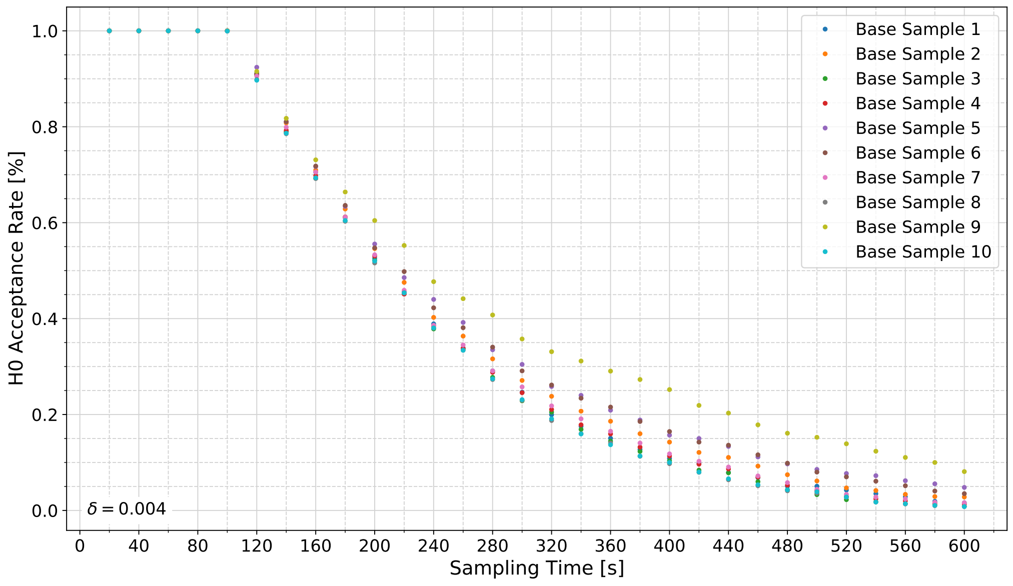


Figure A.46.: Null hypothesis acceptance rates for the long-duration measurement in wall proximity and vertical position of $0.6\,\mathrm{m}$ over bottom at $\delta = 0.004$.

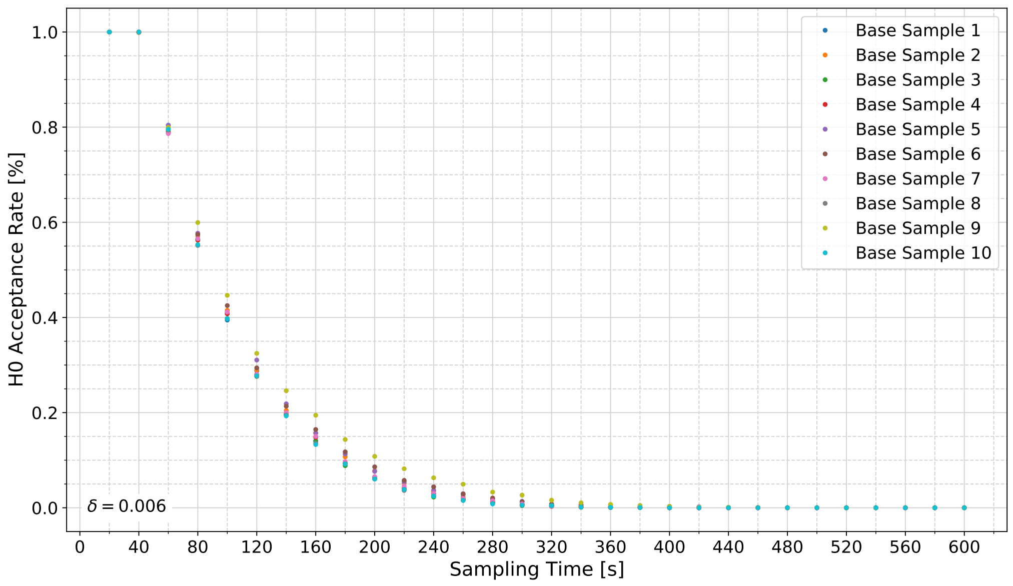


Figure A.47.: Null hypothesis acceptance rates for the long-duration measurement in wall proximity and vertical position of $0.6\,\mathrm{m}$ over bottom at $\delta = 0.006$.

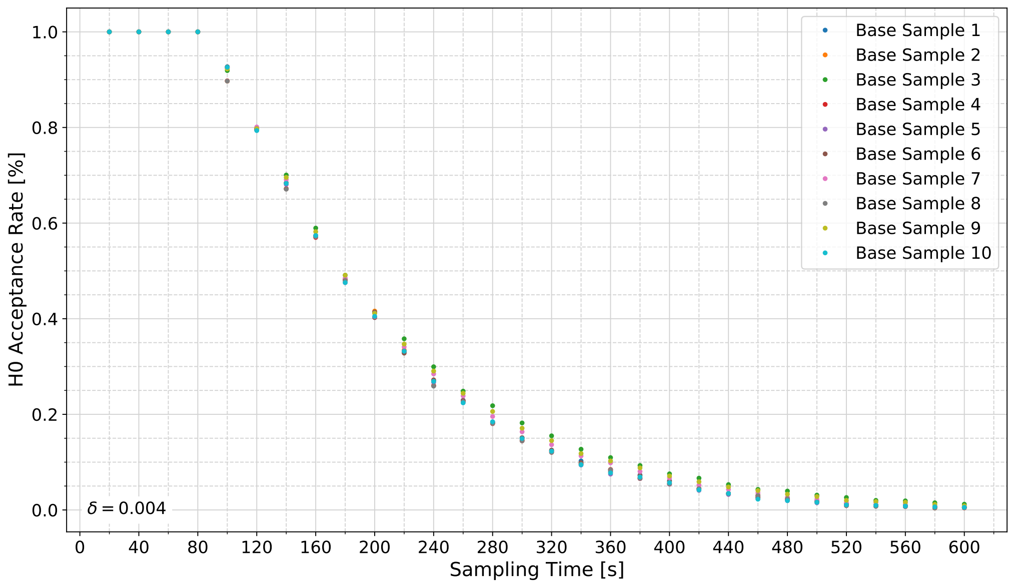


Figure A.48.: Null hypothesis acceptance rates for the long-duration measurement in central horizontal position and vertical position of $0.6\,\mathrm{m}$ over bottom at $\delta = 0.004$.

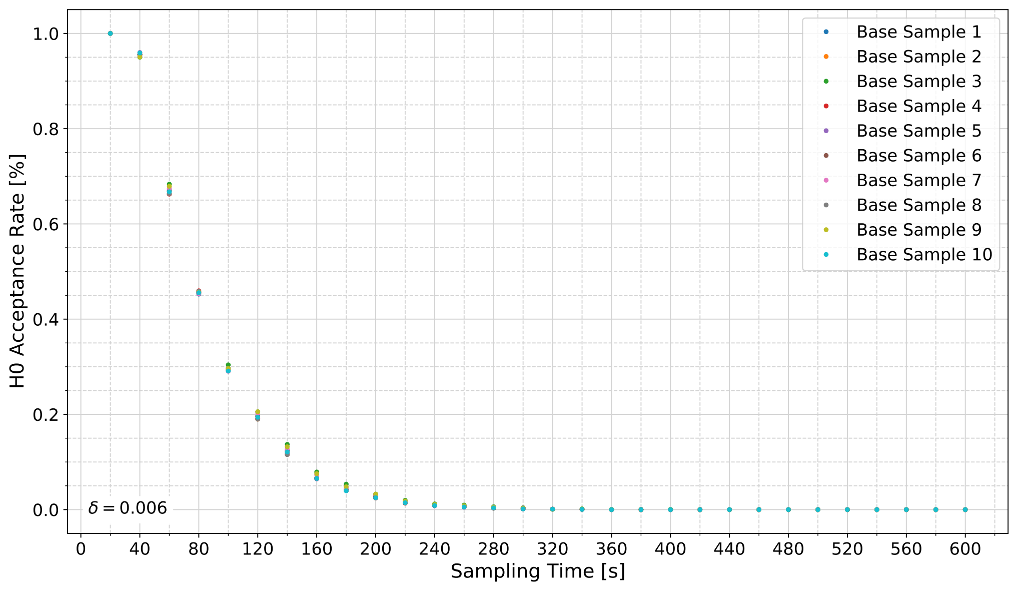


Figure A.49.: Null hypothesis acceptance rates for the long-duration measurement in central horizontal position and vertical position of $0.6\,\mathrm{m}$ over bottom at $\delta = 0.006$.

A.2. Code

```python
#Input
directory_dat = r'F:\Uni\Masterthesis\Python\Daten\Vortex_Shedding\
    Messung_04_ohne\dat'
directory_csv = r'F:\Uni\Masterthesis\Python\Daten\Vortex_Shedding\
    Messung_04_ohne\csv'
#Script
all_files = glob.glob(directory_dat + "/*.dat")
for filename in all_files: #converts all .dat files in directory_dat to .csv
    files and saves them in directory_csv
  df = pd.read_csv(filename, delimiter = ' ', skipinitialspace=True)
  df.to_csv(r'{}\{}.csv'.format(directory_csv, os.path.basename(filename).
    split('.')[0]), index=False, sep=',')
```

Source Code A.1: Script ”Make_csv_from_dat”

```python
def get_raw_data(path,fs): #gets raw data converted to .csv files
  dataframe = pd.read_csv(path,
  names=['Time','Ensamble Counter','Status','u','v','w1','w2','Amp_x','Amp_y',
    'Amp_z','Amp_z2','SNR_x','SNR_y','SNR_z','SNR_z2','COR_x','COR_y','COR_z',
    'COR_z2'])
  dataframe['index_col'] = dataframe.index #makes index start at 0
  dataframe['Time'] = dataframe['index_col']*(1/fs) #calculates the
    measurement time steps, necessary because device time steps are not reset
  dataframe = dataframe.set_index('Time') #makes time steps the index
  return dataframe
```

Source Code A.2: Method ”get_raw_data”

```python
def get_KDE_G_B_data(path,fs): #gets data output by KDE, Gauss or Butterworth
    scripts
  df = pd.read_csv(path, names=['u_out','v_out','w_out'])
  df['index_col'] = df.index
  df['Time'] = df['index_col']*(1/fs)
  df = df.set_index('Time')
  df = df.drop(columns=['index_col'])
  return df
```

Source Code A.3: Method ”get_KDE_G_B_data”

```python
def get_PSAT_data(path): #gets data output by KDE, Gauss or Butterworth
    scripts
  df = pd.read_csv(path, names=['Time','u_out','v_out','w_out'], skiprows=1)
  df = df.set_index('Time')
  return df
```

Source Code A.4: Method ”get_PSAT_data”

```python
def get_data(input_file_state, path, fs): #gets relevant arrays according to
    user-specified variable "input_file_state"
  if input_file_state == 1: #get_raw_data
    df = get_raw_data(path, fs)
    u_df = df.iloc[:,2]
    v_df = df.iloc[:,3]
    w1_df = df.iloc[:,4]
  if input_file_state == 2: # get_PSAT_data
    df = get_PSAT_data(path)
    u_df = df.iloc[:,0]
    v_df = df.iloc[:,1]
    w1_df = df.iloc[:,2]
  if input_file_state == 3: #get_KDE_G_B_data
    df = get_KDE_G_B_data(path, fs)
    u_df = df.iloc[:,0]
    v_df = df.iloc[:,1]
    w1_df = df.iloc[:,2]
  u = pd.Series.to_numpy(u_df) #convert to array
  v = pd.Series.to_numpy(v_df) #convert to array
  w = pd.Series.to_numpy(w1_df) #convert to array
  res = np.sqrt(u**2 + v**2 + w**2) #calculate velocity magnitude
  x_t = df.index.values #get time steps for x-axis
  return u, v, w, res, df, x_t
```
Source Code A.5: Method ”get_data”

```python
#Input
directory = r"F:\Uni\Masterthesis\Python\Filtered data\2_7_Time-Series"
all_files = glob.glob("{}/*.png".format(directory))
filenames = get_filenames_to_sort(all_files)
filter_name = "AccelerationThresh_Butterworth_2.5Hz_order1_spectra"
directory_raw = r"F:\Uni\Masterthesis\Python\Daten\Vortex_Shedding\
    Messung_04_ohne\csv"
destination = "ohne"
all_files_raw = glob.glob("{}/*.csv".format(directory_raw))
filenames_raw = get_filenames_original_order(all_files_raw)
#Script
for e in filenames: #moves files from directory to destination directory while
     considering structure of directory_raw
  try:
    if e in filenames_raw:
      shutil.move(r"{}\{}{}.png".format(directory, e, filter_name), r"
    {}\{}\{}{}.png".format(directory, destination, e, filter_name))
  except BaseException as error:
    print('An exception occurred:{}'.format(error))
```
Source Code A.6: Script ”File_mover”

```python
#Input
directory_dat_in = r'F:\Uni\Masterthesis\Python\Daten\Vortex_Shedding\
    Messung_04_ohne\dat'
directory_dat_out = r'F:\Uni\Masterthesis\Python\Daten\Vortex_Shedding\
    Messung_04_ohne\dat_psat'
#Script
all_files = glob.glob(directory_dat_in + "/*.dat")
for filename in all_files: #converts all .dat files in directory_dat_in to dat
     files that can be input to P-SAT framework
  df = pd.read_csv(filename, delimiter = ' ', skipinitialspace=True, header=
    None)
  df.iloc[:, 0] = round(df.iloc[:, 0] - df.iloc[0,0],3)
  df.iloc[:, 1] = df.iloc[:, 1] - df.iloc[0,1] + 1
  df.to_csv(r'{}\{}.dat'.format(directory_dat_out, os.path.basename(filename).
    split('.')[0]), index=False, sep=',', header=None)
```

Source Code A.7: Script "Make_dat_for_P-SAT_from_dat"

```python
#Input
directory = r"F:\Uni\Masterthesis\Python\PSAT"
all_files = glob.glob("{}/*.dat".format(directory))
#Script
filenames = get_filenames(all_files)
for e in filenames:
  print(e + '.dat')
```

Source Code A.8: Script "Filenames_P-SAT"

```python
def get_filenames(all_files): #gets filenames in directory and elimates
    duplicates
  li_basenames_duplicates = []
  for filename in all_files:
    basenames = os.path.basename(filename).split('.')[0]
    basenames = basenames.split('_')[0] + '_' + basenames.split('_')[1] + '_'
     + basenames.split('_')[2]
    li_basenames_duplicates.append(basenames)
  li_basenames = []
  for i in li_basenames_duplicates:
    if i not in li_basenames:
      li_basenames.append(i)
  return li_basenames
```

Source Code A.9: Method "get_filenames"

```python
def calculate_sigma(fs): #calculates sigma for the Gauss filter
  f50 = fs/6 #half-power frequency, equation 4.7
  sig_enum= np.log(np.sqrt(0.5)) #sigma enumerator, equation 4.6
  sig_denom= ((-2*np.pi**2)*(f50**2)) #sigma denominator, equation 4.6
  sig = (sig_enum/sig_denom)**0.5 #equation 4.6
  return sig
```

Source Code A.10: Method ”calculate_sigma”

```python
def get_x_f(x_t,fs): #gets x-axis in frequency domain
  N = len(x_t) #number of measurements
  T = len(x_t)*1/fs - 1/fs #sampling time
  x_f=np.linspace(0.0,N/(T),N) #x-axis at length of sampling frequency
  return x_f
```

Source Code A.11: Method ”get_x_f”

```python
def calculate_R_f(sig, x_f): #calculates frequency response
  R_f = np.exp((-2*np.pi**2)*(sig**2)*(x_f**2)) #equation 4.8
  return R_f
```

Source Code A.12: Method ”calculate_R_f”

```python
def gauss_filter(u, v, w, R_f): #operates Gauss filter
  in_li = [u,v,w]
  out_li= []
  for e in in_li:
    get_fft = fft(e) # gets signal in frequency domain
    output_fft = R_f * get_fft #multiplies signal with filter in freq domain
    out = abs(ifft(output_fft)) # get filtered signal in time domain
    out_li.append(out)
  return out_li
```

Source Code A.13: Method ”gauss_filter”

```python
#Input
directory = r"F:\Uni\Masterthesis\Python\Gauss" #directory that contains the
    files to be filtered
all_files = glob.glob("{}/*.csv".format(directory)) #gets filenames of the
    files to be filtered
input_file_state = 1 #user-specified state of input files (1 = raw, 2 = P-SAT,
     3 = KDE, Gauss, Butterworth)
fs = 25 #sampling frequency
#Script
for filename in all_files:
  try:
    #Get data
    u, v, w, res, df, x_t = get_data(input_file_state, filename, fs)
    x_f= get_x_f(x_t,fs)  #x-axis frequency domain
    #Get sigma
    sig = calculate_sigma(fs)
    #Frequency response in freq domain
    R_f = calculate_R_f(sig, x_f)
    #Filter all components
    out_li = gauss_filter(u, v, w, R_f)
    #Output to dataframe
    output = get_output_df(out_li)
    #Output to CSV
    output_to_csv(output, directory, os.path.basename(filename).split('.')[0])
  except BaseException as error: #exceptions are not filtered
    print('An exception occurred for the file {}: {}.'.format(filename, error)
   )
```

Source Code A.14: Script ”Gauss”

```
def normalize_cutoff(cutoff, fs):
  nyq = 0.5*fs #calculates nyquist frequency
  n_cutoff = cutoff / nyq #normalises cutoff frequency
  return n_cutoff
```

Source Code A.15: Method ”normalize_cutoff”

```
def butterworth_filter(u, v, w, order, n_cutoff):
  in_li = [u,v,w] #gets components as list to make them iterable
  out_li= []
  for e in in_li: #iterates over components
    b, a = butter(order, n_cutoff, btype='low', analog=False) #b = numerator,
   a = denominator, btype sets filter prototype, digital filter --> analog =
   false
    out = lfilter(b, a, e) #filters data along one-dimension
    out_li.append(out) #appends filtered components to out_li
  return out_li
```

Source Code A.16: Method ”butterworth_filter”

```
#Input
directory = r"F:\Uni\Masterthesis\Python\Butterworth" #directory that contains
     the files to be filtered
all_files = glob.glob("{}/*.csv".format(directory)) #gets filenames of the
    files to be filtered
input_file_state = 3 #user-specified state of input files (1 = raw, 2 = P-SAT,
     3 = KDE, Gauss, Butterworth)
fs=25 #sampling frequency
order = 1 #filter order
cutoff = 2.5 #cutoff frequency in Hz
#Script
for filename in all_files:
  try:
    #Script
    #Get data
    u, v, w, res, df, x_t = get_data(input_file_state, filename, fs)
    #Normalize cutoff.
    n_cutoff = normalize_cutoff(cutoff, fs)
    #Filter all components
    out_li = butterworth_filter(u, v, w, order, n_cutoff)
    #Output to dataframe
    output = get_output_df(out_li)
    #Output to CSV
    output_to_csv(output, directory, os.path.basename(filename).split('.')[0],
     cutoff, order)
  except BaseException as error: #exceptions are not filtered
    print('An exception occurred for the file {}: {}.'.format(filename, error)
   )
```

Source Code A.17: Script ”Butterworth”

```
# Copyright (c) 2013, Md Rashedul Islam
# Copyright (c) 2010, Barry Dillon
# Copyright (c) 2015, Dr. Zdravko Botev
# All rights reserved.
# Redistribution and use in source and binary forms, with or without
   modification, are permitted provided that the following conditions are met
   :
# * Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.
# * Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution
# * Neither the name of University of Alberta nor the names of its
   contributors may be used to endorse or promote products derived from this
   software without specific prior written permission.
# * Neither the name of The University of New South Wales nor the names of its
    contributors may be used to endorse or promote products derived from this
    software without specific prior written permission.
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
   AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
   IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
   LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
   CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
   SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
   INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
   CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
   ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
    POSSIBILITY OF SUCH DAMAGE.

# Copyright (C) 2012-present, The Computational Data Science Lab
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
   IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
```

Source Code A.18: Copyright Notices

```
def get_kde_input(e):
  #Step 1: derivatives
  df = [0.0]*n
  db = [0.0]*n
  du = [0.0]*n
  for i in range(n-1):
    df[i] = e[i+1] - e[i] #forward difference equation
  for i in range(1,n):
    db[i] = e[i] - e[i-1] #backward difference equation
  for i in range(len(e)): #exceptions
    if abs(db[i]) > abs(df[i]):
      du[i] = df[i]
    else:
      du[i] = db[i]
  u1 = np.asarray(e, dtype=float)
  w1 = np.asarray(du, dtype=float)

  #Step 2: Axis rotation estimation
  th = np.arctan2((n*sum(np.multiply(u1,w1))-sum(u1)*sum(w1)),(n*sum(np.
   multiply(u1,u1))-sum(u1)*sum(u1)))

  #Step 3 : Data transformation
  ut = (u1)*np.cos(th) + (w1)*np.sin(th)
  wt = (w1)*np.cos(th) - (u1)*np.sin(th)
  return ut, wt
```

Source Code A.19: Method ”get_kde_input”

```
def get_kde_output(ut, wt):
  #Step 5a: Applying kernel density function using Botev et al.(2010)'s
   algorithm
  #gets minima and maxima from input vectors
  max_ut = np.amax(ut)
  min_ut = np.amin(ut)
  max_wt = np.amax(wt)
  min_wt = np.amin(wt)
  #gets max-min for input vectors
  scaling_ut=max_ut-min_ut
  scaling_wt=max_wt-min_wt
  ##(Step 4:) Rescaling the data so that it ranges from 0 to 1.
  ut_t = (ut - min_ut)/scaling_ut
  wt_t = (wt - min_wt)/scaling_wt
```

```python
#Bin data uniformly using regular grid
(binned, xedges, yedges) = np.histogram2d(ut_t, wt_t, bins=N, density=False)
  # range=((min_ut, max_ut), (min_wt, max_wt))
#Discrete cosine transform
a = dctn(binned/n)
a[0, :] /= 2
a[:, 0] /= 2
#Bandwidth
t_y = hy**2
t_x = hx**2
# # Smooth the discrete cosine transform of initial data
k  = np.arange(N, dtype="float")
k2 = k**2
smoothed = a * np.outer(np.exp(-np.pi**2 * k2 * t_x/2), np.exp(-np.pi**2 *
 k2 * t_y/2)) #equation 4.2
# # Apply the inverse discrete cosine transform
smoothed[0, :] *= 2
smoothed[:, 0] *= 2
inverse = idctn(smoothed)
density = np.transpose(inverse) * N/scaling_ut * N/scaling_wt # density
 matrix
#Get meshgrid
#vector x
a_start = min_ut #start a
a_stop = max_ut #stop a
a_step = scaling_ut/(N-1) #value per step for a
a = np.arange(a_start,a_stop,a_step, float) # looses endpoint --> gives only
  array with length 255
endpoint = (a[254]+a_step) # calculate endpoint
a = np.append(a,endpoint) #append endpoint to array
#vector y
b_start = min_wt #start b
b_stop = max_wt #stop b
b_step = scaling_wt/(N-1) #value per step for b
b = np.arange(b_start,b_stop,b_step, float) # looses endpoint --> gives only
  array with length 255
endpoint = (b[254]+b_step) # calculate endpoint
b = np.append(b,endpoint) #append endpoint to array --> array with length
 256
#meshgrid
X, Y = np.meshgrid(a,b)
# #Get outputs from kde
uf = np.diag(X)
wf = np.diag(Y)
return wf, uf, density
```

Source Code A.20: Method ”get_kde_output”

```python
def get_spike_Id(wf, uf, density, ut, wt):
  #Step 5b: Get maximum density values
  dp = np.amax(np.amax(density))
  #Get indices of maximum density
  t_wp_up = np.where(density==dp) # get array indices
  l_wp_up = list(t_wp_up) #turn from tuple to array
  wp = l_wp_up[0] #get first column of indices
  up = l_wp_up[1] #get second column of indices
  #Step6a: Get slopes and iteration specifics, this part becomes clear when
   looking at figure 4.2
  #u
  f_u = density[wp,:]
  lf_f_u = len(np.transpose(f_u)) #get length for row axis
  diff_f_u = np.diff(f_u,axis=1) #difference quotient
  diff_f_u_0 = np.insert(diff_f_u,0,0, axis=1) #add zero at index 0 --> no
   delta possible here
  dk_u = diff_f_u_0*N/dp # slope for velocity component
  #w
  f_w = np.transpose(density[:,up]) #transpose, so that array is normal to f_u
  lf_f_w = len(np.transpose(f_w)) #get length for row axis
  diff_f_w = np.diff(f_w,axis=1) #difference quotient
  diff_f_w_0 = np.insert(diff_f_w,0,0, axis=1) #add zero at index 0 --> no
   delta possible here
  dk_w = diff_f_w_0*N/dp # slope for derivative
  #Get i for loops within grid boundaries
  i_1_u = np.arange(up,1,-1, float)
  i_1_u_li = i_1_u.tolist()
  i_1_u_li_int = [int(i) for i in i_1_u_li]
  i_2_u = np.arange(up+2,lf_f_u,1, float)
  i_2_u_li = i_2_u.tolist()
  i_2_u_li_int = [int(i) for i in i_2_u_li]
  i_1_w = np.arange(wp,1,-1, float)
  i_1_w_li = i_1_w.tolist()
  i_1_w_li_int = [int(i) for i in i_1_w_li]
  i_2_w = np.arange(wp+2,lf_f_w,1, float)
  i_2_w_li = i_2_w.tolist()
  i_2_w_li_int = [int(i) for i in i_2_w_li]
  #Step6b: Calculate cutoff points: iterates over the indices and determines
   where condition is met first for f_u and dk_u
  for i in i_1_u_li_int:
    if f_u[:,i]/f_u[:,int(up)] <= c1 and np.absolute(dk_u[:,i]) <= c2:
      i_1_a_u = i
      break
    else:
      i_1_a_u = i_1_u_li_int[-1]
  for i in i_1_w_li_int:
    if f_w[:,i]/f_w[:,int(wp)] <= c1 and np.absolute(dk_w[:,i]) <= c2:
      i_1_a_w = i
      break
    else:
      i_1_a_w = i_1_w_li_int[-1]
  for i in i_2_u_li_int:
    if f_u[:,i]/f_u[:,int(up)] <= c1 and np.absolute(dk_u[:,i]) <= c2:
      i_2_a_u = i
      break
    else:
      i_2_a_u = i_2_u_li_int[-1]
  for i in i_2_w_li_int:
    if f_w[:,i]/f_w[:,int(wp)] <= c1 and np.absolute(dk_w[:,i]) <= c2:
      i_2_a_w = i
      break
    else:
      i_2_a_w = i_2_w_li_int[-1]
  u_i1 = uf[i_1_a_u] # =ul
  u_i2 = uf[i_2_a_u] # =uu
  w_i1 = wf[i_1_a_w] # =wl
  w_i2 = wf[i_2_a_w] # =wu
  # Step 7: Calculate axes of ellipse and identify spikes
  uu1 = u_i2 - 0.5*(u_i2+u_i1)
  wu1 = w_i2 - 0.5*(w_i2 + w_i1)
  Ut1 = ut - 0.5*(u_i2 + u_i1)
  ul1 = u_i1 - 0.5*(u_i2 + u_i1)
  wl1 = w_i1 - 0.5*(w_i2 + w_i1)
  Wt1 = wt - 0.5*(w_i2 + w_i1)
  F = [0.0]*n
  F = np.asarray(F, dtype=float)
  at = 0.5 * (uu1 - ul1)
  bt = 0.5 * (wu1 - wl1)
  for i in range(n): #sets values of zero-array to 1 if condition for spike is
     met
    if Ut1[i] > uu1 or Ut1[i] < ul1:
      F[i] = 1
    else:
      we = np.sqrt((bt**2)*(1-(Ut1[i]**2)/(at**2)))
      if Wt1[i] > we or Wt1[i] < -we:
        F[i] = 1
  Id_tuple = np.where(F>0) #find flagged values
  Id = np.asarray(Id_tuple[0])
  return Id
```

Source Code A.21: Method ”get_spike_Id”

```python
def KDE_despike(u_li): #summarises all KDE methods into one method that gives
   the spike IDs
  Id_li= []
  for e in u_li:
    ut, wt = get_kde_input(e)
    wf, uf, density = get_kde_output(ut, wt)
    Id = get_spike_Id(wf, uf, density, ut, wt)
    Id_li.append(Id)
  return Id_li
```

Source Code A.22: Method ”KDE_despike”

```python
def replace_spike_lin_interpolation(Id_li, u, v, w):
  u_out = u.tolist()
  u_out = pd.DataFrame(u_out, columns=['u_out'])
  v_out = v.tolist()
  v_out = pd.DataFrame(v_out, columns=['v_out'])
  w_out = w.tolist()
  w_out = pd.DataFrame(w_out, columns=['w_out'])
  #Check if first and/or last data points are spikes, replace by mean, delete
   Id from list
  if Id_li[0][0] == 0:
    u_out.iloc[0] = np.mean(u_out)
    Id_li[0] = np.delete(Id_li[0],0)
  if Id_li[0][len(Id_li[0])-1] == (n-1):
    u_out.iloc[n-1] = np.mean(u_out)
    Id_li[0] = np.delete(Id_li[0],-1)
  if Id_li[2][0] == 0:
    w_out.iloc[0] = np.mean(w_out)
    Id_li[2] = np.delete(Id_li[2],0)
  if Id_li[2][len(Id_li[2])-1] == (n-1):
    w_out.iloc[n-1] = np.mean(w_out)
    Id_li[2] = np.delete(Id_li[2],-1)
  if Id_li[1][0] == 0:
    v_out.iloc[0] = np.mean(v_out)
    Id_li[1] = np.delete(Id_li[1],0)
  if Id_li[1][len(Id_li[1])-1] == (n-1):
    v_out.iloc[n-1] = np.mean(v_out)
    Id_li[1] = np.delete(Id_li[1],-1)
  #Replaces in u and w (Beams 1 and 3)
  for j in [0,2]:
    for i in Id_li[[j][0]]:
      lgv = i-1 #last good value
      ngv = i+1 #next good value
      while lgv in Id_li[[j][0]]: #checks if ID of lgv is a spike
        lgv -= 1
      while ngv in Id_li[[j][0]]: #checks if ID of ngv is a spike
        ngv += 1
      u_out.iloc[i,0] = (u_out.iloc[lgv,0] + u_out.iloc[ngv,0])/2
      w_out.iloc[i,0] = (w_out.iloc[lgv,0] + w_out.iloc[ngv,0])/2
  #Replaces in v (Beams 2 and 4)
  for i in Id_li[[1][0]]: # replaces in v
    lgv = i-1 #last good value
    ngv = i+1 #next good value
    while lgv in Id_li[[1][0]]: #checks if ID of lgv is a spike
      lgv -= 1
    while ngv in Id_li[[1][0]]: #checks if ID of ngv is a spike
      ngv += 1
    v_out.iloc[i,0] = (v_out.iloc[lgv,0] + v_out.iloc[ngv,0])/2
  output = pd.concat([u_out, v_out, w_out], axis=1)
  if output.isnull().sum().sum() > 0: #check if there are nan values in output
     dataframe
    sys.exit("Filtering not successful. Check whether input file state
   corresponds to input file. If yes, check other input parameters.")
  return output
```

Source Code A.23: Method ”replace_spike_lin_interpolation”

```python
#Input
directory = r"F:\Uni\Masterthesis\Python\KDE" #directory that contains the
    files to be filtered
all_files = glob.glob("{}/*.csv".format(directory)) #gets filenames of the
    files to be filtered
input_file_state = 3 #user-specified state of input files (1 = raw, 2 = P-SAT,
     3 = KDE, Gauss, Butterworth)
fs = 25 #sampling frequency
hx = 0.01 #bandwidth (recommended in Islam and Zhu, 2013)
hy = 0.01 #bandwidth (recommended in Islam and Zhu, 2013)
N = 256 #number of bins (recommended in Islam and Zhu, 2013)
c1 = 0.4 #cutoff threshold (recommended in Islam and Zhu, 2013)
c2 = 0.4 #cutoff threshold (recommended in Islam and Zhu, 2013)
#Script
for filename in all_files:
  try:
    #Get data
    u, v, w, res, df, x_t = get_data(input_file_state, filename, fs)
    u_li = [u,v,w]
    n = len(u)
    #KDE despiking
    Id_li = KDE_despike(u_li)
    #Replace spikes
    output = replace_spike_lin_interpolation(Id_li, u, v, w)
    #Output to CSV
    output_to_csv(output, directory, os.path.basename(filename).split('.')[0])
  except BaseException as error: #exceptions are not filtered
    print('An exception occurred for the file {}: {}.'.format(filename, error)
   )
```

Source Code A.24: Script ”KDE_despiking”

```python
#Input
directory = r"F:\Uni\Masterthesis\Python\PDF Bilder" #directory containing the
    original and filtered data files
all_files = glob.glob("{}/*.csv".format(directory)) #gets filenames of all
    files
filter_name = "KDE_Butterworth_2.5Hz_order1" #specifies filenames of all files
input_file_state_original = 1 #user-specified state of input files (1 = raw, 2
    = P-SAT, 3 = KDE, Gauss, Butterworth)
input_file_state_filtered = 3 #user-specified state of input files (1 = raw, 2
    = P-SAT, 3 = KDE, Gauss, Butterworth)
fs = 25 #sampling frequency
#Script
filenames = get_filenames(all_files) # eliminates duplicates from filename
for filename in filenames:
  path_raw, path_filtered = get_file_paths(directory, filename, filter_name)
  u_original, v_original, w_original, res_original, df_original, x_t_original
   = get_data(input_file_state_original, path_raw, fs)
  u_filtered, v_filtered, w_filtered, res_filtered, df_filtered, x_t_filtered
   = get_data(input_file_state_filtered, path_filtered, fs)
  #Plotting Comparison of Velocity Magnitudes
  fig = plt.figure()
  fig.set_figheight(9)
  fig.set_figwidth(16)
  line1, = plt.plot(x_t_original, res_original, 'tab:brown', label='
   res_original')
  line2, = plt.plot(x_t_filtered, res_filtered, 'tab:blue', label='
   res_filtered')
  y_ticks = np.arange(0,4.5,0.5)
  plt.yticks(y_ticks)
  plt.legend([line1,line2],["res_original", "res_filtered"], loc="upper right"
   , prop=dict(size=14))
  plt.xlabel("Time [s]", fontsize=16)
  plt.ylabel("Velocity [m/s]", fontsize=16)
  plt.xticks(fontsize=14)
  plt.yticks(fontsize=14)
  plt.savefig('{}_{}_res.pdf'.format(filename, filter_name), bbox_inches='
   tight')
  plt.close(fig)
  #Plotting Comparison of all velocities (u,v,w,res)
  fig = plt.figure()
  gs = fig.add_gridspec(4, hspace=0.06)
  axs = gs.subplots(sharex=True)
  fig.set_figheight(18)
  fig.set_figwidth(14)
  l5, = axs[0].plot(x_t_original, res_original,'tab:brown', label='
   res_original')
  l0, = axs[0].plot(x_t_filtered, res_filtered, 'tab:blue', label='
   res_filtered', )
  l1, = axs[1].plot(x_t_original, u_original, 'tab:brown', label='u_original')
  l1, = axs[1].plot(x_t_filtered, u_filtered, 'tab:green', label='u_filtered')
  l2, = axs[2].plot(x_t_original, v_original, 'tab:brown', label='v_original')
  l2, = axs[2].plot(x_t_filtered, v_filtered, 'tab:red', label='v_filtered')
  l3, = axs[3].plot(x_t_original, w_original, 'tab:brown', label='w_original')
  l3, = axs[3].plot(x_t_filtered, w_filtered, 'tab:orange', label='w_filtered'
    )
  plt.legend([l5, l0, l1, l2, l3,],["original","res_filtered","u_filtered", "
    v_filtered", "w_filtered"], loc="lower center", ncol = 5, prop=dict(size
    =14))
  axs[0].set_yticks([0,2,4])
  axs[1].set_yticks([-4,-2,0,2,4])
  axs[2].set_yticks([-2,0,2])
  axs[3].set_yticks([-1,0,1])
  for ax in axs:
  ax.set_xlabel("Time [s]", fontsize=16)
  ax.set_ylabel("Velocity [m/s]", fontsize=16)
  ax.label_outer()
  ax.tick_params(axis='both', labelsize=14)
  plt.savefig('{}_{}_all.pdf'.format(filename, filter_name), bbox_inches='
    tight')
  plt.close(fig)
```

Source Code A.25: Script ”Time-series_Visualisation”

```python
#Input
directory = r"F:\Uni\Masterthesis\Python\Filtered data\2_7_Spectra" #directory
     containing the original and filtered data files
all_files = glob.glob("{}/*.csv".format(directory)) #gets filenames of all
    files
filter_name = "KDE" #specifies filenames of all files
input_file_state_original = 1 #user-specified state of input files (1 = raw, 2
     = P-SAT, 3 = KDE, Gauss, Butterworth)
input_file_state_filtered = 3 #user-specified state of input files (1 = raw, 2
     = P-SAT, 3 = KDE, Gauss, Butterworth)
fs = 25 #sampling frequency
#Script
filenames = get_filenames(all_files) # eliminates duplicates from filename
for filename in filenames:
  #Gets file paths for original and filtered data
  path_raw, path_filtered = get_file_paths(directory, filename, filter_name)
  #Gets data for original and filtered data
  u_original, v_original, w_original, res_original, df_original, x_t_original
    = get_data(input_file_state_original, path_raw, fs)
  u_filtered, v_filtered, w_filtered, res_filtered, df_filtered, x_t_filtered
    = get_data(input_file_state_filtered, path_filtered, fs)
  #Computing PSD original samples
  freqs_u_original, psd_u_original = compute_PSD(u_original, fs)
  freqs_v_original, psd_v_original = compute_PSD(v_original, fs)
  freqs_w_original, psd_w_original = compute_PSD(w_original, fs)
```

```
freqs_res_original, psd_res_original = compute_PSD(res_original, fs)
#Computing PSD filtered samples
freqs_u_filtered, psd_u_filtered = compute_PSD(u_filtered, fs)
freqs_v_filtered, psd_v_filtered = compute_PSD(v_filtered, fs)
freqs_w_filtered, psd_w_filtered = compute_PSD(w_filtered, fs)
freqs_res_filtered, psd_res_filtered = compute_PSD(res_filtered, fs)
#Get x_f original
x_f_original = get_x_f(x_t_original, fs)
#Get x_f filtered
x_f_filtered = get_x_f(x_t_filtered, fs)
#Get Kolmogorov -5/3 slopes for slope grid
x, y1, y2, y3, y4, y5, y6, y7, y8, y9, y10, y11, y12, y13, y14 =
  get_Kolmogorov_slope()
#Plotting
fig, ax = plt.subplots(figsize=(8, 8))
#slope grid
line9, = plt.plot(x, y1, 'lightgray', label='-5/3 slope', linestyle='dashed'
  )
line10, = plt.plot(x, y2, 'lightgray', linestyle='dashed')
line11, = plt.plot(x, y3, 'lightgray', linestyle='dashed')
line12, = plt.plot(x, y4, 'lightgray', linestyle='dashed')
line13, = plt.plot(x, y5, 'lightgray', linestyle='dashed')
line14, = plt.plot(x, y6, 'lightgray', linestyle='dashed')
line15, = plt.plot(x, y7, 'lightgray', linestyle='dashed')
line16, = plt.plot(x, y8, 'lightgray', linestyle='dashed')
line17, = plt.plot(x, y9, 'lightgray', linestyle='dashed')
line18, = plt.plot(x, y10, 'lightgray', linestyle='dashed')
line19, = plt.plot(x, y11, 'lightgray', linestyle='dashed')
line20, = plt.plot(x, y12, 'lightgray', linestyle='dashed')
line21, = plt.plot(x, y13, 'lightgray', linestyle='dashed')
line22, = plt.plot(x, y14, 'lightgray', linestyle='dashed')
first_legend = ax.legend(handles=[line9], loc='lower center', prop=dict(size
  =14))
ax.add_artist(first_legend)
#original
line1, = plt.plot(freqs_u_original, psd_u_original, 'tab:green', label='
  u_original', linestyle='dotted')
line2, = plt.plot(freqs_v_original, psd_v_original, 'tab:red', label='
  v_original', linestyle='dotted')
line3, = plt.plot(freqs_w_original, psd_w_original, 'tab:orange', label='
  w_original', linestyle='dotted')
line4, = plt.plot(freqs_res_original, psd_res_original, 'tab:blue', label='
  res_original', linestyle='dotted')
second_legend = ax.legend(handles=[line1,line2,line3,line4,], loc='lower
  left', prop=dict(size=14))
ax.add_artist(second_legend)
#filtered
line5, = plt.plot(freqs_u_filtered, psd_u_filtered, 'tab:green', label='
  u_filtered')
```

```python
  line6, = plt.plot(freqs_v_filtered, psd_v_filtered, 'tab:red', label='
    v_filtered')
  line7, = plt.plot(freqs_w_filtered, psd_w_filtered, 'tab:orange', label='
    w_filtered')
  line8, = plt.plot(freqs_res_filtered, psd_res_filtered, 'tab:blue', label='
    res_filtered')
  third_legend = ax.legend(handles=[line5,line6,line7,line8,], loc='lower
    right', prop=dict(size=14))
  #Specifications
  plt.xlim([0.09,13])
  plt.ylim([0.00000000001,0.1])
  plt.xlabel("Frequency [Hz]", fontsize=16)
  plt.ylabel("E(f) [(m²/s²)/Hz]", fontsize=16)
  plt.xticks(fontsize=14)
  plt.yticks(fontsize=14)
  plt.loglog()
  plt.savefig('{}_{}_spectra.pdf'.format(filename, filter_name), bbox_inches='
    tight')
  plt.close(fig)
```

Source Code A.26: Script ”Spectra_Visualisation”

```python
directory = r"F:\Uni\Masterthesis\Python\Parameter_neu" #directory that
    contains the files to calculate parameters for
all_files = glob.glob("{}/*.csv".format(directory)) #gets filenames of the
    files to be filtered
input_file_state = 2 #user-specified state of input files (1 = raw, 2 = P-SAT,
     3 = KDE, Gauss, Butterworth)
fs=25
#best use this script when only files of one input_file_state are present in
    the directory
for filename in all_files:
  try:
    u, v, w, res, df, x_t = get_data(input_file_state, filename, fs)
    # Mean
    mean_u = sum(u)/len(u)
    mean_v = sum(v)/len(v)
    mean_w = sum(w)/len(w)
    mean_res = sum(res)/len(res)
    # Global Maximum
    max_u = max(u)
    max_v = max(v)
    max_w = max(w)
    max_res = max(res)
    # Global Minimum
    min_u = min(u)
    min_v = min(v)
    min_w = min(w)
    min_res = min(res)
    # Variance
    var_u = sum([(e - mean_u)**2 for e in u]) / (len(u)-1)
    var_v = sum([(e - mean_v)**2 for e in v]) / (len(v)-1)
    var_w = sum([(e - mean_w)**2 for e in w]) / (len(w)-1)
    var_res = sum([(e - mean_res)**2 for e in res]) / (len(res)-1)
    # Standard Deviation
    std_u = np.sqrt(var_u)
    std_v = np.sqrt(var_v)
    std_w = np.sqrt(var_w)
    std_res = np.sqrt(var_res)
    #Covariance
    sum_u_v = 0
    for i in range(0, len(u)):
      sum_u_v += ((u[i] - mean_u) * (v[i] - mean_v))
    cov_u_v = sum_u_v / (len(u)-1)
    sum_u_w = 0
    for i in range(0, len(u)):
      sum_u_w += ((u[i] - mean_u) * (w[i] - mean_w))
    cov_u_w = sum_u_w / (len(u)-1)
    sum_v_w = 0
    for i in range(0, len(v)):
      sum_v_w += ((v[i] - mean_v) * (w[i] - mean_w))
    cov_v_w = sum_v_w / (len(v)-1)
    #Correlation
    cor_u_v = cov_u_v / (std_u * std_v)
    cor_u_w = cov_u_w / (std_u * std_w)
    cor_v_w = cov_v_w / (std_v * std_w)
    #Skewness
    skew_u = ((var_u)**(-3/2)) * sum([(e - mean_u)**3 for e in u]) / (len(u)
   -1)
    skew_v = ((var_v)**(-3/2)) * sum([(e - mean_v)**3 for e in v]) / (len(v)
   -1)
    skew_w = ((var_w)**(-3/2)) * sum([(e - mean_w)**3 for e in w]) / (len(w)
   -1)
    skew_res = ((var_res)**(-3/2)) * sum([(e - mean_res)**3 for e in res])/ (
   len(res)-1)
    #Kurtosis with Pearson
    kurt_u = ((var_u)**(-2)) * sum([(e - mean_u)**4 for e in u]) / (len(u)-1)
    kurt_v = ((var_v)**(-2)) * sum([(e - mean_v)**4 for e in v]) / (len(v)-1)
    kurt_w = ((var_w)**(-2)) * sum([(e - mean_w)**4 for e in w]) / (len(w)-1)
    kurt_res = ((var_res)**(-2)) * sum([(e - mean_res)**4 for e in res]) / (
   len(res)-1)
    #Turbulence kinetic energy TKE
    tke = 0.5*(var_u + var_v + var_w)
    #Arrange values
    output_data = {'mean_u':[mean_u],'var_u':[var_u], 'std_u':[std_u],
      'max_u':[max_u], 'min_u':[min_u],
      'skew_u':[skew_u], 'kurt_u':[kurt_u],
      'mean_v':[mean_v], 'var_v':[var_v], 'std_v':[std_v],
      'max_v':[max_v], 'min_v':[min_v],
      'skew_v':[skew_v], 'kurt_v':[kurt_v],
      'mean_w':[mean_w], 'var_w':[var_w], 'std_w':[std_w],
      'max_w':[max_w], 'min_w':[min_w],
      'skew_w':[skew_w], 'kurt_w':[kurt_w],
      'mean_res': [mean_res], 'var_res':[var_res], 'std_res':[std_res],
      'max_res':[max_res], 'min_res':[min_res],
      'skew_res':[skew_res], 'kurt_res':[kurt_res],
      'cov_u_v':[cov_u_v], 'cov_u_w':[cov_u_w], 'cov_v_w':[cov_v_w],
      'cor_u_v':[cor_u_v], 'cor_u_w':[cor_u_w], 'cor_v_w':[cor_v_w],
      'tke':[tke]}
    #Values to dataframe
    output_df = pd.DataFrame(output_data, index={filename})
    #Output to .csv
    points = [(i, filename[i]) for i in findall('.', filename)] #finds number of '.' in filename, necessary for denotation
    if len(points) == 2:
      #Option for file names with 2 '.' (e.g. 2.5Hz and .csv)
      output_to_csv(output_df, directory, (os.path.basename(filename).split('.')[0]+ '.' +os.path.basename(filename).split('.')[1]))
    if len(points) == 1:
      #Output to .csv for file names with only one '.' (only .csv)
      output_to_csv(output_df, directory, os.path.basename(filename).split('.')[0])
  except BaseException as error:
    print('An exception occurred for the file {}: {}.'.format(filename, error))
```

Source Code A.27: Script "Parameter_Calculation"

```
def get_parameters(directory, filename): #extracts data so that it creates a data frame for all parameters for one sample
  path_original = r"{}\{}_parameters.csv".format(directory, filename)
  path_KDE = r"{}\{}_KDE_parameters.csv".format(directory, filename)
  path_PSAT = r"{}\{}_AccelerationThresh_parameters.csv".format(directory, filename)
  path_Gauss = r"{}\{}_Gauss_parameters.csv".format(directory, filename)
  path_Butterworth = r"{}\{}_Butterworth_2.5Hz_order1_parameters.csv".format(directory, filename)
  path_KDE_Gauss = r"{}\{}_KDE_Gauss_parameters.csv".format(directory, filename)
  path_KDE_Butterworth = r"{}\{}_KDE_Butterworth_2.5Hz_order1_parameters.csv".format(directory, filename)
  path_PSAT_Gauss = r"{}\{}_AccelerationThresh_Gauss_parameters.csv".format(directory, filename)
  path_PSAT_Butterworth = r"{}\{}_AccelerationThresh_Butterworth_2.5Hz_order1_parameters.csv".format(directory, filename)
  parameters_original = pd.read_csv(path_original, index_col=0)
  parameters_KDE = pd.read_csv(path_KDE, index_col=0)
  parameters_PSAT = pd.read_csv(path_PSAT, index_col=0)
  parameters_Gauss = pd.read_csv(path_Gauss, index_col=0)
  parameters_Butterworth = pd.read_csv(path_Butterworth, index_col=0)
  parameters_KDE_Gauss = pd.read_csv(path_KDE_Gauss, index_col=0)
  parameters_KDE_Butterworth = pd.read_csv(path_KDE_Butterworth, index_col=0)
  parameters_PSAT_Gauss = pd.read_csv(path_PSAT_Gauss, index_col=0)
  parameters_PSAT_Butterworth = pd.read_csv(path_PSAT_Butterworth, index_col
   =0)
  all_parameters_df = pd.concat([parameters_original, parameters_PSAT,
   parameters_KDE, parameters_Butterworth, parameters_Gauss,
  parameters_PSAT_Gauss, parameters_PSAT_Butterworth, parameters_KDE_Gauss,
   parameters_KDE_Butterworth], axis=0)
  return all_parameters_df
```

Source Code A.28: Method ”get_parameters”

```
def count_category(li, component, method, category): #counts how often a
   category occurs for each method, used in next method "
   sort_category_counts_by_method"
  count = 0
    for i in range(len(li)):
      if li[i][component][method] == category:
        count += 1
  return count
```

Source Code A.29: Method ”count_category”

```
def sort_category_counts_by_method(number_of_components, number_of_methods,
   number_of_categories,li_cat_per_sample): #sorts mean category count by
   method
  category_by_method = []
  for e in range(number_of_components): #iterates velocity components
    category_by_method_c = []
    for f in range(number_of_methods): #iterates methods
      category_by_method_m = []
      for i in range(1,(number_of_categories+1)): #iterates categories, count
   starts at 1, i.e. ends at number_of_categories+1
        category_by_method_m.append(count_category(li_cat_per_sample,e,f,i))
      category_by_method_c.append(category_by_method_m)
    category_by_method.append(category_by_method_c)
  return category_by_method
```

Source Code A.30: Method ”sort_category_counts_by_method”

```
def categorise_means(filenames, indices, number_of_methods): #categorises the
   mean per sample for every velocity compoent
  li_mean_cat_per_sample = []
  for filename in filenames: #iterates samples
    all_parameters_df = get_parameters(directory, filename)
    li_mean_cat = []
```

```python
    for e in indices: #iterates velocity components
      li_mean_cat_component = []
      for i in range(1,(number_of_methods+1)): #iterates methods, starts at 1, because original samlpes cannot be categorised
        mean_change = ((all_parameters_df.iloc[i,e]-all_parameters_df.iloc[0,e])/all_parameters_df.iloc[0,e])*100 # Equation 6.1 Gives  percentage of realtive change in mean
        if mean_change >= -1 and mean_change <= 1:
          mean_cat = 1
        elif mean_change >= -5 and mean_change <= 5:
          mean_cat = 2
        elif mean_change >= -10 and mean_change <= 10:
          mean_cat = 3
        else:
          mean_cat = 4
        li_mean_cat_component.append(mean_cat)
      li_mean_cat.append(li_mean_cat_component)
    li_mean_cat_per_sample.append(li_mean_cat)
  return li_mean_cat_per_sample, all_parameters_df
```

Source Code A.31: Method ”categorise_means”

```python
def categorise_variance(filenames, indices, number_of_methods): #categorises the mean per sample for every velocity compoent
  li_variance_cat_per_sample = []
    for filename in filenames: #iterates samples
    all_parameters_df = get_parameters(directory, filename)
    li_variance_cat = []
    for e in indices: #iterates velocity components
      li_variance_cat_component = []
      for i in range(1,(number_of_methods+1)): #iterates methods, starts at 1, because original samlpes cannot be categorised
        variance_change = ((all_parameters_df.iloc[i,e]-all_parameters_df.iloc[0,e])/all_parameters_df.iloc[0,e])*100 # Equation 6.2 Gives percentage of realtive change in variance
          if variance_change <= -50:
            variance_cat = 1
          elif variance_change <= -25:
            variance_cat = 2
          elif variance_change <= -10:
            variance_cat = 3
          else:
            variance_cat = 4
        li_variance_cat_component.append(variance_cat)
      li_variance_cat.append(li_variance_cat_component)
    li_variance_cat_per_sample.append(li_variance_cat)
  return li_variance_cat_per_sample, all_parameters_df
```

Source Code A.32: Method ”categorise_variance”

```python
def categorise_skew(filenames, indices, number_of_methods): #categorises skewness values per sample for every velocity component
  li_skew_cat_per_sample = []
  for filename in filenames: #iterates samples
    all_parameters_df = get_parameters(directory, filename)
    li_skew_cat = []
    for e in skew_indices: #iterates velocity components
      li_skew_cat_component = []
      for i in range(number_of_methods+1):#iterates methods, number_of_methods+1 because here the original samples are categorised, too
        if all_parameters_df.iloc[i,e] <= 0.1 and all_parameters_df.iloc[i,e] >= -0.1 :
          skew_cat = 1
        elif all_parameters_df.iloc[i,e] <= 0.5 and all_parameters_df.iloc[i,e] >= -0.5:
          skew_cat = 2
        elif all_parameters_df.iloc[i,e] <= 2 and all_parameters_df.iloc[i,e] >= -2:
          skew_cat = 3
        else:
          skew_cat = 4
        li_skew_cat_component.append(skew_cat)
      li_skew_cat.append(li_skew_cat_component)
  li_skew_cat_per_sample.append(li_skew_cat)
  return li_skew_cat_per_sample, all_parameters_df
```

Source Code A.33: Method ”categorise_skew”

```python
def categorise_kurt(filenames, indices, number_of_methods): #categorises kurtosis values per sample for every velocity component
  li_kurt_cat_per_sample = []
  for filename in filenames: #iterates samples
    all_parameters_df = get_parameters(directory, filename)
    li_kurt_cat = []
    for e in indices: #iterates velocity components
      li_kurt_cat_component = []
      for i in range(number_of_methods+1): #iterates methods, number_of_methods+1 because here the original samples are categorised, too
        if all_parameters_df.iloc[i,e] <= 4  and all_parameters_df.iloc[i,e] >= 2 :
          kurt_cat = 1
        elif all_parameters_df.iloc[i,e] <= 10 and all_parameters_df.iloc[i,e] > 4:
          kurt_cat = 2
        elif all_parameters_df.iloc[i,e] <= 50 and all_parameters_df.iloc[i,e] > 10:
          kurt_cat = 3
        else:
          kurt_cat = 4
        li_kurt_cat_component.append(kurt_cat)
      li_kurt_cat.append(li_kurt_cat_component)
    li_kurt_cat_per_sample.append(li_kurt_cat)
  return li_kurt_cat_per_sample, all_parameters_df
```

Source Code A.34: Method "categorise_kurt"

```python
#Input
directory = r"F:\Uni\Masterthesis\Python\PDF Bilder\parameters" #directory containing the parameter files
all_files = glob.glob("{}/*.csv".format(directory)) #gets filenames of the files to be categorised
filenames = get_filenames(all_files) # eliminates duplicates from filenames
kurt_indices = [6,13,20,27] #see table A.1. in the annex for indices in the parameter dataframe
number_of_methods = 8 # number of methods to compare
number_of_components = 4 # number of components to compare (velocity magnitude included)
number_of_categories = 4 # number of defined categories
number_of_samples = 160 #number of samples to categorise
#Write out method and category names for heatmap, make sure, that method names follow the order of parameters in line 46
methods_by_name = ['Orig','P-SAT','KDE','But','Gau','P-SAT_But','P-SAT_Gau','KDE_But','KDE_Gau'] #defines names of methods to compare
categories_by_name = ['Category 1','Category 2','Category 3','Category 4'] # defines names of categories
#Script
li_kurt_cat_per_sample, all_parameters_df = categorise_kurt(filenames, kurt_indices, number_of_methods ) # gets kurtosis categories by sample and parameter_df
category_by_method = sort_category_counts_by_method(number_of_components, number_of_methods, number_of_categories, li_kurt_cat_per_sample) #sorts kurtosis categories by method
category_by_method[:] = [[[x / number_of_samples * 100 for x in g] for g in k] for k in category_by_method] #calculates percentage of samples per method per category for each component
#Get data for heatmaps
u_kurt = pd.DataFrame(category_by_method[0], columns=categories_by_name,
index = methods_by_name)
v_kurt = pd.DataFrame(category_by_method[1], columns=categories_by_name,
index = methods_by_name)
w_kurt = pd.DataFrame(category_by_method[2], columns=categories_by_name,
index = methods_by_name)
res_kurt = pd.DataFrame(category_by_method[3], columns=categories_by_name,
index = methods_by_name)
#Plot heatmaps
plt.figure()
cbar_ticks = [0,20,40,60,80,100]
heatmap_u_kurt = sns.heatmap(u_kurt,annot=True, cmap="YlGnBu", fmt='.1f',
cbar_kws={'label': 'Samples [%]','orientation': 'vertical','ticks': cbar_ticks
    }, vmin=0, vmax=100)
plt.yticks(rotation=0)
plt.savefig('heatmap_u_kurt.pdf', bbox_inches='tight')
plt.figure()
heatmap_v_kurt = sns.heatmap(v_kurt,annot=True, cmap="YlGnBu", fmt='.1f',
cbar_kws={'label': 'Samples [%]','orientation': 'vertical','ticks': cbar_ticks
    }, vmin=0, vmax=100)
plt.yticks(rotation=0)
plt.savefig('heatmap_v_kurt.pdf', bbox_inches='tight')
plt.figure()
heatmap_w_kurt = sns.heatmap(w_kurt,annot=True, cmap="YlGnBu", fmt='.1f',
cbar_kws={'label': 'Samples [%]','orientation': 'vertical','ticks': cbar_ticks
    }, vmin=0, vmax=100)
plt.yticks(rotation=0)
plt.savefig('heatmap_w_kurt.pdf', bbox_inches='tight')
```

Source Code A.35: Script ”Categorisation_and_Heatmap_Visualisation_Kurtosis”

```
def get_population_df(path): #gets dataframe including all velocity magnitudes
   velocity_df = pd.read_csv(path,
   names=['0','1','velocity_y','velocity_z','velocity_x','5','6','7','8','9','
     10','11','12','13','14','15','16','17'])
   velocity_df.drop(['0','1','5','6','7','8','9','10','11','12','13','14','15',
     '16','17'], axis=1, inplace = True)
   velocity_df['velocity_res']= np.sqrt(velocity_df['velocity_y']**2 +
     velocity_df['velocity_z']**2 + velocity_df['velocity_x']**2)
   population_df = velocity_df['velocity_res']
   return population_df
```

Source Code A.36: Method ”get_population_df”

```
def get_samples(no_measurements, population_df, no_samples): # gets
     discriminable samples from population data frame of initial measurement
     length
   samples = []
   for i in range(no_samples):
      samples.append(population_df.sample(no_measurements, random_state=i))
   return samples
```

Source Code A.37: Method ”get_samples”

```
def extend_samples(no_measurements, no_steps, increment): # extends
     discriminable samples by number of steps of specified increments-->
     prolongs the sampling time
   list_of_samples = []
   for x in range(no_steps):
      list_of_samples.append(get_samples(no_measurements, population_df,
     no_samples))
      no_measurements+=increment
```

```python
  return list_of_samples
```

Source Code A.38: Method ”extend_samples”

```python
def calculate_means(list_of_samples): # calculates means per sample for different sampling times.
  list_of_means = []
  for i in range(len(list_of_samples)):
    means_part = []
    for j in range(len(list_of_samples[i])):
      means_part.append(np.mean(list_of_samples[i][j]))
    list_of_means.append(means_part)
  return list_of_means
```

Source Code A.39: Method ”calculate_means”

```python
def get_base_sample(no_measurements_base, random_state_base_sample, population_df): #gets base samples from population data frame
  base_sample_df = population_df.sample(no_measurements_base, random_state = random_state_base_sample)
  return base_sample_df
```

Source Code A.40: Method ”get_base_sample”

```python
def shuffle_population(random_state_population, population_df):
  population_df_shuffled = population_df.sample(frac=1, random_state = random_state_population) #shuffles population so that samples (get_base_sample, get_samples) produce different samples
  return population_df_shuffled
```

Source Code A.41: Method ”shuffle_population”

```python
def calculate_pvalues(list_of_samples, base_sample_df, low, upp): #performs TOST, gives pvalues by sampling time
  list_of_pvalues = []
  for i in range(len(list_of_samples)):
    pvalues_part = []
    for j in range(len(list_of_samples[i])):
      TOST = weightstats.ttost_ind(list_of_samples[i][j], base_sample_df, low, upp, usevar='unequal')
      pvalues_part.append(TOST[0])
    list_of_pvalues.append(pvalues_part)
  return list_of_pvalues
```

Source Code A.42: Method ”calculate_pvalues”

```python
def sort_pvalues_by_sample(list_of_pvalues): #sorts pvalues by sample
  list_of_pvalues_by_sample = []
  for i in range(len(list_of_pvalues[0])):
    list_of_pvalues_by_sample.append([])
    for e in list_of_pvalues:
      list_of_pvalues_by_sample[i].append(e[i])
  return list_of_pvalues_by_sample
```

Source Code A.43: Method ”sort_pvalues_by_sample”

```
def count_accept_H0(results_df, alpha):
  count_accept_0 = (results_df > alpha).sum()
  count_accept_0 = count_accept_0.to_numpy()
  return count_accept_0
```

Source Code A.44: Method ”count_accept_H0”

```
#Input Parameters
file = "0.1_wall" #long duration measurement
directory = r"F:\Uni\Masterthesis\Ttest\data" #directory of long duration
    measurement
no_measurements = 500 #Defines the number of measurements for the initial step
    .
increment = 500 #Defines the increment for each step to increase the number of
     measurements.
no_steps = 15 # Defines the number of steps to increase the number of
    measurements (prolong the sampling time).
no_samples = 1000 #Number of samples per step.
#Script
path = r"{}\{}.csv".format(directory, file)
population_df = get_population_df(path)
list_of_samples = extend_samples(no_measurements, no_steps, increment)
list_of_means = calculate_means(list_of_samples)
```

Source Code A.45: Script ”Comparison of Means”

```
#Input Parameters
directory = r"F:\Uni\Masterthesis\Ttest\data" #directory of long duration
    measurement
directory_output = r"F:\Uni\Masterthesis\Ttest\Ergebnisse\TOST_Ergebnisse_0
    .003\P_counts_" #directory to output results
file = "0.1_wall" #long duration measurement
no_measurements_base = 30000 #Defines the number of measurements for the base
    sample.
random_state_population = 0 #Defines the reproducable state of the population
    dataframe.
no_measurements = 500 #Defines the number of measurements for samples for the
    initial step.
increment = 500 #Defines the increment for each step to increase the number of
     measurements.
no_steps = 30 #Number of steps to increase the number of measurements (prolong
     the sampling time).
no_samples = 100 #Number of samples to compare against the base sample.
alpha = 0.05 #Significance level of the TOST.
```

```python
iterations_of_script = 10 #Number of base samples.
low = -0.003 #Lower equivalence boundary of the TOST δ(-).
upp = 0.003 # Upper equivalence boundary of the TOST δ(+).
#Script
path = r"{}\{}.csv".format(directory, file)
random_state_population_str = str(random_state_population)
population_df = get_population_df(path)
#Calculate p-values.
results = []
for i in range(iterations_of_script):
  random_state_base_sample = i
  base_sample_df = get_base_sample(no_measurements_base,
    random_state_base_sample, population_df)
  population_df_shuffled = shuffle_population(random_state_population,
    population_df)
  list_of_samples = extend_samples(no_measurements, no_steps, increment)
  list_of_pvalues = calculate_pvalues(list_of_samples, base_sample_df, low,
    upp)
  list_of_pvalues_by_sample = sort_pvalues_by_sample(list_of_pvalues)
  results.append(list_of_pvalues_by_sample)
#Get list of dataframes.
results_df = []
for each_sample in range(iterations_of_script):
  df = pd.DataFrame.from_records(results[each_sample])
  results_df.append(df)
#Count H0 acceptions for each sample.
accept_H0 = []
for each_sample in range(iterations_of_script):
  count = count_accept_H0(results_df[each_sample], alpha)
  accept_H0.append(count)
#Dataframe to array.
df_array_count = pd.DataFrame.from_records(accept_H0)
#Array to .csv.
df_array_count.to_csv(r"{}{}\{}_base_sample_iterations_{}_dataframe_{}
    _nosamples_{}.csv".format(directory_output, file, file,
    iterations_of_script, random_state_population_str, no_samples), index=
    False, header=False)
```

Source Code A.46: Script ”TOST”

```
#Parameters
file = '0.1_wall'
low = '-0.002'
upp = '+0.002'
delta = '0.004'
path = r"F:\Uni\Masterthesis\Python\Ttest\Ergebnisse\TOST_Ergebnisse_{}\
    P_counts_{}".format(delta,file)
all_files = glob.glob(path + "/*.csv")
#Script
li = []
for i in range(10):
  for filename in all_files:
  df = pd.read_csv(filename, names=['20','40','60','80','100','120','140','160
    ','180','200','220','240','260','280','300','320','340','360','380','400',
    '420','440','460','480','500','520','540','560','580','600'])
  df_row = df.iloc[[i]]
  li.append(df_row)
df = pd.concat(li)
df = df.transpose()
acc_rate_0 = df.iloc[:,0:20].sum(axis=1)/2000
acc_rate_1 = df.iloc[:,20:40].sum(axis=1)/2000
acc_rate_2 = df.iloc[:,40:60].sum(axis=1)/2000
acc_rate_3 = df.iloc[:,60:80].sum(axis=1)/2000
acc_rate_4 = df.iloc[:,80:100].sum(axis=1)/2000
acc_rate_5 = df.iloc[:,100:120].sum(axis=1)/2000
acc_rate_6 = df.iloc[:,120:140].sum(axis=1)/2000
acc_rate_7 = df.iloc[:,140:160].sum(axis=1)/2000
acc_rate_8 = df.iloc[:,160:180].sum(axis=1)/2000
acc_rate_9 = df.iloc[:,180:200].sum(axis=1)/2000
x =
    [20,40,60,80,100,120,140,160,180,200,220,240,260,280,300,320,340,360,380,400,420,440,

#Plotting
fig= plt.figure()
fig.set_figheight(8)
fig.set_figwidth(14)
plt.xlabel("Sampling Time [s]", fontsize=16)
plt.ylabel("H0 Acceptance Rate [%]", fontsize=16)
plt.plot(x,acc_rate_0, marker='.', linestyle='None', label='Base Sample 1')
plt.plot(x,acc_rate_1, marker='.', linestyle='None', label='Base Sample 2')
plt.plot(x,acc_rate_2, marker='.', linestyle='None', label='Base Sample 3')
plt.plot(x,acc_rate_3, marker='.', linestyle='None', label='Base Sample 4')
plt.plot(x,acc_rate_4, marker='.', linestyle='None', label='Base Sample 5')
plt.plot(x,acc_rate_5, marker='.', linestyle='None', label='Base Sample 6')
plt.plot(x,acc_rate_6, marker='.', linestyle='None', label='Base Sample 7')
plt.plot(x,acc_rate_7, marker='.', linestyle='None', label='Base Sample 8')
plt.plot(x,acc_rate_8, marker='.', linestyle='None', label='Base Sample 9')
plt.plot(x,acc_rate_9, marker='.', linestyle='None', label='Base Sample 10')
plt.text(5, 0.01, ' = '.format(delta), fontsize = 14, backgroundcolor= 'white'
    )
plt.legend(loc="upper right", fontsize=14)
plt.minorticks_on()
ax = plt.gca()
ax.xaxis.set_major_locator(MultipleLocator(40))
ax.xaxis.set_minor_locator(MultipleLocator(20))
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(b=True, which='major', color='lightgrey', linestyle='-')
plt.grid(b=True, which='minor', color='lightgrey', linestyle='--')
plt.savefig(r'F:\Uni\Masterthesis\Python\PDF Bilder\{}_{}.pdf'.format(delta,
    file), bbox_inches='tight')
plt.close(fig)
```

Source Code A.47: Script ”Visualisation_TOST”

A.3. Data and Tables

Table A.1.: Indices for statistical parameters in the parameter dataframe.

	u	**v**	**w**	**res**
mean	0	7	14	21
variance	1	8	15	22
standard deviation	2	9	16	23
global maximum	3	10	17	24
global minimum	4	11	18	25
skewness	5	12	19	26
kurtosis	6	13	20	27

	uv	**uw**	**vw**
cov	28	29	30
cor	31	32	33

	res
TKE	34

Table A.2.: Null hypothesis acceptance rates for the long-duration measurement in wall proximity and vertical position of $0.1\,\mathrm{m}$ over bottom with $\delta = 0.004$.

	0	1	2	3	4	5	6	7	8	9
20	1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
40	1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
60	1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
80	1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
100	1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
120	1	0.9995	0.9995	1.0	0.999	1.0	0.999	1.0	0.9995	0.999
140	0.9325	0.9225	0.935	0.926	0.926	0.923	0.9285	0.9285	0.9205	0.931
160	0.8455	0.836	0.863	0.831	0.8405	0.831	0.8395	0.8345	0.833	0.8635
180	0.768	0.7455	0.773	0.747	0.7565	0.7495	0.757	0.7435	0.75	0.7685
200	0.6715	0.671	0.7075	0.6605	0.6715	0.6535	0.669	0.6665	0.656	0.698
220	0.606	0.61	0.6445	0.6075	0.5975	0.6055	0.5975	0.6075	0.608	0.637
240	0.53	0.542	0.5925	0.5195	0.5205	0.515	0.5245	0.528	0.5155	0.5735
260	0.474	0.489	0.5335	0.4665	0.4665	0.4555	0.465	0.473	0.4565	0.519
280	0.4185	0.44	0.485	0.408	0.4115	0.407	0.413	0.4225	0.41	0.46
300	0.3785	0.4025	0.4515	0.3735	0.369	0.363	0.3695	0.384	0.3655	0.4245
320	0.3315	0.37	0.4205	0.3315	0.3265	0.3155	0.3275	0.344	0.317	0.3935
340	0.2985	0.3145	0.391	0.299	0.286	0.2815	0.285	0.3035	0.2815	0.3655
360	0.269	0.2825	0.3585	0.2545	0.256	0.2465	0.257	0.262	0.2485	0.328
380	0.2385	0.2525	0.3375	0.2295	0.221	0.21	0.223	0.2325	0.2135	0.3115
400	0.2015	0.2215	0.3245	0.1875	0.187	0.185	0.189	0.199	0.185	0.2975
420	0.181	0.2035	0.2975	0.1605	0.1685	0.1535	0.171	0.17	0.1525	0.271
440	0.1555	0.1755	0.2775	0.1445	0.144	0.135	0.1475	0.1595	0.137	0.2415
460	0.135	0.159	0.247	0.13	0.1225	0.1215	0.1205	0.137	0.122	0.222
480	0.116	0.1365	0.226	0.1155	0.1035	0.094	0.1055	0.12	0.094	0.1945
500	0.0965	0.117	0.2075	0.09	0.0905	0.0775	0.091	0.103	0.079	0.1735
520	0.0885	0.1005	0.1845	0.0775	0.0785	0.068	0.0785	0.0855	0.0695	0.1515
540	0.0785	0.0905	0.1675	0.0655	0.067	0.0545	0.068	0.0735	0.055	0.1335
560	0.071	0.077	0.161	0.053	0.058	0.0425	0.062	0.063	0.0435	0.1305
580	0.054	0.064	0.153	0.045	0.0435	0.0385	0.046	0.0465	0.0385	0.126
600	0.0505	0.0525	0.1435	0.036	0.0385	0.0315	0.04	0.039	0.0305	0.112

Table A.3.: Null hypothesis acceptance rates for the long-duration measurement in wall proximity and vertical position of $0.1\,\mathrm{m}$ over bottom with $\delta = 0.006$.

	0	1	2	3	4	5	6	7	8	9
20	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
40	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
60	0.9115	0.905	0.911	0.906	0.9135	0.9045	0.9115	0.909	0.905	0.914
80	0.7255	0.7265	0.722	0.712	0.7215	0.715	0.722	0.7145	0.7155	0.725
100	0.543	0.556	0.568	0.5465	0.5415	0.548	0.541	0.551	0.5475	0.5605
120	0.406	0.408	0.441	0.403	0.403	0.395	0.402	0.407	0.395	0.43
140	0.302	0.3135	0.332	0.2985	0.2945	0.296	0.2955	0.3025	0.296	0.3185
160	0.222	0.228	0.2635	0.2185	0.215	0.2115	0.2165	0.226	0.2115	0.248
180	0.1605	0.1605	0.2045	0.145	0.152	0.1395	0.1535	0.148	0.1395	0.1895
200	0.1125	0.1215	0.157	0.1135	0.11	0.1105	0.1105	0.118	0.111	0.1425
220	0.078	0.098	0.123	0.088	0.0795	0.089	0.077	0.0915	0.089	0.112
240	0.063	0.0685	0.0965	0.0625	0.0585	0.053	0.059	0.0615	0.054	0.085
260	0.0465	0.046	0.071	0.038	0.0445	0.0345	0.0445	0.0395	0.034	0.064
280	0.0305	0.0315	0.06	0.0255	0.0295	0.0235	0.0295	0.028	0.0235	0.0525
300	0.0215	0.0255	0.048	0.021	0.019	0.017	0.019	0.0215	0.017	0.038
320	0.018	0.02	0.0365	0.0145	0.015	0.0125	0.0155	0.016	0.0115	0.0325
340	0.011	0.0145	0.0285	0.011	0.0095	0.0095	0.01	0.013	0.01	0.0225
360	0.007	0.012	0.0215	0.008	0.007	0.006	0.006	0.0085	0.006	0.017
380	0.006	0.0085	0.0145	0.006	0.004	0.005	0.004	0.0065	0.005	0.011
400	0.0045	0.006	0.0125	0.004	0.004	0.003	0.004	0.0045	0.003	0.0085
420	0.002	0.002	0.0095	0.0015	0.002	0.002	0.002	0.002	0.0015	0.008
440	0.002	0.0015	0.006	0.001	0.0015	0.0015	0.0015	0.0015	0.0015	0.0045
460	0.002	0.0005	0.0045	0.0005	0.002	0.0005	0.002	0.0005	0.001	0.0035
480	0.002	0.001	0.0045	0.0005	0.001	0.0005	0.001	0.0005	0.0005	0.0035
500	0.001	0.0005	0.0025	0.0	0.0	0.0	0.0005	0.0	0.0	0.002
520	0.0005	0.0	0.002	0.0	0.0	0.0	0.0	0.0	0.0	0.0015
540	0.0	0.0	0.001	0.0	0.0	0.0	0.0	0.0	0.0	0.0005
560	0.0	0.0	0.0005	0.0	0.0	0.0	0.0	0.0	0.0	0.0
580	0.0	0.0	0.0005	0.0	0.0	0.0	0.0	0.0	0.0	0.0
600	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table A.4.: Null hypothesis acceptance rates for the long-duration measurement in central horizontal position and vertical position of $0.1\,\mathrm{m}$ over bottom with $\delta = 0.004$.

	0	1	2	3	4	5	6	7	8	9
20	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
40	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
60	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
80	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
100	0.9985	0.9985	0.996	0.997	0.997	0.9985	0.998	0.9995	0.9965	0.997
120	0.8955	0.897	0.908	0.9025	0.9005	0.896	0.8905	0.896	0.9035	0.9025
140	0.7795	0.7805	0.8005	0.7955	0.781	0.7805	0.7855	0.7855	0.7965	0.793
160	0.68	0.6885	0.716	0.7095	0.685	0.6875	0.6785	0.687	0.7125	0.7065
180	0.5985	0.602	0.624	0.626	0.6025	0.6015	0.6045	0.6125	0.6265	0.627
200	0.494	0.499	0.564	0.5595	0.493	0.4985	0.5025	0.514	0.5615	0.5555
220	0.432	0.4375	0.4885	0.478	0.432	0.4365	0.446	0.4585	0.482	0.477
240	0.379	0.382	0.4325	0.4245	0.3805	0.382	0.381	0.393	0.4295	0.4185
260	0.3335	0.3365	0.381	0.369	0.3345	0.3355	0.325	0.332	0.373	0.3625
280	0.2875	0.292	0.3345	0.324	0.292	0.291	0.2875	0.2965	0.328	0.3225
300	0.243	0.247	0.297	0.287	0.2425	0.2465	0.2495	0.2525	0.291	0.2835
320	0.207	0.2085	0.2805	0.2695	0.205	0.2085	0.212	0.231	0.2735	0.2655
340	0.167	0.1645	0.2515	0.2375	0.1635	0.163	0.173	0.19	0.242	0.23
360	0.1335	0.136	0.223	0.2135	0.135	0.136	0.142	0.1575	0.2155	0.2115
380	0.109	0.1105	0.2045	0.1905	0.1045	0.1105	0.1255	0.14	0.194	0.184
400	0.0825	0.0865	0.187	0.1705	0.085	0.0855	0.094	0.118	0.1765	0.164
420	0.0725	0.0755	0.158	0.1445	0.07	0.0755	0.0785	0.097	0.1475	0.1435
440	0.056	0.0555	0.1375	0.122	0.057	0.0555	0.064	0.0805	0.127	0.1205
460	0.051	0.0535	0.128	0.114	0.0515	0.053	0.0555	0.071	0.1175	0.1115
480	0.043	0.044	0.108	0.0975	0.044	0.044	0.0425	0.0665	0.1	0.093
500	0.0335	0.0365	0.0925	0.0825	0.036	0.036	0.04	0.0495	0.086	0.081
520	0.0325	0.0315	0.0785	0.069	0.0315	0.031	0.036	0.041	0.0725	0.066
540	0.023	0.0235	0.066	0.0565	0.0235	0.023	0.0295	0.036	0.0585	0.055
560	0.0195	0.0205	0.0615	0.0525	0.0195	0.0205	0.0245	0.033	0.0555	0.051
580	0.013	0.014	0.0555	0.0475	0.014	0.014	0.021	0.0265	0.05	0.043
600	0.0115	0.011	0.052	0.0455	0.0115	0.011	0.0145	0.0245	0.047	0.0425

Table A.5.: Null hypothesis acceptance rates for the long-duration measurement in central horizontal position and vertical position of $0.1\,\mathrm{m}$ over bottom with $\delta = 0.006$.

	0	1	2	3	4	5	6	7	8	9
20	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
40	0.999	0.999	0.9995	0.999	0.9985	0.999	1.0	0.999	0.9995	0.999
60	0.765	0.7595	0.758	0.7595	0.7635	0.759	0.766	0.766	0.7605	0.7565
80	0.5555	0.554	0.563	0.5575	0.554	0.554	0.5425	0.548	0.5585	0.555
100	0.388	0.3865	0.42	0.4085	0.388	0.387	0.392	0.394	0.4135	0.4075
120	0.2725	0.2835	0.314	0.3055	0.276	0.2835	0.2785	0.2805	0.3075	0.3045
140	0.202	0.2035	0.225	0.222	0.204	0.203	0.2055	0.204	0.2225	0.2205
160	0.1415	0.137	0.167	0.165	0.1405	0.1365	0.1465	0.148	0.1655	0.1605
180	0.093	0.093	0.119	0.1135	0.0935	0.093	0.091	0.098	0.114	0.113
200	0.0615	0.0595	0.0925	0.085	0.0605	0.0595	0.066	0.072	0.087	0.083
220	0.0355	0.0395	0.0575	0.0535	0.036	0.0395	0.037	0.043	0.055	0.053
240	0.025	0.024	0.0415	0.037	0.0245	0.024	0.026	0.028	0.0375	0.0365
260	0.0145	0.015	0.028	0.025	0.0155	0.015	0.0165	0.017	0.026	0.024
280	0.007	0.0085	0.0205	0.0185	0.0065	0.0085	0.01	0.0105	0.0185	0.0165
300	0.0055	0.0045	0.014	0.013	0.005	0.0045	0.0065	0.007	0.0135	0.012
320	0.003	0.0035	0.0075	0.007	0.003	0.0035	0.003	0.0035	0.007	0.007
340	0.0015	0.002	0.006	0.005	0.0015	0.002	0.002	0.0025	0.005	0.005
360	0.0005	0.001	0.0045	0.004	0.0005	0.001	0.0015	0.002	0.0045	0.004
380	0.0	0.0	0.003	0.003	0.0	0.0	0.0	0.001	0.003	0.003
400	0.0	0.0	0.0025	0.002	0.0	0.0	0.0	0.0005	0.0025	0.0015
420	0.0	0.0	0.0015	0.001	0.0	0.0	0.0	0.0	0.0015	0.0005
440	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
460	0.0	0.0	0.0005	0.0005	0.0	0.0	0.0	0.0	0.0005	0.0005
480	0.0	0.0	0.0005	0.0005	0.0	0.0	0.0	0.0	0.0005	0.0
500	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
520	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
540	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
560	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
580	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
600	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table A.6.: Null hypothesis acceptance rates for the long-duration measurement in wall proximity and vertical position of $0.4\,\mathrm{m}$ over bottom with $\delta = 0.004$.

	0	1	2	3	4	5	6	7	8	9
20	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
40	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
60	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
80	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
100	0.999	0.9985	0.9985	1.0	0.9995	0.9995	0.999	0.999	0.999	0.999
120	0.904	0.9125	0.903	0.898	0.9	0.8995	0.9015	0.895	0.894	0.9005
140	0.789	0.7895	0.788	0.792	0.792	0.7895	0.7915	0.7925	0.792	0.794
160	0.698	0.7025	0.697	0.704	0.7025	0.6965	0.708	0.702	0.706	0.7135
180	0.602	0.615	0.6025	0.6365	0.6245	0.614	0.6435	0.628	0.6385	0.6485
200	0.5295	0.5305	0.5365	0.5535	0.547	0.538	0.557	0.547	0.554	0.568
220	0.471	0.4695	0.472	0.489	0.4795	0.472	0.5025	0.481	0.4935	0.505
240	0.3845	0.3905	0.3855	0.4275	0.405	0.3965	0.438	0.407	0.4345	0.4465
260	0.325	0.3315	0.329	0.3805	0.3635	0.351	0.398	0.371	0.3915	0.4085
280	0.2835	0.286	0.2825	0.336	0.298	0.294	0.3595	0.3075	0.346	0.365
300	0.243	0.2445	0.24	0.287	0.2685	0.2515	0.3105	0.274	0.296	0.3275
320	0.2005	0.201	0.202	0.2545	0.235	0.217	0.273	0.2425	0.2635	0.286
340	0.171	0.179	0.1685	0.2145	0.1915	0.176	0.234	0.197	0.2185	0.244
360	0.1495	0.1525	0.1505	0.179	0.1635	0.1545	0.197	0.1635	0.1835	0.208
380	0.1225	0.1295	0.126	0.1645	0.142	0.1295	0.1865	0.147	0.172	0.1965
400	0.107	0.107	0.1055	0.139	0.126	0.1135	0.16	0.132	0.146	0.1715
420	0.079	0.08	0.0775	0.1165	0.102	0.0885	0.139	0.1055	0.123	0.1485
440	0.063	0.066	0.063	0.107	0.09	0.072	0.1285	0.0975	0.1155	0.137
460	0.0485	0.053	0.0505	0.094	0.0705	0.0585	0.109	0.0745	0.0985	0.1195
480	0.0355	0.0405	0.0355	0.067	0.0505	0.039	0.0905	0.055	0.0735	0.1025
500	0.024	0.037	0.026	0.0585	0.041	0.032	0.0755	0.045	0.0635	0.084
520	0.0195	0.0285	0.021	0.0495	0.0345	0.028	0.0635	0.04	0.0535	0.0715
540	0.014	0.024	0.015	0.048	0.031	0.0175	0.0575	0.0365	0.0515	0.065
560	0.0135	0.0165	0.014	0.0405	0.027	0.0165	0.051	0.031	0.045	0.06
580	0.012	0.0145	0.0115	0.0335	0.025	0.0165	0.0445	0.0275	0.038	0.0485
600	0.0095	0.0135	0.0105	0.0275	0.019	0.0145	0.0385	0.022	0.029	0.044

Table A.7.: Null hypothesis acceptance rates for the long-duration measurement in wall proximity and vertical position of $0.4\,\mathrm{m}$ over bottom with $\delta = 0.006$.

	0	1	2	3	4	5	6	7	8	9
20	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
40	1.0	0.9995	1.0	1.0	0.9995	0.9995	1.0	1.0	1.0	1.0
60	0.765	0.7745	0.7665	0.7695	0.7715	0.7695	0.7775	0.771	0.7705	0.7815
80	0.5725	0.5765	0.5685	0.579	0.5695	0.5685	0.5835	0.573	0.583	0.586
100	0.406	0.407	0.407	0.406	0.4085	0.4075	0.4145	0.407	0.411	0.422
120	0.2745	0.281	0.2765	0.2965	0.292	0.282	0.309	0.295	0.3	0.31
140	0.19	0.195	0.1925	0.206	0.1995	0.195	0.2125	0.2025	0.2075	0.2185
160	0.126	0.131	0.127	0.1425	0.133	0.129	0.154	0.1375	0.1455	0.1585
180	0.0865	0.09	0.088	0.0965	0.0885	0.088	0.102	0.0925	0.097	0.105
200	0.059	0.064	0.0595	0.0695	0.0645	0.061	0.072	0.067	0.0715	0.0785
220	0.04	0.0375	0.04	0.0515	0.046	0.0435	0.0565	0.048	0.0525	0.0585
240	0.025	0.027	0.024	0.0325	0.028	0.027	0.0375	0.029	0.0335	0.041
260	0.016	0.015	0.0155	0.02	0.017	0.017	0.0265	0.0185	0.0205	0.0285
280	0.008	0.0095	0.009	0.017	0.012	0.011	0.0175	0.014	0.0175	0.0185
300	0.003	0.0075	0.004	0.01	0.008	0.0065	0.0105	0.008	0.01	0.011
320	0.003	0.003	0.0035	0.007	0.006	0.004	0.0085	0.006	0.0075	0.009
340	0.0025	0.001	0.0025	0.0035	0.0025	0.0035	0.0045	0.0035	0.0045	0.0055
360	0.001	0.001	0.002	0.0015	0.002	0.002	0.003	0.002	0.002	0.0035
380	0.001	0.001	0.001	0.0015	0.0	0.0005	0.0025	0.0	0.002	0.003
400	0.0	0.001	0.0	0.001	0.0005	0.0	0.001	0.0005	0.001	0.002
420	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.001
440	0.0	0.0005	0.0	0.0005	0.0	0.0	0.0005	0.0	0.0005	0.0005
460	0.0	0.0	0.0	0.0	0.0	0.0	0.0005	0.0	0.0	0.0005
480	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
500	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
520	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
540	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
560	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
580	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
600	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table A.8.: Null hypothesis acceptance rates for the long-duration measurement in central horizontal position and vertical position of $0.4\,\mathrm{m}$ over bottom with $\delta = 0.004$.

	0	1	2	3	4	5	6	7	8	9
20	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
40	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
60	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
80	0.9915	0.991	0.9875	0.99	0.9895	0.9885	0.992	0.9895	0.988	0.9915
100	0.8395	0.842	0.84	0.844	0.842	0.841	0.8465	0.8505	0.845	0.845
120	0.71	0.6945	0.6925	0.698	0.692	0.698	0.7035	0.6975	0.7285	0.7085
140	0.6105	0.5735	0.58	0.575	0.579	0.5755	0.5815	0.573	0.6335	0.5845
160	0.5225	0.485	0.5	0.4845	0.4975	0.4935	0.4925	0.485	0.5425	0.4965
180	0.4265	0.4085	0.389	0.413	0.3865	0.383	0.4235	0.3885	0.455	0.428
200	0.362	0.3395	0.3325	0.3425	0.3325	0.3285	0.352	0.3275	0.399	0.3665
220	0.303	0.2815	0.2745	0.287	0.271	0.2735	0.298	0.272	0.3475	0.3085
240	0.251	0.2275	0.2265	0.231	0.2235	0.224	0.242	0.225	0.2875	0.253
260	0.204	0.1905	0.1735	0.197	0.1735	0.1655	0.211	0.177	0.253	0.217
280	0.1815	0.1485	0.143	0.156	0.139	0.1345	0.1675	0.132	0.2135	0.1795
300	0.1495	0.1185	0.121	0.123	0.1155	0.112	0.138	0.103	0.1815	0.1465
320	0.1175	0.1005	0.094	0.1055	0.093	0.0875	0.1135	0.093	0.1485	0.1225
340	0.0895	0.0845	0.068	0.0865	0.0675	0.0645	0.098	0.0725	0.1175	0.103
360	0.0715	0.069	0.0495	0.0715	0.0485	0.048	0.082	0.052	0.1005	0.089
380	0.0555	0.0535	0.04	0.059	0.0405	0.041	0.0695	0.045	0.086	0.0765
400	0.052	0.0455	0.035	0.051	0.0355	0.033	0.057	0.033	0.069	0.062
420	0.0435	0.0305	0.029	0.034	0.027	0.0245	0.043	0.029	0.069	0.047
440	0.037	0.024	0.0235	0.025	0.0205	0.017	0.034	0.0185	0.0595	0.0385
460	0.0315	0.017	0.0155	0.02	0.0145	0.0145	0.0245	0.013	0.0495	0.028
480	0.0235	0.0135	0.0115	0.0175	0.012	0.0105	0.0225	0.0105	0.0385	0.0265
500	0.019	0.0135	0.008	0.0155	0.008	0.0075	0.021	0.007	0.029	0.0235
520	0.014	0.0105	0.0055	0.0125	0.0055	0.005	0.016	0.0055	0.0225	0.0205
540	0.011	0.0075	0.0045	0.0095	0.0045	0.004	0.015	0.005	0.02	0.0185
560	0.0075	0.0045	0.003	0.0055	0.002	0.002	0.009	0.002	0.02	0.0115
580	0.0065	0.0035	0.002	0.004	0.0015	0.0005	0.006	0.0015	0.012	0.0095
600	0.0045	0.003	0.001	0.0035	0.001	0.001	0.0055	0.0005	0.01	0.007

Table A.9.: Null hypothesis acceptance rates for the long-duration measurement in central horizontal position and vertical position of $0.4\,\mathrm{m}$ over bottom with $\delta = 0.006$.

	0	1	2	3	4	5	6	7	8	9
20	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
40	0.8915	0.884	0.889	0.886	0.886	0.8895	0.8865	0.888	0.8865	0.886
60	0.6015	0.5735	0.576	0.573	0.5745	0.572	0.576	0.5725	0.6045	0.5755
80	0.392	0.3865	0.3765	0.39	0.3765	0.379	0.395	0.382	0.402	0.3935
100	0.244	0.2255	0.225	0.2245	0.225	0.2205	0.225	0.222	0.267	0.2315
120	0.141	0.144	0.1325	0.149	0.133	0.133	0.155	0.1365	0.1585	0.159
140	0.102	0.085	0.0865	0.0875	0.0865	0.0845	0.097	0.082	0.107	0.099
160	0.0645	0.0535	0.0555	0.053	0.052	0.0495	0.0565	0.0495	0.0745	0.06
180	0.0385	0.033	0.0335	0.0335	0.0335	0.03	0.035	0.029	0.0475	0.0365
200	0.025	0.0195	0.015	0.02	0.014	0.0145	0.024	0.017	0.0315	0.027
220	0.015	0.0145	0.0095	0.015	0.0105	0.009	0.0165	0.0115	0.02	0.02
240	0.0085	0.0085	0.0055	0.0085	0.005	0.0045	0.0105	0.006	0.0125	0.0125
260	0.004	0.004	0.0035	0.0035	0.0035	0.0025	0.0045	0.003	0.0045	0.0055
280	0.002	0.0025	0.001	0.0025	0.001	0.001	0.003	0.0015	0.0035	0.004
300	0.0005	0.001	0.0	0.001	0.0	0.0	0.0015	0.0005	0.002	0.002
320	0.0005	0.0005	0.0	0.0005	0.0	0.0	0.0005	0.0	0.0015	0.0015
340	0.0005	0.0	0.0	0.0	0.0	0.0	0.0005	0.0	0.0005	0.0005
360	0.0	0.0005	0.0	0.0005	0.0	0.0	0.0005	0.0	0.0	0.001
380	0.0	0.0005	0.0	0.0005	0.0	0.0	0.0005	0.0	0.0	0.0005
400	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0005	0.0005
420	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
440	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
460	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
480	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
500	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
520	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
540	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
560	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
580	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
600	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table A.10.: Null hypothesis acceptance rates for the long-duration measurement in wall proximity and vertical position of $0.6\,\mathrm{m}$ over bottom with $\delta = 0.004$.

	0	1	2	3	4	5	6	7	8	9
20	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
40	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
60	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
80	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
100	0.9995	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9995
120	0.912	0.9075	0.908	0.91	0.924	0.9105	0.9055	0.8985	0.9155	0.897
140	0.7885	0.808	0.7895	0.7925	0.8105	0.8105	0.799	0.7855	0.8175	0.7865
160	0.7055	0.7095	0.6925	0.698	0.717	0.718	0.7055	0.692	0.731	0.6935
180	0.6105	0.6285	0.6115	0.612	0.634	0.636	0.6115	0.603	0.664	0.6045
200	0.526	0.546	0.5215	0.5305	0.5555	0.5475	0.533	0.516	0.6045	0.52
220	0.4555	0.4755	0.4525	0.451	0.4855	0.498	0.459	0.454	0.5525	0.4535
240	0.389	0.4025	0.3785	0.385	0.44	0.4225	0.386	0.3805	0.477	0.3805
260	0.3385	0.3635	0.3365	0.338	0.392	0.381	0.345	0.334	0.4415	0.334
280	0.291	0.316	0.2775	0.288	0.335	0.3405	0.291	0.273	0.4075	0.2745
300	0.2465	0.271	0.2285	0.2455	0.3045	0.291	0.2575	0.2295	0.3575	0.231
320	0.1995	0.238	0.205	0.2105	0.2585	0.2615	0.218	0.1875	0.331	0.191
340	0.1755	0.207	0.169	0.1785	0.24	0.234	0.191	0.16	0.3115	0.1595
360	0.1505	0.186	0.1445	0.1595	0.209	0.2155	0.165	0.1405	0.2905	0.137
380	0.127	0.16	0.123	0.132	0.1885	0.1855	0.1405	0.113	0.273	0.1135
400	0.1085	0.1425	0.1035	0.1135	0.157	0.1645	0.118	0.0975	0.252	0.1
420	0.0975	0.121	0.084	0.0965	0.15	0.1425	0.1025	0.0795	0.219	0.08
440	0.079	0.1105	0.0785	0.0865	0.1335	0.136	0.0905	0.064	0.203	0.066
460	0.0685	0.0925	0.06	0.0695	0.1115	0.116	0.072	0.0515	0.1785	0.0535
480	0.0505	0.0745	0.043	0.0525	0.097	0.0985	0.058	0.041	0.161	0.043
500	0.0505	0.0615	0.033	0.041	0.0855	0.08	0.044	0.036	0.1525	0.039
520	0.0425	0.047	0.0225	0.029	0.077	0.07	0.0335	0.0275	0.139	0.028
540	0.035	0.0415	0.018	0.024	0.0725	0.061	0.028	0.0175	0.1235	0.0175
560	0.0285	0.0335	0.014	0.02	0.062	0.0515	0.024	0.0135	0.1105	0.014
580	0.019	0.029	0.0125	0.0155	0.0555	0.0405	0.017	0.01	0.1	0.0105
600	0.016	0.028	0.009	0.013	0.048	0.035	0.016	0.008	0.081	0.008

Table A.11.: Null hypothesis acceptance rates for the long-duration measurement in wall proximity and vertical position of $0.6\,\mathrm{m}$ over bottom with $\delta = 0.006$.

	0	1	2	3	4	5	6	7	8	9
20	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
40	1.0	1.0	1.0	0.999	0.9995	1.0	0.9995	1.0	0.9995	1.0
60	0.804	0.792	0.791	0.788	0.804	0.797	0.7865	0.794	0.7995	0.7955
80	0.5645	0.571	0.563	0.5625	0.577	0.5745	0.5655	0.5515	0.5995	0.553
100	0.3945	0.415	0.408	0.408	0.4115	0.425	0.4115	0.398	0.4465	0.3975
120	0.288	0.2895	0.276	0.2805	0.3105	0.294	0.2805	0.2765	0.3245	0.2775
140	0.1975	0.2045	0.197	0.195	0.2185	0.2135	0.2	0.195	0.246	0.193
160	0.1405	0.157	0.1405	0.148	0.157	0.1645	0.15	0.1355	0.1945	0.133
180	0.094	0.107	0.0885	0.0935	0.113	0.1175	0.097	0.0925	0.1435	0.092
200	0.062	0.0775	0.064	0.0635	0.0765	0.086	0.0655	0.0605	0.108	0.0605
220	0.0385	0.0515	0.038	0.041	0.0545	0.0575	0.0455	0.0365	0.082	0.0385
240	0.03	0.0345	0.0225	0.029	0.037	0.044	0.03	0.025	0.063	0.0245
260	0.0195	0.0235	0.017	0.0185	0.028	0.0295	0.0195	0.0155	0.0495	0.0155
280	0.0115	0.0165	0.011	0.0125	0.019	0.0205	0.0145	0.009	0.033	0.008
300	0.007	0.0085	0.0045	0.0065	0.0135	0.0125	0.008	0.005	0.0265	0.0055
320	0.005	0.004	0.003	0.003	0.008	0.007	0.003	0.0045	0.016	0.0045
340	0.003	0.003	0.001	0.0015	0.007	0.005	0.0015	0.0015	0.0105	0.0015
360	0.0025	0.001	0.0005	0.001	0.0065	0.001	0.001	0.0005	0.007	0.001
380	0.001	0.0005	0.0005	0.0005	0.0045	0.0005	0.0005	0.0005	0.0045	0.0005
400	0.0015	0.0	0.0	0.0	0.003	0.0	0.0	0.0	0.003	0.0
420	0.0005	0.0	0.0	0.0	0.002	0.0	0.0	0.0	0.001	0.0
440	0.0005	0.0	0.0	0.0	0.0005	0.0	0.0	0.0	0.001	0.0
460	0.0005	0.0	0.0	0.0	0.0005	0.0	0.0	0.0	0.0005	0.0
480	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
500	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
520	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
540	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
560	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
580	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
600	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table A.12.: Null hypothesis acceptance rates for the long-duration measurement in central horizontal position and vertical position of $0.6\,\mathrm{m}$ over bottom with $\delta = 0.004$.

	0	1	2	3	4	5	6	7	8	9
20	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
40	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
60	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
80	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
100	0.927	0.897	0.919	0.8975	0.925	0.9265	0.924	0.897	0.9225	0.926
120	0.7965	0.798	0.797	0.7985	0.7945	0.7945	0.801	0.7975	0.7975	0.7935
140	0.684	0.6715	0.7005	0.672	0.681	0.684	0.691	0.671	0.6955	0.683
160	0.572	0.572	0.5895	0.57	0.5735	0.574	0.572	0.5715	0.582	0.574
180	0.4805	0.4815	0.491	0.4825	0.4755	0.48	0.485	0.48	0.491	0.4755
200	0.4035	0.416	0.414	0.4135	0.4025	0.4025	0.4045	0.4125	0.411	0.405
220	0.3305	0.3345	0.358	0.332	0.3295	0.328	0.341	0.3325	0.347	0.332
240	0.2705	0.26	0.2995	0.26	0.267	0.272	0.2845	0.259	0.2905	0.269
260	0.2295	0.2275	0.2485	0.2285	0.2245	0.228	0.2385	0.2265	0.244	0.224
280	0.183	0.1815	0.218	0.1805	0.184	0.183	0.1955	0.181	0.206	0.1845
300	0.151	0.146	0.182	0.1445	0.1475	0.151	0.1635	0.1445	0.171	0.1495
320	0.125	0.1235	0.155	0.1205	0.122	0.125	0.1365	0.1205	0.145	0.123
340	0.1025	0.098	0.127	0.0965	0.094	0.099	0.113	0.096	0.118	0.0945
360	0.082	0.0845	0.1095	0.0845	0.075	0.0795	0.0985	0.084	0.1025	0.0775
380	0.073	0.0675	0.093	0.0655	0.066	0.072	0.0805	0.066	0.088	0.069
400	0.061	0.056	0.0755	0.054	0.054	0.06	0.067	0.0545	0.071	0.057
420	0.0445	0.0445	0.0665	0.0445	0.041	0.0435	0.052	0.0445	0.059	0.042
440	0.0345	0.034	0.053	0.0345	0.0325	0.0345	0.043	0.034	0.048	0.0345
460	0.027	0.0305	0.043	0.03	0.0235	0.026	0.0355	0.03	0.0405	0.0225
480	0.0215	0.0235	0.0395	0.023	0.0195	0.0215	0.0265	0.0235	0.033	0.0195
500	0.0185	0.0165	0.031	0.017	0.015	0.017	0.0225	0.0165	0.028	0.016
520	0.0125	0.0095	0.026	0.009	0.01	0.012	0.0155	0.009	0.02	0.0105
540	0.0105	0.008	0.02	0.008	0.0085	0.01	0.016	0.0075	0.0175	0.0095
560	0.01	0.007	0.019	0.007	0.0075	0.0085	0.015	0.007	0.016	0.008
580	0.007	0.004	0.015	0.0045	0.006	0.0065	0.009	0.004	0.0115	0.0065
600	0.0055	0.0045	0.012	0.0045	0.0055	0.0055	0.008	0.0045	0.0085	0.0055

Table A.13.: Null hypothesis acceptance rates for the long-duration measurement in central horizontal position and vertical position of $0.6\,\mathrm{m}$ over bottom with $\delta = 0.006$.

	0	1	2	3	4	5	6	7	8	9
20	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
40	0.9555	0.956	0.95	0.956	0.96	0.955	0.9525	0.9555	0.9505	0.959
60	0.671	0.663	0.6835	0.6625	0.668	0.669	0.6745	0.663	0.679	0.6675
80	0.4525	0.4585	0.456	0.4595	0.4565	0.454	0.4535	0.4585	0.4565	0.4555
100	0.297	0.293	0.304	0.2935	0.2925	0.2965	0.2905	0.2925	0.2965	0.291
120	0.1965	0.19	0.2055	0.191	0.191	0.196	0.2025	0.19	0.2055	0.194
140	0.124	0.116	0.137	0.116	0.1185	0.122	0.127	0.1155	0.132	0.121
160	0.0655	0.066	0.079	0.0655	0.0645	0.066	0.073	0.0655	0.0755	0.0655
180	0.0425	0.0405	0.0535	0.04	0.04	0.0415	0.0455	0.04	0.048	0.0395
200	0.028	0.025	0.0325	0.0245	0.025	0.028	0.031	0.0245	0.0325	0.025
220	0.0155	0.014	0.0195	0.0135	0.0145	0.0155	0.017	0.014	0.018	0.0145
240	0.0085	0.01	0.012	0.0095	0.008	0.0085	0.009	0.0095	0.011	0.0085
260	0.006	0.0075	0.0095	0.0075	0.005	0.006	0.0075	0.0075	0.008	0.0055
280	0.004	0.003	0.006	0.003	0.0035	0.0035	0.005	0.003	0.005	0.0035
300	0.0015	0.0035	0.004	0.003	0.0015	0.0015	0.002	0.0035	0.003	0.0015
320	0.001	0.001	0.001	0.001	0.001	0.001	0.0005	0.001	0.0005	0.001
340	0.0005	0.0005	0.0015	0.0005	0.0005	0.0005	0.0005	0.0005	0.001	0.0005
360	0.0	0.0	0.0005	0.0	0.0	0.0	0.0005	0.0	0.0005	0.0
380	0.0	0.0	0.0005	0.0	0.0	0.0	0.0	0.0	0.0	0.0
400	0.0	0.0	0.0005	0.0	0.0	0.0	0.0	0.0	0.0005	0.0
420	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
440	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
460	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
480	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
500	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
520	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
540	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
560	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
580	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
600	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table A.14.: Sampling times corresponding to null hypothesis acceptance rates under 0.05 for all base samples.

No	Position_h	Position_v [m]	Sampling Time [min] $\delta = 0,004$	Sampling Time [min] $\delta = 0,006$
1	Center	0.1	10	4
2	Center	0.4	7	3
3	Center	0.6	7.67	3.33
4	Wall	0.1	>10	5
5	Wall	0.4	9.67	4
6	Wall	0.6	10	4.33

Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Teresa Schnellbach, die vorliegende Masterarbeit gemäß §22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 29. Mai 2022

[illegible]

T. Schnellbach