

Diese Arbeit wurde vorgelegt am
Lehr- und Forschungsgebiet Theorie der hybriden Systeme

**Effiziente und skalierbare
Schallausbreitungsmodellierung zur Vorhersage von
Lärmimmissionen durch Windkraftanlagen**
**Efficient and Scalable Sound Propagation Modeling for
Predicting Wind Turbine Noise Immission**

Bachelorarbeit
Informatik

November 2024

Vorgelegt von Presented by	Peter Schlosshan Matrikelnummer: 422304 peter.schlosshan@rwth-aachen.de
Erstprüfer First examiner	Prof. Dr. rer. nat. Erika Ábrahám Lehr- und Forschungsgebiet: Theorie der hybriden Systeme RWTH Aachen University
Zweitprüfer Second examiner	Prof. Dr. rer. nat. Jürgen Giesl Lehr- und Forschungsgebiet: Programmiersprachen und Verifikation RWTH Aachen University
Betreuer Supervisor	Dr. rer. nat. Pascal Richter Lehr- und Forschungsgebiet: Theorie der hybriden Systeme RWTH Aachen University

Contents

1. Introduction	1
1.1. Motivation	1
1.2. Related work	2
1.3. Contribution	2
1.4. Outline of this work	3
2. Fundamentals of sound and noise	3
2.1. Key acoustic quantities	3
2.2. Wave characteristics	4
2.3. Human perception	5
2.4. Equivalent continuous sound levels	5
3. Predicting noise immission with sound propagation models	6
3.1. Factors affecting sound propagation	6
3.2. Overview of existing models	8
3.2.1. The Harmonoise sound propagation model	9
3.2.2. Comparing Harmonoise to Nord2000	10
3.3. Noise modeling standards in Germany	11
4. The ISO 9613-2 model	12
4.1. Setup	12
4.2. Attenuation factors	13
4.3. Modeling of sound barriers	14
4.3.1. Calculation of the diffracted sound paths	15
4.3.2. Reflection attenuation	18
4.4. Adjusting for wind turbines with the Interimsverfahren	20
4.5. Noise assessment guidelines in TA Lärm	20
5. Implementation and software design	21
5.1. Optimizing data manipulation through vectorization	21
5.2. Efficient data handling with Xarray	22
5.3. Parallel and memory-conscious computation with Dask	23
5.4. Digital elevation model	25
5.5. Overview of the simulation software	26
5.5.1. Configuration	26
5.5.2. Loading the input assets	27
5.5.3. Preparing the input	28
5.5.4. Defining the computation	29
5.5.5. Processing the result	30
6. Verification and performance analysis	31
6.1. Runtime comparison with WindPRO	33
6.2. Evaluating the effectiveness of parallel processing	33

6.3. Accuracy at specific receiver locations	33
6.4. Analysis of contour alignment	37
7. Conclusion and outlook	41
7.1. Impact of our contribution	41
7.2. Future directions	42
A. Appendix	44
A.1. Verification setup details	44
References	46

1. Introduction

Wind is a clean and inexhaustible resource that is available globally. As the movement of air masses, it carries kinetic energy that wind turbines can capture and convert into electricity [1]. However, as demand for renewable energy grows, fewer suitable land-based sites remain for new wind farms, pushing wind turbines closer to residential and community areas [2, 3]. Noise generated by wind turbines is one of the primary obstacles to gaining broad public acceptance of wind energy, as it can lead not only to environmental disturbances and social concerns but also to significant impacts on sleep quality and overall well-being [2, 4, 5]. Systematic reviews have found that individuals exposed to turbine noise are more likely to report sleep disturbances and reduced quality of life [5]. Addressing these challenges is essential to ensure that wind energy can be sustainably integrated into populated regions without compromising public health and quality of life.

1.1. Motivation

The motivation for this work is to improve public acceptance of wind turbines by providing a more accurate and context-specific understanding of their noise impact. In countries like Germany, regulatory guidelines have been established to protect communities from harmful environmental noise effects [6]. These guidelines are essential not only for planning and installing new turbines, but also for building public trust in wind energy infrastructure. A key component in addressing public concerns and adhering to regulations is estimating wind turbine sound emissions before construction begins. This requirement is formalized in German regulations, which mandate a noise assessment report as part of the pre-construction process for new turbines [6].

Producing a reliable noise assessment requires a mathematical model to describe how sound propagates from turbines to various receiver locations, providing quantitative data on expected noise levels. These models differ in their levels of detail and computational intensity [7]. To support public acceptance, the model should avoid overly conservative assumptions and instead be as precise and accurate as possible. The German guidelines, however, mandate a specific sound propagation model that must be followed for compliance in noise assessments [6, 8].

Given the complexity of these calculations, developing the model as software is essential for several reasons. First, accurately calculating sound levels involves multiple variables, such as turbine specifications, atmospheric conditions, and the topography around each receiver. Without software, processing the interactions between these variables at numerous receiver locations would be time-consuming and prone to error. Software allows for the fast and automated computation of sound levels at many receiver locations at once and ensures both accuracy and consistency across all calculations. Moreover, software enables scalable bulk computations that go beyond single receiver locations to provide comprehensive noise maps. By mapping noise levels across an area, planners can visually interpret the spatial distribution of noise and assess compliance with regulatory guidelines. Additionally, a noise simulation program may be

integrated with other simulation tools to allow planners to optimize turbine placements based on multiple constraints, such as noise impact, shadow flicker, and energy efficiency. Such an integrated approach helps with decision-making and ensures that wind energy projects are both effective and publicly accepted.

1.2. Related work

Research in wind turbine noise modeling focuses on achieving sound propagation predictions that are as accurate as possible. Häfele’s study [9] considers the German adaptation of the ISO 9613-2 sound propagation model for high-altitude sources (Interimsverfahren) and validates it with field measurements. Cand et al. [7] examine various sound propagation models and contrast empirical methods like ISO 9613-2 with more complex, computationally demanding numerical models, providing context on the trade-offs between model complexity and accuracy. Bouma [10] implements wind turbine noise modeling in MATLAB using the Harmonoise model, which includes an advanced ground model and atmospheric effects such as wind shear, highlighting an alternative model’s handling of more complex conditions. Bresciani’s model [11] also considers Harmonoise and presents a numerical sound propagation approach that factors in turbine design and complex environmental conditions. Finally, Ushizima’s work [12], while not directly related to sound modeling, demonstrates the value of the computational frameworks Xarray and Dask for efficient, scalable data processing, which is a method also applied in this thesis to achieve high-resolution noise mapping with reduced computational costs.

1.3. Contribution

Our contribution to improving public acceptance of wind turbines lies in the development of a simulation program that implements the ISO 9613-2 sound propagation model, specifically adapted for wind turbines. This program is capable of predicting wind turbine sound immissions at specific receiver locations and can also generate detailed noise maps to illustrate spatial sound distribution. Due to the flexible architectural design of our program, it can be easily extended to allow for adaptations such as different operational modes and placement scenarios. To further improve the accuracy of predictions, we developed an algorithm to compute sound paths diffracted around barrier edges. The program serves as a robust tool for understanding turbine noise immissions, while strictly adhering to German regulatory standards and assessment methodologies.

Our goals for the program are twofold: to achieve high accuracy and resource-efficient performance, and to ensure scalability for handling large computational workloads. This was accomplished through the use of multi-dimensional labeled data structures and the implementation of lazy, parallelized operations across multiple workers and threads. Verification of the noise simulation program demonstrates that it produces outputs highly consistent with established modeling approaches. Furthermore, measurements indicate the program’s strong performance and suggest a high quality in the

solution we have developed.

1.4. Outline of this work

This thesis begins with foundational concepts of sound and noise in Section 2, which covers their physical properties, human impact, and essential acoustic quantities for noise assessment. Next, we provide an overview of sound propagation models in Section 3 by explaining core propagation mechanics and comparing models, with an emphasis on Harmonoise, Nord2000, and German noise modeling standards. In Section 4, we present ISO 9613-2 as the foundation for our simulation program and detail its methodology, adaptations for wind turbine noise, and regulatory adjustments for German compliance. The design and implementation of our program are illustrated in Section 5, where we describe the program’s core structures, efficient data processing techniques, and phases of the workflow. We then evaluate the accuracy and performance of our program in Section 6 and compare it to established tools to analyze its computational efficiency. Finally, in Section 7, we conclude with a summary of our contribution and discuss potential refinements and areas for further development.

2. Fundamentals of sound and noise

To understand the sound propagation models that form the foundation of wind turbine noise immission prediction programs, a basic understanding of the physical and psychological principles of sound and noise is necessary.

Sound is the propagation of pressure waves radiating from a vibrating source through an elastic medium, such as air, water, or solid structures. When particles in the medium are displaced, they cause neighboring particles to move and create a wave of energy that travels outward from the source. The human ear can detect these pressure changes, which leads to the sensation of hearing. Sound waves consist of alternating regions of higher pressure and lower pressure, which move in rhythmic patterns. This propagation occurs at a constant rate known as the speed of sound, with each displaced particle transferring energy to its neighbor in a continuous chain reaction. [13]

2.1. Key acoustic quantities

The physical properties of sound are characterized by the quantities *sound pressure*, *sound power* and *sound intensity*. Each of these can also be expressed as a *level*, which refers to the logarithmic magnitude of the quantity relative to a standard reference value and is usually measured in decibels (dB) [14]. The definition of a level depends on whether the quantity is directly proportional to power or if its square is proportional to power. For a quantity P directly proportional to power, the level is defined as $10 \log_{10}(\frac{P}{P_0})$ dB. In contrast, for a quantity F whose square is proportional to power, the level is defined as $10 \log_{10}(\frac{F^2}{F_0^2})$ dB, which simplifies to $20 \log_{10}(\frac{F}{F_0})$ dB [15].

Sound Pressure Sound pressure is the local pressure variation from the ambient atmospheric pressure caused by a sound wave. It is measured in pascals (Pa) and represents the force per unit area exerted by the sound wave on a surface perpendicular to the direction of the wave. Sound pressure decreases inversely with distance from the sound source and indicates the loudness at a specific location. It is typically measured with a microphone or sound level meter at that location. The human threshold of hearing is approximately 2×10^{-5} Pa, while the threshold of pain is around 20 Pa. Since the square of sound pressure is proportional to power, the corresponding sound pressure level is defined as $L_p = 20 \log_{10}(\frac{p}{p_0})$ dB, with a reference pressure p_0 of 2×10^{-5} Pa [16]. Consequently, a level of 0 dB corresponds to the threshold of human hearing, and only a level of $-\infty$ dB represents the complete absence of sound. [14, 17]

Sound Power The sound power P measures the total acoustic energy emitted by a sound source per unit time and is an intrinsic attribute of the source, independent of distance or surrounding environment. It is measured in watts (W) and represents the rate at which sound energy is emitted in all directions into the environment. Since it is directly proportional to power, its corresponding level is defined as $L_W = 10 \log_{10}(\frac{P}{P_0})$ dB, with P_0 being the reference sound power of usually 10^{-12} W [16]. [14, 17]

Sound intensity The sound intensity I describes the flow of acoustic energy per unit area in a direction perpendicular to that area, making it a vector quantity. For a sound power P passing through an area A , the sound intensity is obtained as $I = \frac{P}{A}$. It is therefore directly proportional to power, such that its level is given by $L_I = 10 \log_{10}(\frac{I}{I_0})$ dB [16]. The reference sound intensity I_0 is usually fixed at 10^{-12} W m⁻². [14, 17]

2.2. Wave characteristics

Since sound is a wave, it also has an amplitude, a wavelength and a frequency. The amplitude represents the peak pressure value of the wave and corresponds to the energy it carries, with a greater amplitude implying greater sound pressure and sound intensity. Wavelength is the distance a wave travels during one complete oscillation and reflects its physical size. The frequency, measured in Hertz (Hz), indicates how many times a wave oscillates each second, with a higher frequency resulting in a higher pitch. However, sound generally does not have a single frequency, but rather consists of a range of frequencies. To reduce the complexity of sound calculations that differentiate between frequencies, these frequencies are typically grouped into bands. Common groupings include octave bands, where each band's upper frequency is twice its lower frequency, and one-third octave bands, which divide each octave into three narrower bands. Each band is represented by its center frequency. [14]

2.3. Human perception

The perception of sound is complex and highly subjective, such that it is difficult to define a loudness quantity that accurately reflects individual variations in hearing sensitivity and the diverse ways in which people perceive sound intensity across different frequencies and environments [18]. Some research suggests that perceived loudness is proportional to the logarithm of sound intensity, while others propose a more gradual power-based relationship, approximated as $L_{\text{perceived}} \propto I^{0.3}$ [17, 19]. Since I is proportional to sound pressure squared, this can also be expressed as $L_{\text{perceived}} \propto p^{0.6}$, and a doubling of the perceived loudness roughly corresponds to a sound pressure level increase of about 10 dB.¹

A more quantifiable method for representing sound as perceived by the human ear is provided by *A-weighting*, which applies a frequency-specific correction to the sound pressure level. This correction accounts for the ear's varying sensitivity to different frequencies, resulting in an A-weighted sound pressure level. To indicate that the measurement is A-weighted, the unit is typically denoted as dB(A). [14, 20]

The term *noise* describes a category of sounds that can negatively disrupt human life, and it is characterized as being unwanted, inappropriate, interfering, distracting and irritating. Intermittent noises that start and stop frequently are considered more bothersome than a constant background hum. Moreover, when a single frequency stands out distinctly from the others, it can increase the level of annoyance [14]. Some standards that describe noise immission, such as the German *TA Lärm*, account for this by incorporating tonality or impulsiveness factors to further adjust the sound pressure level [6].

2.4. Equivalent continuous sound levels

The *equivalent continuous sound pressure level* $L_{p,\text{eq}}$ represents the constant sound pressure level that would deliver the same total acoustic energy as the actual fluctuating sound over a specified period. It effectively averages varying noise levels into a single value to quantify overall noise exposure and is defined as

$$L_{p,\text{eq}} = 10 \log_{10} \left(\frac{1}{T} \int_0^T \frac{p(t)^2}{p_0^2} dt \right), \quad (1)$$

where T is the total time interval in seconds, $p(t)$ is the instantaneous sound pressure in pascals, and p_0 is the reference sound pressure of 2×10^{-5} Pa. An A-weighted version $L_{p,\text{A},\text{eq}}$, which uses the A-weighted sound pressure $p_{\text{A}}(t)$ in place of the unweighted $p(t)$, can also be defined. Lastly, a more differentiated view, broken down by frequency, is also possible. It is given by $L_{p,f,\text{eq}}$, the equivalent continuous octave band sound pressure level, that replaces $p(t)$ with a frequency-dependent $p_f(t)$. [14, 21]

¹Doubling the perceived loudness, i.e. $L_{\text{perceived,after}} = 2L_{\text{perceived,before}}$, implies $p_{\text{after}} = 2^{\frac{1}{0.6}} p_{\text{before}}$. Thus: $L_{p,\text{after}} = 20 \log_{10} \frac{p_{\text{after}}}{p_0} = 20 \log_{10} (2^{\frac{1}{0.6}} \frac{p_{\text{before}}}{p_0}) = 20 \log_{10} 2^{\frac{1}{0.6}} + 20 \log_{10} \frac{p_{\text{before}}}{p_0} \approx L_{p,\text{before}} + 10.03 \text{ dB}$.

3. Predicting noise immission with sound propagation models

The phenomenon of sound can be understood in three distinct stages: emission, propagation and immission. First, sound energy is emitted from the source with some amount of energy and certain characteristics. Next, this sound is modified as it propagates through the environment, until finally, a sound immission occurs at various receiver locations as the sound arrives with a potentially different energy level and character. Besides the question of how the sound source and receiver in the first and last stage should be modeled, the main challenge lies in the propagation stage [7]. To accurately predict the sound pressure at a receiver location for any given sound energy input, a model of the propagation path is needed [2].

There are two different approaches to such a model. One way is to model the real underlying physical mechanisms, which requires knowledge of physical parameters like humidity, air temperature or wind velocity to control how each physical mechanism behaves. The other approach is to use empirical measurements instead, developing equations that best fit the input to the output with the use of possibly abstract parameters. [7]

Even when both source and receiver are constant, the influences that modify sound during propagation can vary both spatially and temporally, which leads to fluctuating immissions. To address this, models often consider typical average or worst-case scenarios, such as equivalent continuous sound levels described in Section 2.4. [7, 22]

3.1. Factors affecting sound propagation

As sound travels from its source to the receiver, it is altered through its physical properties and the surrounding environment. These factors include geometric divergence, atmospheric absorption, ground effects, atmospheric refraction, atmospheric turbulence, and terrain and barrier effects [7].

Geometric divergence Geometric divergence describes the phenomenon that sound spreading from a point source under free-field conditions does so as a sphere with increasing area. This area increase is proportional to the square of the distance r from the source. The sound intensity, describing sound power per unit area, is thus inversely proportional to the square distance and decreases with increasing distance ($I \propto \frac{1}{r^2}$). It follows that sound pressure, whose square is proportional to sound intensity, is inversely proportional to the distance ($p \propto \frac{1}{r}$), and doubling the distance from the source decreases the sound pressure level by about 6 dB.² The geometric divergence is also independent of sound frequency. [7, 23, 11]

²Doubling the distance, i.e. $r_{\text{after}} = 2r_{\text{before}}$, implies $p_{\text{after}} = \frac{1}{2}p_{\text{before}}$. Thus: $L_{p,\text{after}} = 20 \log_{10} \frac{p_{\text{after}}}{p_0} = 20 \log_{10} (\frac{1}{2} \frac{p_{\text{before}}}{p_0}) = 20 \log_{10} \frac{1}{2} + 20 \log_{10} \frac{p_{\text{before}}}{p_0} \approx L_{p,\text{before}} - 6.02 \text{ dB}$.

Atmospheric absorption Sound is the oscillation of air molecules. This movement requires energy, which leads to a loss of transmitted sound energy. Higher frequencies are attenuated more than lower frequencies, meaning that their influence on the overall sound pressure level diminishes with increasing distance [7, 23].

Ground effects Under free-field conditions, there is only a single propagation path for each receiver location by which it can be reached. Adding a ground surface to the model means that the sound can not only propagate in a direct straight line, but is also reflected off the ground, introducing a second propagation path to the receiver. The ground-reflected wave travels a longer distance, which leads to a lower amplitude and a time delay. The time delay then causes a frequency-dependent phase shift, meaning that the crests and troughs of the wave occur at different points in time than those in the direct wave. The direct wave and the reflected wave may then interfere at the receiver location. This interference is either *constructive*, where the wave crests and troughs somewhat align and produce a combined wave that has a higher amplitude than each of the two input waves, or *destructive*, in which case the crests of one wave align with the troughs of the other and vice versa, leading to a combined wave with an amplitude lower than the input waves. Constructive interference thus increases sound pressure, while destructive interference decreases it. Because of this frequency-dependent interference, each sound-receiver pair attenuates some frequencies more than others. However, since sounds typically consist of a mix of frequencies, ground effects become less relevant in the overall sound perception, as the combined effect tends to smooth out frequency-specific interferences. [7, 23, 11]

Atmospheric refraction The atmosphere is inhomogeneous and dynamic, with variations in temperature and wind speed at different altitudes, influencing the speed of sound. Refraction, which bends sound waves toward regions where they propagate more slowly, causes sound to curve based on these varying speeds in the atmosphere. When traveling downwind (with the wind), the sound speed increases with the wind speed, whereas traveling upwind (against the wind) decreases it. Because wind speed usually increases with altitude, it follows that higher altitudes cause sound to move faster when traveling downwind and slower in upwind direction. Sound is thus refracted downwards when traveling downwind and upwards when traveling upwind. Sound speed also increases with temperature, meaning that sound is refracted downwards when the temperature increases with altitude (temperature inversion, e.g. during the night) and upwards when temperature decreases with altitude (temperature lapse; normal condition). In these ways, both wind and temperature affect the sound propagation path. In fact, when sound is refracted downward and bends toward the ground, it may be reflected off the ground multiple times. This typically leads to a moderate sound pressure amplification of less than 3 dB. Upward propagation, on the other hand, can create an acoustic shadow region where sound levels are significantly lower, causing a reduction in sound pressure of as much as 20 dB. [7, 23, 11]

Atmospheric turbulence The main effect of atmospheric turbulence is that it causes random small-scale changes in sound propagation. The sound waves are scattered into different directions, such that, for example, even acoustic shadow regions caused by downward refraction can be reached. Atmospheric turbulence also causes the sound pressure level to fluctuate slightly around a mean value, making it necessary for sound propagation models to perform some temporal and possibly also spatial averaging. [7, 23, 11]

Terrain and barrier effects Any solid object between source and receiver is a barrier. Barriers interrupt the sound propagation path and usually reduce the sound pressure level at the receiver. They are either a natural part of the terrain, or are placed specifically for attenuation. Sound waves can, however, still reach the space behind them, as frequency-dependent diffraction effects cause sound to bend at barrier edges. Barriers may also block ground-reflected paths, effectively nullifying any ground effects and, if ground reflection caused destructive interference at the receiver location, potentially even increasing the sound immission. For wind turbines, the high altitude of sound emission means that dedicated sound barriers are not really feasible and barriers are mainly topographic features. [7, 23]

3.2. Overview of existing models

The main objective of a noise assessment, as outlined in Section 1.1, is to determine the sound pressure levels at various receiver locations. This requires a mathematical model that describes how sound propagates from the turbines to these receivers.

Because the atmospheric conditions affecting sound propagation are highly complex, it is impractical to consider all attributes and their effects. Every practical sound propagation model therefore has some amount of assumptions and simplifications, resulting in a wide range of more or less different models [7]. They can be divided into two main categories [11]. One category are the *engineering methods*, which are primarily based on assumptions and simplifications. It is typical for such models to compute multiple sound attenuation values that are summed up sequentially. They vary in complexity, some even considering the terrain and atmospheric effects in a simplified manner. The other category are *numerical methods*, which are often based on solving complex wave equations. Another approach often followed by numerical models is to view sound not as a wave, but rather as rays or particles. They also take detailed atmospheric conditions into account, and some even support non-flat terrain. Models that fall into this category are useful for calculating more complex propagation scenarios under very specific conditions, but their results cannot easily be generalized to other circumstances, limiting their applicability to a narrow scope. One use-case for such models is the development and validation of engineering methods. [7]

Greater complexity in a model typically involves more input parameters, which must be precise. If they are inaccurate, the results become less meaningful, and a simpler method may be more appropriate [7]. This is illustrated by Monika Agatz [24], who warns that sophisticated, highly detailed propagation models relying on meteorological

	Nord2000	Harmonoise	ISO 9613-2	NMPB
Computing time	Fast	Fast	Very fast	Very fast
Propagation in specific adverse conditions	Yes	Yes	No	No
Meteorological conditions	Yes	Yes	No	No
Terrain model	Yes	Yes	Flat	Flat
Accuracy	Good	Good	Basic	Basic

Table 1: A comparison of the most common engineering methods. Nord2000 and Harmonoise are similar in every listed aspect, as they are both comparatively fast to compute, take meteorological factors and model of the terrain as input and provide relatively good accuracy. Likewise, ISO 9613-2 and NMPB share every compared characteristic. They are both very fast to compute, do not take specific atmospheric conditions into account and assume a simplified, flat terrain, providing only a basic accuracy. The table is adapted from Reference [7], which includes further details about the compared models.

datasets and flexible modeling parameters are impractical for standardized comparisons and difficult to assess. Due to this trade-off between accuracy and generalizability, the accessibility of input data, and the fact that engineering methods are standardized, e.g. in ISO 9613-2, and adopted in regulatory frameworks such as the German TA Lärm, this work focuses on engineering methods.

Four of the most common engineering methods for sound prediction are ISO 9613-2, NMPB³, Nord2000, and Harmonoise, which are compared in Table 1. Both ISO 9613-2 and NMPB provide basic predictions of sound pressure at receivers and are computationally efficient. In contrast, Nord2000 and Harmonoise are more advanced models that offer greater accuracy but are also more complex to implement. The principles underlying the Nord2000 model are discussed in [25], with more comprehensive descriptions available in [26] and [27]. The most detailed account is provided by [28]. Nord2000 has been explicitly validated for wind turbine noise in [29]. The Harmonoise model is introduced in the following section and is subsequently compared to Nord2000.

3.2.1. The Harmonoise sound propagation model

The Harmonoise model was developed with the goal of predicting environmental noise and was intended as a foundation for a future standardized European method for

³Nouvelle Méthode de Prévision du Bruit

environmental noise mapping. It was mainly developed from 2001 to 2004 under the European *Harmonoise* project, which focused specifically on road and rail traffic noise. The *Harmonoise* project has also benefited from insights gained during the Scandinavian Nord2000 project, which ran a few years prior, and adapted some of its results into the *Harmonoise* model. To assert accuracy, the model was validated against typical scenarios, such as sources close to the ground, propagation distances up to 1 km, and moderate atmospheric refraction conditions. A detailed description of this state of the model is available in reference [30]. After its initial development, the model was improved upon during the European *Imagine* follow-up project (2004–2006). The project extended the prediction model to cover noise sources at higher altitudes and greater distances, which made it applicable to aircraft and industrial noise. The model as described in reference [31] comprises these updates and includes further modifications and optimizations made following a series of test calculations. [30, 31, 32]

One of the main features of the model is a more accurate physical description of sound propagation along the ground, which reduces errors from incorrect assumptions about ground properties [33]. It can be applied to different terrain profiles and is capable of predicting noise levels under any meteorological situation, provided that the conditions can be represented by an equivalent linear gradient of sound speed versus height [34]. The model furthermore uses a Fresnel weighting approach that was initially based on Nord2000 and handles frequencies between 25 Hz and 10 kHz in one-third octave bands [33, 30].

3.2.2. Comparing *Harmonoise* to Nord2000

In 2009, test calculations were conducted to compare *Harmonoise* with both precise numerical reference models and the Nord2000 model. The findings indicated that *Harmonoise* is an accurate engineering tool for outdoor sound propagation, marking a significant improvement over older models like ISO 9613-2. Although both *Harmonoise* and Nord2000 are limited to linear sound speed profiles, *Harmonoise* generally performs better, particularly in scenarios involving downward refraction over flat ground. A key distinction between *Harmonoise* and Nord2000 is their handling of atmospheric refraction: in Nord2000, sound rays are modeled as curving toward or away from the ground, while in *Harmonoise*, the sound rays remain straight, with the ground itself treated as if it curves. This approach is effective for short and moderate distances, but it limits *Harmonoise*'s accuracy over large distances where multiple ground reflections occur; Nord2000 incorporates a correction term here. Overall, the test calculations concluded that *Harmonoise* is typically slightly more accurate than Nord2000. However, it is important to note that the tests did not specifically consider sound emitted by wind turbines, whereas Nord2000 has been explicitly validated for wind turbine noise in [29]. [31, 32]

Another difference to Nord2000 is a simplification that reduces computational complexity. Both models represent the ground as ground segments with diffracting edges, where N is the number of ground segments and M is the number of edges. *Harmonoise* simplifies the model by avoiding a specific calculation, which results in a computational

complexity of $\mathcal{O}(N + M)$ compared to Nord2000's $\mathcal{O}(N^M)$. This makes Harmonoise more efficient, especially in complex environments with multiple diffraction effects, and simplifies implementation by treating various diffraction scenarios equally. [30]

3.3. Noise modeling standards in Germany

In Germany, wind turbines taller than 50 meters require a permit, which includes a mandatory sound emission prognosis that must comply with the *Technical Instructions on Noise Protection*⁴, commonly abbreviated as *TA Lärm*. The goal of TA Lärm is to protect the public and local residents from harmful environmental effects caused by noise, as well as to prevent such harmful effects in the future. Harmful environmental impacts, as defined by this regulation, include noise emissions that, due to their nature, intensity, or duration, have the potential to cause danger, significant disadvantages, or considerable annoyance to the public or the neighborhood. [6]

The TA Lärm requires sound prognosis calculations to be conducted using ISO 9613-2. However, this approach is primarily designed for sound sources close to the ground and is not intended for high-altitude sources like wind turbines. As a result, actual sound levels are louder than predicted, because less ground attenuation occurs at higher altitudes. For example, a 2015 study conducted by the German *State Agency for Nature, Environment, and Consumer Protection of North Rhine-Westphalia*⁵ [35] demonstrated a considerable discrepancy between wind turbine noise levels predicted by ISO 9613-2 and actual sound measurements. Real-world sound levels have been found to be much louder than predicted, mainly because the model for ground attenuation is not applicable to wind turbines. This emphasized the need for a new model. [36, 6]

Based on these findings and comparisons with more advanced noise propagation models, the German *Standards Committee for Acoustics, Noise Reduction, and Vibration Technology*⁶ released the *Interim Procedure for Predicting Noise Immissions from Wind Turbines*⁷ in 2015 as a temporary solution until they develop a sound propagation model specifically for wind turbines, which has not yet happened. This procedure, commonly referred to as the *Interimsverfahren*, contains modifications to ISO 9613-2 to make it applicable to wind turbines. In 2016, the German *Federal-State Working Group for Immission Control*⁸ updated their *Guidelines for Noise Immission Control of Wind Turbines*⁹ to recommend the Interimsverfahren instead of the unmodified ISO 9613-2. [37, 8]

⁴Technische Anleitung zum Schutz gegen Lärm (TA Lärm)

⁵Landesamt für Natur, Umwelt und Verbraucherschutz Nordrhein-Westfalen (LANUV NRW)

⁶Normenausschuss Akustik, Lärminderung und Schwingungstechnik (NALS)

⁷Interimsverfahren zur Prognose der Geräuschimmissionen von Windkraftanlagen

⁸Bund/Länder-Arbeitsgemeinschaft für Immissionsschutz (LAI)

⁹Hinweise zum Schallimmissionsschutz bei Windkraftanlagen

4. The ISO 9613-2 model

The ISO 9613-2 model, with adjustments from the Interimsverfahren, is selected for our wind turbine noise simulation software due to its regulatory acceptance and suitability for assessing environmental noise from high-altitude sources. Compared to NMPB, ISO 9613-2 provides an internationally recognized, standardized approach that ensures compliance with German noise assessment regulations.

While Harmonoise and Nord2000 offer more sophisticated modeling of atmospheric and terrain influences and include complex meteorological effects, they also require significantly more detailed input data and computational resources. These models are excellent for specialized scenarios but are often impractical for routine assessments due to their complexity and data requirements. In contrast, ISO 9613-2 offers a balanced approach and achieves acceptable accuracy with simpler, more accessible inputs. This makes it more feasible for broader applications. The model's alignment with regulations and practical trade-off between accuracy and usability makes it a suitable choice for this application.

4.1. Setup

The model is limited to predicting under a single representative meteorological condition, "moderate downwind", which is similar to temperature inversions typically occurring at night [34]. More specifically, the wind is assumed to blow from the source towards the receiver, in a direction that deviates not more than 45° from the direct source-receiver line. The wind speed, measured at a height between 3 m and 11 m above the ground, is expected to lie within a range of 1 m s^{-1} to 5 m s^{-1} . As explained in Section 3.1, these conditions correspond to a downward refraction of the sound, leading to higher sound levels at the receivers. To adjust for the real climatic conditions of the area, a single meteorological factor C_{met} may be incorporated at the end of the calculation.

For any given octave band center frequency f , ISO 9613-2 defines the equivalent continuous octave band sound pressure level under downwind conditions as

$$L_{p,f,\text{eq}}(\text{DW}) = L_W + D_C - A. \quad (2)$$

L_W is the sound power level of the point source for the octave band with center frequency f , relative to a reference sound power of 1 pW. D_C is a directivity correction factor that indicates how much the sound level from the point sound source deviates in a specific direction compared to an omnidirectional source. For an omnidirectional source emitting into free space, D_C is 0 dB. A represents the frequency-dependent total attenuation that affects the sound in the given octave band as it propagates from the source to the receiver.

Using $L_{p,f,\text{eq}}$, the equivalent continuous A-weighted sound pressure level under down-

wind conditions is calculated as

$$L_{p,A,eq}(DW) = 10 \log_{10} \left(\sum_{i=1}^n \left(\sum_{j=1}^8 10^{0.1(L_{p,f,eq}(ij) + A_f(j))} \right) \right), \quad (3)$$

where n is the number of sound sources, $L_{p,f,eq}(ij)$ is the value of $L_{p,f,eq}$ for source i and octave band j , and $A_f(j)$ is the A-weighting of octave band j . First, the sound pressure levels $L_{p,f,eq}$ for every source and every octave band are A-weighted. These frequency-dependent weighted levels are then summed up for each source, resulting in a single weighted sound pressure level per source. Finally, these levels are combined to the total A-weighted sound pressure level at the receiver location.

As mentioned, ISO 9613-2 assumes conditions that are favorable for sound propagation and may lead to immission estimates that are too high. This can be corrected through long-term averaging that accounts for varying meteorological conditions, such that the *long-term average A-weighted sound pressure level* is given as

$$L_{p,A,eq}(LT) = L_{p,A,eq}(DW) - C_{met}, \quad (4)$$

where C_{met} is a meteorological correction factor that depends on local meteorological statistics for wind speed, wind direction and temperature gradients.

4.2. Attenuation factors

The total attenuation, as required for Equation (2), consists of multiple attenuation factors that are computed individually. It is defined as

$$A = A_{div} + A_{atm} + A_{gr} + A_{bar} + A_{misc}, \quad (5)$$

where A_{div} is the attenuation caused by geometrical divergence, A_{atm} is the attenuation resulting from atmospheric absorption, A_{gr} is the attenuation due to ground effects, A_{bar} is the attenuation caused by barriers, and A_{misc} is the attenuation from miscellaneous other effects.

Geometric divergence Given the distance d from the source to the receiver, the geometrical divergence for spherical free-field spreading is

$$A_{div} = 20 \log_{10} \left(\frac{d}{d_0} \right) + 11, \quad (6)$$

where the reference distance d_0 is fixed at 1 m. A_{div} is frequency-independent.

Atmospheric absorption The attenuation due to atmospheric absorption is given as

$$A_{atm} = \frac{\alpha d}{1000}, \quad (7)$$

with atmospheric attenuation coefficient α and sound-receiver distance d . Coefficient α depends on atmospheric conditions like temperature and humidity and can be found in ISO 9613-2 [21] table 2. A_{atm} is highly frequency-dependent.

Ground attenuation ISO 9613-2 also considers ground effects, albeit in a simplified manner. Because the model assumes downwind conditions and the sound path curves downward, this attenuation is mainly influenced by the ground sections close to the source and receiver. The calculation of ground attenuation is thus divided into a source region, a receiver region, and a less relevant middle region between them. The surface of each ground region is described through a ground factor G , which ranges from $G = 0$ for hard ground like concrete and water to $G = 1$ for porous ground like grass and soil. Values between 0 and 1 represent mixed ground. The ground factors are then used to calculate the region-specific attenuations A_s , A_r and A_m , as shown in ISO 9613-2 table 3. Summing these up yields the total ground attenuation for the given octave band:

$$A_{\text{gr}} = A_s + A_r + A_m. \quad (8)$$

The model also specifies an alternative method for calculating the ground attenuation. When only the total A-weighted sound power level is available, and the sound travels over mixed ground with no particularly dominant frequency (i.e., low tonality), A_{gr} may also be calculated independent of frequency as

$$A_{\text{gr}} = 4.8 - \frac{2h_m}{d} \left(17 + \frac{300}{d} \right). \quad (9)$$

The value h_m is the mean height of the source-receiver path, and d is the distance between source and receiver. When using the alternative method, the directivity correction D_C used in Equation (2) must include a value D_Ω as defined in ISO 9613-2 Equation 11.

Barrier attenuation Barrier effects are expressed through the attenuation value A_{bar} . Any object that has a closed surface without wide cracks or gaps, a sufficiently large surface density, and a horizontal extent greater than the wavelength, qualifies as a screening obstacle. It is represented by a barrier with vertical side edges, rising directly up, and straight top edges that may be sloped. Because the barrier attenuation as described in ISO 9613-2 lacks generalizability and is partially ill-defined, ISO/TR 17534-3 [38] was released to clarify and extend the original standard. Barrier attenuation is discussed with more detail in the following Section 4.3.

Miscellaneous attenuation Effects of sound propagation through vegetation, through industrial sites, and through housing areas are covered by the attenuation A_{misc} . Each of these effects may be modeled as a sound ray that bends over the obstructed area with a curving radius of 5 km. Further details can be found in ISO 9613-2 Annex A.

4.3. Modeling of sound barriers

The barrier attenuation is based on the diffraction of sound at barrier edges. Diffraction over the top and around the sides of barriers are computed separately and combined as

$$A_{\text{bar}} = -10 \log_{10} \left(10^{-0.1A_{\text{bar,top}}} + 10^{-0.1A_{\text{bar,side1}}} + 10^{-0.1A_{\text{bar,side2}}} \right). \quad (10)$$

Barrier attenuation over the top is given as $A_{\text{bar,top}} = D_{\text{Z,top}} - A_{\text{gr}}$, clipped to be nonnegative. The subtraction of the ground attenuation reflects the fact that ground effects become less relevant when barriers are considered, as mentioned in Section 3.1, while the value $D_{\text{Z,top}}$ describes the diffraction effects. It is calculated in the unique vertical plane E_V that is perpendicular to the x-y-plane and includes both the source S and the receiver R , which corresponds to a normal vector of

$$\vec{n}_{E_V} = \vec{SR} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

The plane can then be written as $E_V = \{X \mid \vec{SX} \cdot \vec{n}_{E_V} = 0\}$, where $\cdot$ denotes the dot product and X is any point on the plane. This equation is fulfilled exactly by those points X that lie in the plane. [38]

The barrier attenuations around the sides are $A_{\text{bar,side1}} = D_{\text{Z,side1}}$ and $A_{\text{bar,side2}} = D_{\text{Z,side2}}$, clipped to be at least 0. These values are determined in the lateral plane E_L , which contains both source and receiver and is perpendicular to the plane E_V . A normal vector is therefore $\vec{n}_{E_L} = \vec{SR} \times \vec{n}_{E_V}$, and the plane can be written as $E_L = \{X \mid \vec{SX} \cdot \vec{n}_{E_L} = 0\}$.

The calculation of the D_Z values is based on the difference between the length of the direct source-receiver path that ignores all barriers and the diffracted path around the barrier edges. The diffracted path is the shortest possible polygon line that reaches from source to receiver and avoids barriers. This construction can be compared to a spanned rubber band that is fixed at points S and R and encloses as few blocking obstacles as possible [38]. The problem is closely related to the Euclidean Shortest Path problem, where a shortest path from a point S to a point R needs to be found such that it does not intersect with any polygon barriers [39]. However, as it is described in ISO/TR 17534-3 [38], the problem of finding the diffracted path is a special case of the Euclidean Shortest Path problem where the polygon path is required to be convex, curving either only left or only right. This may reduce the problem to the iterative construction of a convex hull, as outlined in the following Section 4.3.1 [39].

4.3.1. Calculation of the diffracted sound paths

The idea of using a convex hull for computing the diffracted paths is outlined in Algorithm 1 for the case of the two lateral paths. A similar construction can be used to determine the top path.

Algorithm 1 defines a function `LATERALPATHS3D`, which takes the three-dimensional barriers and the source and receiver locations as input and returns the two lateral diffraction paths. The function first determines the normal vector n_{E_L} and computes the intersection of E_L with the barriers. This three-dimensional section is then flattened to two dimensions through a transformation that translates and rotates it into the xy plane. Together with the flattened source and receiver locations, the resulting polygons are passed to `LATERALPATHS2D`, which determines the diffraction paths in

the plane E_L in two dimensions. These paths are finally transformed back into three dimensions and get returned.

LATERALPATHS2D first filters the polygons for those crossing the direct source-receiver line. If there are no such polygons, then both diffraction paths are equivalent to the direct line, and the function returns. Otherwise, the vertices of the filtered polygons are partitioned into two subsets, with each subset containing the vertices on one side of the line. Using these vertex sets, two convex hull objects are constructed that allow for points to be added incrementally. The idea behind this is that each hull represents a diffraction path as it is being constructed. Initially, each hull contains the flattened source and receiver points, and the vertices of one side of the polygons crossing the direct line. Under the assumption that any points added to the hull are on that same side of the direct line, the hull never extends to the other side and the direct line is always part of the hull. Removing the direct line yields a path that connects S and R while avoiding all points stored in the hull. Repeatedly expanding the convex hull with all polygon vertices that are crossed by its corresponding path finally results in the diffraction path.

This approach is realized by EXPANDTOPATH, which takes a convex hull object, the polygons, and the direct line as input. It first obtains the current path by removing the direct line from the hull and then determines the polygons that this path crosses. Next, a loop is entered. In each iteration, the vertices of the crossing polygons are added to the hull, the new path is determined, and the crossing polygons are updated accordingly. This is repeated until the path does not cross any polygons, and the path is returned.

One limitation of this algorithm is that it may fail to identify one or both lateral paths in certain edge cases. The line connecting the source and receiver can be divided into three segments: a segment before the source, a segment between the source and receiver, and a segment after the receiver. While the algorithm successfully handles cases where barriers intersect the middle segment, it fails to determine a diffraction path if a barrier encountered during the expansion of the convex hull crosses either the first or last segment. Future work thus requires refinement of the proposed algorithm.

The full algorithm for both vertical and lateral diffraction has been implemented in our program as part of the sound prognosis model. Three-dimensional operations in LATERALPATHS3D are mainly performed with the *Trimesh* package, which provides a class that represents 3D objects as collections of triangles. The package allows for computing plane-mesh intersections and contains predefined functions for creating, combining or inverting various transformation matrices [40]. Operations in two dimensions are performed through the *Shapely* library, which contains classes to represent 2D objects such as points, line segments and polygons. It also provides a spatial indexing data structure based on R-trees that allows for fast geometrical lookup operations, such as finding all polygons crossed by a line [41]. The convex hull computation uses the *SciPy* package, which defines a Convex Hull class that determines a hull incrementally using the Quickhull algorithm [42].

To illustrate and verify the computation of the diffracted paths, two random locations in the region around the German city of Herzogenrath have been chosen as the

Algorithm 1 Construction of the diffracted propagation path in the lateral plane E_L .

```

1: function LATERALPATHS3D( $s, r, barriers$ )
2:    $normal \leftarrow (r - s) \times ((r - s) \times (0, 0, 1))$ 
3:    $origin \leftarrow s$ 
4:    $section \leftarrow \text{PLANESECTION}(barriers, normal, origin)$ 
5:    $flatten \leftarrow \text{GETFLATTENTRANSFORMATION}(normal, origin)$ 
6:    $polygons \leftarrow \text{TRANSFORM}(section, flatten)$ 
7:    $sFlat \leftarrow \text{TRANSFORM}(s, flatten)$ 
8:    $rFlat \leftarrow \text{TRANSFORM}(r, flatten)$ 
9:    $leftPath, rightPath \leftarrow \text{LATERALPATHS2D}(sFlat, rFlat, polygons)$ 
10:   $unflatten \leftarrow \text{INVERTTRANSFORMATION}(flatten)$ 
11:  return  $\text{TRANSFORM}(leftPath, unflatten), \text{TRANSFORM}(rightPath, unflatten)$ 
12: end function

13: function LATERALPATHS2D( $s, r, polygons$ )
14:   $line \leftarrow \text{LINE}(s, r)$ 
15:   $crossingPolygons \leftarrow \text{GETCROSSINGPOLYGONS}(polygons, line)$ 
16:  if  $crossingPolygons$  is empty then
17:    return  $line, line$ 
18:  end if
19:   $vertices \leftarrow \text{POLYGONS TO VERTICES}(crossingPolygons)$ 
20:   $leftVertices \leftarrow \text{FILTERLEFTFROMLINE}(vertices, line)$ 
21:   $rightVertices \leftarrow \text{FILTERRIGHTFROMLINE}(vertices, line)$ 
22:   $leftHull \leftarrow \text{CONVEXHULL}(\{sFlat, rFlat\} \cup leftVertices)$ 
23:   $rightHull \leftarrow \text{CONVEXHULL}(\{sFlat, rFlat\} \cup rightVertices)$ 
24:   $leftPath \leftarrow \text{EXPANDTOPATH}(leftHull, polygons, line)$ 
25:   $rightPath \leftarrow \text{EXPANDTOPATH}(rightHull, polygons, line)$ 
26:  return  $leftPath, rightPath$ 
27: end function

28: function EXPANDTOPATH( $hull, polygons, line$ )
29:   $path \leftarrow \text{REMOVEFROMHULL}(hull, line)$ 
30:   $crossingPolygons \leftarrow \text{GETCROSSINGPOLYGONS}(polygons, path)$ 
31:  while  $crossingPolygons$  is not empty do
32:     $vertices \leftarrow \text{POLYGONS TO VERTICES}(crossingPolygons)$ 
33:     $hull \leftarrow \text{EXPANDHULL}(hull, vertices)$ 
34:     $path \leftarrow \text{REMOVEFROMHULL}(hull, line)$ 
35:     $crossingPolygons \leftarrow \text{GETCROSSINGPOLYGONS}(polygons, path)$ 
36:  end while
37:  return  $path$ 
38: end function

```

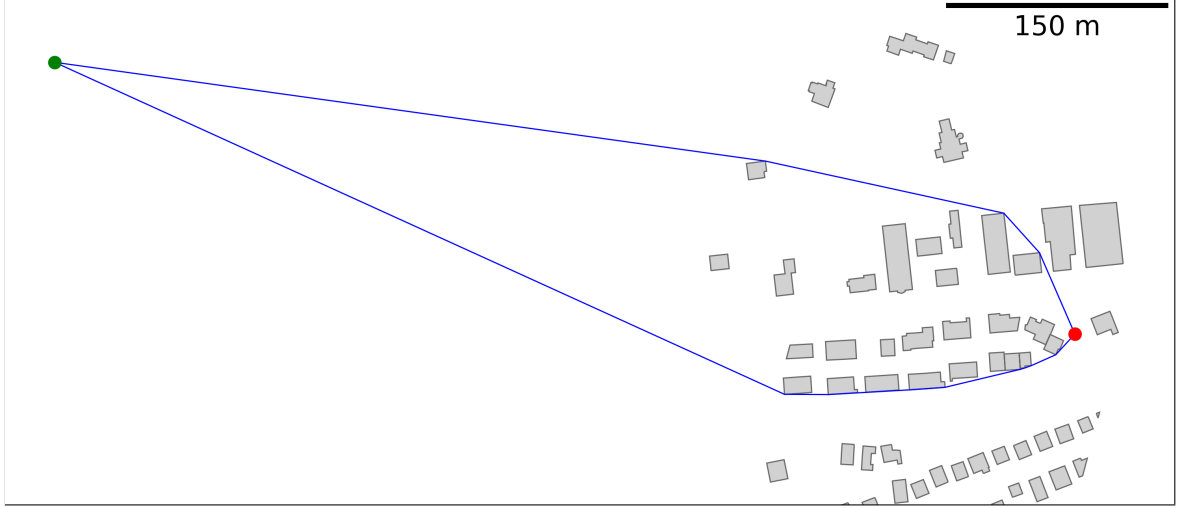


Figure 1: The plane E_L with the two lateral diffraction paths. Source and receiver are marked by a green and red dot, respectively. Because the sound source is considerably higher than the buildings, the plane does not intersect any barriers around the source, and polygons only appear in proximity of the receiver, where the ground is closer. The intersection of E_L with the ground becomes visible in the bottom right corner, as the polygons abruptly end at a line orthogonal to the direct S - R line.

source and receiver positions. The source has been set to a height of 50 m, and the receiver is 2 m above the ground. While the contours of the buildings surrounding them are taken from OpenStreetMap, the building heights were not available and have been randomly generated according to a normal distribution with a mean height of 15 m and a standard deviation of 4 m. Figure 1 shows the two lateral diffraction paths in the plane E_L as computed by our implementation of Algorithm 1. The top path in the E_V plane is visualized in Figure 2. Finally, a 3D rendering of the situation is shown in Figure 3.

4.3.2. Reflection attenuation

As discussed in Section 3.1, sound can be reflected off the ground surface, a phenomenon accounted for in ISO 9613-2 through the ground attenuation factor A_{gr} . However, the model can also incorporate reflections off barriers, such as building facades. This effect is modeled using *image sources*, which are auxiliary sound sources positioned so that the direct path from each image source to the receiver mimics the path of the sound wave reflected off the barrier surface from the real source. Each image source is treated like a separate sound source, with adjusted sound power levels that depend on the material properties of the reflective surface and the angle of incidence.

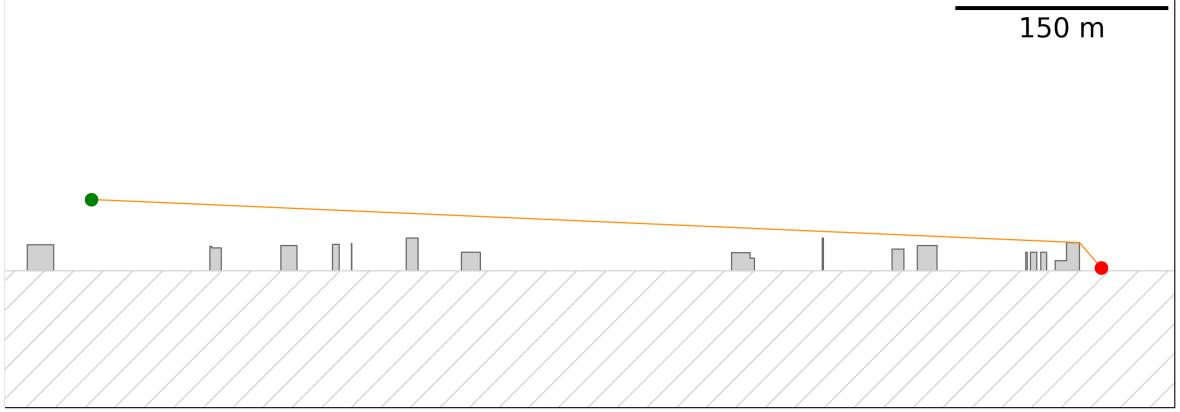


Figure 2: The plane E_V with the propagation path for diffraction over top edges. The height difference between source, shown in green, and receiver, marked red, becomes clearly visible and causes a reduced number of diffraction edges. The only diffraction edge is the top-right edge of the rightmost building.

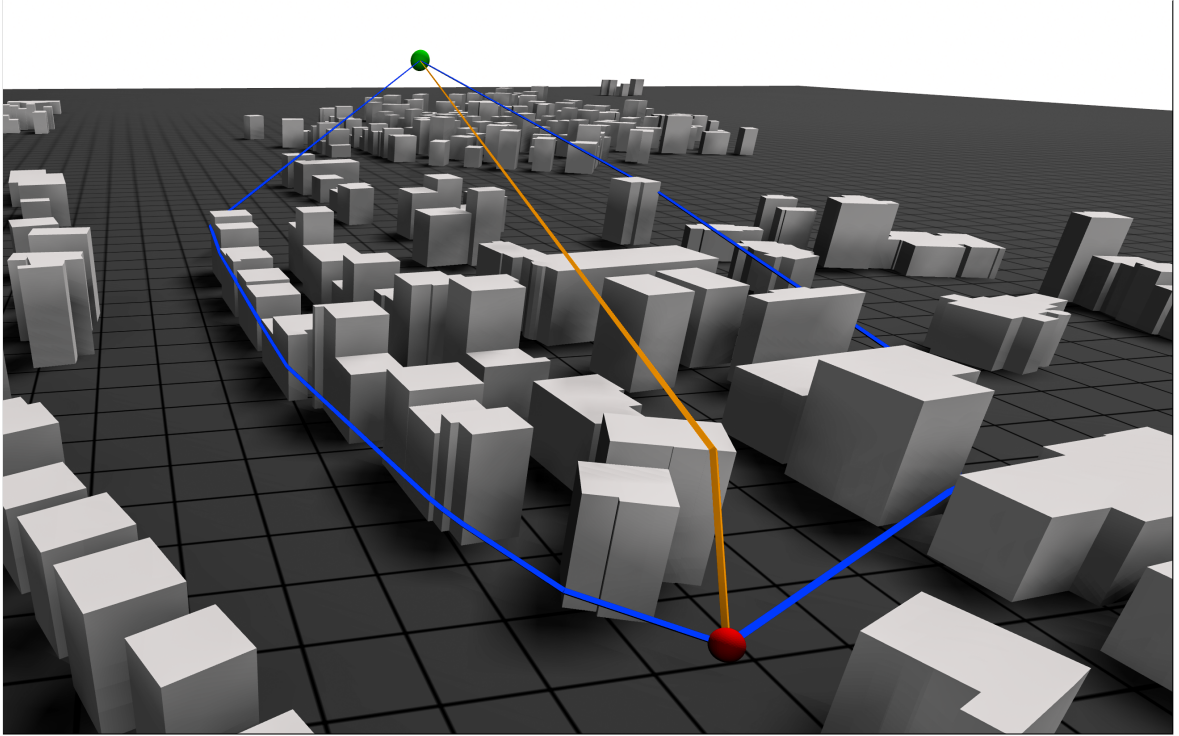


Figure 3: A 3D rendering of the buildings, the top diffraction path (orange) and the two lateral diffraction paths (blue). Source and receiver are represented by a green and red sphere, respectively. The image was created with the *Trimesh* and *Pyrender* Python packages.

4.4. Adjusting for wind turbines with the Interimsverfahren

The *Interimsverfahren*, released in 2015 by the German *Standards Committee for Acoustics, Noise Reduction, and Vibration Technology*¹⁰, adapts ISO 9613-2 specifically for high-altitude sources and introduces several adjustments.

Firstly, sound calculations must be frequency-dependent, unlike in ISO 9613-2, which allows for frequency-independent analysis. ISO 9613-2 permits users to apply attenuation terms based on a single sound power level of 500 Hz and also use simplified ground attenuation with the alternative method. However, the *Interimsverfahren* mandates the frequency-dependent approach. When sound power levels are not available for each octave band, a reference spectrum from the LAI guidelines may be applied to the overall sound power level.

Secondly, ground attenuation (A_{gr}) is fixed at -3 dB, whereas the ISO 9613-2 alternative method would yield values ranging from 0 dB to 4.8 dB. Furthermore, the directivity correction (D_C) is set to 0, in contrast to ISO 9613-2's alternative method, which typically assigns a value of approximately 3 dB. Additionally, the *Interimsverfahren* specifies fixed atmospheric conditions for calculations and sets the temperature at 10°C and the relative humidity at 70%. Finally, the meteorological correction (C_{met}) is disregarded, which corresponds to constant downwind conditions. [37, 8, 43]

4.5. Noise assessment guidelines in TA Lärm

The German *TA Lärm* is a regulatory guideline that sets noise emission limits to protect the public from harmful environmental noise. It provides criteria to assess and manage noise from industrial and commercial activities, which ensures that noise levels stay within acceptable limits for human health and the environment. [6]

For the assessment of noise immssions, *TA Lärm* builds on top of ISO 9613-2 to provide a prognosis procedure, the result of which are assessment levels, generally denoted as L_r . The assessment level L_G of the total noise exposure expected after the commissioning of a sound-emitting facility, is determined by combining the assessment level L_V of the pre-existing noise exposure with the assessment level L_Z of the additional noise exposure:

$$L_G = 10 \log_{10}(10^{0.1L_V} + 10^{0.1L_Z}). \quad (11)$$

Each assessment level L_r is defined over an assessment period T_r , which is divided into N sub-periods T_j . Sound emissions and adjustments remain constant during each sub-period, as in cases of distinct operational modes. For each sub-period, the equivalent continuous A-weighted sound pressure level $L_{p,A,eq}$, combined from all sources, is determined. The level is then adjusted using the meteorological correction C_{met} and three psychoacoustic factors. These factors – adjustments for tonality and informational content (K_T), impulsiveness (K_I), and time periods with increased sensitivity (K_R) – are designed to account for the way sound is perceived by humans and may vary for each sub-period. The assessment level is thereby defined through a weighted

¹⁰Normenausschuss Akustik, Lärminderung und Schwingungstechnik (NALS)

sum of the sound levels during each sub-period as

$$L_r = 10 \log_{10} \left(\frac{1}{T_r} \sum_{j=1}^N T_j \cdot 10^{0.1(L_{p,A,eq,j} - C_{met} + K_{T,j} + K_{I,j} + K_{R,j})} \right). \quad (12)$$

The time that a sound source k is active during sub-period T_j is denoted as $T_{E,k,j}$. The A-weighted sound pressure level for sub-period T_j is then given by the weighted sum of each source's A-weighted sound pressure during that period:

$$L_{p,A,eq,j} = 10 \log_{10} \left(\frac{1}{T_j} \sum_k T_{E,k,j} \cdot 10^{0.1L_{p,A,eq,k,j}} \right) \quad (13)$$

The value $L_{p,A,eq,k,j}$ is equivalent to $L_{p,A,eq}(DW)$ as specified in the ISO 9613-2 model and is calculated accordingly. The calculation should be performed for each turbine k with its corresponding sound power levels during sub-period T_j .

5. Implementation and software design

As outlined in Section 1.3, the goal of this work is the creation of a program that reliably predicts wind turbine noise immissions while being both efficient and scalable. The program is developed in Python 3.10 and based on the German variant of the ISO 9613-2 sound propagation model detailed in Section 4. It is designed for seamless integration into a broader ecosystem of simulation tools, including user-facing web applications, shadow cast simulations, and optimization engines that assess combined simulation results to identify ideal wind turbine locations.

Much of the data handled in sound modeling is inherently multidimensional (see Section 5.5.2), which makes tools necessary that can efficiently process complex datasets. Our program is therefore built around the open-source Python framework *Xarray* for handling labeled multidimensional data and utilizes *Dask* for performance optimization through explicit parallel processing and efficient memory management. This approach allows our program to overcome runtime and memory constraints that would otherwise make such large-scale simulations impractical.

5.1. Optimizing data manipulation through vectorization

A primary requirement for our simulation software is high performance. However, while Python is flexible and user-friendly, it is often regarded as slow for computationally intensive tasks due to its interpreted and dynamically typed nature. Unlike compiled languages like C or C++, Python executes code line by line at runtime, which incurs performance overhead. This is especially noticeable in operations within loops, where Python's need to manage data types and interpret instructions dynamically can significantly slow down execution. As a result, Python's flexibility often comes at the cost of speed, particularly in numerically intensive scenarios. [44, 45]

A common strategy to overcome performance limitations in Python is to offload heavy computations to lower-level languages like C or C++, which can execute operations much faster. Python provides an API to call such compiled code, enabling developers to write performance-critical code in a compiled language while retaining Python’s readability and usability for the broader program structure. The *NumPy* library uses this approach by implementing most of its core operations in C. This design allows NumPy to perform array operations much faster than native Python, mainly by avoiding the overhead of Python loops. Instead of iterating through arrays element-by-element, NumPy uses *vectorization*, a technique where operations are applied to entire arrays or large chunks of data at once [44]. This approach leverages highly optimized C routines and, in some cases, can take advantage of CPU-level optimizations such as SIMD (Single Instruction, Multiple Data) when supported by the underlying libraries [46, 44]. By offloading array operations to compiled code, NumPy achieves notable performance gains, which makes it well-suited for numerical and scientific computing [44].

Most calculations defined in the ISO 9613-2 model and its adjustments can be expressed as vectorized operations. However, due to limitations in the used libraries, our implementation of the barrier attenuation algorithm proposed in Section 4.3.1 cannot be vectorized across arbitrary dimensions. This presents a significant performance challenge for the program, as the interpreted nature of Python becomes a bottleneck. If computational efficiency is a priority, users have the option to disable barrier attenuation in the configuration settings.

5.2. Efficient data handling with Xarray

Despite the advantages of vectorized operations, NumPy arrays lack the concept of labeled dimensions or coordinates. This means that vectorized operations require all input arrays to share the same dimensions in the same order, which makes it challenging to extend NumPy arrays for higher dimensions with meaningful labels. To address this, we use Xarray, an open-source Python library specifically designed to efficiently manipulate labeled, multidimensional data.

The fundamental data structure in Xarray is the *Variable*, a multidimensional array that holds data along with labels for each dimension and optional metadata attributes. Typically, users interact with Xarray’s more comprehensive structures, *DataArrays* and *Datasets*, which encapsulate *Variables* and provide a high-level, user-friendly interface. By default, *Variables* store data using NumPy arrays, but Xarray also supports alternative array backends like Dask, as discussed in Section 5.3. [47, 48]

A *DataArray* extends a *Variable* by associating coordinates along each dimension, describing the contained data even further. Each coordinate is itself a *Variable*, typically linked to a single dimension of the same name. [47]

Multiple related *Variables* can be organized within a *Dataset*, where all variables share common dimensions. *Datasets* can also include shared coordinates, which allows for simultaneous operations across all contained *Variables*. For example, given a climate *Dataset* `ds`, data recorded in February can be selected with

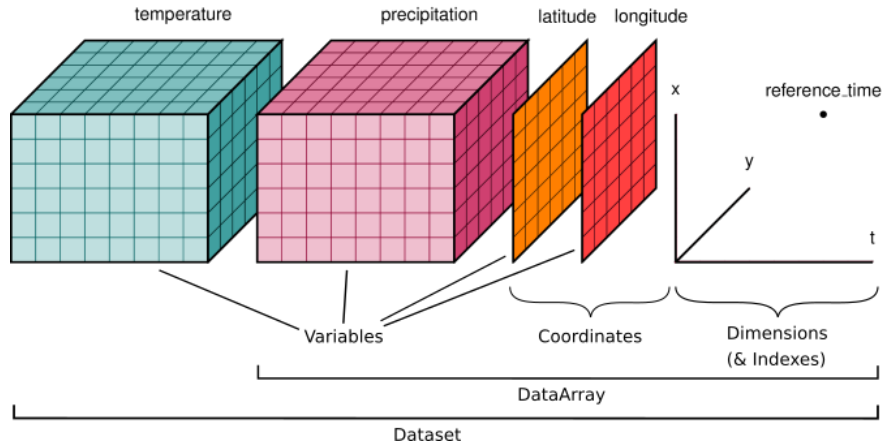


Figure 4: The fundamental Xarray concepts. Two three-dimensional data variables *temperature* and *precipitation* are shown, with dimensions x , y , and t . The data variables are described by two 2D variables *latitude* and *longitude* that span the x and y dimensions and serve as coordinates, assigning a latitude-longitude-label to each x - y position. They can be combined with a single data variable to form a `DataArray`, or with multiple data variables to form a `Dataset` where coordinates are shared. The figure is taken from [48].

`ds.sel(time='february')`, which filters all data variables, such as `temperature` and `precipitation`. Similarly, element-wise operations like calculating square roots of all temperature and precipitation values, can be performed across the `Dataset` using expressions like `np.sqrt(ds)`. When accessing individual `Variables` from a `Dataset`, they are dynamically wrapped as `DataArrays` that include relevant coordinates, ensuring users interact only with fully-labeled data. [47, 48]

These fundamental Xarray concepts are illustrated in Figure 4 using a climate dataset. The `DataArray` and `Dataset` structures allow Xarray to manage data operations more intuitively than pure NumPy. Before performing any operation, Xarray automatically aligns dimensions by their labels and coordinates. If one object has a dimension that the other lacks, Xarray adjusts the data structure as needed, allowing operations to work seamlessly without manual intervention [47]. This automatic alignment and adjustment may reduce cognitive load and make the code less error-prone. The Xarray API closely resembles that of NumPy, supporting functions like `mean`, `sum`, `transpose`, and `dot`, and many NumPy functions, such as `sqrt` and `arccos`, can be directly applied to `DataArrays` and `Datasets` [48].

5.3. Parallel and memory-conscious computation with Dask

While vectorization and Xarray improve our program's efficiency for handling large, multidimensional data, further performance gains require parallel and memory-efficient computation. To achieve this, we use Dask, a Python library for scalable computation on a single machine or distributed system.

Dask provides a lazy computation model by wrapping NumPy arrays in a data structure called a *Dask array*. Dask arrays act as a high-level abstraction over NumPy arrays and offer nearly the same interface, with the difference that operations on Dask arrays are executed lazily. Instead of performing computations immediately, Dask stores them as tasks within a task graph to be executed later. Each Dask array corresponds to a node in this task graph, which allows complex workflows to be divided into stages. This design enables Dask to compute only the necessary parts of a workflow, and to discard intermediary steps when they are not required. For instance, if only the final result of a computation is needed, Dask can compute the final array alone, while processing and discarding intermediary steps along the way. If an intermediate result is also needed, Dask efficiently computes both intermediate and final results without recomputing tasks, by reusing intermediate steps to construct the final output. [49, 50]

Another core feature of Dask arrays is *chunking*, which divides a large array into smaller arrays along specified dimensions. For example, a 3D Dask array can consist of multiple smaller 3D chunks. When a computation runs, Dask processes each chunk individually and executes operations across the required chunks to complete the task. The independence of these chunks allows computations to be executed in parallel across multiple workers, each running with a specific number of threads. These workers may run on local CPU cores or be distributed across a network. Additionally, chunking enables *out-of-core computation* to allow large datasets to be processed without loading everything into memory at once. Instead, data is loaded or created “on demand”, and unnecessary data can be offloaded to disk to conserve memory. [49, 50]

This approach significantly improves memory efficiency, avoiding the need to keep all data in memory at once. For instance, when processing a noise simulation for a 15 km by 15 km area with a resolution of one meter, approximately $15\,000^2 = 225\,000\,000$ sound receivers are created. Storing this dataset along with all intermediate results would require around 205 GB of memory. However, with Dask’s out-of-core computation, only the necessary data is loaded and processed at any given moment.

To help monitor and optimize these computations, Dask offers a dashboard that visualizes the task graph, profiles task execution, and provides insights into performance. The dashboard is particularly helpful for ensuring that all workers are utilized adequately, that memory usage remains manageable, and that excessive inter-worker communication is avoided. It can also help verify that the task graph is sufficiently parallelized. Observing the dashboard’s metrics often helps identify further optimization possibilities, such as adjusting chunk sizes. Setting the chunk sizes requires careful consideration, as chunk sizes that are too small create scheduling overhead that slows execution, while chunk sizes that are too large can increase memory usage and limit parallel processing. It is often necessary to experiment with different chunk sizes to identify an optimal size for a given workload. [50]

Dask arrays can serve as a backend for Xarray, enabling lazy operations on Xarray DataArrays and Datasets that are executed by the Dask scheduler [47]. Functions that are not inherently lazy can be adapted to work with Dask by wrapping them with the Xarray functions `apply_ufunc` or `map_blocks`. This approach applies the

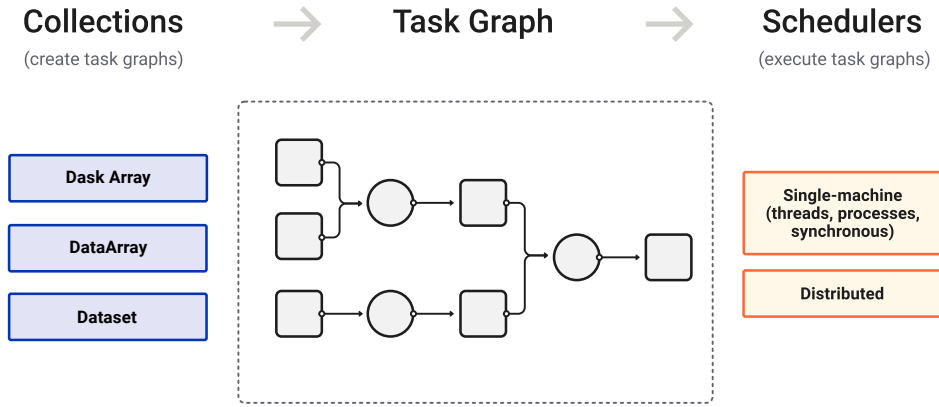


Figure 5: Workflow for computation with Dask. Initially, collections such as plain Dask Arrays or Xarray DataArrays and Datasets are created, which define data structures and computations. Operations on these collections form a task graph that organizes the computation into individual tasks and dependencies. A Dask scheduler then executes this task graph, either on a single machine using multiple threads and processes, or distributed across multiple machines. This architecture enables Dask to efficiently handle large-scale computations by breaking them into manageable tasks. The figure is adapted from [50].

function lazily across all chunks in the DataArray or Dataset, while also ensuring that Xarray’s dimension labels, coordinates, and metadata remain correct and aligned [48]. The Dask computation workflow is outlined in Figure 5.

5.4. Digital elevation model

The ISO 9613-2 sound propagation model involves calculating the distances between sound sources and receiver locations. Each position must therefore be defined not only by 2D coordinates (x, y) but also by a height above ground level and the elevation of the ground itself. While the height above ground is provided as an input to the model along with the x, y coordinates, the ground elevation can be determined using a digital terrain model.

A *Digital Terrain Model (DTM)* represents the Earth’s bare surface by assigning unique height values to discrete points on a 2D grid, approximating natural landforms without including objects like buildings or vegetation. DTMs are distinct from *Digital Surface Models (DSMs)*, which include both natural and man-made features like trees and buildings. Another related term, *Digital Elevation Model (DEM)*, is often used as an umbrella term to describe both DTMs and DSMs. DTMs are commonly generated using ground surveys, aerial imaging, lidar (light detection and ranging), and radar-based methods. Notable global DEM datasets such as SRTM and Copernicus primarily offer DSMs rather than DTMs due to the inclusion of vegetation and built structures [51]. However, the ISO 9613-2 sound propagation model handles forests and buildings separately and therefore requires a bare-earth elevation model [21].

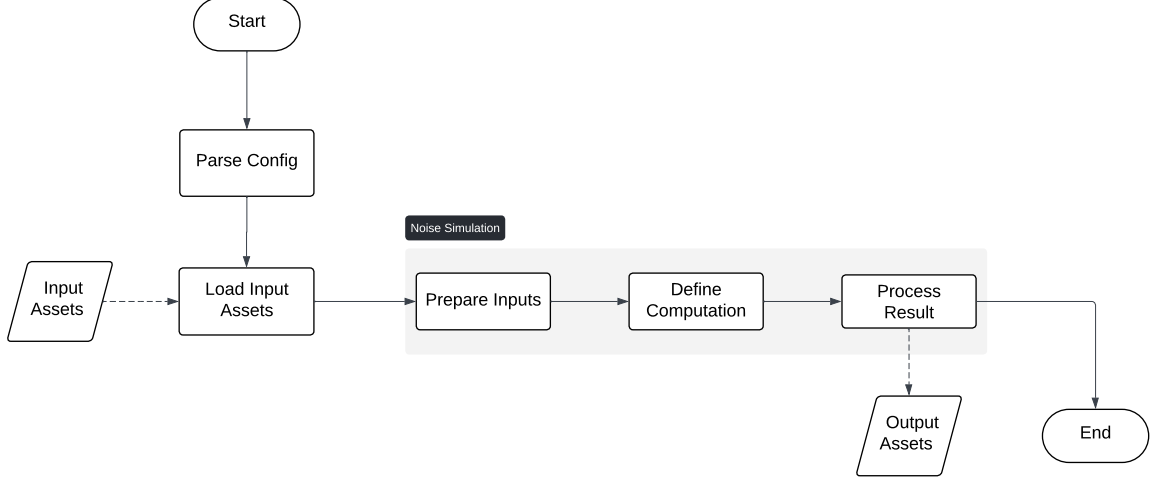


Figure 6: Abstract structure of the simulation software workflow. The process begins by creating a shared configuration, followed by loading the relevant input assets from disk or the network. The core noise propagation module then modifies these inputs adequately, defines the specified computation steps lazily, and finally executes the calculations to generate the requested output assets, such as a noise map.

FABDEM (Forest And Buildings removed Copernicus DEM) [52] builds on the Copernicus COP30 DSM, which is considered one of the best global DSMs currently available, by removing artifacts from buildings and vegetation to approximate a pure DTM. Developed by Laurence Hawker et al. using machine-learning techniques, FABDEM provides a resolution of approximately 30 meters (1 arc-second) and outperforms other global elevation models, such as those based on SRTM [52, 53].

5.5. Overview of the simulation software

Xarray, with Dask as its backend, is combined with other libraries to create an efficient and scalable program capable of predicting wind turbine noise immissions using the ISO 9613-2 sound propagation model. The program consists of five main components, as outlined in Figure 6 and further detailed in the following sections.

5.5.1. Configuration

The simulation software is designed to be stateless. There is no overarching **Simulation** class that manages the simulation; instead, the software is structured around pure functions. Each function operates independently and keeps no information between calls, thereby relying solely on the parameters provided at the time of execution. This stateless approach simplifies testing by isolating functions, and only their inputs and outputs need to be considered, without the risk of hidden state or unintended side

effects. Stateless functions also improve code predictability, as they consistently produce the same output for a given input. Furthermore, since each function call is self-contained, this design enables easier parallelization and concurrent execution. An example for this are the Xarray operations, which are parallelized through Dask.

To manage configuration without passing numerous parameters, an immutable `Config` object provides a global context for functions. Created at the start of the application, the `Config` object derives its state from a Python dictionary containing only basic data types (string, integer, float, boolean, list, or dict). This dictionary can either be loaded from a JSON or TOML file on disk, offering a human-readable configuration file for the simulation, or be received over the network.

The configuration object includes various parameters essential for the simulation, including the chunk size, the types of output assets to generate, the number of processes and threads for the parallel Dask computation, the coordinate reference system for calculations, and the area of interest. While many parameters are fully parsed right away, certain properties, such as the chunk size, coordinate reference system, and area of interest, are only fully determined during the preparation phase, after the input assets have been loaded. For example, if the area of interest is defined as an extent in meters from the bounding box of the wind turbines, it requires knowledge of the turbine locations and the coordinate reference system to be accurately determined.

5.5.2. Loading the input assets

The input data for the simulation is provided as JSON files, which are either fetched from online sources or generated by other applications, such as a front-end interface. Among other things, this data includes precise descriptions of the turbine placements and the receiver locations to be considered. Coordinates are specified in the WGS84 coordinate reference system, using latitude and longitude in degrees.

This data varies in dimensionality. It is often zero-dimensional (e.g., project settings) or one-dimensional (e.g., existing turbines, receivers, scenarios, turbine types), but can also include two-dimensional data (e.g., sound power levels). During the simulation, certain data may extend beyond two dimensions, such as a four-dimensional array representing the ground attenuation per turbine-receiver pair, frequency and sub-period. Representing both input and result data as multidimensional Xarray DataArrays, potentially grouped into Datasets, is therefore a natural fit for this.

To maintain compatibility with other wind turbine simulation modules that might be used together with our program and that use all or some of the same input data, we chose to retain input data in its original JSON format when passed to each simulation module. Each module then decides whether to convert the data into a more suitable structure, such as a NumPy array, Xarray DataArray/Dataset, or a Dataclass. To ensure seamless interoperability and smooth communication across modules, a translation layer between different data representations might be required. This transformation from raw JSON files to a format suitable for calculations is part of a broader *input preparation phase*, during which data may be expanded, filtered, interpolated, or otherwise modified to meet the needs of the core calculation.

Each type of input JSON file is represented by a subclass of `JsonAsset`, which defines each asset by its relative path, human-readable name, and content. To annotate the content with specific keys and value types, the `TypedDict` class from the `typing` module is used. A collection of `JsonAsset` subclasses can then be passed to the `load_assets()` function, which loads the requested assets efficiently and returns a mapping of asset classes (asset types) to class instances (concrete assets). This mapping is read-only and can be shared between modules, reducing memory usage by eliminating the need to reload raw input data multiple times.

5.5.3. Preparing the input

The noise propagation simulation specifies the input assets it requires, which are provided as arguments to its `prepare()` function. This function processes and transforms the assets into a format optimized for the core calculations described in Section 5.5.4.

In this preparation phase, the input assets undergo significant reshaping and stacking to convert them into Dask-backed DataArrays and Datasets. Chunk size, which affects parallel processing and memory efficiency, is configured through the program configuration. By default, it is set such that each thread receives approximately three chunks and is automatically adjusted based on the number of available cores. Alternatively, users can specify a maximum chunk size or a precise size to best suit the computational resources.

Geometric data Geometric inputs, such as building locations, are validated and converted into Shapely classes representing points, line segments, and polygons, which enables detailed geometric calculations. These two-dimensional shapes are then extended to a certain height, making them three-dimensional. This is done using the Trimesh package, along with a triangulation engine and an engine for Boolean operations to merge adjacent buildings as needed. Additionally, debugging options are available to randomize turbine positions or building heights, which enables rapid generation of diverse simulation scenarios.

Coordinate system setup The preparation phase also involves reprojecting the input data from the WGS84 coordinate reference system to a projected system that minimizes distortions in the area of interest and uses meters as units, rather than degrees, for improved accuracy in calculations. A suitable choice are the projections given by the Universal Transverse Mercator (UTM) system, which divides the world into zones and assigns a locally accurate coordinate system to each zone [54]. For instance, Germany is covered by zones 31N in the west, 32N in central areas, and 33N in the east. Our program can automatically select the optimal UTM zone for the area being simulated. Alternatively, the program can construct a custom local Transverse Mercator projection, centered precisely on the area of interest to minimize distortions. While this approach may provide the highest local accuracy, it may limit comparability with data in standardized coordinate systems such as UTM. Finally, users can

also specify an exact coordinate reference system in the configuration to meet specific requirements.

Elevation data After the area of interest and the coordinate reference system have been determined, elevation data from the FABDEM elevation model, described in Section 5.4, is prepared. Missing data is retrieved from a server at the University of Bristol. These data tiles are provided as raster files and use the WGS84 coordinate system, based on longitude and latitude, with each tile covering one degree in both width and height [55]. The tiles required to represent the area of interest are fetched, merged, and subsequently transformed to the local coordinate system used for computation. This merged raster is represented as a chunked `DataArray` with x and y coordinates. During the later creation of the receivers, bilinear interpolation is applied to this `DataArray` at each receiver position to approximate ground elevations.

Receivers Our program supports two types of receivers: *specific receivers* and *grid receivers*. Specific receivers have predefined coordinates provided as input, where sound pressure levels are calculated precisely. In contrast, grid receivers are arranged in a regular grid pattern with constant spacing between them. These grid receivers allow for interpolation to produce a noise map with contour lines. During the preparation phase, the program generates grid receivers based on the specified area of interest and resolution. Since the number of receivers can become quite large at high resolutions, most `Datasets` and `DataArrays` in the simulation are chunked along the **receiver** dimension.

5.5.4. Defining the computation

The prepared inputs are passed to `define_computation()`, where the Dask task graph is constructed, storing the sequence of calculation steps needed to transform input data into results.

To make the noise propagation calculation more manageable, it is divided into discrete computation steps, organized within the `computation` module by sound propagation model. For example, the module `computation.iso9613.basic` contains the original ISO 9613-2 calculation steps, while adjustments specific to the German Interimsverfahren are included in `computation.iso9613.interim`. The module `computation.harmonoise` contains drafts of computation steps for the Harmonoise model [31]. Each step can utilize the results from previous steps but never directly invokes another step. This restriction simplifies the control flow, leaving it to `define_computation()` to assemble these steps in a sensible way that respects dependencies and configuration parameters. For instance, if the Interimsverfahren is disabled, the function calls `computation.iso9613.basic.A_gr()`; if enabled, it calls `computation.iso9613.interim.A_gr()` instead.

The results of the computation steps are stored in a single `simulation` `Dataset`, which starts empty. This structure not only allows for the final result to be calculated but also makes intermediary results like ground attenuation accessible. If this `Dataset`

used eager computation with NumPy arrays instead of Dask arrays, it would consume significant memory even if only the final result was needed. However, with lazy Dask arrays, intermediary results are simply nodes in the task graph and require minimal memory. If an intermediary result in the Dataset has already been computed, the final calculation will reuse this data rather than recompute it.

Finally, `define_computation()` returns the `simulation` Dataset. Since it only stores the operations without executing them, the entire phase of defining the computations completes very quickly.

5.5.5. Processing the result

After the simulation result has been defined as a Dask task graph, the function `process_result()` takes the lazy result and processes it into output assets. If these assets are not based on Dask arrays, this processing involves the partial computation of the result by executing sections of the task graph. There are currently two output assets implemented.

Firstly, the result can be serialized to the NetCDF format. NetCDF (Network Common Data Form) is a self-describing, machine-independent file format that efficiently stores large multi-dimensional scientific data [56]. Its data model comprises labeled dimensions and metadata and was even used as a conceptual basis for Xarray’s development [57, 48]. Xarray objects thus map directly to netCDF files, making it ideal for serialization. Such a byte stream of serialized data could then, for instance, be stored to disk or directly transmitted over a network for further processing.

Secondly, a noise map can be created from the result. A noise map is a visual representation of the spatial distribution of sound pressure levels and can be used to illustrate the impact of noise emitted by wind turbines on the surrounding environment. To generate such a map, the sound pressure levels must have been determined for several receiver positions distributed over the entire area of interest. These values can then be interpolated to also cover the areas between the receivers with sound levels. The closer the distance between these receiver positions, the less interpolation is required and the more accurate the map will be. During the preparation phase of the simulation, such receivers are automatically created to form a grid with adjustable intermediate spacing. The interpolated sound pressure levels are finally resampled into discrete steps with intervals in between. These steps should be chosen according to the simulated scenario and the specific aspects intended to be emphasized; for example, the steps {20 dB(A), 30 dB(A), 40 dB(A), 50 dB(A)} may be a sensible choice for wind turbine noise. The discrete steps are then visualized either as contour lines, showing the steps as colored lines, or as filled contours, which color the intervals between steps.

The noise map is highly customizable via the configuration file. By default, map tiles from Stadia Maps [58] are fetched at a specified zoom level to provide contextual background. These tiles are typically in the Web Mercator projection, which may not align with the locally precise coordinate reference system used in the computation. To prevent potential artifacts or distortions when displaying the map, the tiles are kept in Web Mercator, and the computed results are transformed to match this projection

during rendering. In addition to the contour lines, the wind turbines and specific receiver locations are also displayed. If there are fewer than ten specific receiver locations, each is distinguished in the legend with a unique color; otherwise, they are shown as red dots. The computed A-weighted sound pressure levels can be annotated above both the receiver locations and the contour lines. A color bar on the side aids in differentiating the contours, and a scale bar in the corner provides a sense of distance. Examples of noise maps generated by our program are shown in Figures 8 and 11, which visualize a scenario introduced in Section 6.

The result processing can be easily extended to additional modifications or further processing beyond generating output files and noise maps. However, care should be taken to avoid redundant calculations and ensure that values are computed only once for optimal efficiency.

6. Verification and performance analysis

To verify the accuracy of the simulation software described in Section 5 and to assess its performance relative to a professional software solution, it is benchmarked against WindPRO.

WindPRO, created by EMD International, is a specialized tool for the design and planning of both individual wind turbines and wind farms. The software organizes various functional aspects into distinct modules designed for specific planning needs [59]. For instance, the DECIBEL module enables noise immission calculations based on the ISO 9613-2 model and its variations, such as the Interimsverfahren, and also incorporates noise assessment methods like TA Lärm. Additionally, WindPRO offers the NORD2000 module, which implements the more sophisticated Nord2000 sound propagation model [60]. Given that our simulation uses the ISO 9613-2 propagation model, we consider WindPRO’s DECIBEL module for comparison.

For this benchmark, we simulate a scenario in which four wind turbines are placed in close proximity around Herzogenrath, Germany. The ground elevations at each location are interpolated from the FABDEM elevation model. The turbines have different heights and sound profiles, and are described in more detail in Appendix A.1. Furthermore, fifteen specific receiver locations at which the sound pressure levels should be measured precisely are randomly placed throughout the area. A detailed description of the receivers is given in Table 6. Finally, we enable the generation of a noise map and specify multiple different grid spacings, which leads to the creation of numerous grid receivers. The scenario is mapped out in Figure 7.

Because WindPRO does not implement barrier attenuation, and because of the performance limitations described in Section 5.1 when computing the diffracted sound paths around screening obstacles, barrier attenuation has been disabled for the benchmark simulations described in this section. However, for illustration purposes, the verification scenario has been simulated with barrier attenuation over the top edges of buildings. Outlines of the buildings have been fetched from OpenStreetMap, while their heights have been randomly assigned and greatly exaggerated to emphasize the

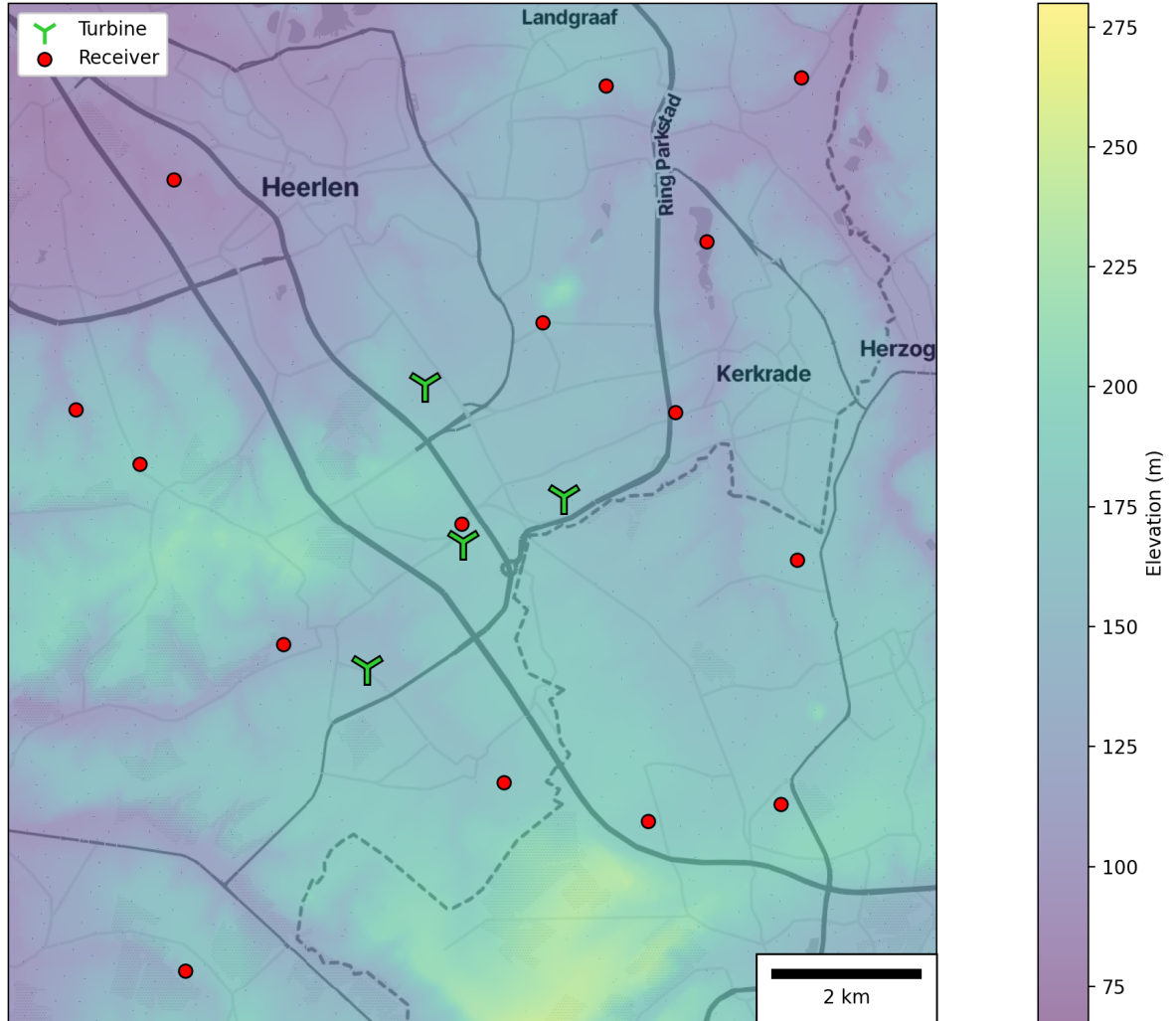


Figure 7: Visualization of the scenario used for verification. The map extends around the bounding box of the simulated wind turbines, placing them in the center. The fifteen randomly selected receiver locations are marked as red dots. Ground elevation is represented by color gradients, based on data from the FABDEM elevation model.

effect of barrier attenuation. The resulting noise map is displayed in Figure 8.

6.1. Runtime comparison with WindPRO

The scenario was simulated using both our software and WindPRO version 4.0.547 on a system with an Intel Core i7-10510U CPU (8 logical cores, base clock 1.8 GHz, turbo boost up to 4.9 GHz), 16 GB DDR4 RAM, and Windows 11 Pro (64-bit). For the 15 specific receiver locations, the runtime was well below one second for both programs and thus considered negligible. However, for grid receivers, the simulation can be executed with varying receiver spacings (resolutions), allowing for a more detailed performance comparison. Receiver grids were computed within an area defined by the bounding box of the wind turbines, expanded by 5000 m on each side, with grid spacings ranging from 1 m to 30 m in increments of 1 m. Furthermore, a local Dask cluster was configured with 2 processes, each with 4 threads. The chunk size was determined automatically as the total number of receivers divided by 24, ensuring that each of the 8 threads received approximately three chunks. To manage resource constraints on the test machine, the chunk size was limited to a maximum of 1 000 000.

The measured runtimes, shown in Figure 9, illustrate that our software consistently outperforms WindPRO, especially at finer grid spacings. While both programs experience increased runtimes with finer grid resolutions, our software demonstrates significantly less runtime growth and offers greater efficiency in both low- and high-resolution scenarios. On average, our software is approximately 60 times faster than WindPRO, performing at least 22 times faster in all cases and achieving up to 108 times faster speeds in high-resolution simulations with large numbers of receivers.

6.2. Evaluating the effectiveness of parallel processing

To evaluate the effectiveness of parallel processing and confirm the utility of Dask for this simulation, the scenario was run with different configurations of process and thread counts on a machine with an AMD EPYC 9634 CPU (12 dedicated cores, up to 3.7 GHz per core), 32 GB DDR5 RAM, and Ubuntu 24.04.1 LTS. As before, the computed area covers the bounding box of the wind turbines extended by 5000 m on each side, with a fixed grid receiver spacing of 2 m. The chunk size was set to *auto* for optimal distribution across available threads. Each measurement represents an average of six runs. The results are shown in Figure 10. The benchmarking validates that parallel processing significantly improves performance, making the calculations nearly 6 times faster by reducing runtime from 71.85 seconds to 12.01 seconds. This substantial improvement justifies the use of Dask to effectively manage the workload.

6.3. Accuracy at specific receiver locations

The equivalent continuous A-weighted sound pressure levels ($L_{p,A,eq}$) were computed at the fifteen specific receiver locations using both our software and WindPRO. The results, shown in Table 2, are very similar, suggesting a high degree of accuracy in our

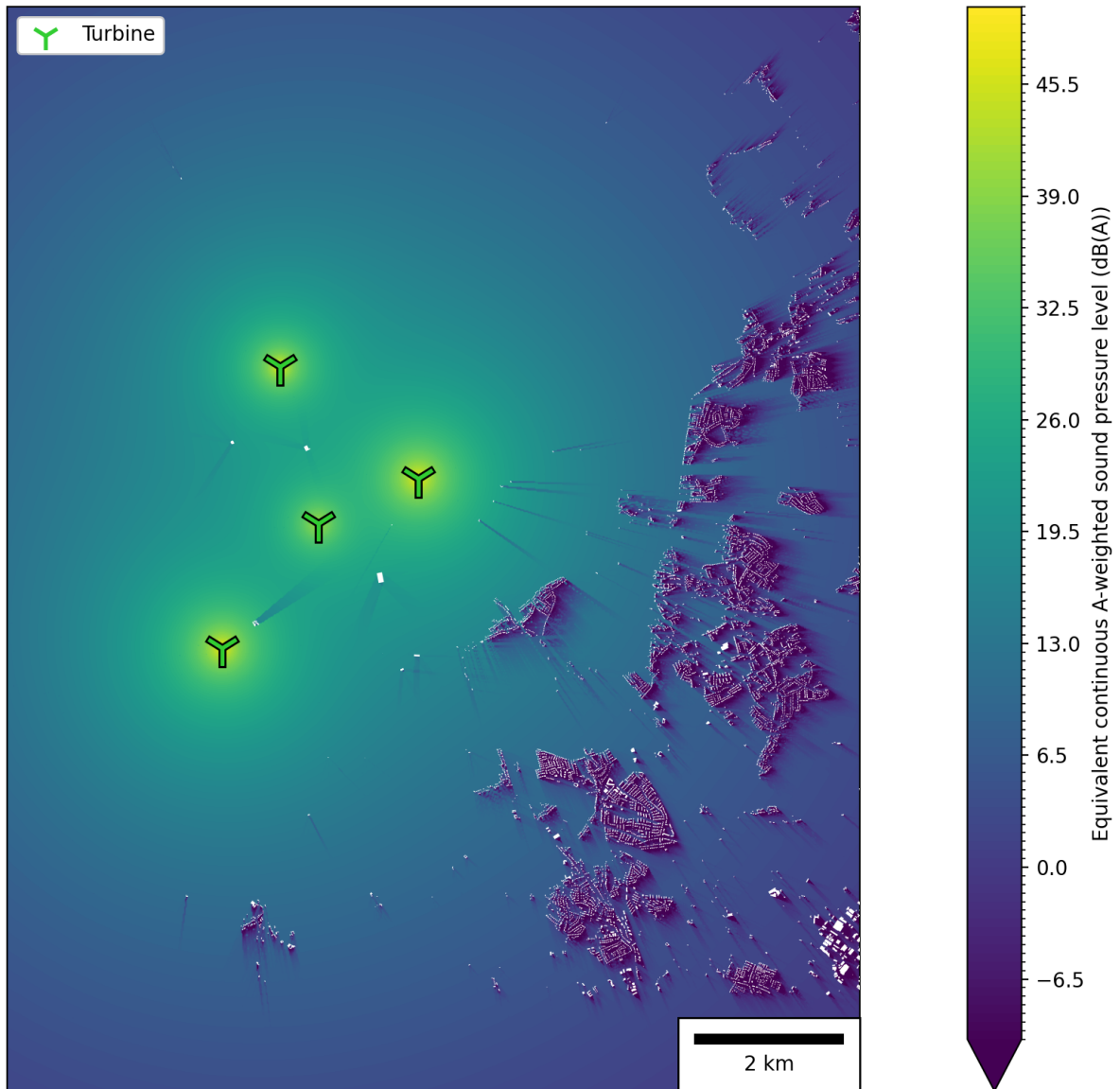
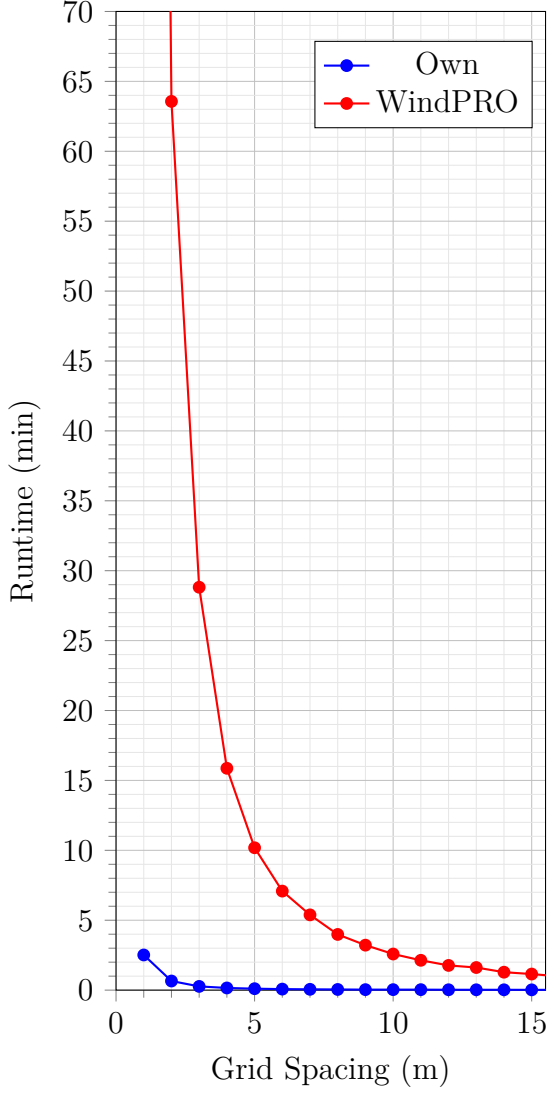
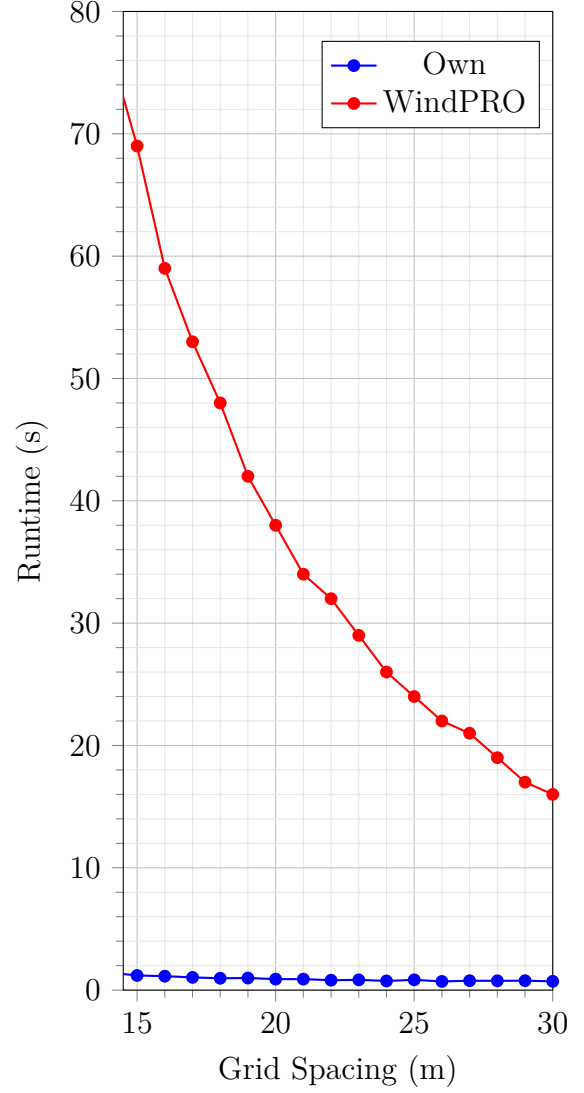


Figure 8: Simulation of the verification scenario demonstrating the effects of barrier attenuation over the top edges of buildings, which are marked in white. The map has been generated with a resolution of 15 m and illustrates the sound pressure resulting from the sound generated by wind turbines in the area. Regions of reduced noise impact are visible, as less wind turbine sound reaches behind the buildings. The building heights have been greatly exaggerated to highlight the attenuation effect, and as such, this map does not represent accurate sound predictions.



(a) Runtimes, in minutes, for grid spacings between 1 m and 15 m.



(b) Runtimes, in seconds, for grid spacings between 15 m and 30 m.

Figure 9: Runtime comparison between our software and WindPRO for noise simulation across various grid receiver spacings. The left plot shows runtimes for high-resolution grid spacings (1 m to 15 m), while the right plot presents runtimes for coarser grid spacings (15 m to 30 m). Each data point represents the average runtime of three simulation runs. The results reveal a steep increase in runtime for WindPRO as grid spacing decreases, while our software maintains a consistently low runtime, with only a slight increase for grid spacings below 4 m. The performance gap between WindPRO and our software becomes especially pronounced at finer grid resolutions, with WindPRO taking approximately 4 h and 10 min for a 1 m grid spacing (exceeding the bounds of this plot), while our software completes the same task in under 3 min.

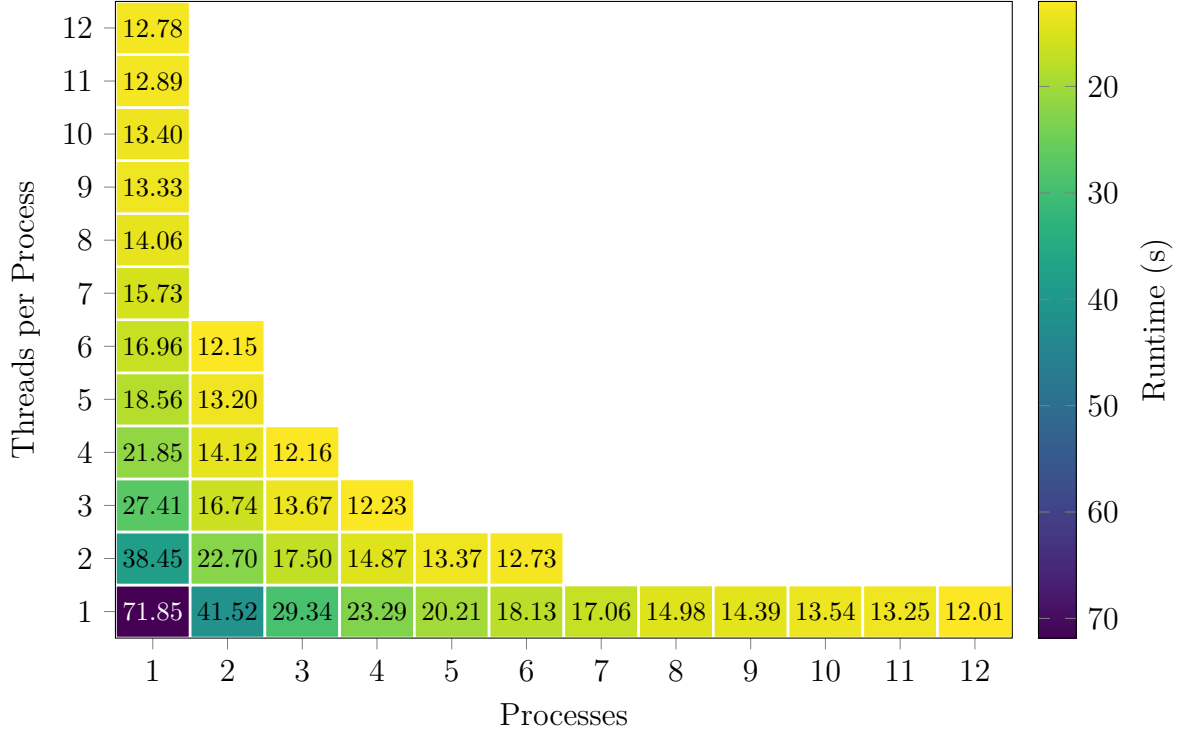


Figure 10: Runtime for different combinations of processes and threads per process using a local Dask cluster. The machine has a CPU with 12 cores, and a fixed grid receiver spacing of 2m was used. Only configurations without a combination directly above or to the right fully utilize all 12 available cores, making them the most efficient in terms of resource use. Other configurations are included for comparison, illustrating the impact of different parallel processing setups on runtime. Each value represents the average runtime across six simulation runs. The plot shows that configurations with fewer processes and threads per process lead to longer runtimes, reaching nearly 72s in the slowest case (1 process with 1 thread, i.e. 1:1). Increasing the number of processes or threads generally reduces runtime, though the reduction per additional thread or process diminishes as parallelism increases. For instance, moving from 1:1 to 1:2 nearly halves the runtime, while the improvement from 1:11 to 1:12 is less significant. Among configurations that fully utilize all cores, the combinations 2:6, 3:4, 4:3, and 12:1 achieve notably short runtimes, with 12:1 being the fastest overall. Interestingly, the 12:1 configuration performs significantly faster than configurations with slightly fewer processes, suggesting possible measurement variability or other influencing factors.

	$L_{p,A,eq}$ Own (dB(A))	$L_{p,A,eq}$ WindPRO (dB(A))	$\Delta L_{p,A,eq}$ (dB(A))	$\Delta L_{p,A,eq}$ (%)
Receiver 1	9.1252	9.1255	-0.0003	-0.0033
Receiver 2	20.3930	20.3930	0	0
Receiver 3	15.5569	15.5569	0	0
Receiver 4	12.6749	12.6752	-0.0003	-0.0024
Receiver 5	37.9949	37.9950	-0.0001	-0.0003
Receiver 6	15.2842	15.2844	-0.0002	-0.0013
Receiver 7	12.0631	12.0632	-0.0001	-0.0008
Receiver 8	23.3634	23.3637	-0.0003	-0.0013
Receiver 9	11.3628	11.3627	+0.0001	+0.0009
Receiver 10	27.4299	27.4299	0	0
Receiver 11	15.4415	15.4418	-0.0003	-0.0019
Receiver 12	12.8193	12.8193	0	0
Receiver 13	11.8359	11.8358	+0.0001	+0.0008
Receiver 14	23.2117	23.2118	-0.0001	-0.0004
Receiver 15	17.0880	17.0884	-0.0004	-0.0023

Table 2: Sound pressure levels at the receiver locations, as computed by our simulation software and by WindPRO, rounded to four decimal places. The two columns $\Delta L_{p,A,eq}$ indicate the absolute and relative deviation of our simulation’s results from those of WindPRO. The differences are minimal, with minimum deviations of -0.0004 dB(A) and -0.0033 %, maximum deviations of $+0.0001$ dB(A) and $+0.0009$ %, and average deviations of approximately $+0.0001$ dB(A) and -0.0008 %. These results confirm the high accuracy of our simulation, as the differences are negligible and well within acceptable limits.

software when compared to WindPRO. The minor deviations observed might stem from differences in the implementation of mathematical operations, floating-point precision limitations, slight distortions from differences between the chosen coordinate reference systems, or other causes.

6.4. Analysis of contour alignment

Contour lines for the A-weighted equivalent sound pressure levels, ranging from 10 dB(A) to 50 dB(A) in 0.25 dB(A) intervals, were computed in both our program and WindPRO using a grid resolution of 3 m and an immission height above ground of 5 m. Since WindPRO does not support the FABDEM elevation model used in our program, and to ensure a reproducible verification that isolates the accuracy of the calculation procedures rather than the input data source, the ground elevation was set to a constant 140 m.

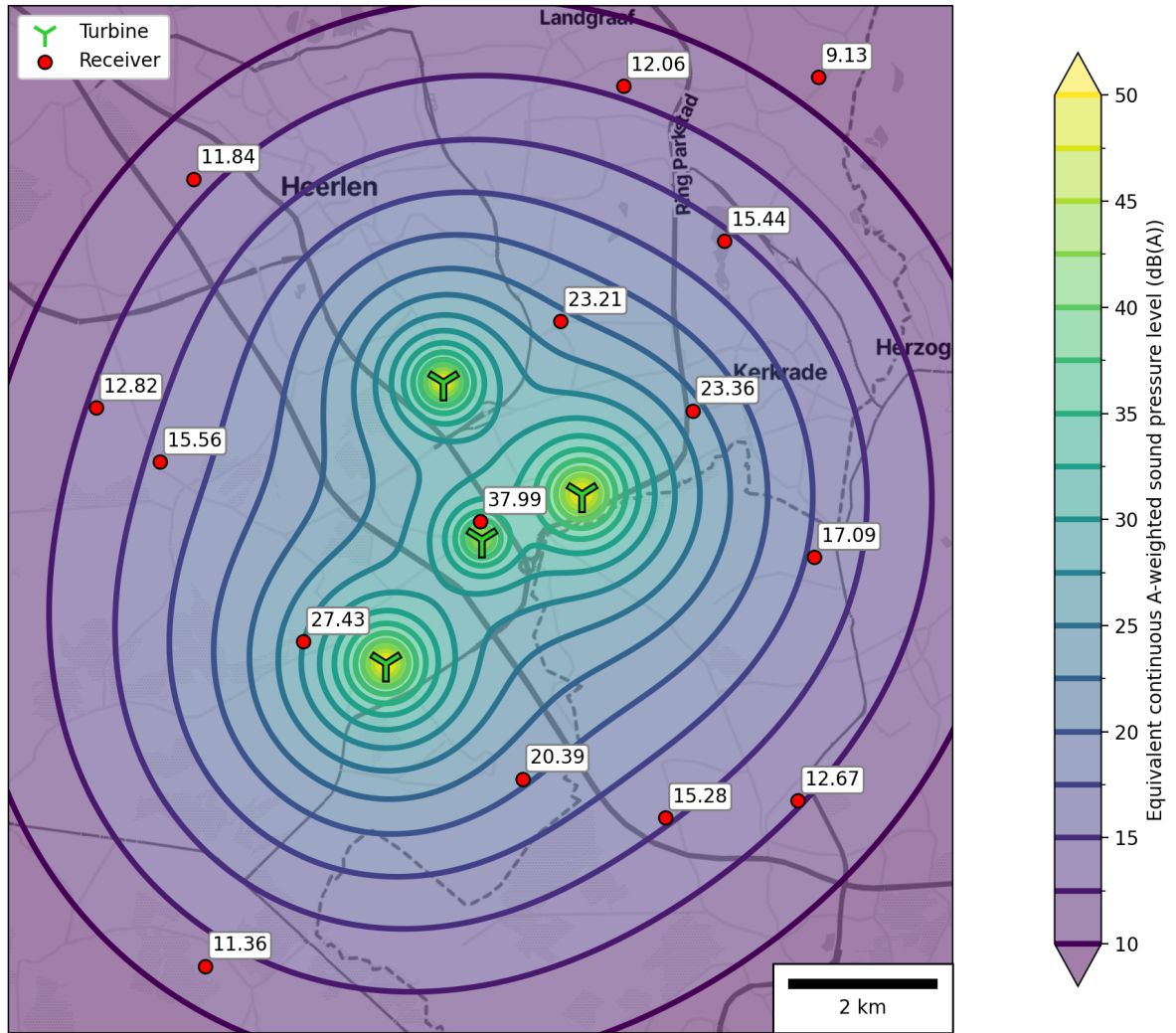


Figure 11: Simulation result for the verification scenario, as generated by our program. The receiver locations are annotated with their predicted A-weighted sound pressure levels, rounded to two decimal places. Contour lines represent noise levels according to the color bar on the right, with labels corresponding to A-weighted sound pressure levels. The intervals between contour lines are shaded with slight transparency to keep the underlying map tiles visible. These contours also make it possible to differentiate between low-noise turbines and those with higher sound power levels. A scale bar in the bottom right corner provides a reference for distances, with all map features transformed to Web Mercator projection for display.

To compare these two sets of contours, a suitable measure must be defined. All contour lines are assumed to be closed, non-intersecting rings. If this is not the case, such as at the boundaries of the computed area, the area can be extended to fully enclose the relevant contours. The contours of a result X are denoted by C_X , and the set of contour rings for a given level $k \in L$ is written as $C_{X,k}$. Each contour $C_{X,k}$ is represented by a finite set of vertices, which are interpolated to ensure even spacing along the rings, avoiding under- or over-representation of certain sections.

For any point p in space, we define the signed distance $d_k(p, X)$ as the shortest distance from p to the contours $C_{X,k}$. This distance is zero if p lies exactly on $C_{X,k}$; it is negative if p is inside the area enclosed by $C_{X,k}$, and positive if p is outside. To compare the contours from a result A to those of result B , the distance $d_k(p, B)$ may be computed for points p lying on $C_{A,k}$. For a fixed level $k \in L$, we define:

$$d_{k,\min}(A, B) = \min\{d_k(p, B) \mid p \in C_{A,k}\}, \quad (14)$$

$$d_{k,\max}(A, B) = \max\{d_k(p, B) \mid p \in C_{A,k}\}, \quad (15)$$

$$d_{k,\text{avg}}(A, B) = \frac{1}{|C_{A,k}|} \sum_{p \in C_{A,k}} d_k(p, B). \quad (16)$$

These metrics describe the deviation of the contour lines at level k in result A relative to result B . For example, if $d_{k,\max}(A, B) < 0$, the contours at level k in A are strictly smaller and lie inside those of B . If $d_{k,\min}(A, B) = d_{k,\max}(A, B) = d_{k,\text{avg}}(A, B) = 0$, the contours for A and B exactly coincide at level k . To assess the overall deviation across all levels $k \in L$, the following values may be calculated:

$$d_{\min}(A, B) = \min\{d_{k,\min}(A, B) \mid k \in L\}, \quad (17)$$

$$d_{\max}(A, B) = \max\{d_{k,\max}(A, B) \mid k \in L\}, \quad (18)$$

$$d_{\text{avg}}(A, B) = \frac{1}{|L|} \sum_{k \in L} d_{k,\text{avg}}(A, B). \quad (19)$$

The presented distance measures have been computed to compare the contour lines generated by our simulation (result A) to those from WindPRO (result B). The results are displayed in Table 3.

Contour level k (dB(A))	$d_{k,\min}$ (m)	$d_{k,\max}$ (m)	$d_{k,\text{avg}}$ (m)
10.0	-0.2070	+1.0582	+0.5989
12.5	-0.1711	+1.0375	+0.6096
15.0	-0.1170	+1.0231	+0.6198
17.5	-0.9884	+1.0102	+0.5751
20.0	-1.0301	+1.0110	+0.4849
22.5	-1.0307	+1.0069	+0.4124
25.0	-1.0281	+1.0028	+0.3704
27.5	-1.0213	+1.0030	+0.3517
30.0	-1.0199	+1.0009	+0.5866
32.5	-1.0155	+1.0048	+0.5809
35.0	-0.0145	+1.0042	+0.6283
37.5	-0.0098	+1.0010	+0.6197
40.0	-0.0125	+1.0049	+0.6208
42.5	-0.0105	+1.0043	+0.6141
45.0	-0.0121	+0.9953	+0.5951
47.5	-0.0177	+0.9962	+0.5798
50.0	-0.1545	+0.9326	+0.4124
Total	-1.0307	+1.0582	+0.5484

Table 3: Minimum, maximum, and average signed distances from our contour lines to those of WindPRO, calculated for various sound pressure levels and rounded to four decimal places. The bottom row shows the values of $d_{\min}$, $d_{\max}$, and d_{avg} , which summarize the overall contour deviations across all levels. Both results were generated with a grid resolution of 3 m. Due to possible differences in the specific grid receiver locations between simulations, an uncertainty of up to 1.5 m is expected. The measurements indicate that our contour lines deviate from WindPRO’s by a maximum of 1.06 m, with an average deviation of approximately 55 cm. All deviations fall within the expected uncertainty range of 1.5 m, meaning that our results align with WindPRO’s results as closely as measurable at this grid resolution.

7. Conclusion and outlook

To summarize this work, this section concludes the contributions of this work and discusses potential areas for future enhancements.

7.1. Impact of our contribution

This work contributes to the public acceptance of wind turbines by presenting an efficient and scalable simulation solution for predicting wind turbine noise. The software is based on the robust ISO 9613-2 model and incorporates modifications for high-altitude sources to comply with German regulatory standards and meet practical accuracy requirements.

To ensure that our program is grounded in precise scientific understanding, this thesis provides a comprehensive overview on wind turbine sound and noise, as well as the challenges in determining sound pressure levels through sound propagation modeling. This foundational knowledge serves as the basis for the development of our program.

The significant capabilities of our program are highlighted through a comparative analysis with the established software WindPRO. Our program's sound immission predictions align closely with those of WindPRO, with an average deviation in predicted sound pressure levels of only 0.0008 %. Similarly, the spatial sound distribution is as accurate as could be measured, showing an average distance difference between contour lines of equal sound pressure of merely 55 cm.

A factor contributing to accuracy is the use of state-of-the-art, high-quality elevation models. We employ elevation data that has been thoroughly adjusted to make it more suitable for sound propagation modeling by removing buildings and vegetation from the original dataset. This refinement ensures that the terrain data accurately represents ground elevation, which is critical for precise sound propagation calculations. Additionally, our program intelligently and automatically fetches the necessary data based on the selected area, which simplifies the setup process and reduces the potential for user error.

The program has been carefully optimized for efficient handling of geospatial aspects and seamlessly integrates three different coordinate reference systems: WGS84 for input coordinates (longitude and latitude), a computation projection that is automatically chosen – either UTM for compatibility or a custom projection positioned to optimize for local accuracy and minimal distortions – and Web Mercator for mapping purposes. This careful coordinate system management ensures compatibility with various data sources and mapping services, facilitating interoperability.

Furthermore, we have developed an algorithm to determine diffracted sound paths used in ISO 9613-2 barrier attenuation calculations. Unlike WindPRO, which ignores this aspect, our program can account for barrier effects, potentially providing more accurate results in scenarios where diffraction plays a significant role. While the precise influence of buildings on wind turbine noise requires further investigation, this capability already gives our program an advantage over other noise simulation solutions.

Most importantly, the program demonstrates significant computational efficiency.

It consistently outperforms WindPRO, running at least 22 times faster and up to 108 times faster in high-resolution simulations, thereby saving over four hours of processing time when simulating at a resolution of 1 meter. By employing advanced vectorization techniques with Xarray and Dask for parallel processing and efficient memory management, we made the program almost 6 times faster than single-threaded execution, all while maintaining acceptable memory consumption levels. This performance is achieved without compromising accuracy, which makes our program both powerful and practical for large-scale simulations.

Together with its flexible and modular architecture that utilizes advanced multi-dimensional data structures, these enhancements make the program broadly applicable and allow for integration with other simulation tools, such that users can simulate multiple placement scenarios simultaneously or explore different operational modes. With its ability to generate both precise point predictions and detailed noise maps, the program serves as a valuable tool for wind energy planners to optimize turbine placement and helps to promote public acceptance of wind energy projects by effectively communicating potential noise impacts.

7.2. Future directions

For the simulation software presented in this work, we have mainly considered the ISO 9613-2 propagation model. We have also established groundwork for including obstacles in the computation. However, propagation factors such as reflection and miscellaneous attenuation effects (e.g., due to vegetation) are not yet addressed. The proposed algorithm for determining the diffracted sound paths has certain edge cases where it fails to identify one or both lateral paths and needs to be refined. Currently, the algorithm also cannot be fully vectorized due to the complexity of 3D intersections and 2D spatial analysis, making Python a performance bottleneck. This could potentially be improved by using Numba [61], which is a Python package that offers just-in-time compilation and allows some functions written in native Python and NumPy to run at near-C speeds. While Numba integrates well with Xarray and Dask, certain packages like Trimesh and Shapely are not compatible, which may require further work.

An updated version of the ISO 9613-2 model was recently released [62]. Although the core calculations remain largely the same, the update includes clarifications and additional details, such as a section specifically addressing wind turbine noise and a more detailed discussion of wind effects on sound propagation. However, German regulations still require the previous ISO 9613-2 version for noise assessments, and German adjustments are based on that version. The updated version has therefore not been considered in this work, but may be incorporated in the future. Given that the current Interimsverfahren adjustments are only a temporary solution, it may be advisable to wait for official guidance from German regulatory bodies if a new propagation model is to be mandated.

Looking forward, future work could explore more advanced sound propagation models, such as Harmonoise and Nord2000, in more detail. These models may yield greater

accuracy in modeling sound propagation paths and thus improve immission predictions, provided that the input data is of high quality and accuracy. They could be adapted to the software architecture presented in this work to enable vectorization and parallel processing, though not all operations may be vectorizable, which can potentially result in high runtimes. Numba may again provide a way to counter these performance challenges.

Besides propagation, the sound emission may also be modeled in a more differentiated way. In this work, we adhered to the ISO 9613-2 framework and modeled each wind turbine as a point source at hub height, radiating uniformly in all directions. The source characteristics are represented through a frequency-dependent sound power spectrum, which may vary across time periods and operational modes. Further research is needed to assess whether a point source is the most accurate approximation for a wind turbine and if sound power levels could be predicted more precisely. For instance, Reference [63] indicates that wind turbines emit sound at varying intensities based on current wind profiles. This variability is already considered in WindPRO but has yet to be incorporated into our model.

A. Appendix

A.1. Verification setup details

For the verification of the simulation software developed in the course of this work, an exemplary scenario has been created. The scenario involves four wind turbines, with positional details given in Table 4 and sound power levels given in Table 5. Additionally, fifteen specific receiver locations have been randomly placed in the area as described in Table 6. Section 6 presents the verification results.

	Longitude (°)	Latitude (°)	Elevation (m)	Hub height (m)
Turbine 1	5.999 119	50.864 383	149.0	80.0
Turbine 2	5.988 147	50.830 616	153.8	100.0
Turbine 3	6.025 418	50.850 970	162.3	120.0
Turbine 4	6.006 380	50.845 577	176.5	140.0

Table 4: The wind turbines used in the verification scenario. Longitude and latitude were chosen arbitrarily. The hub heights have been selected to resemble typical wind turbine heights. The elevation was interpolated from the FABDEM elevation model.

	$L_{W,f}$ (dB)								$L_{W,A}$ (dB(A))
	63 Hz	125 Hz	250 Hz	500 Hz	1000 Hz	2000 Hz	4000 Hz	8000 Hz	
Turbine 1	100	102	98	94	91	85	80	74	96.34
Turbine 2	103	105	100	97	94	88	83	76	99.15
Turbine 3	105	106	101	98	95	90	85	79	100.31
Turbine 4	98	100	96	93	90	86	82	75	95.46

Table 5: Sound power levels for each wind turbine and octave band. The levels have been chosen to resemble typical wind turbine noise levels. The last column provides the total A-weighted sound power level, rounded to two decimal places, as an informative summary of sound power. It is computed as $L_{W,A} = 10 \log_{10}(\sum_f 10^{0.1(L_{W,f} + A_f)})$, where A_f is the A-weighting for frequency f .

	Longitude (°)	Latitude (°)	Elevation (m)	Height (m)
Receiver 1	6.070 367	50.901 019	107.0	5.0
Receiver 2	6.014 140	50.816 737	171.7	5.0
Receiver 3	5.945 218	50.854 842	181.9	5.0
Receiver 4	6.066 531	50.814 123	179.7	5.0
Receiver 5	6.006 097	50.847 702	175.5	5.0
Receiver 6	6.041 348	50.812 101	181.2	5.0
Receiver 7	6.033 375	50.899 988	142.1	5.0
Receiver 8	6.046 492	50.860 972	136.0	5.0
Receiver 9	5.953 777	50.794 227	156.0	5.0
Receiver 10	5.972 423	50.833 247	132.4	5.0
Receiver 11	6.052 496	50.881 367	105.5	5.0
Receiver 12	5.933 062	50.861 341	134.8	5.0
Receiver 13	5.951 603	50.888 804	84.9	5.0
Receiver 14	6.021 413	50.871 726	154.9	5.0
Receiver 15	6.069 639	50.843 394	166.9	5.0

Table 6: The receiver locations considered in the verification scenario. Their longitudes and latitudes have been selected randomly, while the elevation data is interpolated from the FABDEM elevation model. The immission heights in meters above the ground level have been set to 5 m for all receivers.

References

- [1] Fernando D. Bianchi, Ricardo J. Mantz, and Hernán De Battista. *Wind Turbine Control Systems*. Springer London, 2007. ISBN: 9781846284939. DOI: 10.1007/1-84628-493-7.
- [2] Emre Barlas et al. “Consistent modelling of wind turbine noise propagation from source to receiver”. In: *The Journal of the Acoustical Society of America* 142.5 (Nov. 2017), pp. 3297–3310. ISSN: 0001-4966. DOI: 10.1121/1.5012747.
- [3] Na’ama Teschner and Rachelle Alterman. “Preparing the ground: Regulatory challenges in siting small-scale wind turbines in urban areas”. In: *Renewable and Sustainable Energy Reviews* 81 (2018), pp. 1660–1668. ISSN: 1364-0321. DOI: <https://doi.org/10.1016/j.rser.2017.05.256>.
- [4] Hartmut Ising and Barbara Kruppa. “Health Effects caused by Noise: Evidence in the Literature from the Past 25 Years”. In: *Noise and Health* 6.22 (2004), pp. 5–13.
- [5] Igho J. Onakpoya et al. “The effect of wind turbine noise on sleep and quality of life: A systematic review and meta-analysis of observational studies”. In: *Environment International* 82 (2015), pp. 1–9. ISSN: 0160-4120. DOI: <https://doi.org/10.1016/j.envint.2015.04.014>.
- [6] *Sechste Allgemeine Verwaltungsvorschrift zum Bundes-Immissionsschutzgesetz (Technische Anleitung zum Schutz gegen Lärm - TA Lärm)*. Technical Guideline GMBI Nr. 26/1998, p. 503. Revised Jun. 2017. Bundesministerium für Umwelt, Naturschutz und Reaktorsicherheit, Aug. 1998.
- [7] Matthew Cand, Andrew Bullmore, and Timothy Van Renterghem. “Wind Turbine Noise Propagation”. In: *Handbook of Wind Energy Aerodynamics*. Springer International Publishing, 2021, pp. 1–30. ISBN: 9783030054557. DOI: 10.1007/978-3-030-05455-7_71-1.
- [8] *Hinweise zum Schallimmissionsschutz bei Windkraftanlagen (WKA)*. Guideline. Revised draft from March 17, 2016, with amendments from June 23, 2016. Länderausschuss für Immissionsschutz (LAI), June 2016.
- [9] Lorenz Karl Häfele. *Berechnung und Bewertung der Schallausbreitung von Windenergieanlagen mit Hilfe von Messdaten*. Studienarbeit. Stuttgart, Germany: Universität Stuttgart, Stuttgarter Lehrstuhl für Windenergie (SWE) am Institut für Flugzeugbau, Apr. 2023.
- [10] Lennart Bouma. “Atmospheric sound propagation modeling with the Harmonoise model applied to wind turbine noise”. MA thesis. University of Twente, 2023.
- [11] Andrea P.C. Bresciani. “Physical and perceptual prediction of wind turbine noise”. PhD thesis. University of Twente, 2024. ISBN: 9789036559546. DOI: 10.3990/1.9789036559546.

- [12] Daniela Ushizima, Matthew McCormick, and Dilworth Parkinson. “Accelerating Microstructural Analytics with Dask for Volumetric X-ray Images”. In: *2020 IEEE/ACM 9th Workshop on Python for High-Performance and Scientific Computing (PyHPC)*. 2020, pp. 41–48. DOI: 10.1109/PyHPC51966.2020.00010.
- [13] Robert T. Sataloff and Joseph Sataloff. “The Physics of Sound”. In: *Hearing Loss*. 4th Edition. Boca Raton: CRC Press, 2005. Chap. 2, pp. 3–18. ISBN: 9780429134494. DOI: 10.1201/b21609.
- [14] Enda Murphy and Eoin A King. *Environmental Noise Pollution: Noise Mapping, Public Health, and Policy*. Elsevier, 2014. ISBN: 9780124115958. DOI: 10.1016/c2012-0-13587-0.
- [15] *Quantities and units – Part 3: Space and time*. Standard ISO 80000-3. First edition. Geneva, Switzerland, Mar. 2006.
- [16] *Quantities and units – Part 8: Acoustics*. Standard ISO 80000-8. First edition. Geneva, Switzerland, Aug. 2007.
- [17] Gerhard Müller and Michael Möser. *Handbook of Engineering Acoustics*. Springer Berlin Heidelberg, 2013. ISBN: 9783540694601. DOI: 10.1007/978-3-540-69460-1.
- [18] Hugo Fastl. “Psycho-Acoustics and Sound Quality”. In: *Communication Acoustics*. Springer-Verlag, 2005, pp. 139–162. ISBN: 354022162X. DOI: 10.1007/3-540-27437-5_6.
- [19] S. S. Stevens. “The Measurement of Loudness”. In: *The Journal of the Acoustical Society of America* 27.5 (Sept. 1955), pp. 815–829. ISSN: 1520-8524. DOI: 10.1121/1.1908048. URL: <http://dx.doi.org/10.1121/1.1908048>.
- [20] *Electroacoustics – Sound level meters – Part 1: Specifications*. Standard IEC 61672-1. Edition 2.0. Geneva, Switzerland: International Electrotechnical Commission, Sept. 2013.
- [21] *Acoustics — Attenuation of sound during propagation outdoors — Part 2: General method of calculation*. Standard ISO 9613-2. First edition. Geneva, Switzerland: International Organization for Standardization, Dec. 1996.
- [22] Bertrand Lihoreau et al. “Outdoor sound propagation modeling in realistic environments: Application of coupled parabolic and atmospheric models”. In: *The Journal of the Acoustical Society of America* 120.1 (July 2006), pp. 110–119. ISSN: 0001-4966. DOI: 10.1121/1.2204455.
- [23] Keith Attenborough and Timothy Van Renterghem. *Predicting Outdoor Sound*. 2nd Edition. Boca Raton: CRC Press, 2021. ISBN: 9780429470806. DOI: 10.1201/9780429470806.
- [24] Monika Agatz. *Ein Rechtsrahmen für den Windenergieausbau*. Windenergie-Handbuch. Germany: Self-published, Aug. 2020.

- [25] Jørgen Kragh et al. *Nordic Environmental Noise Prediction Methods, Nord2000 Summary Report: General Nordic Sound Propagation Model and Applications in Source-Related Prediction Methods*. Preliminary Report AV 1719/01. Revised May 2002. Lyngby, Denmark: DELTA Danish Electronics, Light & Acoustics, Dec. 2001.
- [26] Birger Plovsing and Jørgen Kragh. *Nord2000. Comprehensive outdoor sound propagation model. Part 1: Propagation in an atmosphere without significant refraction*. Technical Report AV 1849/00. Revised Dec. 2001. Lyngby, Denmark: DELTA Danish Electronics, Light & Acoustics, Dec. 2000.
- [27] Birger Plovsing and Jørgen Kragh. *Nord2000. Comprehensive outdoor sound propagation model. Part 2: Propagation in an atmosphere with refraction*. Technical Report AV 1851/00. Revised Dec. 2001 and Mar. 2006. Hørsholm, Denmark: DELTA Danish Electronics, Light & Acoustics, Dec. 2000.
- [28] Birger Plovsing. *Proposal for Nordtest Method: Nord2000 – Prediction of Outdoor Sound Propagation*. Technical Report AV 1106/07. Revised Jan. 2014. Lyngby, Denmark: DELTA Danish Electronics, Light & Acoustics, Mar. 2007.
- [29] B. Søndergaard and B. Plovsing. *Validation of the Nord2000 Propagation Model for Use on Wind Turbine Noise*. Technical Report AV 1238/09. Revised Oct. 2009. Hørsholm, Denmark: DELTA, Sept. 2009.
- [30] Dirk Van Maercke and Jérôme Defrance. “Development of an analytical model for outdoor sound propagation within the Harmonoise project”. In: *Acta Acustica united with Acustica* 93.2 (2007), pp. 201–212.
- [31] Erik Salomons et al. “The Harmonoise Sound Propagation Model”. In: *Acta Acustica united with Acustica* 97.1 (Jan. 2011), pp. 62–74. ISSN: 1610-1928. DOI: 10.3813/aaa.918387.
- [32] Dirk van Maercke and Erik Salomons. “The Harmonoise sound propagation model: further developments and comparison with other models”. In: *Proceedings NAG-DAGA conference, Rotterdam, The Netherlands*. 2009.
- [33] Fabio Strigari, Michael Chudalla, and Wolfram Bartolomaeus. “Weather-corrected immission levels — Comparison of sound propagation models”. In: *Proceedings of the DAGA 2018*. Deutsche Gesellschaft für Akustik (DEGA). Munich, Germany, 2018.
- [34] Stylianos Kephalopoulos et al. “Advances in the development of common noise assessment methods in Europe: The CNOSSOS-EU framework for strategic environmental noise mapping”. In: *Science of The Total Environment* 482–483 (June 2014), pp. 400–410. ISSN: 0048-9697. DOI: 10.1016/j.scitotenv.2014.02.031.
- [35] *Schallprognosen für Windenergieanlagen nach dem „Interimsverfahren“*. Factsheet. Ministerium für Umwelt, Landwirtschaft, Natur- und Verbraucherschutz des Landes Nordrhein-Westfalen (MULNV NRW), Nov. 2017.

- [36] Detlef Piorr. “Erste Erfahrungen mit den LAI-Hinweisen zum Schallimmissionsschutz bei Windkraftanlagen”. In: *Proceedings of the DAGA 2018*. Deutsche Gesellschaft für Akustik (DEGA). Munich, Germany, 2018.
- [37] *Dokumentation zur Schallausbreitung – Interimsverfahren zur Prognose der Geräuschimmissionen von Windkraftanlagen*. Technical Report. Berlin, Germany: Normenausschuss Akustik, Lärminderung und Schwingungstechnik (NALS), May 2015.
- [38] *Acoustics — Software for the calculation of sound outdoors — Part 3: Recommendations for quality assured implementation of ISO 9613-2 in software according to ISO 17534-1*. Technical Report ISO/TR 17534-3. First edition. Geneva, Switzerland: International Organization for Standardization, Jan. 2015.
- [39] Fajie Li and Reinhard Klette. *Euclidean Shortest Paths*. Springer London, 2011. ISBN: 9781447122562. DOI: 10.1007/978-1-4471-2256-2.
- [40] Michael Dawson-Haggerty et al. *Trimesh*. Version 4.4.7. 2024. URL: <https://github.com/mikedh/trimesh>.
- [41] Sean Gillies et al. *Shapely*. Version 2.0.6. 2024. DOI: 10.5281/zenodo.5597138. URL: <https://github.com/shapely/shapely>.
- [42] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.
- [43] VDE VERLAG GmbH. *Interimsverfahren zu Schallimmissionen: Schutz der Anwohner nicht immer erhöht*. Accessed: 2024-09-17. 2018. URL: <https://www.energie.de/sonne-wind-waerme/news-detailansicht/nsctrl/detail/News/interimsverfahren-zu-schallimmissionen-schutz-der-anwohner-nicht-immer-erhoeht>.
- [44] Stefan van der Walt, S. Chris Colbert, and Gael Varoquaux. “The NumPy Array: A Structure for Efficient Numerical Computation”. In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30. DOI: 10.1109/MCSE.2011.37.
- [45] Stefan Behnel et al. “Cython: The Best of Both Worlds”. In: *Computing in Science & Engineering* 13.2 (2011), pp. 31–39. DOI: 10.1109/MCSE.2010.118.
- [46] NumPy Developers. *NumPy reference: CPU/SIMD optimizations*. Accessed: 2024-11-01. URL: <https://numpy.org/devdocs/reference/simd/index.html>.
- [47] S. Hoyer and J. Hamman. “xarray: N-D labeled arrays and datasets in Python”. In: *Journal of Open Research Software* 5.1 (2017). DOI: 10.5334/jors.148.
- [48] xarray Developers. *Xarray documentation*. Accessed: 2024-10-04. URL: <https://docs.xarray.dev>.
- [49] Matthew Rocklin. “Dask: Parallel Computation with Blocked algorithms and Task Scheduling”. In: *Proceedings of the 14th Python in Science Conference*. SciPy. SciPy, 2015, pp. 126–132. DOI: 10.25080/majora-7b98e3ed-013. URL: <http://dx.doi.org/10.25080/majora-7b98e3ed-013>.

- [50] Dask Developers. *Dask Documentation*. Accessed: 2024-10-12. URL: <https://docs.dask.org>.
- [51] Christian Hirt. “Digital Terrain Models”. In: *Encyclopedia of Geodesy*. Springer International Publishing, 2014, pp. 1–6. ISBN: 9783319023700. DOI: 10.1007/978-3-319-02370-0_31-1.
- [52] Laurence Hawker et al. “A 30 m global map of elevation with forests and buildings removed”. In: *Environmental Research Letters* 17.2 (2022), p. 024016. DOI: 10.1088/1748-9326/ac4d4f.
- [53] Christopher B Marsh, Phillip Harder, and John W Pomeroy. “Validation of FAB-DEM, a global bare-earth elevation model, against UAV-lidar derived elevation in a complex forested mountain catchment”. In: *Environmental Research Communications* 5.3 (Mar. 2023), p. 031009. DOI: 10.1088/2515-7620/acc56d.
- [54] Richard B Langley. “The UTM grid system”. In: *GPS world* 9.2 (1998), pp. 46–50.
- [55] Jeffrey Neal and Laurence Hawker. *FABDEM V1-2*. 2023. DOI: 10.5523/bris.s5hqmjcdj8yo2ibzi9b4ew3sn.
- [56] NSF Unidata. *Network Common Data Form (NetCDF)*. Accessed: 2024-10-04. 2024. URL: <https://www.unidata.ucar.edu/software/netcdf>.
- [57] NSF Unidata. *The NetCDF Data Model*. Accessed: 2024-10-04. 2023. URL: https://docs.unidata.ucar.edu/netcdf-c/current/netcdf_data_model.html.
- [58] Stadia Maps. *Stadia Maps: Map Tiles and Geolocation API*. Accessed: 2024-11-04. 2024. URL: <https://stadiamaps.com>.
- [59] Tom Acker and Arshiya Hoseyni Chime. *Wind Modeling using WindPro and WAsP Software*. Tech. rep. Flagstaff, AZ, USA: Sustainable Energy Solution Lab, Mechanical Engineering Department, Northern Arizona University, July 2011.
- [60] EMD International A/S. *windPRO 4.0 User Manual: Environmental Calculations*. Chapter 6: Noise and Reflections Calculations. EMD International A/S. Denmark, Sept. 2023.
- [61] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. “Numba: A llvm-based python jit compiler”. In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. 2015, pp. 1–6.
- [62] *Acoustics – Attenuation of sound during propagation outdoors – Part 2: Engineering method for the prediction of sound pressure levels outdoors*. ISO 9613-2. Geneva, Switzerland: International Organization for Standardization, Jan. 2024.
- [63] *V80-2.0 MW: Versatile Megawattage*. Available at: <https://www.vestas.com>. Vestas Wind Systems A/S. Randers, Denmark, 2024.