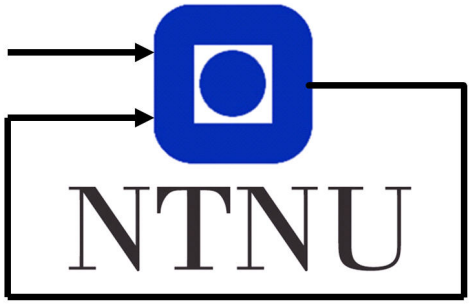
Approximating linear filters to simulate stochastic aerodynamic models

Tormod Gjerde Nes

May 2022



Department of Engineering Cybernetics

Contents

Abstract

Current methods of realizing stochastic turbulence models rely on the inverse Fourier transform. This makes it impossible to generate turbulence online. As an alternative this project proposes a method which realizes the Liepmann[8] turbulence model through discrete filters. To do this the target correlation function array is decomposed using the Cholesky decomposition and equated to a set of basic filter correlation functions to approximate it. The resulting filter array can then be excited using independent sources of Gaussian white noise to generate the target auto-correlation and cross-correlation.

1 Introduction

In any industrial engineering application, accurate simulation is essential. This is especially the case for structures under constant unpredictable load like wind turbines. This project will focus on the aerodynamic part of this simulation, in other words the unpredictable turbulence. It is essential that this simulated turbulence is accurate enough such that the combined simulation is indicative of physical implementation.
Known methods for generating turbulence rely on the frozen turbulence hypothesis[13]. This allows any simulation to be represented as generating a turbulence box, which is then projected onto the simulated sensors with the mean wind speed. Previous methods rely on taking the inverse Fourier transform of the spectral tensor[10], which was proposed as a slight improvement to the power spectral density approach used by TurbSim[16]. This limits the turbulence box to finite length, as the inverse Fourier transform can not create infinite samples from a finite sampling of its source spectra[9].
This project proposes a method which generates signals using digitally filtered white noise. This allows the turbulence box to be generated online with infinite length, this can shorten the necessary memory drastically. Instead of having to allocate space for entire turbulence boxes that can reach incredibly high file sizes. In our proposed scheme we can limit these files to the filter itself and the continuous turbulence boxes stretching back only as far as the filter order requires.
The precursor to this project[11] managed to generate turbulence in some cases. It was often limited by instability caused by the system identification scheme which was outsourced to a MatLab function [2].
In this project, we refine the approach by designing a dedicated method of approximating linear discrete systems to target correlation functions. We will first calculate the correlation functions of common linear filters in section 3. These functions will then be equated to the target correlation functions found from the Cholesky decomposition for correlated noise 4.This allows us to then extract the necessary parameters in the original filters to approximate the target correlation function 5. This filter can then be excited using Gaussian white noise to generate an array of signals with accurate autocorrelation and cross-correlation 6.
The main theoretical contributions of this project can be summarized as:

- Noted correlation functions for common discrete filters
- A method for realizing a Cholesky decomposition for correlated noise
- A method of approximating correlation functions to linear systems

2 Theory

2.1 Stochastic turbulence models

Turbulence is often mentioned as one of the primary examples of chaotic behaviour. Most of our physical models can predict the future with a good amount of certainty. This is not the case for general turbulence, especially wind turbulence. The established method of modelling turbulent behaviour is solving the Navier-Stokes equations [12]. This method requires extreme resolution, and subsequently, extreme computational power to simulate.
Consider now a single point placed in the turbulent wind that we measure. The signal that results from this measurement has statistical traits which are then noted. Thus instead of modelling the physical interaction as wind shears and forms vortices, we generate a signal which replicates the statistical traits we expect. Several stochastic models have been made in an attempt to most

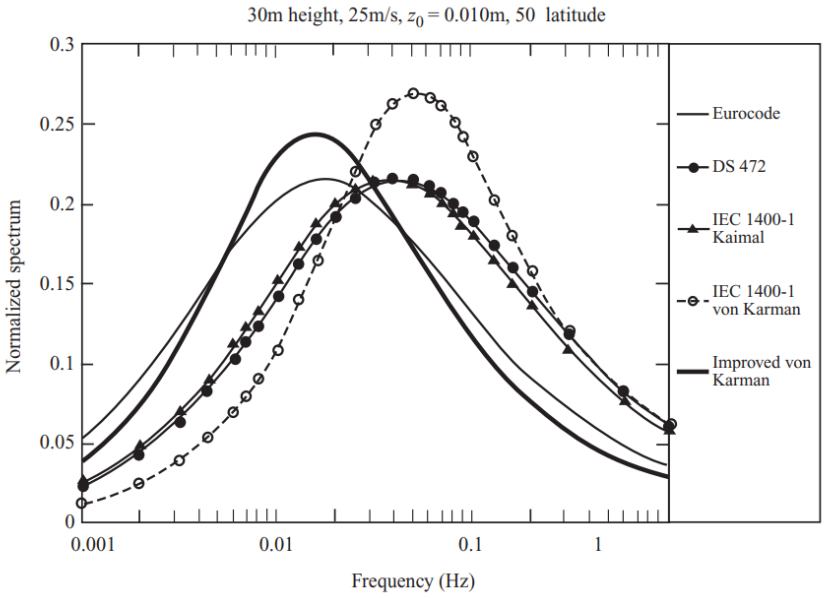


Figure 1: Common stochastic models for intense wind

accurately mimic real turbulence [14]. As can be seen in figure 1 they are mostly similar, but do have slight differences. We will only be working with one stochasitc model in this project, not shown above. Because our method loses some accuracy, the minor differences between stochastic models would also be lost should we try to distinguish between them.

2.1.1 Current methods of realization

The existing methods for realizing these models rely on the inverse Fourier-transform to generate fixed horizon time-series. The specifics of this operation have been noted in a previous related project[11] and shall not be repeated here. However, for continued consistency, some comparison to the main method from the previous project, TurbSim[16], will be made.

2.2 Signal generation

The objective of any simulation boils down to the resulting signal, whether that be the heat of a house or the speed of a car. In our case we need a method for generating a signal for every simulated point in turbulent wind.

2.2.1 Gaussian white noise

White noise is the source for all turbulence simulation in this project. It is defined as zero-mean and temporally uncorrelated[15]. This means that any two samples, regardless of how close they are, will be uncorrelated and randomly distributed with unit variance. In mathematical terms for white noise time series x.

$$E[x(n)x(n-\tau)] = 0 \quad \forall \quad \tau \neq 0 \tag{1}$$

A white noise discrete signal will have equal power at every frequency, this trait allows us to shape the signal to the desired correlation using the internal correlation function of filters alone.

2.2.2 Correlation functions

The wind cannot change infinitely fast. As such any two samples of said wind will be correlated proportionally to the time between them. Thus any time series measured by a node placed in turbulence can be represented by a correlation function[8]. A correlation function is defined as

$$R(\tau) = E[x(n)x(n-\tau)] \tag{2}$$

Which can readily be compared to (1). The center term for any correlation function, i.e when $\tau = 0$, is the variance of the respective signal. The correlation function for unit white noise would therefore be a single point of 1 at $\tau = 0$ and zero for every other value. This is where correlated noise differs. Since two samples are no longer completely uncorrelated the off center values of the correlation function will be non-zero. Naturally, this correlation will decay with time. Any delayed correlation can never exceed the center term, as that term represents the maximum power, or variance for that signal[4]. For the purposes of this project, the correlation will scale with distance. This

distance is linked to time as the mean wind speed of said turbulence will propagate it parallel to its direction. In addition to this temporal direction we will also have spatial correlation between nodes in a plane. This makes it such that every node will have its internal correlation function, often dubbed auto-correlation, and every node will affect every other node scaling with distance, dubbed cross-correlation.

2.2.3 Difference equations

From previous section we now have our source gaussian white noise, and the correlation that we wish to shape it to. To shape it we will use the method of discrete filtering. This method takes a source input, u, and calculates an output, x, based on said input. Let us take a look at a simple example

$$x_n = 0.5u_n + 0.5u_{n-1} \tag{3}$$

In this case the filter output yields a finite impulse response (FIR) which takes the mean of the last two inputs. For this project we want the output to be generated recursively in real time. As such it is important that our filters are all causal[5]. This implies that the output is generated from present and past input only, as relaying on future input would be impossible for an online approach.
The easy example of a FIR filter is insufficient in our case, as the necessary sample horizon to cover our desired correlation function would make it computationally unrealistic. That is where the existence of infinite impulse response (IIR) filters prove useful.

$$x_n = 0.5u_n + 0.5x_{n-1} \tag{4}$$

In this case the output is a function of the present input and the previous output. Thus any output sample will correlate proportionally to the sample difference to past samples. This is exactly the trait we noted present in turbulence in general. The equations above are considered difference equations, and they each yield defined impulse responses.

2.3 The Z-transform

To represent digital filters in more compact form we introduce the Z-transform. The Z-transform can be computed from impulse responses, $x(n)$, not to be confused with the difference equations in last section.

$$X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n} \tag{5}$$

This definition is not used in this project, as we will refer to the following table whenever transformation between impulse response and Z-transform

	Impulse Response	z-transform
1	$p^n u(n)$	$\frac{1}{1-pz^{-1}}$
2	$(p^n \cos(\omega n))u(n)$	$\frac{1-pz^{-1}\cos(\omega)}{1-2pz^{-1}\cos(\omega)+p^2z^{-1}}$
3	$(p^n \sin(\omega n))u(n)$	$\frac{pz^{-1}\sin(\omega)}{1-2pz^{-1}\cos(\omega)+p^2z^{-1}}$

is necessary. Only the relevant impulse responses are noted as opposed to more common, more extensive Z-transform tables [6].

The variable p is what is referred to as the pole in all cases. The pole represents the roots of the denominator and is the the most essential variable in a filter for fitting correlation functions. The output can then be generated as

$$x(z) = H(z)u(z) \tag{6}$$

As this operation is the main source of computation in this project it should be noted that the actual calculation involved is a difference equation. Let us look at the example

$$H(z) = \frac{K}{1 - c_1 z^{-1} + c_2 z^{-2}} \tag{7}$$

Which by inserting into (6) and cross multiplying gives

$$\begin{aligned} x(z)(1 - c_1 z^{-1} + c_2 z^{-2}) &= Ku(z) \\ x(z) &= Ku(z) + x(z)[c_1 z^{-1} - c_2 z^{-2}] \end{aligned} \tag{8}$$

Which we can then transform using the simplified rule that z^{-k} is a sample delay by k [6].

$$x_n = Ku_n + c_1 x_{n-1} - c_2 x_{n-2} \tag{9}$$

We see that the resulting difference equation generates an output using two previous outputs. The order of a filter is the same as the largest sample delay in the difference equation [5]. This is always equal to the largest ordered term in the denominator of the Z-transform. There are two aspects to the computational cost of a difference equation, one is the amount of terms computed which scales with both the numerator and the denominator order. But the largest impact quickly comes from the memory required for a machine to recursively simulate the filter, the memory is only dependent on the order in the denominator. In other words, how many previous outputs does the filter need to remember to generate the current output. Keeping this order low is therefore paramount to efficient simulation.

While the analysis in this project is mainly based on correlation functions, it should be noted that the power spectra of a filter is found as:

$$S(z) = |H(z)H(z^{-1})| \tag{10}$$

2.3.1 Convolution

In future sections we will extend the table to include the respective correlation functions. This operation relies on taking the impulse response $x(n)$ and convolving[4] it with its own delayed impulse response $x(n-\tau)$. This operation can be noted as the following

$$R(\tau) = \sum_{n=-\infty}^{\infty} x(n)x(n-\tau) \tag{11}$$

2.4 Sum of geometric series

A geometric series is a series where every subsequent term is equal to the previous term multiplied by a common ratio p[7].

$$a + ap + ap^2 + ap^3 \dots ap^n = \sum_{n=0}^{\infty} ap^n \tag{12}$$

In discrete signal processing, the sum of this series often comes up. The sum is only defined in the converging case when $|p| < 1$. To see why this is the case we derive the expression for the closed form sum, S.

$$\begin{aligned} S &= a + ap + ap^2 + ap^3 \dots ap^n \\ pS &= ap + ap^2 + ap^3 + ap^4 \dots ap^n + ap^{n+1} \\ S - pS &= a - ap^{n+1} \end{aligned} \tag{13}$$

In line three we assign values to the sums and algebraically add them. In our case these sums come from infinite series and they therefore must converge to set values for this line to be mathematically proper. If this is the case we can solve for our sum as:

$$\begin{aligned} S(1-p) &= a - ap^{n+1} \\ S &= \frac{a - ap^{n+1}}{1-p} \end{aligned} \tag{14}$$

Since our series are infinite we let n go to infinity and find.

$$\sum_{n=0}^{\infty} ap^n = \frac{a}{1-p} \quad \forall \quad |p| < 1 \tag{15}$$

We can then extend this equation by letting the common ratio be complex and we find

$$\begin{aligned} p_c^n &= p^n e^{in\omega} \\ p_c^n &= p^n \left(\cos(n\omega) + i\sin(n\omega)\right) \end{aligned} \tag{16}$$

Where we have utilized Euler's formula for complex numbers[3]. By insertion we then find the very relevant equations.

$$\begin{aligned}\sum_{n=0}^{\infty} p^n \cos(\omega n) &= \Re\left(\frac{1}{1-p_c}\right) \\ \sum_{n=0}^{\infty} p^n \sin(\omega n) &= \Im\left(\frac{1}{1-p_c}\right)\end{aligned} \tag{17}$$

We then need only find the real and imaginary components respectively of the RHS term.

$$\begin{aligned}\frac{1}{1-p_c} &= \frac{1}{1-pe^{-i\omega}} \\ \frac{1}{1-p(\cos(\omega)-i\sin(\omega))} &= \frac{1}{1-p\cos(\omega)-ip\sin(\omega)}\end{aligned} \tag{18}$$

Following

$$\begin{aligned}\frac{1}{1-p(\cos(\omega)-i\sin(\omega))} &= \frac{1}{1-p\cos(\omega)-ip\sin(\omega)}\frac{1-p\cos(\omega)+ip\sin(w)}{1-p\cos(\omega)+ip\sin(\omega)} \\ \frac{1-p\cos(\omega)+ip\sin(\omega)}{1+(p\cos(\omega))^2+(p\sin(\omega))^2-2p\cos(\omega)} &= \frac{1-p\cos(\omega)+ip\sin(\omega)}{1-2p\cos(\omega)+p^2}\end{aligned} \tag{19}$$

Where the complex terms in the final denominator cancel. We can then use this result to find the final result for the complex case.

$$\begin{aligned}\sum_{n=0}^{\infty} p^n \cos(\omega n) &= \frac{1-p\cos(\omega)}{1-2p\cos(\omega)+p^2} \\ \sum_{n=0}^{\infty} p^n \sin(\omega n) &= \frac{p\sin(\omega)}{1-2p\cos(\omega)+p^2}\end{aligned} \tag{20}$$

The constant term a is associative in any case, and can therefore be moved outside the sum term and the results above will be multiplied by the constant.

2.5 The Cholesky Decomposition

Given a real and positive definite matrix $\mathbf{A}$, the Cholesky decomposition yields the following:

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T \tag{21}$$

$\mathbf{L}$ is then a lower triangular matrix[1]. The necessary requirement that $\mathbf{A}$ is positive definite will always hold in context of this project, as the elements in $\mathbf{A}$ will be signal variance and covariance.
The Cholesky decomposition is useful for generating covariant noise. Given

that $\mathbf{A}$ is made up of N by N elements of desired covariance, then this can be simulated as

$$x = \mathbf{L}u \tag{22}$$

Where u is an N by 1 uncorrelated white noise signal source. This is the general approach when every signal is temporally uncorrelated, in a later section we explore how the cholesky decomposition can be altered to function for correlated noise.

3 Filter correlation functions

Any correlation function must be represented by a sum of exponentials since the goal is to minimize memory. In this section we will deduce the exponential correlation functions of common discrete filters. These are our baseline tools in fitting various target correlation function, and as such must be varied enough to cover some base traits, while also not exceeding our limitation on filter order.

3.1 Base correlation functions

3.1.1 1st order filter

At first let us consider the simplest and computationally cheapest filter.

$$H_1(z) = \frac{K}{1 - pz^{-1}} \tag{23}$$

The impulse response can be found from its inverse Z-transform.

$$h(n) = \mathcal{Z}^{-1}[H(z)] = Kp^n \tag{24}$$

We can then use convolution to find the correlation function resulting from such a filter.

$$R(k) = \sum_{n=0}^{\infty} h(n)h(n+k) \tag{25}$$

$$R(k) = K^2 \sum_{n=0}^{\infty} p^n p^{n+k} \tag{26}$$

$$R(k) = K^2 p^k \sum_{n=0}^{\infty} p^{2n} \tag{27}$$

Where using the theorem for geometric sums gives us

$$R(k) = \frac{K^2 p^{|k|}}{1 - p^2} \tag{28}$$

And the final $|k|$ results from the correlation function being necessarily symmetric. The absolute value operator will be omitted in future results to yield easier to read equations. This filter gives us a simple peak which decays exponentially.

3.1.2 Cosine filter

For the case of a rotating turbulence box, it is necessary to represent oscillation, proportional to the angular velocity w we wish to model. Consider the following filter:

$$H(z) = K\frac{1 - p\cos(\omega)z^{-1}}{1 - 2p\cos(\omega)z^{-1} + p^2z^{-2}} \tag{29}$$

Which has the following impulse response.

$$h(n) = Kp^n\cos(\omega n) \tag{30}$$

Thus the correlation function is found from

$$R(k) = \sum_{n=0}^{\infty} Kp^n\cos(\omega n)Kp^{n+k}\cos(\omega(n+k)) \tag{31}$$

$$R(k) = K^2p^k\sum_{n=0}^{\infty} p^{2n}\cos(\omega n)[\cos(\omega k)\cos(\omega n) - \sin(\omega k)\sin(\omega n)] \tag{32}$$

Where we have applied the following trigonometric identity:

$$\cos(\omega(n+k)) = \cos(\omega k)\cos(\omega n) - \sin(\omega k)\sin(\omega n) \tag{33}$$

$$R(k) = K^2p^k\sum_{n=0}^{\infty} p^{2n}[\cos(\omega k)\cos^2(\omega n) - \sin(wk)\sin(\omega n)\cos(\omega n)] \tag{34}$$

Now applying two further trigonometric identities.

$$\cos^2(\omega n) = \frac{1 + \cos(2\omega n)}{2} \tag{35}$$

$$\sin(\omega n)\cos(\omega n) = \frac{\sin(2\omega n)}{2} \tag{36}$$

$$R(k) = \frac{K^2p^k}{2}\left[\cos(\omega k)\sum_{n=0}^{\infty}[p^{2n}(1 + \cos(2\omega n))] - \sin(\omega k)\sum_{n=0}^{\infty}[p^{2n}\sin(2\omega n)]\right] \tag{37}$$

And we find the correlation function as

$$R(k) = \frac{K^2p^k}{2}\left[\cos(\omega k)A - \sin(\omega k)B\right] \tag{38}$$

Where

$$A = \frac{1 - p^2\cos(2\omega)}{1 - 2p^2\cos(2\omega) + p^4} + \frac{1}{1 - p^2} \tag{39}$$

$$B = \frac{p^2\sin(2\omega)}{1 - 2p^2\cos(2\omega) + p^4} \tag{40}$$

Where A and B come from the geometric sum for complex numbers (20).

3.1.3 Sine filter

Consider the following filter:

$$H(z) = K\frac{p\sin(\omega)z^{-1}}{1 - 2p\cos(\omega)z^{-1} + p^2z^{-2}} \tag{41}$$

Which gives the following impulse response

$$h(n) = Kp^n\sin(\omega n) \tag{42}$$

Through largely the same arguments as for the cosine filter, we find the following correlation function.

$$R(k) = \frac{K^2p^k}{2}\left[\cos(\omega k)A + \sin(\omega k)B\right] \tag{43}$$

$$A = \frac{1}{1-p^2} - \frac{1 - p^2\cos(2\omega)}{1 - 2p^2\cos(2\omega) + p^4} \tag{44}$$

$$B = \frac{p^2\sin(2\omega)}{1 - 2p^2\cos(2\omega) + p^4} \tag{45}$$

In both cases we can simplify further by noting that the oscillating term is a linear combination of harmonic waveforms. Thus we can represent the correlation as.

$$R(k) = \frac{K^2p^k}{2}\left[G\cos(\omega k + \phi)\right] \tag{46}$$

$$\begin{aligned} G &= sgn(A)\sqrt{A^2 + B^2} \\ \phi &= \tan^{-1}\left(-\frac{B}{A}\right) \end{aligned} \tag{47}$$

This goes for the cosine case as well of course. This step is helpful when we wish to analyze where the resulting extremities would fall.

3.2 Filter combinations

If a signal is generated from two independent sources of correlated noise, then the resulting correlation function is a direct sum. In our case the signal is generated from a single source and as such the filter combinations will affect one another, dubbed crossterms, this limiting factor is explored further in this subsection.

3.2.1 Second order filter

This filter does not include every kind of second order filter, but rather the one which can be represented by two exponential functions added together, as such our filter can be represented as:

$$H_2(z) = \frac{K_1}{1 - p_1 z^{-1}} + \frac{K_2}{1 - p_2 z^{-}1} \tag{48}$$

Thus the impulse response of this filter will be the sum of two exponentials.

$$h(n) = K_1 p_1^n + K_2 p_2^n \tag{49}$$

And convuluting this impulse response yields

$$R(k) = \sum_{n=0}^{\infty} p_1^k [K_1^2 p_1^{2n} + K_1 K_2 (p_1 p_2)^n] + p_2^k [K_2^2 p_2^{2n} + K_1 K_2 (p_1 p_2)^n] \tag{50}$$

And applying the geometric sum theorem gives us.

$$R(k) = p_1^{|k|} \left(\frac{K_1^2}{1 - p_1^2} + \frac{K_1 K_2}{1 - p_1 p_2} \right) + p_2^{|k|} \left(\frac{K_2^2}{1 - p_2^2} + \frac{K_1 K_2}{1 - p_1 p_2} \right) \tag{51}$$

Thus the new form has two exponentials, but the gain associated with each is dependent on the other, this becomes important as we wish to keep our filters strictly real.

3.2.2 Cosine and first order

Consider the following filter:

$$H(z) = K_o \frac{1 - p_1 \cos(\omega) z^{-1}}{1 - 2 p_1 \cos(\omega) z^{-1} + p_1^2 z^{-2}} + K_d \frac{1}{1 - p_2 z^{-1}} \tag{52}$$

Which gives the following impulse response

$$h(n) = K_o p_1^n \cos(\omega n) + K_d p_2^n \tag{53}$$

And we can find the autocorrelation as:

$$R(k) = \sum_{n=0}^{\infty} (K_o p_1^n \cos(\omega n) + K_d p_2^n)(K_o p_1^{n+k} \cos(\omega (n + k)) + K_d p_2^{n+k}) \tag{54}$$

Which for simplicity we decompose into four seperate terms where:

$$R_1(k) = \frac{K_o^2 p_1^k}{2} \left[\cos(\omega k) A - \sin(\omega k) B \right] \tag{55}$$

$$R_4(k) = \frac{K_d^2 p_2^k}{1 - p_2^2} \tag{56}$$

and

$$R_2(k) = \sum_{n=0}^{\infty} K_o p_1^n \cos(\omega n) K_d p_2^{n+k} \tag{57}$$

$$R_3(k) = \sum_{n=0}^{\infty} K_d p_2^n K_o p_1^{n+k} \cos(\omega(n+k)) \tag{58}$$

R_2 is a first order component found through inspection with the pole p_2.

$$R_2(k) = K_o K_d p_2^k C \tag{59}$$

R_3 can be found through similar arguments as R_1 with the pole p_1.

$$R_3(k) = K_o K_d p_1^k \left[\cos(\omega k) C - \sin(\omega k) D\right] \tag{60}$$

$$C = \frac{1 - p_1 p_2 \cos(w)}{1 - 2p_1 p_2 \cos(\omega) + (p_1 p_2)^2} \tag{61}$$

$$D = \frac{p_1 p_2 \sin(\omega)}{1 - 2p_1 p_2 \cos(\omega) + (p_1 p_2)^2} \tag{62}$$

3.2.3 Sine and first order

$$H(z) = K_o \frac{p_1 \sin(\omega) z^{-1}}{1 - 2p_1 \cos(\omega) z^{-1} + p_1^2 z^{-2}} + K_d \frac{1}{1 - p_2 z^{-1}} \tag{63}$$

Where R_1 and R_4 is the same as found in last section.

$$R_2(k) = K_o K_d p_2^k D \tag{64}$$

$$R_3(k) = K_o K_d p_1^k [\cos(\omega k) D + \sin(\omega k) C] \tag{65}$$

3.2.4 Time delayed filter combinations

Consider now a filter with the impulse response

$$h(n) = h_1(n) + h_2(n - \tau) \tag{66}$$

Where τ symbolizes a delay of the impulse response. The correlation function of this impulse response can then be found as

$$R(k) = R_1(k) + R_2(k) + \sum_{n=0}^{\infty} h_1(n) h_2(n - \tau + k) + h_1(n + k) h_2(n - \tau) \tag{67}$$

Where R_1 and R_2 are the correlation functions of impulse response h_1 and h_2 respectively. The remaining cross terms will be equal to the cross terms found in the last section, with some alterations. The cross term that represents the delayed impulse response, in other words its respective pole, will

be reflected around the exterior of the correlation function. The non-delayed cross impulse response will be reflected within the interior. We define the interior as the part of the function that is within the delay on both sides, and the exterior as the outside. This concept is shown clearly in figure 7. This description is done to avoid lengthy repetitive calculations, the simplified method has been used to calculate the comparison shown in figure 8.

3.3 Function plots

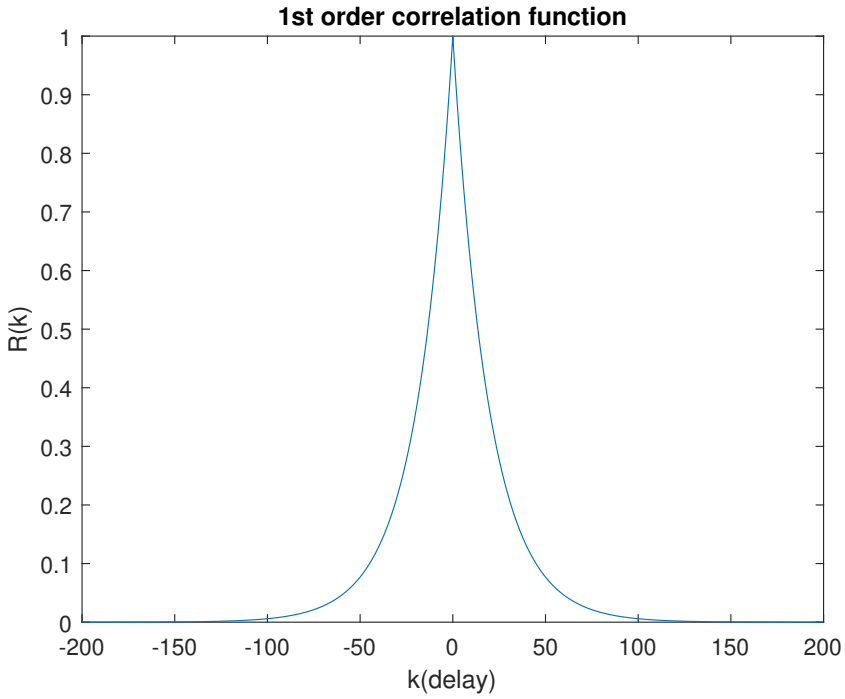


Figure 2: Function form for first order discrete filter

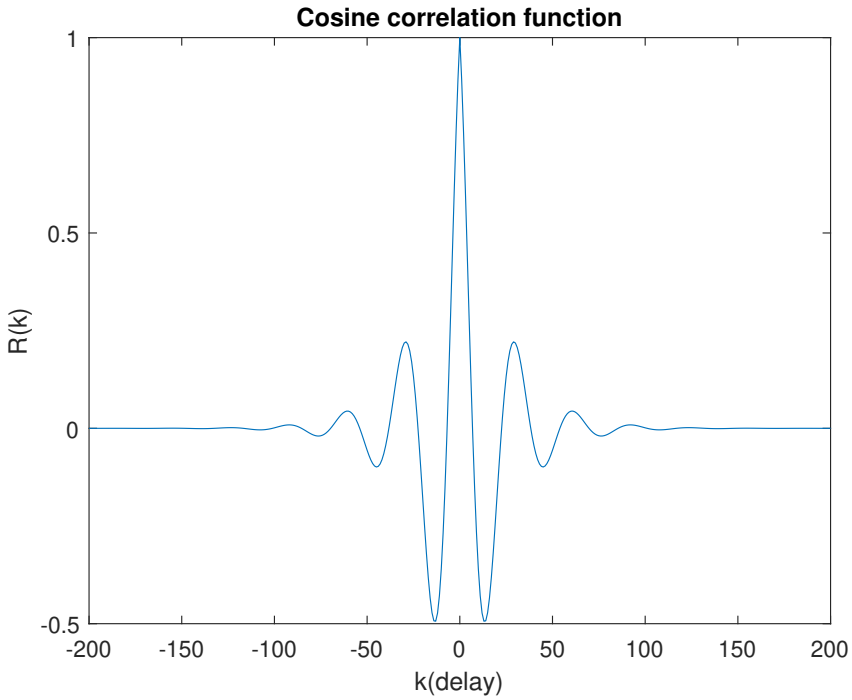


Figure 3: Function form for filter with cosine impulse response

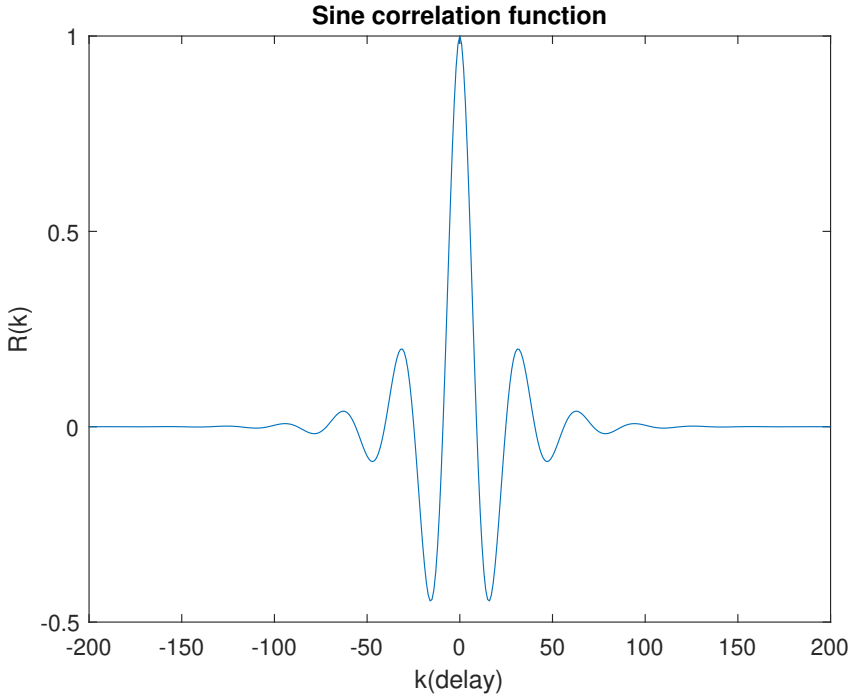


Figure 4: Function form for filter with sine impulse responser

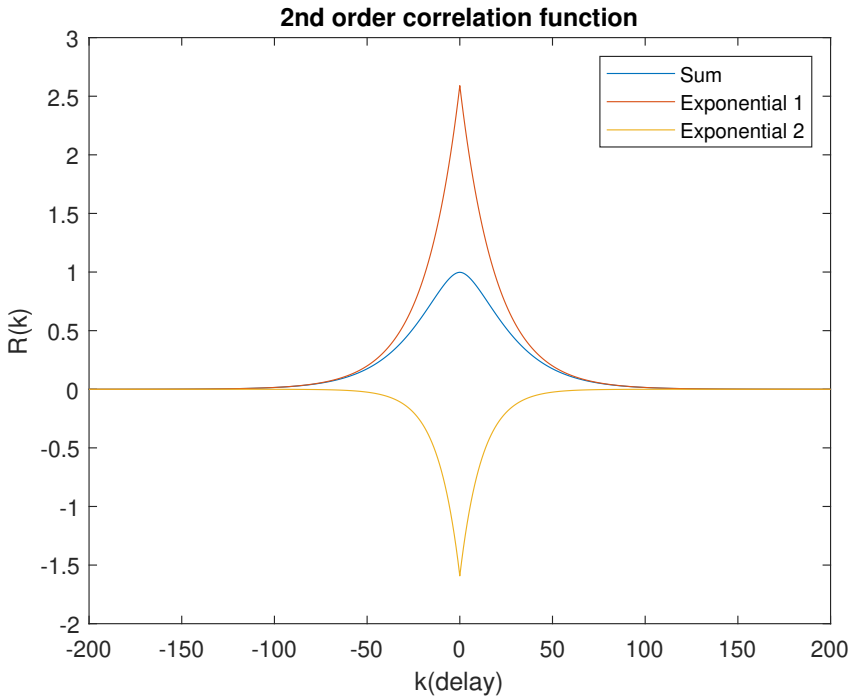


Figure 5: Function form for second order discrete filter

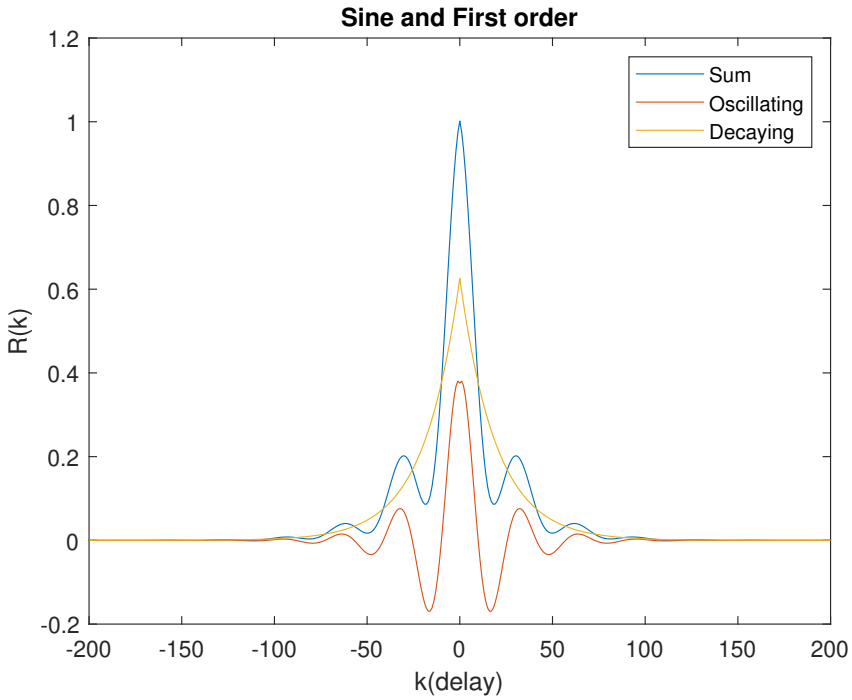


Figure 6: Function form for combined first order and cosine impulse response

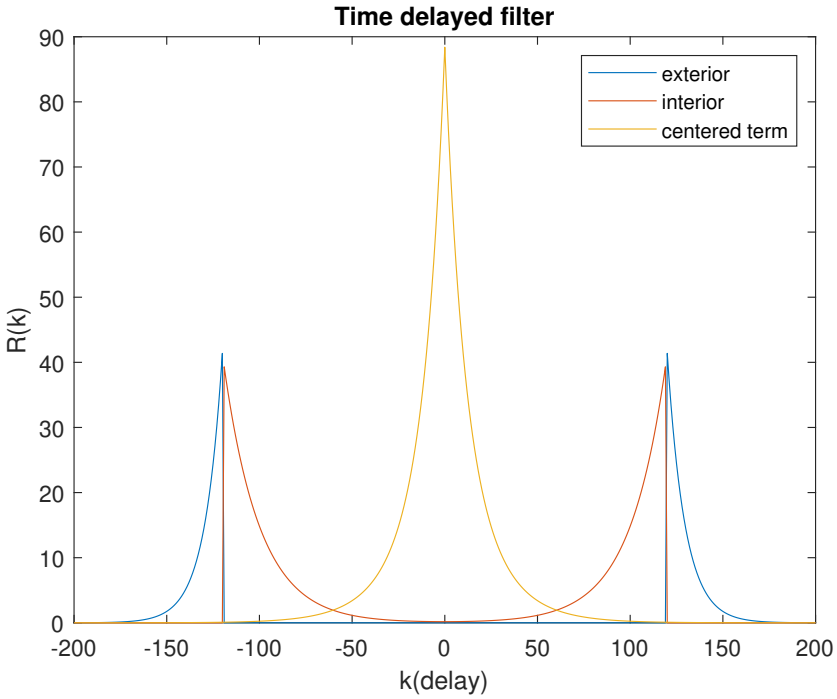


Figure 7: Showing what is defined as interior and exterior for time delayed filter combinations

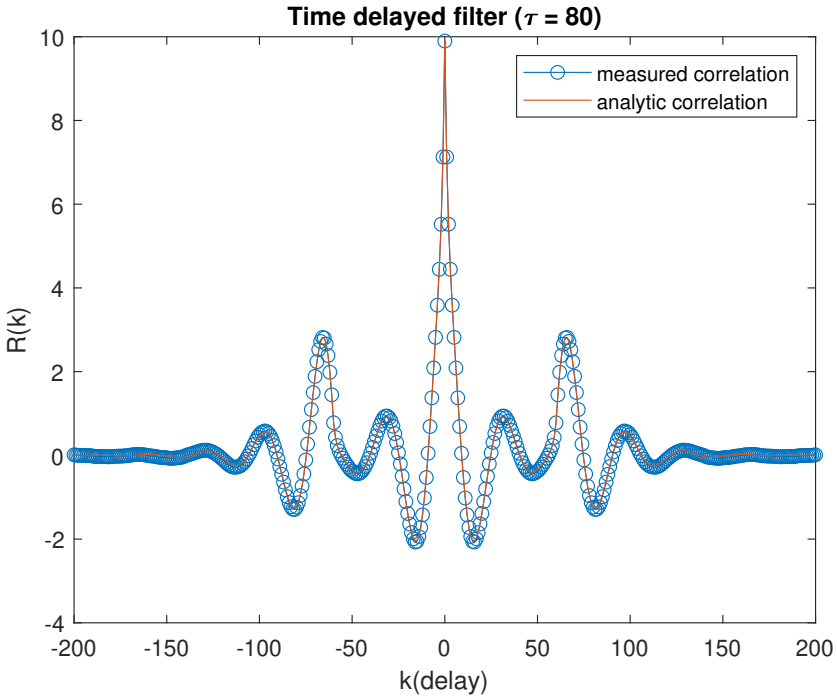


Figure 8: Comparison of the empirically measured correlation and the analytic correlation

4 The turbulence box

We first define the coordinate system that forms the base of our turbulence box. We consider z our temporal axis and y as our spatial plane. As such y is composed of y_1 and y_2. The stationary prospect of the spatial plane is something that is expanded upon once we try to rotate our turbulence box.

4.1 Turbulence model

The stochastic turbulence model will be largely focused on Liepmann's analysis [8]. The equations that govern this is shown below.

$$d = \begin{bmatrix} z^i - z^j \\ y_1^i - y_1^j \\ y_2^i - y_2^j \end{bmatrix} = \begin{bmatrix} kvT_s \\ \tilde{y_1} \\ \tilde{y_2} \end{bmatrix} \tag{68}$$

$$R(d) = \mu \left[g(|d|) + \frac{f(|d|) - g(|d|)}{|d|^2} d(m)^2 \right] \tag{69}$$

$$f(d) = e^{-\frac{d}{L_I}}, \qquad g(d) = \left(1 - \frac{d}{2L_I}\right) e^{-\frac{d}{L_I}} \tag{70}$$

Where d represents the distance between nodes on the 3 dimensions. m notes which dimensional component is being calculated. For the longitudinal component it would be $d(1) = kvT_s$. This neglects the cross interference between dimensional components, this is not an uncommon practice in turbulence simulation, but does lose some information. The main parameters to consider for this model is the length integral scale L_I and the mean wind speed v. Together with the time step T_s they compute the necessary delay horizon to be computed. For the remainder of this project, the mean wind speed will be 20 meters per second and the length integral scale will be 40. The time step has been chosen as $T_s = 0.1$ since that captures most, if not all of the high frequency dynamics of turbulence, while keeping computation to a minimum.

While most computation time is dominated by the filter identification specified in a later section, generating the correlation functions in (69) for N_y^2 node combinations can be quite costly. The absolute distance d is the only factor in the function, and for the spatially stationary case, this is non-unique for many node combinations. As such you can severely reduce the computation time by saving results for each unique distance calculated and inserting it should a new distance fall within a rounding tolerance. Depending on the resolution N of the frame the necessary computations Q are upper and lower bound by

$$N\sqrt{2} \leq Q \leq \frac{N^2}{2} + N \tag{71}$$

As the resolution becomes infinitely detailed, the distance between two neighbour nodes becomes infinitesimally small. As such any distance found in the frame will be infinitesimally close to any distance found along the diagonal. The upper bound comes from the necessary symmetry of the frame, any distance withing the frame can be found from the upper or lower triangle, including the diagonal.
In the cumulative computation, this optimization has minimal effect, but it is quite substantial on its own as the brute force computation has N^4 correlation functions that are calculated.

4.2 The Cholesky decomposition for correlated noise

At first we will consider the simpler alternative where the turbulence box is projected directly. The distance between all nodes will be constant and we can therefore calculate $\tilde{y}$ to benefit computation time. Applying the equation above for an N by N frame gives us an array with $N_y = N^2$ correlation functions. The necessary horizon for these correlation functions are proportional to L_I/v. This array will then be our desired auto-correlation or cross correlation for each node combination in $\mathbf{R}$. To generate such a target we will use the Cholesky decomposition. Simply applying the Cholesky decomposition directly does not work, we must consider each correlation function its own entry and use them in a modified Cholesky-Banachiewicz algorithm[1]. The entries of the resulting lower triangular matrix is found as:

$$\begin{aligned} \mathrm{L}_{j,j} &= \sqrt{R_{j,j} - \textstyle\sum_{k=1}^{j-1} L_{j,k}^2} \\ \mathrm{L}_{i,j} &= \tfrac{1}{L_{j,j}} \left(R_{i,j} - \textstyle\sum_{k=1}^{j-1} L_{i,k} L_{j,k} \right), \quad \forall \quad i > j \end{aligned} \tag{72}$$

This specific scheme is designed for uncorrelated noise and as such only operates in scalars. A clear example of this is the proposed division in (72), as all of our correlation functions go to zero such division is not ideal. The observation to make is that the rows, represented by i are the sums which eventually add to our resulting target correlation, the elements in the columns j, are scaling elements. Using this we can rewrite equation (72) as

$$L_{i,j} = \frac{1}{M_{j,j}} \left(R_{i,j} - \sum_{k=1}^{j-1} L_{i,k} M_{j,k} \right), \quad \forall \quad i > j \tag{73}$$

' Where $\mathbf{M}$ contains the center term for each component, in other words the respective variance for each output. Furthermore, the elements resulting from the above scheme are scaled as square root elements. The center term of each should be it's variance. We alter the scheme to the following

$$L_{j,j} = R_{j,j} - \sum_{k=1}^{j-1} L_{j,k} M_{j,k} \tag{74}$$

$$L_{i,j} = \frac{M_{i,j}}{M_{j,j}} \left(R_{i,j} - \sum_{k=1}^{j-1} L_{i,k} M_{j,k} \right), \quad \forall \quad i > j \tag{75}$$

' The elements of **M** are found by taking the Cholesky decomposition of $\mathbf{R}(0)$. We must now generate filters which realize each individual correlation function contained in **L**.

4.2.1 Off center maxima

The elements of **M** were described to be the center term, and not the maximum above. This is an important distinction as not all of the elements in **L** will have its maximum in the center. Take for example the example below.

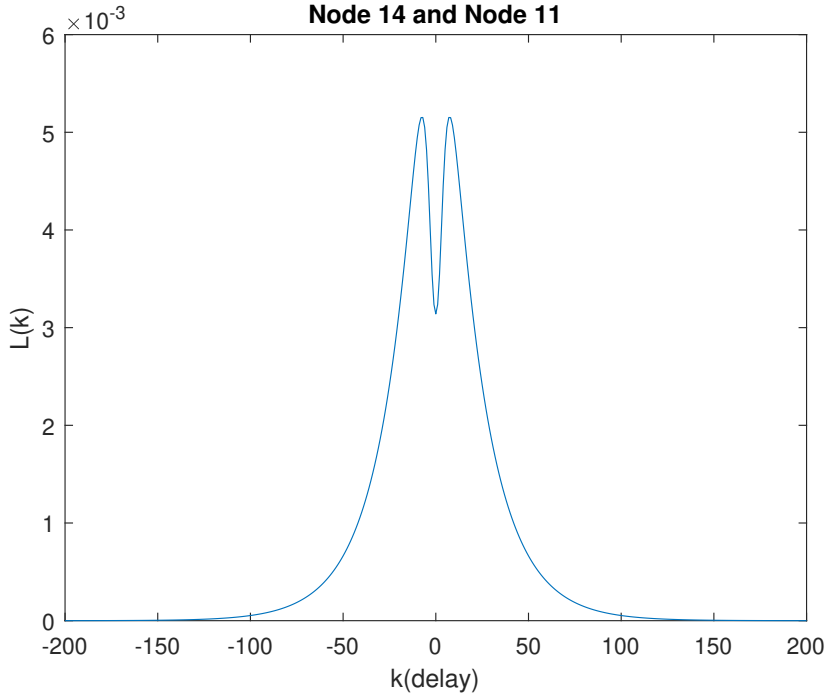


Figure 9: Special case where an element in L does not have its maximum centered

In future sections we will attempt to realize all the elements in **L** using discrete filters. The correlation function depicted above cannot be realized as such, since any signal will always correlate more with its immediate term than any other. However, should the function be considered as a cross correlation, then the function is somewhat realizable using time delays. This is not an exact analytic method by any means, but the cases where this occurs are always relatively small as can be observed from the y-axis in 9.

5 Filter Identification

We now have our target functions calculated from the cholesky decomposition of the Liepmann distribution. What remains is a method for realizing these target functions with relatively good accuracy.

5.1 Form identification

The base function forms derived in 3 is the basis of every correlation function we can estimate. The forms can differ quite a bit from one another and we must first identify what form our target function takes. For this purpose we introduce a function we define as the relative pole function.

$$T_d(k) = \frac{T(k+1)}{T(k)} \tag{76}$$

This function will then give us the relative pole at every point in the target function. We will use this information to distinguish between the forms used in section 3. For non-oscillating functions, this will converge to the dominating pole. This is effectively like taking the derivative of the target function.

5.2 Parameter identification

Every form in 3 has several variables which subtly alter how well it will fit with any target function. Once the form has been selected we need a scheme to identify all the parameters in that form.

5.2.1 First order filter

It is quite rare for the elements within the cholesky decomposition to be best estimated by a first order filter. When it is the case the relative pole in (76) will be constant. Calculating the gain and pole from then is trivial. Should there be a need to estimate a correlation function to first order when it does not fit perfectly, then the residual calculations noted below should be of use.

5.2.2 Second order filter

For non-oscillating filters, every function will be represented by a dominating pole and a secondary pole. In many cases this secondary pole is insufficient to fully match the residue left by the dominating pole. As such we propose a way to estimate a first order function which matches the variance and the stationary gain of said filter. The residual is given by

$$T_r = T - \frac{G_1 p_1^k}{1 - p_1^2} \tag{77}$$

Where T represents our target function. The equation that must now be estimated is:

$$\frac{G_2 p_2^k}{1-p_2^2} = T_r \tag{78}$$

Which for for variance and stationary gains form the following set of equations.

$$\frac{G_2}{1-p_2^2} = T_r(0) \tag{79}$$

$$\frac{G_2}{1-2p_2+p_2^2} = S_r \tag{80}$$

Where S_r represents the sum of the full residual. To get this we have used the relationship in (10) after finding the respective Z-transform of the residuals correlation function. Equating and rearranging then yields the following polynomial.

$$f(x) = \begin{bmatrix} x^2 & x & 1 \end{bmatrix} \begin{bmatrix} -T_r(0) - S_r \\ 2S_r \\ T_r(0) - S_r \end{bmatrix} \tag{81}$$

This polynomial will always have a trivial solution at $p_2 = 1$ which yields a critically stable filter. Thus the solution we are after is the other root. We then find the respective gain as

$$G_2 = T_r(0)(1-p_2^2) \tag{82}$$

We now recall the form of the 2nd order correlation function in (51). Equating G_1 and G_2 to their respective terms we find.

$$\frac{K_1^2}{1-p_1^2} + \frac{K_1 K_2}{1-p_1 p_2} = \frac{G_1}{1-p_1^2} \tag{83}$$

$$\frac{K_2^2}{1-p_2^2} + \frac{K_1 K_2}{1-p_1 p_2} = \frac{G_2}{1-p_2^2} \tag{84}$$

Which can be transformed to the following set of equations:

$$\begin{aligned} c_1 x_1 + c_3 x_3 &= r_1 \\ c_2 x_2 + c_3 x_3 &= r_2 \\ x_3^2 &= x_1 x_2 \end{aligned} \tag{85}$$

Where $x_1 = K_1^2$, $x_2 = K_2^2$ and $x_3 = K_1 K_2$. Substituting and rearranging gives us the following polynomial

$$f(x_3) = \begin{bmatrix} x_3^2 & x_3 & 1 \end{bmatrix} \begin{bmatrix} 1 - c_3^2 \\ c_3(r_1 + r_2) \\ -r_1 r_2 \end{bmatrix} \tag{86}$$

K_1K_2 is the variable x_3 and it will have the same sign as G_1G_2. After finding this variable we can now find K_1 and K_2.

$$\begin{bmatrix} K_1 \\ K_2 \end{bmatrix} = \begin{bmatrix} sgn(G_1) & 0 \\ 0 & sgn(G_2) \end{bmatrix} \begin{bmatrix} \sqrt{\frac{r_1 - c_3 x_3}{c_1}} \\ \sqrt{\frac{r_2 - c_3 x_3}{c_2}} \end{bmatrix} \tag{87}$$

What remains is our method of identifying this scheme as the proper one. This is where the relative pole function comes in, as this will give us both the ideal scheme and the dominating pole required to calculate the residual.

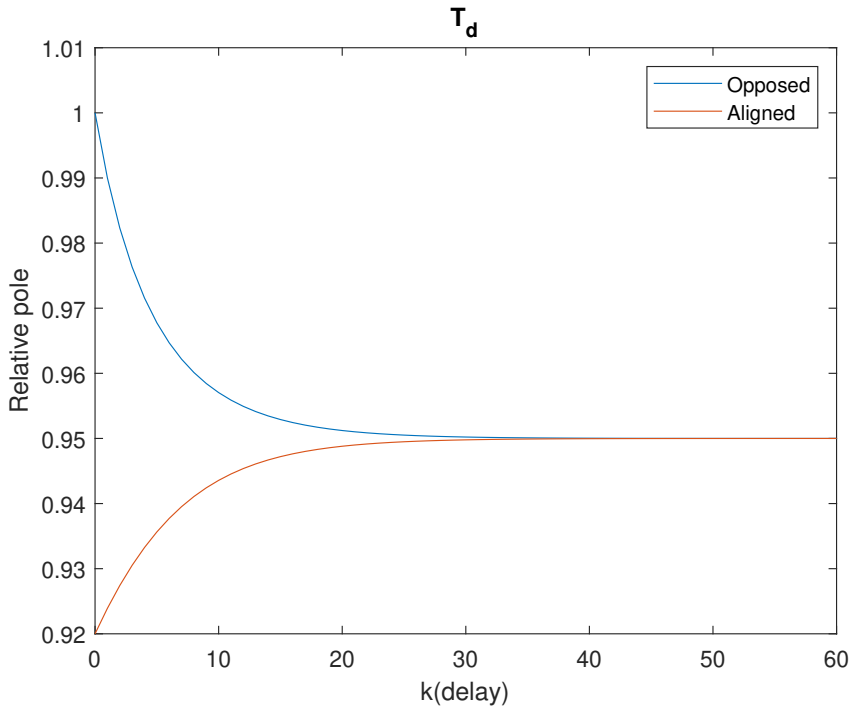


Figure 10: Relative pole of 2nd order filter scheme

From the above figure one can see that there is at least 2 poles in play as it is not in steady state initially. One can also read that the dominant pole $p_1 = 0.95$ and that the secondary pole settles after about 40 samples. Additionally, the rate of change informs us whether or not the secondary pole is directionally aligned or opposed to the dominating pole.

5.2.3 Oscillating filter

The most important difference in form selection is the difference between oscillating filters and non-oscillating filters. Generally, we don't expect oscillating filters for stationary spatial coordinates, but the relative pole function does give clear indication of oscillation if it is present.

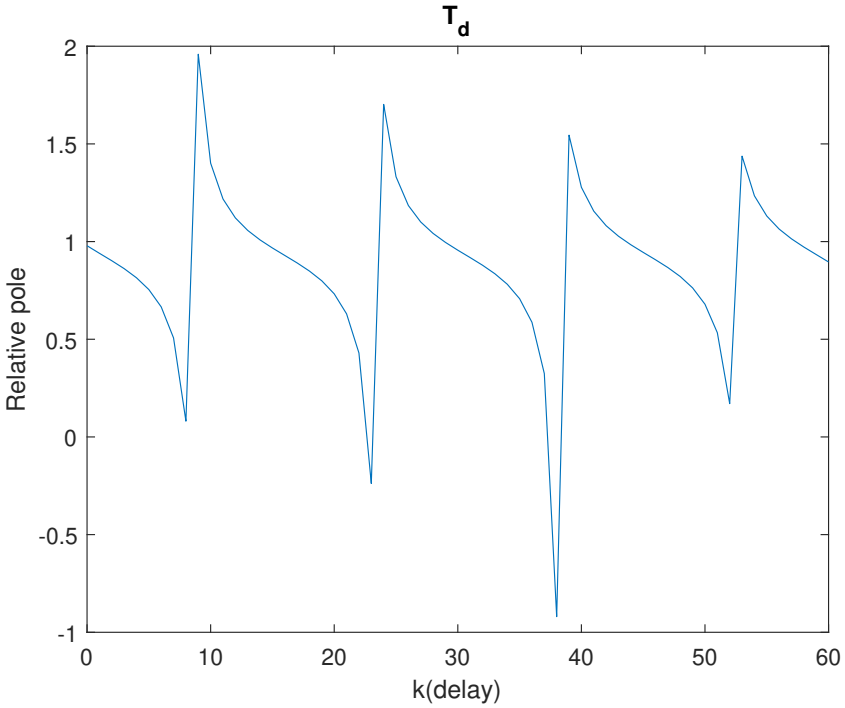


Figure 11: Relative pole function displaying oscillation

Particularly the zero crossing causes a tangential wave shape in the relative pole function which is a clear indication of oscillation. While there is no easily readable dominating pole in this case, the frequency of the oscillation is easily deciphered from the zero crossings. Preserving the variance yields.

$$\frac{K^2}{2}\left(\frac{1}{1-p^2} \pm \frac{1-p^2\cos(2\omega)}{1-2p^2\cos(2\omega)+p^4}\right) = T_r(0) \tag{88}$$

The indeterminate sign depends on whether we are approximating a sine filter or a cosine filter. Equating the preservation of stationary gain does not yield a pretty result like for the first order filter. Our estimation will therefore take a more numerical approach, similar to how we find the dominating pole for the first order case. The sampled angular velocity w is given by the zero crossings found in 11. Given that enough oscillation is present, we can calculate the relative change in peak to peak over the delay difference to find our pole. Once the pole and the sampled angular velocity have been found we find the gain as:

$$K = \sqrt{\frac{2T_r(0)}{A}} \tag{89}$$

The method proposed here is rather lackluster, which as we will find in later sections, also insufficient. The cholesky decomposition for the case of ro-

tating spatial coordinates yields many complex, if not downright impossible correlation functions. The method here is still documented as it does give good results for the more approachable cases.

5.2.4 Combination filter

In many cases the oscillation does not cross zero, but rather decays while oscillating as seen in 6. In such a case, the relative pole function will alter its rate of change without showing the traits of a zero crossing. An example of this can be seen below.

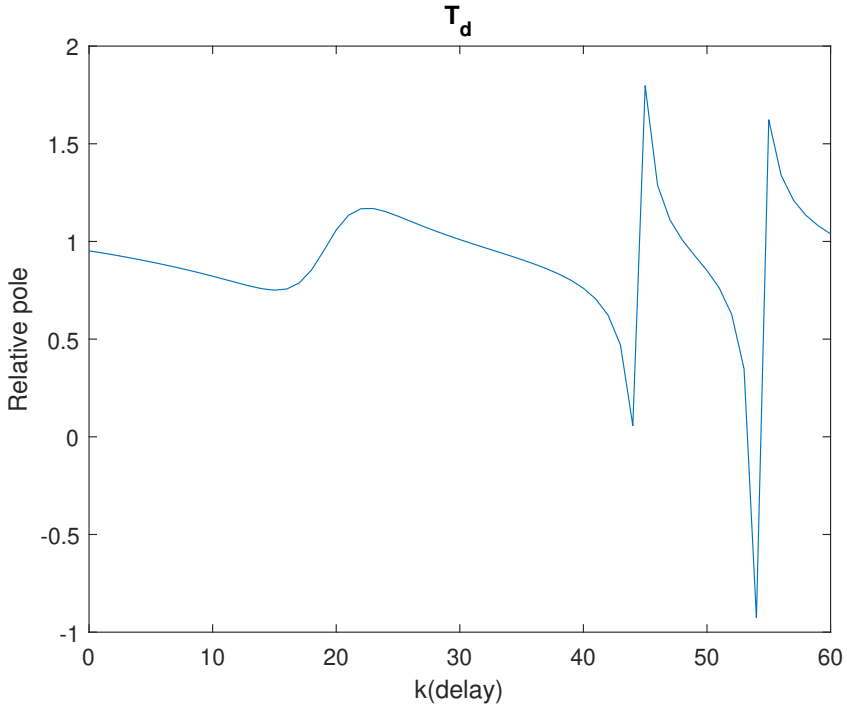


Figure 12: Relative pole function when oscillating pole is dominating

As in 6, the idea is to separate this function into the oscillating one and the strictly decaying one and identifying their parameters seperately through the methods noted above. Every oscillating target correlation function has a hidden imaginary component. If this was included in the relative pole function then the output would be more readable. Because we don't have this component we must use the peaks like we did for the simple oscillating filter. This is because the imaginary component is zero where the real component is at a "peak" which can be seen in 13. Thus we find the oscillating pole as

$$p_1 = \sqrt[m]{\frac{T(k+m)}{T(k)}} \tag{90}$$

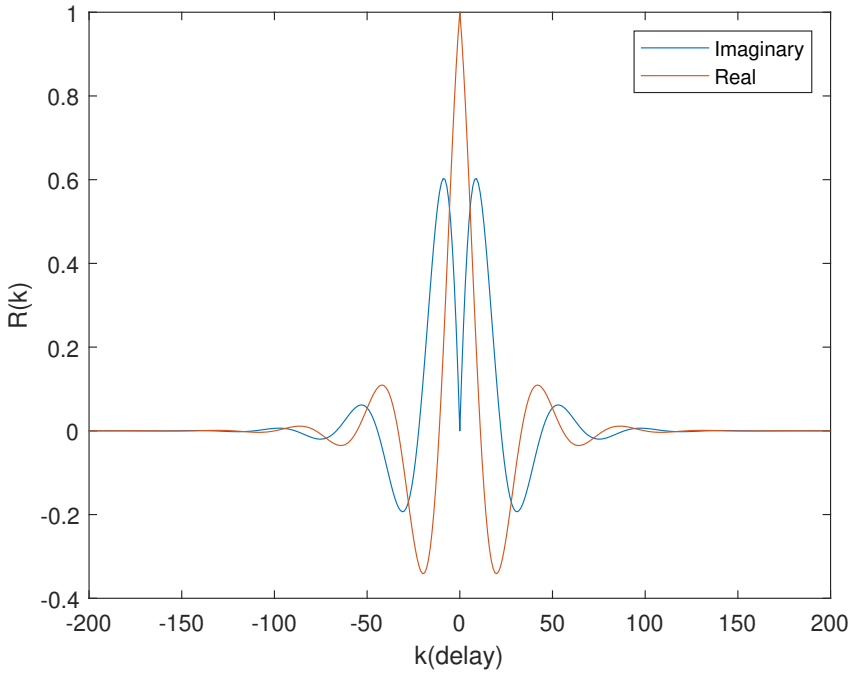


Figure 13: Showing the hidden imaginary component in any oscillating correlation function

Where m is the sample oscillation period, which can be read from the original relative pole function. For this to give the desired detail the secondary pole must decay in time for the oscillating pole to be isolated. This will be the case when the relative pole function shows delayed zero crossing as it does in 12.
If the oscillating pole is dominated, then the relative pole function will rarely show zero crossing, but will oscillate around the dominating pole. And thus the dominating pole can either be found from convergence, if the oscillating pole is sufficiently small, or approximated by taking the mean of the relative pole function.
In either case, the secondary pole can be found by analyzing the residual, which is found by calculating the dominating pole and its respective gain and subtracting it from the target function. When the oscillating pole is dominated the strictly decaying function can be easily found by aligning it with the part of the target function where the oscillation is gone.

In 3 we found that the strictly decaying term is

$$R_d(k) = K_d p_2^{|k|} \left(\frac{K_d}{1 - p_2^2} + K_o C \right) \tag{91}$$

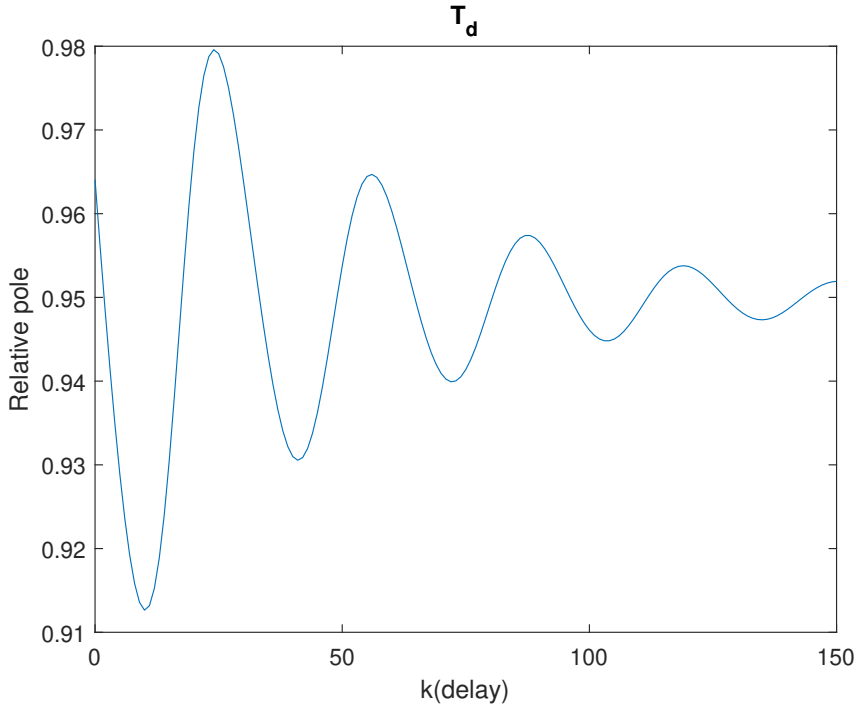


Figure 14: Relative pole function when the oscillating pole is dominated

And the oscillating term can be found to be.

$$R_o(k) = K_o p_1^k \left[\left(\frac{AK_o}{2} + K_d D \right) \cos(\omega k) + \left(\frac{BK_o}{2} + K_d C \right) \sin(\omega k) \right] \quad (92)$$

Which simplifies to

$$R_o(k) = K_o p_1^k \left(G \cos(\omega k + \phi) \right) \quad (93)$$

Where

$$G = sgn \left(\frac{AK_o}{2} + K_d D \right) \sqrt{\left(\frac{AK_o}{2} + K_d D \right)^2 + \left(\frac{BK_o}{2} + K_d C \right)^2} \quad (94)$$

and

$$\phi = \tan^{-1} \left(- \frac{\frac{AK_o}{2} + K_d D}{\frac{BK_o}{2} + K_d C} \right) \quad (95)$$

Which then gives us relative maxima and minima at

$$k_e = -\frac{\phi}{\omega} + n\pi \quad (96)$$

We can then use these points of extremities beyond the decayed pole to align our oscillating correlation function. Using equations (91), (94) and (95) we actually have an over determined set of equations. What should be noted here is that this set of equations don't necessarily have a solution, even when we disregard one of the equations, we might not be able to find a solution for K_o and K_d.
The goals in starting this project was to minimize computation, and therefore order of our filters have been limited. Even when extending the order to allow filter combinations to an order of three the oscillation resulting from the cholesky decomposition of the spatially rotating Liepmann distribution yields too much variation to be fitted by the base functions listed in 3.

5.3 Scheme flaws

The scheme noted in this section is not perfect by any means. Some of the common issues this method encounters are noted in this section. The roots and resulting values found in (87) and similar equations will sometimes yield complex results. This is a major issue as simulating complex filters takes far more computation than the strictly real case. To avoid this a few extra rules are applied.
If the secondary pole found from (81) is larger than the dominating pole, then it will be considered non-existent and the dominating gain will be adjusted such that the entire target function is estimated by a first order filter. This gives substantial error for the individual case but in the compound output this has hardly any effect. The filters would only end up with complex gains when attempting to approximate zeros where there should be none. For the real case in 5, the secondary pole is opposite to the dominating pole. The resulting filter should then be of order two, and relative order two.

$$H(z) = \frac{K}{(1 - p_1 z^{-1})(1 - p_2 z^{-1})} \tag{97}$$

Where K_1 and K_2 can be found, and relate to one another as

$$\begin{aligned} K_1 &= \frac{K}{1 - \frac{p_2}{p_1}} \\ K_2 &= \frac{K}{1 - \frac{p_1}{p_2}} \end{aligned} \tag{98}$$

The automated scheme implemented in MatLab will know when the poles are opposite to one another, but slight numerical errors will sometimes cause it to miss the relation above through some slight error. This is when its calculations yield complex zeros in the filter, when there should be none. When this occurs the scheme will return to a backup computation which simply forces the relation in (98).

5.3.1 Oscillating case

To see why the identification scheme for the oscillating case does not function we will follow its procedure on a common example. In 15 we see how the

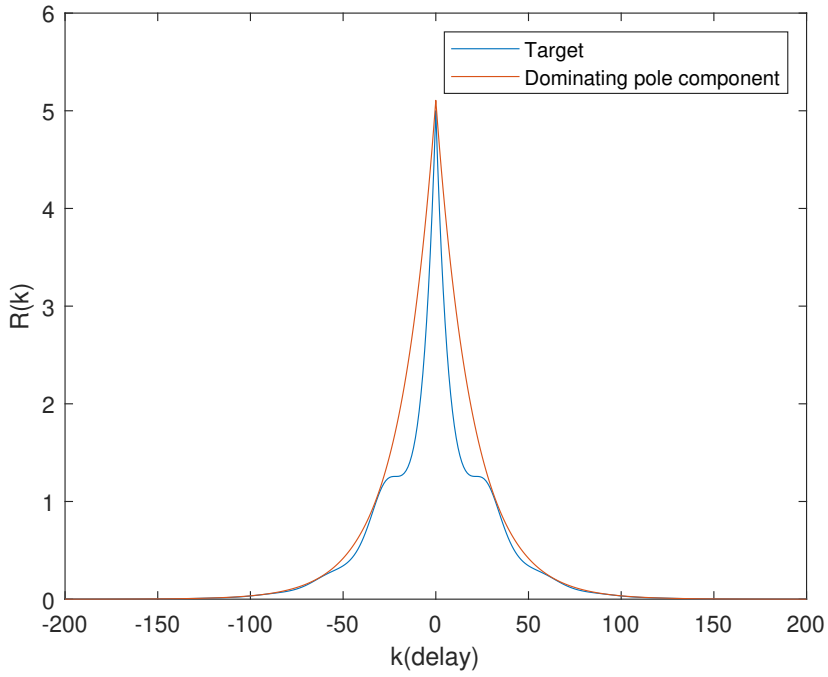


Figure 15: First step in identifying an oscillating combination filter

target function is being fitted by a single dominating pole at the largest delays. This is then subtracted from the original target function to give us our oscillating residue.

The resulting residue can be seen in 16 and it is immediately evident that this form has no representation in our list of base correlation forms in section 3. Since the correlation function is entirely negative we must invert the notion of maxima and minima. The minima is off center, which can not be represented by a time delay since this is a partial autocorrelation and not an individual cross correlation.

This residue will vary wildly, but it is very common for its form to be out of reach for our base forms. The reason this is not the case for the non-oscillating case is likely because there is less complexity in general. Since every correlation function is centered and strictly decreasing the same variation does not appear.

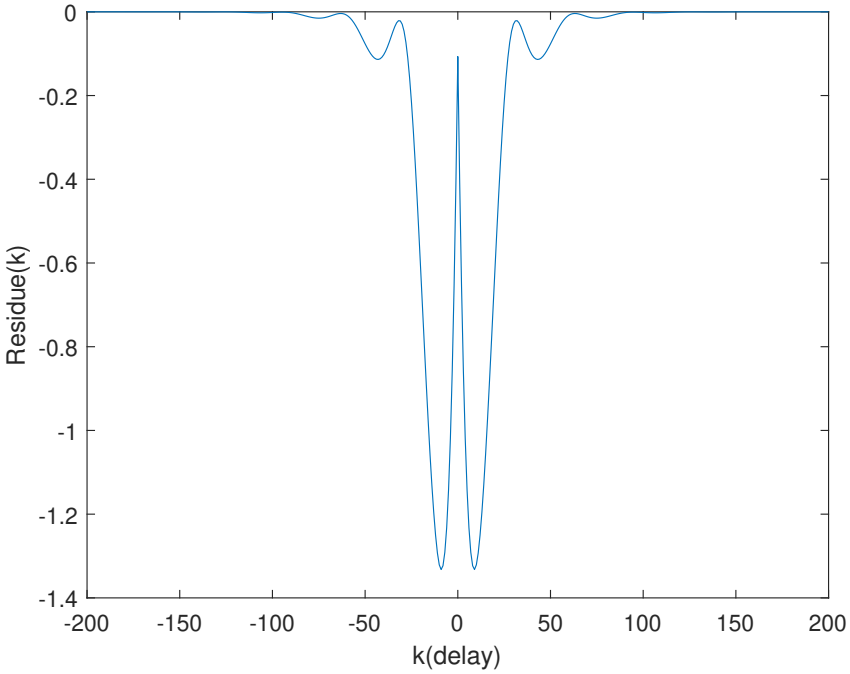


Figure 16: Residue resulting from fitting the dominating pole

6 Simulation

Now that we have a method of generating our N_y by N_y discrete filter. We will excite this filter with N_y unique sources of gaussian white noise. This will realize the correlation functions shown in section 3 and add them together. The calculation required for this simulation is oN_y^2 additions where o is the highest filter order found in the combined filter.

6.1 Stationary spatial coordinates

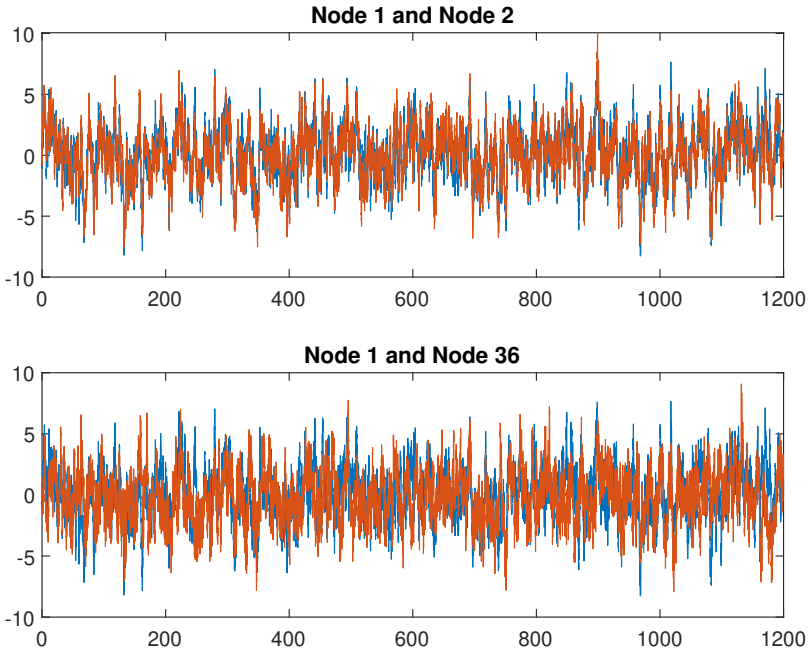


Figure 17: Longitudinal component comparison for close nodes (top) and nodes with far lateral seperation (bottom)

It can be seen that the nodes closely overlap when they are close, and overlap less so when they are further apart. This visual observation becomes clearer when we calculate the cross correlation between nodes. In 18 are the cross correlations for four randomly selected node combinations. The measured correlation is very close to the desired correlation, it should be noted that the functions will match on the right hand side, as cross correlation is not symmetric and has been calculated to match on that side. The slight discrepancies can be attributed to the flaws noted in 5.3 to some degree. We have neglected to mention the phase in any of the previous derivations. It is actually not possible to realize the correlation functions in section 3 exactly

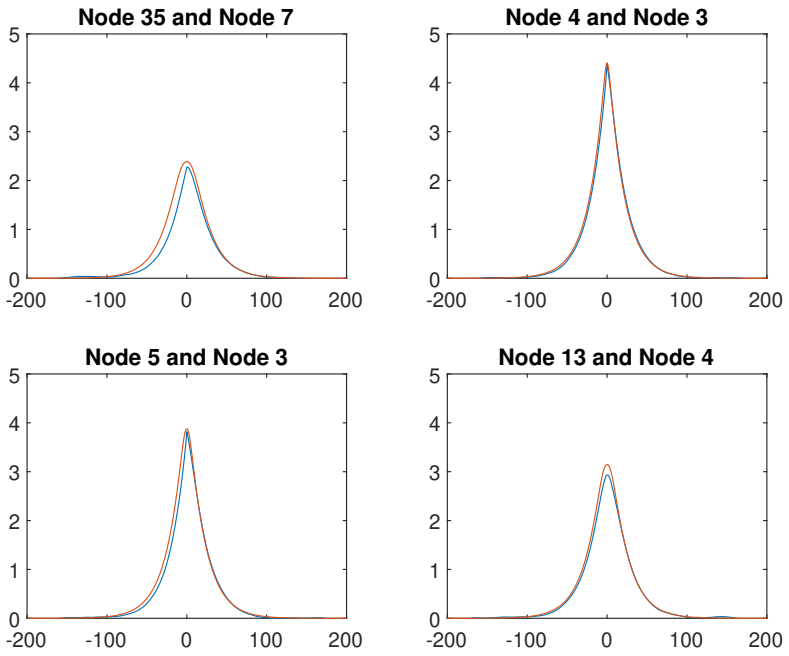


Figure 18: Desired correlation functions in red and measured correlation functions in blue

as varying degrees of phase shift will also shift the correlation lower. This is most evident for nodes which are generated by fewer unique sources of white noise.
In (72) we can see that the first column will be the sum of a single filter only.

In 19 we can see the clear effects phase shift has on the results. This is the main source of the error one sees in 18. As mentioned the error is reduced for nodes that have more contributing unique white noise sources, because they are added together by many filters with only slight differences in phase shift.
One method of realizing correlation without the phase shift would be to filter the white noise both ways through the filter. This will effectively double the order of the filter [17] and also make it impossible to generate online. Effectively limiting computation time and removing the most desirable trait this scheme has to reduce minimal error. This has been judged as unnecessary in this project.

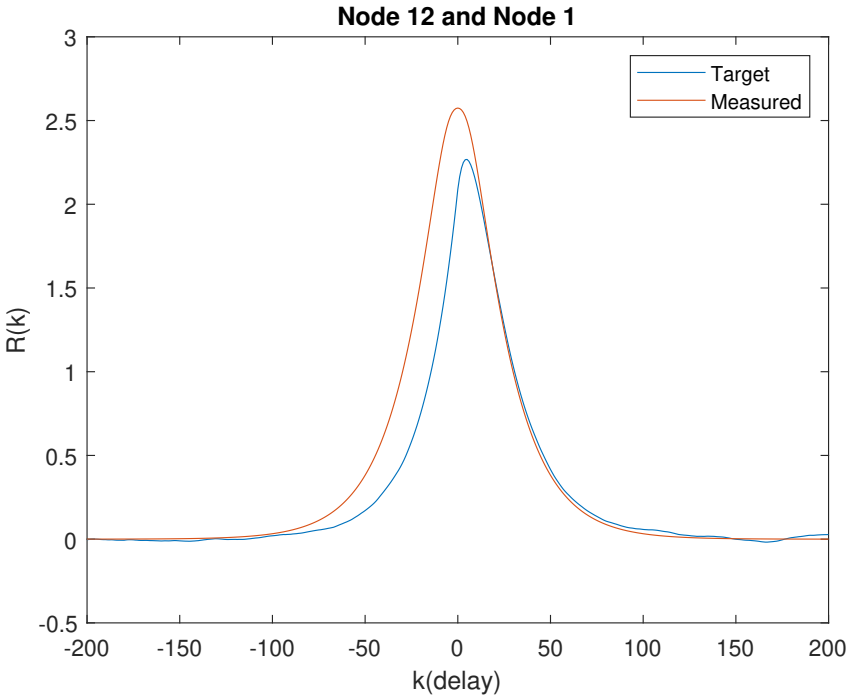


Figure 19: Comparison of target for generation and result

6.2 Approximating rotation

Originally, the rotation was meant to be calculated in 69 and adapted in sub filters after the cholesky decomposition. Because of the lackluster results this is not realistic with low end order filters. Instead, we will attempt to approximate the rotating behavior by first simulating nodes with stationary spatial coordinates, and then sampling those nodes through a rotating point. This rotation is reminiscent to how the blade of a windmill would sample the turbulence in such a box.
This method causes some extra computation, and we face an issue with the spatial resolution of our frame.

6.2.1 Spatially moving sensor

Our sensor node will be moving in the spatial plane following a preset function. In the case where we wish to mimic the movement of a wind turbine blade, this can be represented as.

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} r\sin(\omega T_s + \phi) \\ r\cos(\omega T_s + \phi) \end{bmatrix} \tag{99}$$

Where r represents the absolute distance from the origin we wish to place our node. The phase shift will be inconsequential for single node simulation, but if one wishes to simulate more than one turbine blade, then this will be set to the angle that the blades are distributed at.
This gives the position of the node in the y-plane at each sample point. One issue we face at this stage is that these coordinates do not necessarily overlap with the turbulence box we have generated and are sampling. In fact, as long as T_s is not π divided by an integer then there is no node generation which gives perfect axis overlap. With high spatial resolution this is a non-factor, but it should be explored how this impacts the simulation in lower resolution cases.

The angular velocity has been increased to 5 to show more detailed oscillation, for mean wind speed 20 we expect angular velocity close to 2. In the high resolution case, rounding the coordinates to the closest node yields acceptable results. We expect marginal error for reasons discussed above. As the resolution lowers then an increased information is lost since the distance increases. This error can be lowered by estimating an intermediary point using weighted contributions by the closest nodes. To minimize necessary computation, this will be simple interpolation. We did explore methods to calculate intermediary nodes with expected co-variance in relation to the closest nodes, but the additional information did not outweigh the cost of computation. With said scheme we end up with similar detail to the high resolution case without intermediary nodes.

One benefit of sampling our turbulence box this way is that it lets us move our sensor arbitrarily withing the turbulence box. If we would generate it

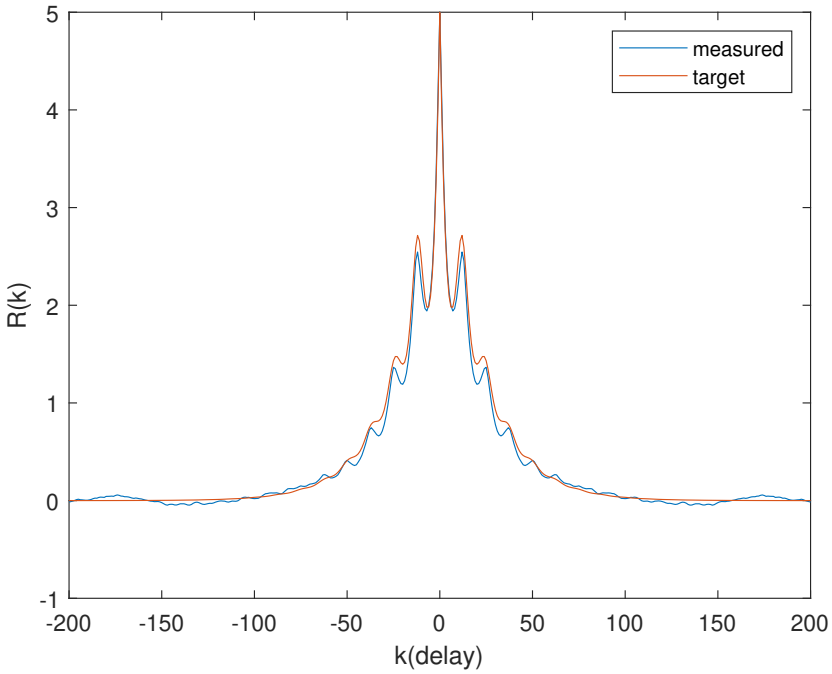


Figure 20: Rotational sampling for high spatial resolution (20 by 20)

directly from (69) then the angular velocity would have to be unchanging to fulfill the requirement of a stationary process. While it is not immediately relevant to the wind turbine simulation, an arbitrarily moving node could also be simulated this way. Such a node could also move in the longitudinal direction under the assumption of Taylor's frozen turbulence hypothesis.

6.3 Computational complexity

The computation is a recursive sum as one would expect from a linear discrete filter. The size of this single time step sum is upper bound by oN_y^2 where o is the highest order filter in the combined filtler. Minimizing the order term has been a large focus of this project and the result is a computation which can simulate beyond real time for filter sizes as large as $N_y = 4800$. The number 4800 comes from three 40 by 40 spatial frames simulating the three disjoint velocity components. Should interconnections between the velocity components be desirable then the necessary target correlation functions multiplies by three to create a three by three structure, which would then be halved to create the lower triangular Cholesky decomposition.
As a point of comparison, we will be comparing the generation time for a finite time horizon turbulence box with the existing method of using the inverse Fourier transform. Using the same computer for multiple generations

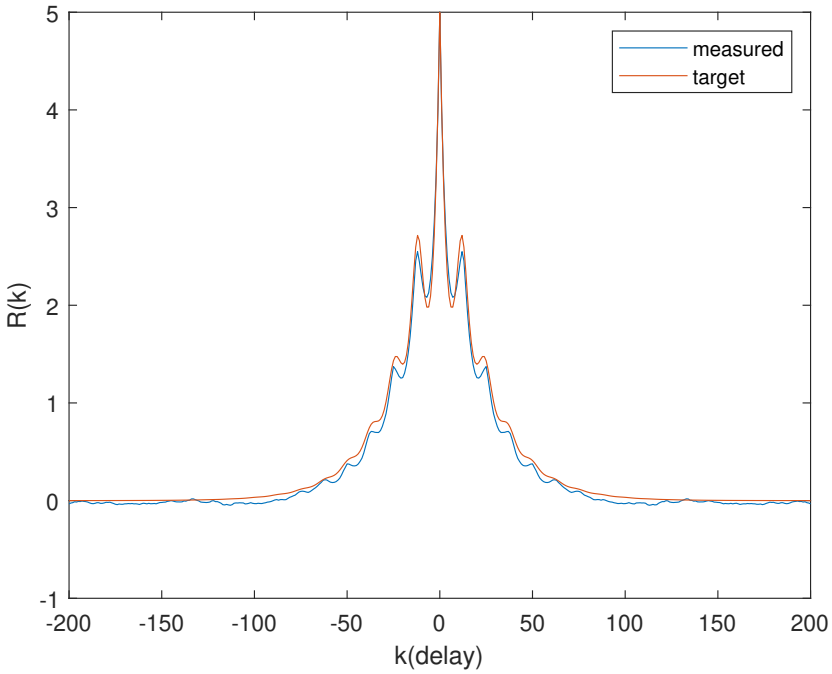


Figure 21: Rotational sampling for low spatial resolution (6 by 6) with intermediary nodes

for both methods it was found that the method proposed in this project is very close to the one used by TurbSim. Our method can create a turbulence box three times faster than TurbSim, but this is without the cross correlation between velocity components that TurbSim uses. Adjusting for this by a factor of three causes the results to be about the same. This brings us to the final cost benefit comparison.

The necessary computation time to generate the filter for this project is extremely high. The computation scales as N_y^2 which in terms of resolution is N^4. The 40 by 40 frame mentioned above took multiple days to compute for a low end CPU. This computation cost is purely pre-generated, and does not impact the real time computation in any way. The benefit one expects after this has been generated is of course the possibility to generate turbulence online. If one considers the benefit of online generation greater than the cost of upfront computation time, then the method proposed here should be considered.

7 Future work

In the precursor to this project [11], we managed to do online simulation if we managed to avoid instability in the filter identification step. In this project we have managed to remove this instability entirely. While we can now accurately and efficiently simulate turbulence online, we are severely limited by the quite simple approach of fitting linear filter correlation functions to Liepmann's model(69).
Every system will be some recursive difference function, as the memory required for a purely FIR filter would leave the scheme cripplingly slow. However, there are possible extensions that can be made to the base functions 3 by possibly including non-linear difference equations.
It should also be noted that a lot of the identification scheme for the oscillating case is numerical and therefore not as accurate as it could be. The oscillating base functions are likely insufficient regardless, but improvements to the proposed method could be made by possibly equating function traits which we missed.
The upfront computational cost of generating the filter can be quite high, especially as the resolution grows. In equation(71) we can reduce this cost by several magnitudes, but only for the target correlation functions, not the ones resulting from the Cholesky decomposition. The Cholesky decomposition is effectively multiplication and addition of the target correlation function. In an attempt to utilize the optimization in (71) we tried to identify poles and respective gains in the target matrix $\mathbf{R}$ and then defining the base operations as a function of said poles and gains. This showed good results for the first elements in $\mathbf{L}$, but as the recursive nature of the decomposition repeated the operations many times over, the accuracy got lost. Should this be expanded upon, or perhaps an alternative method which lets the filter identification happen for the target correlation $\mathbf{R}$ instead of the lower triangular decomposition $\mathbf{L}$, upfront cost could be reduced drastically.

7.1 Kalman Filter

Another application of generating turbulence this way is Kalman filtering. This method relies on known covariance of the states and a system model. In this project we have found an approximate system model in the generated filter. The recursive estimated state covariance in a Kalman filter is given by the following equation.

$$\mathbf{P}_{k+1} = \mathbf{A}\mathbf{P}_k\mathbf{A}^T + \mathbf{Q} \tag{100}$$

$\mathbf{A}$ is the state space representation of the system model and $\mathbf{Q}$ is the covariance of these states. It may seem obvious that we already have the elements in $\mathbf{Q}$ from the Liepmann model. However the state space representation includes every internal state in our filter. For every filter with an order greater

than one, there will be unseen internal states which are also included as elements in $\mathbf{Q}$. A method to either account for these internal states, or one which circumvents them entirely could yield an efficient way to filter turbulence. In this case the turbulence is not filtered down to the mean wind speed component, but rather the measurement noise is filtered as to yield the physical turbulence itself.

8 Appendix

MATLAB Code

Filter identification

```
function [F] = polegain1x(T)
T = T(:);
z = tf('z',0.1);
J = floor(length(T)/2); %Horizon

[m,I] = max(T);
delay = 0;
if I ~= J+1 %Center target and save delay
    delay = abs(I-J-1);
    T(J+1-delay:J+delay) = [];
    pad = zeros(1,delay);
    T = [pad'; T; pad'];
end

T_1 = T/max(abs(T)); %Normalize

L = (find(abs(0.02-abs(T_1))<0.005)); %Settling
   horizon for secondary pole
L = max(abs(J+1-L));

p1 = T(J+2+L)/T(J+1+L); %Find dominant pole and gain
G1 = (1-p1^2)*T(J+1+L)/(p1^L);
k = -J:J;
T_2 = T' - G1*p1.^abs(k) /(1-p1^2); %Calculate
   residual

m = T_2(J+1);
S = sum(T_2);
poly = [-m-S 2*S m-S]; %characteristic polynomial

a = poly(1);
b = poly(2);
c = poly(3);
pc(1) = (-b+sqrt(b^2 - 4*a*c))/(2*a);
pc(2) = (-b-sqrt(b^2 - 4*a*c))/(2*a);

p2 = min(pc(pc>0)); %Find secondary pole

```

```
37 G2 = m*(1-p2^2); %Find secondary gain
38 if  abs(m)/max(T) < 0.01 || p2 >= p1 %Failsafe to
      avoid instability
39 G1 = max(T)*(1-p1^2);
40 F = sqrt(G1)  /(1-p1*z^-1); %Set filter first order
41 elseif abs(S) < 10e-3 %Tolerance
42 F = 0;
43 else
44 if G1 > 0 && S < 0 %If gains are opposite aligned
45     a = (1-p1^2)*(1-p2/p1)^2;
46     b = (1-p1*p2)*(1-p1/p2)*(1-p2/p1);
47     c = (1-p2^2)*(1-p1/p2)^2;
48     K = sqrt(max(T)/(1/a + 2/b + 1/c));
49     F = K/((1-p1*z^-1)*(1-p2*z^-1)); %Filter output
50 else %If gains are aligned
51 a = 1/(1-p1^2);
52 d = 1/(1-p2^2);
53 b = 1/(1-p1*p2);
54
55 rhs1 =  G1/(1-p1^2);
56 rhs2 =  G2/(1-p2^2);
57
58 ak = 1- b^2/(a*d);
59 bk = (rhs1+rhs2)*b/(a*d);
60 ck = -rhs1*rhs2/(a*d);
61 rr(1) = (-bk+sqrt(bk^2 - 4*ak*ck))/(2*ak);
62 rr(2) = (-bk-sqrt(bk^2 - 4*ak*ck))/(2*ak);
63
64 if abs(G1*G2) > 10e-6 %Tolerance to avoid complex
      solutions
65     if G1*G2 >= 0
66     u3 = min(rr(rr>0));
67     else
68     u3 = max(rr(rr<0));
69     end
70 else
71     u3 = 0;
72 end
73 u1 = (rhs1 - b*u3)/a;
74 u2 = (rhs2 - b*u3)/d;
75
76 C1 = sqrt(u1)*sign(G1);
77 C2 = sqrt(u2)*sign(G2);
78
```

```
79  F = C1/(1-p1*z^-1) + C2/(1-p2*z^-1);%Filter output
80  end
81  F = F*z^-(delay); %Add delay if any
82
83  end
```

Target correlation generation

```
function [R] = fetchcovw(y_1,y_2,k,Ts,L,v,u_s,w,comp)
ffunc = @(r) exp(-r/L); %Nested functions
gfunc = @(r) exp(-r/L)*(1-(0.5*r/L));

y_2 = y_2*ones(1,length(y_2)); %Coordinate arrays
y_1 = ones(length(y_1),1)*y_1;


N = length(y_1); %Resolution
for m1 = 1:N
for m2 = 1:N
    sub1 = (m1-1)*N + m2; %First node index
for i = 1:N
    for j = 1:N
        sub2 = (i-1)*N + j; %Second node index
        Ro = [1, 0, 0;
            0, cos(k*w*Ts), -sin(k*w*Ts);
            0, sin(k*w*Ts), cos(k*w*Ts)];

        d = abs([Ts*v*k;y_1(m1,m2);y_2(m1,m2)] - Ro
           *[0;y_1(i,j);y_2(i,j)]); %Distance
        dd = norm(d);
        if dd == 0
            R(sub1,sub2) = u_s;
            %Define function for |d| = 0
        else

        R(sub1,sub2) = u_s*(gfunc(dd) + (ffunc(dd)-
           gfunc(dd))/(dd^2) * d(comp)^2); %
           correlation at k
        end
    end
end
end
end
end
```

Modified cholesky decomposition

```
function [coh2,filter] = chold(Coh)

[Ny,~,Nx] = size(Coh); %Node amount and full horizon
   length
coh2 = zeros(Ny,Ny,Nx);
J = floor(Nx/2); %symmetric horizon
g = chol(Coh(:,:,J+1))'; %Center term cholesky d
G = g.^2; %Square to adjust for filter ID root

gMax = Coh(:,:,J+1); %Max of target functions

for i = 1:Ny
    for j = 1:i
        summ = zeros(1,2*J+1); %set sum vector 0
        if j == 1 %No cumulation for first column
            coh2(i,j,:) = G(i,j)*Coh(i,j,:)/gMax(i,j)
               ;
        else
        for k = 1:j-1
            m = coh2(i,k,:)/g(i,k);
            mz = m(:)';
            summ = summ + mz*g(j,k);
            %cumulate along column for each row
            if i == j
            Targ = Coh(i,i,:); %Target
            coh2(i,j,:) = Targ(:)' - summ;
            %Chol diagonal
            else
            Tx = Coh(i,j,:); %Target
            coh2(i,j,:) = (Tx(:)' - summ)/g(j,j);
            %Chol
            coh2(i,j,:) = coh2(i,j,:)*g(i,j);
            %Square by scalar
            end
        end
        end
        [filter(i,j)] = polegain1x(coh2(i,j,:)); %
           Filter ID
      end
end
end
```

References

[1] *Cholesky decomposition - Wikipedia*. https://en.wikipedia.org/wiki/Cholesky_decomposition. (Accessed on 05/12/2022).

[2] *Estimate state-space model using subspace method with time-domain or frequency-domain data - MATLAB n4sid*. https://www.mathworks.com/help/ident/ref/n4sid.html. (Accessed on 11/18/2021).

[3] *Euler's formula - Wikipedia*. https://en.wikipedia.org/wiki/Euler%27s_formula. (Accessed on 05/06/2022).

[4] Bojana Gajic. *Korrelasjon og energispektraltetthet for sekvenser med endelig energi*.

[5] Bojana Gajic. *Tidsdiskrete systemer - beskrivelse i tidsdomenet*.

[6] Bojana Gajic. *z-transformen og analyse av digitale systemer*.

[7] *Geometric series - Wikipedia*. https://en.wikipedia.org/wiki/Geometric_series. (Accessed on 05/06/2022).

[8] Stewart Glegg and William Devenport. *Aeroacoustics of low Mach number flows: fundamentals, analysis, and measurement*. Academic Press, 2017.

[9] Magne H. Johnsen and Torbjørn Svendsen. *DFT/IDFT og FFT samt anvendelser*.

[10] Jakob Mann. "Wind field simulation." In: *Probabilistic engineering mechanics* 13.4 (1998), pp. 269–282.

[11] Tormod Gjerde Nes. *Recursive generation of stochastic aerodynamic models*.

[12] Steven A Orszag. "Numerical methods for the simulation of turbulence." In: *The Physics of Fluids* 12.12 (1969), pp. II–250.

[13] G.I. Taylor. *The Spectrum of Turbulence*.

[14] Nick Jenkins Ervin Bossanyi Tony Burton David Sharpe. *Wind Energy Handbook*. John Wiley Sons, LTD, 2001.

[15] *White noise - Wikipedia*. https://en.wikipedia.org/wiki/White_noise. (Accessed on 05/08/2022).

[16] *Wind field simulation (text-based input) - File Exchange - MATLAB Central*. https://www.mathworks.com/matlabcentral/fileexchange/50041-wind-field-simulation-text-based-input?s_tid=mwa_osa_a. (Accessed on 10/12/2021).

[17] *Zero-phase digital filtering - MATLAB filtfilt*. https://www.mathworks.com/help/signal/ref/filtfilt.html. (Accessed on 04/26/2022).