Omer Mohamed Babiker

2D wavelet analysis of the free surface with subsurface turbulence

Master's thesis in Energy and Environmental Engineering
Supervisor: Simen Andreas Ådnøy Ellingsen
June 2021

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Energy and Process Engineering

Omer Mohamed Babiker

2D wavelet analysis of the free surface with subsurface turbulence

Master's thesis in Energy and Environmental Engineering
Supervisor: Simen Andreas Ådnøy Ellingsen
June 2021

Norwegian University of Science and Technology
Faculty of Engineering
Department of Energy and Process Engineering

Preface

This is the Master's thesis for Omer Babiker, and is the end of his study work at Energy and Environmental Engineering course at the Norwegian University of Science and Technology (NTNU). This thesis was supervised by Simen Andreas Ådnøy Ellingsen, PhD PhD, and done under the Department of Energy and Processing Engineering at NTNU. It represents a total of 30 ECTS.

The goal was to investigate the use of wavelet transformation to identify near surface turbulent structures by looking at their imprint on the surface elevation. The data for this work was given by Xin Gou and Lian Shen from the University of Minnesota and the main software used for the analysis of the data was MATLAB.

I would like to give special thanks to my supervisor, Simen Andreas Ådnøy Ellingsen, PhD PhD, and to Sander Holum, Stefan Weichert and Benjamin Keeler Smeltzer, PhD, for help and support along the way.

Abstract

Wavelet transformations are a powerful tool in data analysis and computer vision. Here we use them to identify near surface turbulent structures by looking at the imprint they have on the surface. The 2D wavelet transformations using the MATLAB Wavelet Toolbox was implemented on the surface of Direct Numerical Simulation (DNS) data to find both attached vortices and up- and downwellings.

The data contained both the values for the surface elevation and the velocity and pressure field underneath. Using the velocity field the attached vortices and the up- and downwellings where located. The attached vortices were identified using a method for finding vortex cores for a given velocity field. For the up- and downwellings the surface divergence was used, as these structures appear in areas of very large (upwellings) or very small (downwellings) divergence.

The 2D wavelet transform of the surface elevation was performed and compared with the location of the near surface turbulent structures found using the velocity field. The Mexican hat wavelet was the main wavelet used. It turned out that, with this wavelet, the attached vortices gave strong localised positive values in wavelet space, while the up- and downwellings gave negative values. When only looking at the positive values above a certain threshold, other structures then the attached vortices also appeared, probably lingering structures close to up and downwellings. However, the attached vortices have a characteristic circular shape, while these other structures are mainly elongated. The strength of the vortices was also compared to the response they had in wavelet space and some correlation was found, However, only a limited number of vortices were tested in this case.

For the up- and downwellings looking only at the negative values of the threshold below a certain value filtered out these structures quite well. Since these surface structures are sometimes elongated, an anisotropic version of the Mexican hat was tested as well to see if it resulted in a better fit. For the smallest scale, squeezing the wavelet gave slightly better results than just using the isotropic version.

Lastly, the mean surface curvature bared a good resemblance to the small scale wavelet transform and this was investigated. It turns out that transforming with the Mexican hat wavelet is close to a high order sum of the second derivatives of the surface.

Sammendrag

Wavelet-transformasjoner er et kraftig verktøy i dataanalyse og datasyn. Her skal vi bruke dem til å identifisere turbulente strukturer i nærheten av overflaten ved å se på avtrykket de har på overflaten. 2D-wavelet-transformasjonen ved hjelp av MATLAB Wavelet Toolbox ble implementert på overflaten av "Direct Numerical Simulation" (DNS) data for å finne både virvler og opp- og nedvellinger.

Dataene inneholdt både overflatehøyden og hastighets- og trykkfeltene under. Ved hjelp av hastighetsfeltet ble virvlene og opp- og nedvellingene plassert. Virvlene ble identifisert ved hjelp av en metode for å finne virvelkjerner i et gitt hastighetsfelt. For opp- og nedvellingene ble overflatedivergensen brukt, da disse strukturene dukker opp i områder med svært stor (oppvelling) eller svært små (nedvelling) divergens.

2D-wavelet-transformasjonen av overflatehøyded ble utført og sammenlignet med plasseringen av de turbulente strukturer i nærheten av overflaten som ble funnet ved hjelp av hastighetsfeltet. "Mexican hat" waveleten var den som ble brukt for det meste av arbeidet. Med denne waveleten, ga virvlen sterke lokaliserte positive verdier i wavelet-rommet, mens opp- og nedvellingene ga negative verdier. Når vi bare ser på de positive verdiene over en viss terskel dukket det op andre stukturere enn virvlene, sannsynligvis strukturer nær opp- og nedvellinger. Imidlertid har virvlene en karakteristisk sirkulær form, mens disse andre strukturene er langstrakte. Styrken til virvler ble også sammenlignet med responsen de hadde i wavelet-rom, og det ble funnet noe sammenheng, men bare et begrenset antall virvler ble testet i dette tilfellet.

For opp- og nedvellingene, ble de filtrert gangske bra når man ser kun på verdier i wavelet rom under en viss terskel. Siden disse overflatestrukturene er noen ganger langstrakte, ble en anisotrop versjon av "Mexican hat" også testet for å se om den ga bedre resultater. For den minste skala, en klemt versjon av waveleten ga litt bedre resultater enn bare å bruke den isotropiske versjonen.

Til sist, ble det oppdaget at overflatekurvaturen lignet veldig på små skala wavelet transformasjonen og dette ble undersøkt. Det visste seg at å transformere med "Mexican hat" er veldig nærme en høy ordens sum av de andrederiverte.

Table of Contents

Nomenclature

p_1, p_2	=	Pressure at point at 1 or 2
ρ	=	Fluid density
g	=	Gravitational acceleration
h_1, h_2	=	Relative height difference from reference at point 1 or 2
v_1, v_2	=	Velocity at point 1 or 2
ψ	=	Wavelet function
$f(x), h(x)$	=	General function
$\psi_{u,s}$	=	Wavelet function scaled by s and translated by u
$\|\|f\|\|, \|\|\psi\|\|$	=	Magnitude of f and ψ
f^*, h^*, ψ^*	=	Complex conjugate of the functions f, h and ψ
$\widehat{f}, \widehat{h}, \widehat{\psi}$	=	Fourier transforms of the functions f, h and ψ
x, t	=	General variables
ω, f	=	Frequency
s	=	Wavelet scaling variable, $\in \mathbb{R}^+$
u	=	Wavelet translation variable, $\in \mathbb{R}$
c	=	Constant
W_f	=	Wavelet transform of a function f
C_ψ	=	Admissibility condition for the wavelet function ψ
$T^{\vec{u}}$	=	2D translation operator
D^s	=	2D dilation operator
R^θ	=	2D rotation operator
r_θ	=	2D rotation matrix
$\psi_{\vec{u},s,\theta}$	=	Wavelet function rotated by θ, scaled by s and translated by $\vec{u}$
$\psi_{\vec{u},s}$	=	Wavelet function scaled by s and translated by $\vec{u}$
L_x, L_y	=	Domain size in respectively the x and y directions
$\overline{H}$	=	Domain height (in the z direction)
N_x, N_y, N_z	=	Number of cells in respectively the x, y and z directions
SD	=	Surface divergence
$\mathcal{E}$	=	Enstrophy

$\omega_x, \omega_y, \omega_z$	=	Vorticity in respectively the x, y and z directions
H	=	Mean surface curvature
K	=	Total or Gaussian curvature
R_1, R_2	=	Principal radii of curvature

List of Figures

1 Introduction

1.1 Background

The transfer of heat, gas an momentum between the sea or ocean and the air around it happens at the surface. Christopher et al., 2007 state that gas transfer between the air and the water influences the flux of CO_2 and other climatically important gasses on a global scale. Jähne et al., 1987 says that this gas transfer is important both on large scales, for global systems, but also on a smaller scale, for recycling gasses in rivers and lakes. Currently the gas exchange is estimated using approximation based on the wind speed, Wanninkhof et al., 2009, Frew et al., 2004. However, these methods can be quite inaccurate, especially for strong winds. Moreover it is now known that the source of the exchange is not the wind, but near-surface turbulence (Frew et al., 2004, Turney and Banerjee, 2013). The wind may agitate the fluid near the surface, increasing near surface turbulence, and increasing the exchange particles and heat, however, the structures right under the free surface are just as important.

It has been found that slow long-lived near surface turbulent structures are the most effective, Turney and Banerjee, 2013, Kermani et al., 2011, Khakpour et al., 2012. They renew the surface keeping it from reaching an equilibrium and facilitate the exchange. What is not know, however, is a good way to identify these turbulent structures, and quantifying their effect on the interactions between surface and air. Identifying these structures usually means looking into the velocity field right under the surface, but that is not easily done on a large scale, as it means measuring the velocity field of seas, oceans, rivers and lakes. These near surface turbulent structures agitate the surface, locally changing the elevation, so, another idea is to look at the imprint they leave on the surface like Savelsberg and Van De Water, 2009, have done. This means that one could take a picture of the ocean from the air or from satellites or drones and find all the fluid structures only by looking at the surface elevation. In Savelsberg and Van De Water, 2009, the surface and velocity fields were measured separately and they did not detect distinct structures on the surface, they performed statistical and qualitative analysis of near surface turbulent structures. The goal in this project was to identify specific structures with the help of wavelet transformations.

1.2 Motivation for wavelets

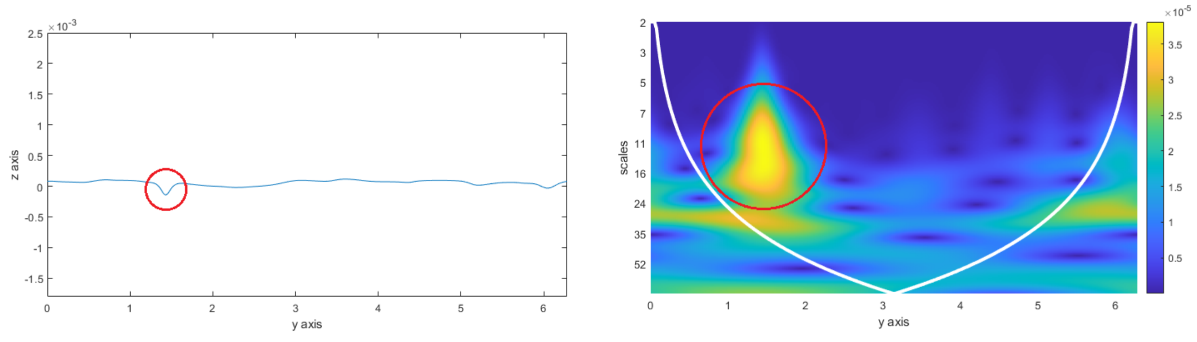
Wavelet transformations are a method of representing a signal or a function based on a mathematical construct called wavelets, Mallat and Peyre ́, 2008. This is similar to Fourier transformations where the signal is represented as the sum of sinusoidal waves. But the Fourier transform of a signal is only a function of the frequencies of the waves it is composed of; it gives no information on where these frequencies are localised. Wave-like phenomena are well described using Fourier transformations, whereas non-dispersive structures, like vortices, are localised. Wavelet transformations try to amend this by using both a frequency and a localisation/time parameter in the transform. This will prove useful when we are trying to filter out stationary long lived structures.

Dolcetti and García Nava, 2019 used 1D wavelet analysis to measure the surface elevation. They used what's called the Wavelet Directional Method, WMD, (Donelan et al., 1996). This method is used in measurements of the surface elevation by minimising the number of measurements locations. The transforms in time at some points where calculated and used to describe the rest of the surface. While in this work, the transforms of the whole surface are of interest.

Some might ask why machine learning is not used, as it has proven to be a very effective method to identify shapes and object in images, Li et al., 2020. The reason is simply because wavelets are a straight forward mathematical concept like Fourier transformations, and it is quite simple to see how and why they perform and how they detect what they detect. In machine learning, the algorithm is very much a black box, and it is not always possible to know what is happening. Having simple to understand methods also makes it easier to correlate with the physics behind what we are observing.

1.3 Project work

This work was preceded by a project where the wavelet transforms where tested and used on the surface to get an idea of how they worked and what kind of information could be gathered from them. In the project one dimensional wavelet transforms where implemented along straight lines on the surface. This work focuses on 2D wavelet transform of the whole surface. In the project, however, lines crossing the surface where chosen and a one dimensional wavelet transform was done. The lines where chosen so that they crosses a dimple on the surface. This is one of the long-lived turbulent structures that facilitate the exchange of heat and gas mentioned earlier. These are the response to a small turbulent vortex right underneath the surface that is attached to id and forms a small dimple. Figure 1.1 shows the surface elevation along the y-axis of the direct numerical simulation done by Guo and Shen (the same dataset used in this work), Figure 1.1a is the surface elevation, and Figure 1.1b is it's transform. The red circles outline a dimple on the surface, both in the original signal and in the wavelet transform. It turns out that, in the transform, these dimples make these pyramid like shapes, and are fairly simple to recognise. Structures of different scales and shapes gave different responses in the wavelet transform. How to read these images will be explained in more detail in later sections.



(a) (b)

Figure 1.1: *Surface elevation along the y-axis of the surface(a), and it's 1D wavelet transform(b). The red circles outline the a dimple on the surface created by an attached vortex underneath*

The 2D wavelet transform was also performed in the project work, but little could be derived other that the fact that it could differentiate between structures of different length scales. This gave the motivation for this work, where the 2D transform is cultivated further, and more typical turbulent structures are investigated.

1.4 Outline

Chapter 2 gives a general explanation of the theory behind how turbulent motion right under the surface can deform the surface, and presents the typical near surface turbulent structures to be investigated. It also gives an introduction to wavelets and wavelet transforms, with some mathematical definitions. In depth mathematical knowledge of wavelets is unnecessary so only the general formulas are given with some qualitative explanation of how they work. Chapter 3 introduces the free surface Direct Numerical Simulation (DNS) data that was used, and how the velocity field was used to identify the turbulent structures near the surface. This chapter also delves into the MATLAB Wavelet Toolbox, which contains the functions for wavelet transformation that were used. Chapter 4 Then looks at the transforms of the surface elevation and compares them with the turbulent structures that were found using the velocity field. Various results are given and discussed along the way. Chapter 5 concludes the analysis and gives some ideas for future work. Lastly, the Appendix contains the MATLAB scripts.

2 Theory

2.1 Surface deformations and near surface turbulence

The Bernoulli equation in fluid dynamics (Çengel and Cimbala, 2014) states that

$$p_1 + \frac{1}{2}\rho v_1^2 + \rho g h_1 = p_2 + \frac{1}{2}\rho v_2^2 + \rho g h_2, \tag{2.1}$$

along a streamline. Where $p_1,\ p_2$ are static pressures at two points on the streamline, ρ is the density, $v1,\ v2$ are the velocities at the two points; g is the gravitational acceleration and $h_1,\ h_2$ are the heights of the two points relative to a reference.

Let us now look at the free surface of a body of fluid, for example water in the sea or the ocean. We can assume that there is only vacuum above it since air has much smaller density and viscosity. Nonetheless there is an approximately constant pressure above the surface, the atmospheric pressure. Now assume, also, that the stresses are continuous at the surface (Phillips, 1977), this is because discontinuous stresses lead to infinitely large forces, which does not make sense in practice. There are two stresses working at the surface: the pressure and the surface tension. These two must then, in sum, be continuous at the surface. If the fluid at the surface is still or at a constant velocity the surface elevation will be a flat surface. The surface tension works tangential to the surface and is continuous and constant, while the pressure works in the vertical directions and is also continuous from the atmospheric pressure above, and increases with pascals law (Çengel & Cimbala, 2014) the further we descend below the surface.

If there is a velocity fluctuation close to the surface, caused by for example and obstacle or turbulent motion, the local pressure near the surface decreases, relative to the fluid around it. This is because to satisfy the Bernoulli equation 2.1, an increase in velocity causes a decrease in pressure. However, the pressure right above the fluctuation is the constant atmospheric pressure. This atmospheric pressure then presses down on the surface causing a dimple. When the surface is deformed surface tension kicks in to amend that. A stress force is then generated to counteract the pressure difference. The surface tension works only tangential to the surface, so that when it is bent downwards by the pressure difference, there is a component in both the vertical and the horizontal direction. This results in a balancing act between the surface tension created by the deformed surface and the pressure difference across the surface. A deceleration causes a pressure increase and a hump on the surface. A more in-depth mathematical explanation can be found in Phillips, 1977.

With that in mind let us look at some of the structures that are common in near surface turbulence. The structures we look at here are the ones we find in gravity dominated turbulence, as Brocchini and Peregrine, 2001 define it. The surface ripples and elevation changes caused by the subsurface turbulence are relatively small, and at a small scale. On top of waves travelling on the surface we also have vortex dimples and scars. The vortex dimples are circular dips in the surface caused by vertically oriented vortices attached to the surface. When turbulent structures near the surface and are obstructed by it they sometimes break up and their vorticity is redirected so that they attach themselves to the surface causing this dimple. They are also called bathtub vortices as they are similar to the vortices one gets when emptying a bathtub.

The other main feature of interest are what Brocchini and Peregrine, 2001, call scars, or up- and downwellings (Kumar et al., 1998). They have also been called splats and antisplats (Guo and Shen, 2010) or up- and downdraughts (Pan and Banerjee, 1995). The scars are technically the features one observes in the vicinity of up- downwellings, but are the most visible structures. Shen et al., 2004, and Kumar et al., 1998, notice that in channel-like flow hairpin vortices generated by the shear layer from the bottom of the channel move towards the surface causing regions with strong vertical motion. When these updraughts reach the surface they cause a small hump on the surface, as the fluid that is being rushed upwards meets the boundary to the air and is redirected horizontally to the sides. Here we call these upwellings, but they have also been called splats, as, on the surface, they are shaped similar to something splattered on a wall. Due to mass conservation, some of the fluid that is dispersed to the sides at the top of an upwelling goes back down again, making areas with strong downward motion. These generate downwellings on the surface, where the opposite happens, and fluid from the sides is pushed inward and back down below the surface. The vortices that cause these rushes of upward or downward moving fluid may break up and attach themselves to the surface, causing the attached vortices mentioned before. In DNS simulation these up- and downwellings appear with the inclusion of a shear layer under the surface. Shen et al., 1999, and Kumar et al., 1998, mention that the hairpin vortices that cause these upwellings are only present if these is a some kind of shear layer underneath. Pan and Banerjee, 1995, did simulations where they changed the boundary condition away from the surface and noticed that upwellings are found to disappear when a slip boundary condition is used for the wall opposite to the free surface, causing the shear layer to disappear.

These have the clearest impact on the surface divergence, SD. It is calculated by

$$SD = \frac{du}{dx} + \frac{dv}{dy}, \tag{2.2}$$

here u and v are the horizontal velocities at the surface, with x and y as their respective directions. By continuity this has to be equal to $\frac{-dw}{dz}$. At an upwelling the fluid at the surface is pushed strongly to the sides, giving strong positive surface divergence; at downwellings the fluid is pushed strongly inward giving strong negative surface divergence.

The important thing to note is that these turbulent structures are the main cause for heat and gas exchange between the ocean and the air around it, Turney and Banerjee, 2013. These near-surface structures continuously renew the surface keeping the exchange reaction from reaching equilibrium. And, as mentioned in chapter 1, stable long-lived eddies are believed to be the main proponent of this exchange. In some flow conditions by Kumar et al., 1998, they estimated that the upwellings contribute to 51% of the mass transfer rate, and the attached vortices up to 20%. The attached vortices and up- and downwellings occur at a very small scales on the surface (smaller than the turbulent perturbations that cause them, Brocchini and Peregrine, 2001) and are sometimes hard to identify and quantify. Here we look at how a simple computer vision tool, more specifically wavelet analysis, can be used on the surface elevation to find these structures. So what are wavelets, and how are they used?

2.2 Wavelet analysis

Most of the theory on wavelet transformations is taken from a book called "A wavelet tour of signal analysis" by Mallat and Peyre ́, 2008, unless something else is specified. As this book is more mathematical than is necessary for this work, only a simplified version of the main concepts and definitions is given here.

The simplest way to give an idea of wavelets is by addressing their similarities with Fourier transformations. Here, we decompose the signal into its frequencies using a sum of sinusoidal waves. The transform then becomes a function of the frequencies. However, this gives no information on where these frequencies are localised. There is more than one way to amend this. One is to do a Fourier transformation only on a windowed portion of the signal, then moving this windows throughout the signal. This is called windowed Fourier transformations or short-time Fourier transformations. Another is wavelet transformations.

Wavelet transformations started in 1910 when Alfred Haar constructed the function

$$\psi(t) = \begin{cases} 1 & \text{if } 0 \leq t < \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} \leq t < 1 \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

Shown also in Figure 2.1. Dilating and translating this generates a basis for all functions of finite energy. A function of finite energy has a finite squared magnitude meaning

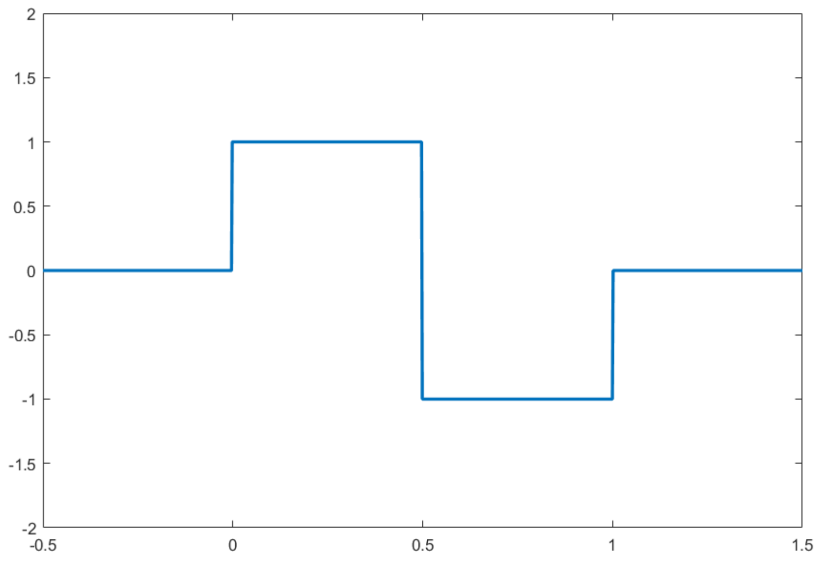


Figure 2.1: *The Haar wavelet.*

$$\int_{-\infty}^{\infty} |f(t)|^2 dt < \infty. \tag{2.4}$$

This is also denoted as $||f||^2 < \infty$.

The function 2.3 was called the Haar wavelet after it's creator. Dilation an translation are the two fundamental operation one can do to a function of one variable. Dilating and translating means a variable shift using two parameters like so

$$\psi_{u,s}(t) = \frac{1}{\sqrt{s}}\psi\left(\frac{t-u}{s}\right), \tag{2.5}$$

where the variable $u \in \mathbb{R}$ is used to translate the function, or wavelet, in the time/space axis, so that the wavelet can be centred for at any value of t. And the variable $s \in \mathbb{R}^+$ is used for the scaling, this term has the effect of squeezing and stretching the whole wavelet. $\psi_{u,s}(t)$ is the resulting wavelet after the manipulation. Scaling and translating the wavelet, and summing over all possible scales and translations, can generate any function of finite energy. The reason for adding the $\frac{1}{\sqrt{s}}$ term to the start is to keep the squared magnitude of the wavelet the same: the original wavelet has a magnitude of 1, so $||\psi||^2 = ||\psi_{u,s}||^2 = 1$. To keep this the case for all possible scaling factors, the term was added.

Similarly to Fourier transformations, we use a convolution to calculate the transform. Defining the transform of a function, $f(t)$, as $W_f(u,s)$, evaluating the integral

$$W_f(u,s) = \int_{-\infty}^{\infty} f(t)\frac{1}{\sqrt{s}}\psi^*\left(\frac{t-u}{s}\right)dt, \tag{2.6}$$

gives the transform. Here ψ^* means the complex conjugate of the wavelet function, since the wavelet and the signal can in general be complex functions. Notice that the transform, $W_f(u,s)$, is a function of two variables, the scaling s and the shifting u. The Haar wavelet in Equation 2.3 is a so called a "mother" wavelet. This mother is then scaled and shifted to get all the other possible variations of the mother needed for the transform, $\psi_{u,s}$ from Equation 2.5. At each value for s and u we have a coefficient, a value that the wavelet transform takes. This tells how prominent the wavelet is at each scale and location. Analogous to how in Fourier transformations the transforms told us how prominent each frequency was.

This is easier to calculate in Fourier space. To do this we use Plancherel's theorem

$$\int_{-\infty}^{\infty} f(t)h^*(t)dt = \frac{1}{2\pi}\int_{-\infty}^{\infty} \widehat{f}(\omega)\widehat{h}^*(\omega)d\omega, \tag{2.7}$$

where $\widehat{f}$ is the Fourier transform of f. This is the general form of the theorem, and accounts for the fact that f and h can be complex valued. Here, in $\widehat{h}^*(\omega)$ means the complex conjugate of the Fourier transform of h. Meaning that, for our wavelet transform integral, we need to evaluate the Fourier transform of $\psi\left(\frac{t-u}{s}\right)$ first. From Fourier analysis we have that (Rottmann, 2019)

$$\mathscr{F}\left(\psi\left(\frac{t-u}{s}\right)\right) = e^{-iu\omega}\mathscr{F}\left(\psi\left(\frac{t}{s}\right)\right) = \widehat{\psi}\left(\frac{\omega}{s}\right)e^{-iu\omega}$$

It's complex conjugate becomes

$$\widehat{\psi}^*\left(\frac{\omega}{s}\right)e^{iu\omega}$$

Plugging this back into Equation 2.6 we get

$$W_f(u,s) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\widehat{f}(\omega)\frac{1}{\sqrt{s}}\widehat{\psi}^*\left(\frac{\omega}{s}\right)e^{iu\omega}d\omega, \tag{2.8}$$

If we look at Equation 2.8 we notice that the variable s is not integrated over, but does contribute to scaling of $\widehat{\psi}^*$. So, for every value of s, Equation 2.8 becomes the inverse Fourier transform of

$$\widehat{f}(\omega)\frac{1}{\sqrt{s}}\widehat{\psi}^*\left(\frac{\omega}{s}\right).$$

The reason for mentioning this is that the functions from the MATLAB Wavelet Toolbox use this to calculate the transforms. See chapter 3 for more on the Wavelet Toolbox.

The inverse is not as straight forward as with Fourier transforms, since the transform is a function of two variables, u, s. First we need to calculate what is defined as the admissibility condition

$$C_\psi = 2\pi\int_0^{\infty}\frac{|\widehat{\psi}(\omega)|^2}{\omega}d\omega, \tag{2.9}$$

$\widehat{\psi}$ is the Fourier transform of ψ, which is why we integrate over ω, usually representing frequency. It is a criterion that $C_\psi < \infty$ for the inverse transform to exist, and is therefore a general criterion for wavelets.

The inverse transform then becomes

$$f(t) = \frac{1}{C_\psi}\int_0^{\infty}\int_{-\infty}^{\infty}W_f(u,s)\frac{1}{\sqrt{s}}\psi\left(\frac{t-u}{s}\right)du\frac{ds}{s^2}. \tag{2.10}$$

How the term C_ψ comes into play for the inverse and more details in general can be found chapters 2 and 4 of "A wavelet tour of signal processing" by Mallat and Peyre´, 2008, or other literature on wavelets. Deep mathematical knowledge on wavelets is unnecessary and the base concepts and definitions given here will suffice going forward.

An example of wavelet transformation is still useful. Figure 2.2 is taken from the book mentioned before, "A wavelet tour of signal analysis" by Mallat and Peyre´, 2008.

The Mexican hat wavelet is used in this example, another mother wavelet. This is defined as

$$\psi(x) = (1-x^2)e^{\frac{-x^2}{2}} \tag{2.11}$$

and can be seen in Figure 2.3.

In Figure 2.2 the vertical axis on the bottom plot is the logarithm of the scale, $\log_2(s)$, while the horizontal axis is the shifting variable, u. u matches the variable used in the signal t. We can think of the transform as the fitting between the wavelet and the original signal. For every value of the scaling s we have a more contracted or dilated wavelet. This is then convolved with the signal to see where there is the most overlap between the wavelet and the signal. Black areas in Figure 2.2 represent places where the value of the fitting is positive, in grey areas there is little overlap and the values are close to 0, and in the white areas the value for overlap is negative. Since the transform is a function of two variables, s and u,

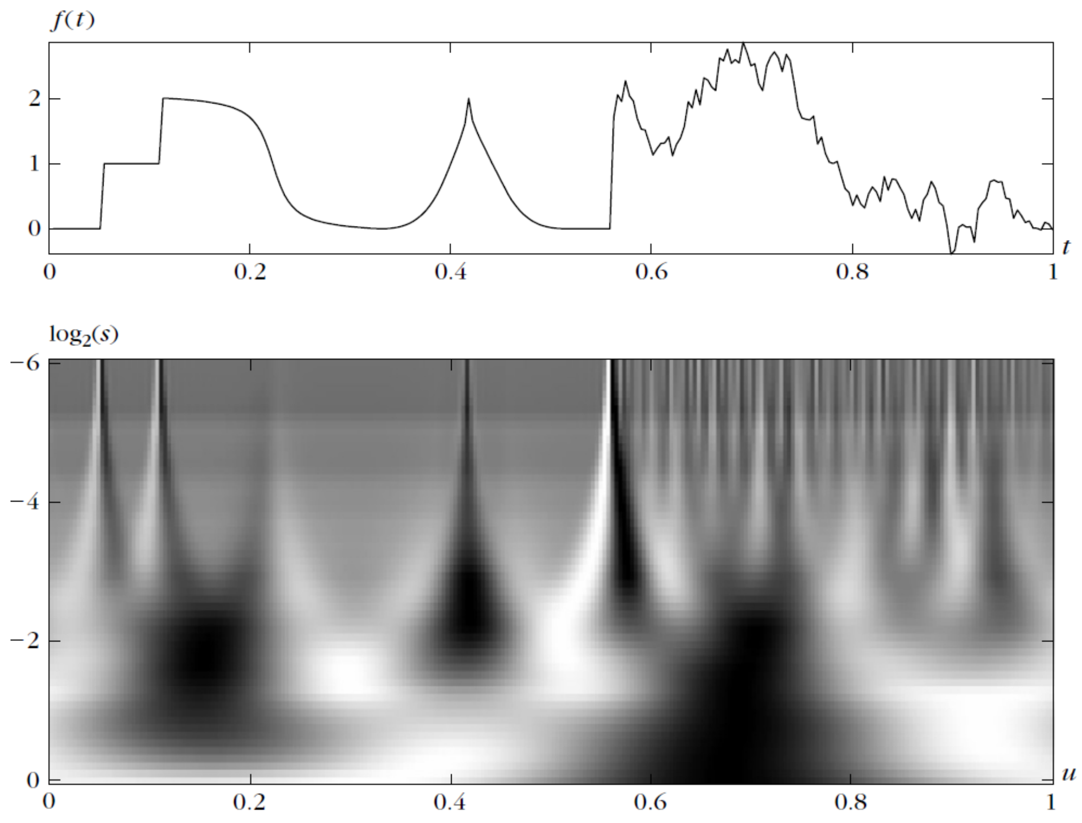


Figure 2.2: *Example from "A wavelet tour of signal processing" by Mallat and Peyre ́, 2008, page 104.*

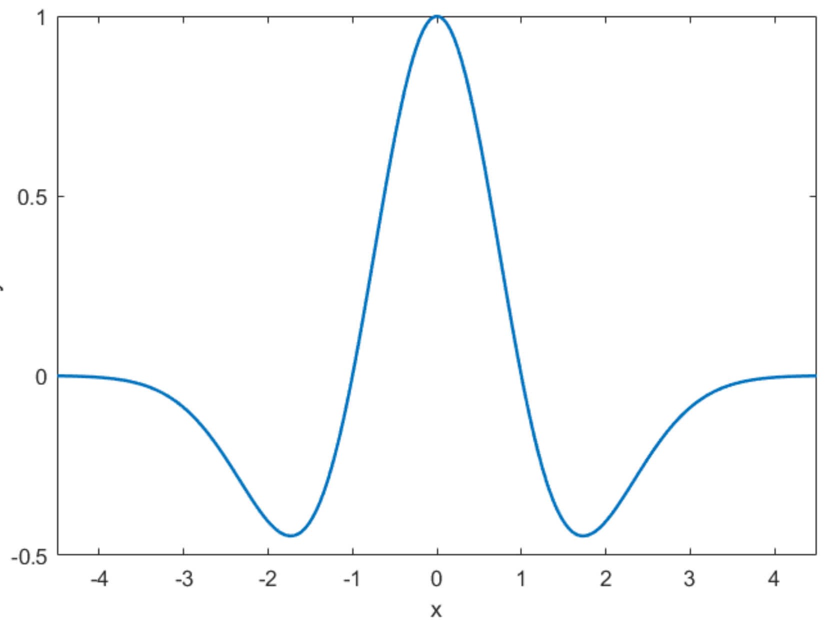


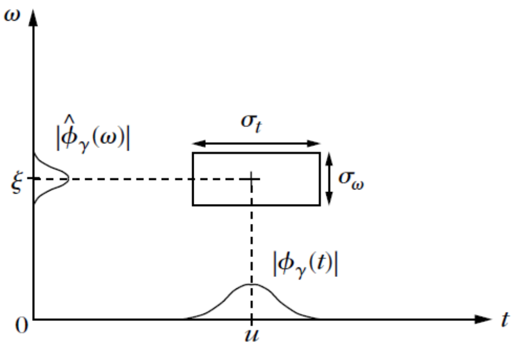
Figure 2.3: *The Mexican hat wavelet.*

it has 1 more dimension than the original signal; in the bottom plot of Figure 2.2 we are therefore looking at a surface in 3D from the top down.

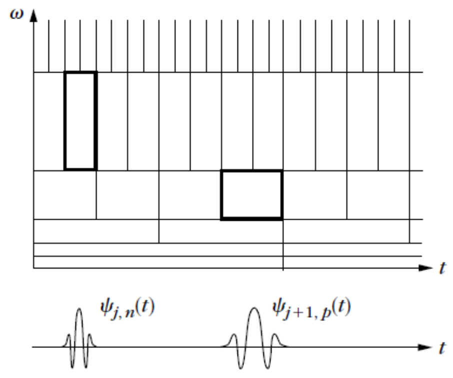
Looking at the signal, $f(t)$, we see that for $u = t \in [0.6, 1]$ there is a lot of noise on top of a general signal. The noise is captured in the transform by the structures we see at the lower scales (top right of the bottom plot of Figure 2.2, $\log_2(s) \leq -4, u \geq 0.6$), while the general signal is captured at the larger scales ($\log_2(s) \geq -3$).

Looking at the transform we notice that some of the structures are quite blurry and smeared out. The jumps at $t \approx 0.05$ and 0.1 are very localised phenomena in the signal, but in the transform it is apparent that they give values for the transform also if we move slightly away from them, especially for larger scales. This makes it hard to judge where some structures in the signal are localised sometimes. This is especially hard when there are various structures close to each other.

This is similar to a famous idea in quantum mechanics, certain pairs of variables cannot be predicted exactly. This is called the Heisenberg's uncertainty principle, or just the uncertainty principle (Sen, 2014). In this case, the variables that can't be predicted exactly are the time, and the scale, which is analogous the the frequency in Fourier transforms. For every signal one can either know exactly it's time variation or it's frequency spectrum. It's impossible to know where each frequency is localised in the time axis, or which frequencies are present at a certain time. In wavelet transformations one works with scales instead of frequencies but the problem remains the same. Predictions of both the time/space and the scale cause uncertainty in both. Meaning that to get an idea of where a frequency is localised, we have to give up some accuracy in exactly which scale we are looking at, and where exactly it is localised. This is also the case for other time-frequency transformations, like the windowed Fourier transform, mentioned at the start of this section.



(a) *Time-frequency box, also called Heisenberg box*



(b) *Heisenberg boxes in wavelet transforms.*

Figure 2.4: *Time-frequency boxes, also called Heisenberg boxes. Figures from "A Wavelet Tour of Signal Processing" by Mallat and Peyre´, 2008, pages 15 and 20.*

Let's look at this graphically. Figure 2.4a shows a box in the time-frequency plane (for wavelets it's the time-scale plane). Predicting ϕ in both time and frequency causes uncertainties. If these uncertainties are set to be the sides of a rectangle in this grid, we get what is called a Heisenberg box. This box is mathematically restricted to have a area larger than $\frac{1}{2}$. This is the resolution for the transform, a kind of "pixel". Inside this box the values are uncertain, but we know that they are most likely somewhere inside of this box. If we want to know the exact signal in the time domain we can reduce the time uncertainty and increase the frequency uncertainty. When there is no uncertainty in time, we need infinite uncertainty in frequency for this box to have an area larger than $\frac{1}{2}$, so we have no information about the frequencies in the signal. The same is the case when there is no uncertainty in frequency. Proof is in chapter 2 of "A wavelet tour of signal processing" by Mallat and Peyre´.

In wavelet transformations the uncertainties are dependent on the scale we are looking at. These Heisenberg boxes don't need the to have the same sides on the whole time-frequency plane. In wavelet transformations this is not the case, and the boxes are arranged such that for smaller scales there is little uncertainty in time but large uncertainty in frequency; these boxes can be seen as tall and thin in our time-frequency plane. But for larger scale there is little uncertainty in frequency but large uncertainty in time and can be seen as short and wide. These boxes can be seen in Figure 2.4b. Here the y-axis is the frequency not the scale, however, these are analogous: small scales mean large frequencies (narrow wavelets), and large scales mean small frequencies (wide wavelets).

The figure shows how for every scale range the box is half the width but twice the length of the boxes in the scale range below it, keeping the area of the box the same, but varying the uncertainties depending on the scale. This means that for large scales (or low frequencies) the transform has good scale resolution as the uncertainty in the scale is small. However, it has bad time resolution, with large uncertainty in the time axis. For small scales (or high frequencies) the opposite is true. The transform has then good time resolution, but bad scale (frequency) resolution.

2.2.1 2D Wavelet transforms

When transforming a two-dimensional surface, an extension of the normal 1D wavelet transform is used. As the book by Mallat and Peyre´, 2008, does not contain much about 2D wavelet transforms, the information given in this section is mostly taken from Antoine et al., 2008, and Wang and Lu, 2010.

For 1D wavelets we used the two transformations on a function of one variable: translation and dilation. On the plane, however, there are three fundamental transformations: translation, dilation and rotation. We then need three variable shifts of the mother wavelet $\psi(\vec{x})$ in 2D wavelet transformations

$$\begin{aligned} \text{Translation: } T^{\vec{u}}\psi &= \psi(\vec{x}-\vec{u}), \quad \vec{u} \in \mathbb{R}^2, \\ \text{Dilation: } D^{s}\psi &= \frac{1}{s}\psi\left(\frac{\vec{x}}{s}\right), \quad s \in \mathbb{R}^+, \\ \text{Rotation: } R^{\theta}\psi &= \psi(r_{-\theta}\vec{x}), \quad \theta \in [0, 2\pi); \end{aligned} \tag{2.12}$$

here there is a vector $\vec{u}$, instead of the scalar in 1D wavelet transforms, since there are two directions to translate in. The scaling is done similarly, but the term in front is now $\frac{1}{s}$ instead of $\frac{1}{\sqrt{s}}$ to keep the squared magnitude of the function the same. The $r_{-\theta}$ term is the rotational matrix in two dimensions (Lay et al., 2016):

$$r_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \tag{2.13}$$

The terms $T^{\vec{u}}, D^s, R^\theta$ are then operators for translation, dilation and rotation on the plane respectively.

Transforming the wavelet to include all of these variable shifts gives

$$\psi_{\vec{u},s,\theta} = \frac{1}{s}\psi\left[r_{-\theta}\left(\frac{\vec{x}-\vec{u}}{s}\right)\right]. \tag{2.14}$$

Having said that, there is a simplification that can be done when working with isotropic wavelets. Isotropic means that the functions is the same no matter the orientation of the axes, it is rotation invariant and the rotation term has no effect. This simplifies the variable shift to

$$\psi_{\vec{u},s} = \frac{1}{s}\psi\left(\frac{\vec{x}-\vec{u}}{s}\right). \tag{2.15}$$

The 2D wavelet transformation then becomes a simple extension of the 1D transformation

$$W_f(\vec{u},s) = \int_{R^+}\int_{\mathbb{R}^2} f(\vec{t})\frac{1}{s}\overline{\psi}\left[r_{-\theta}\left(\frac{\vec{t}-\vec{u}}{s}\right)\right]d\vec{t}, \tag{2.16}$$

or in Fourier space

$$W_f(\vec{u},s) = \int_{\mathbb{R}^2} \widehat{f}(\vec{\omega})\frac{1}{s}\widehat{\overline{\psi}}\left[r_{-\theta}\left(\frac{\vec{\omega}}{s}\right)\right]e^{i\vec{u}\vec{\omega}}d\vec{\omega}.$$

The inverse is also a natural extension of the 1D transform, first we need the admissibility condition

$$C_\psi = (2\pi)^2\int_{\mathbb{R}^2}\frac{|\widehat{\psi}(\vec{\omega})|^2}{\vec{\omega}^2}d\vec{\omega}, \tag{2.17}$$

Then the inverse becomes

$$f(\vec{x}) = \frac{1}{C_\psi} \int_{-\pi}^{\pi} \int_{\mathbb{R}^+} \int_{\mathbb{R}^2} W_f(\vec{u}, s) \frac{1}{s} \psi \left[r_{-\theta} \left(\frac{\vec{x} - \vec{u}}{s} \right) \right] d\vec{u} \frac{ds}{s^2} d\theta. \tag{2.18}$$

We see that the original signal now has two variables, $\vec{t} = [t_1, t_2]$, while the transform has 4: $\vec{u} = [u_1, u_2]$, s and θ, for isotropic wavelets it's three as the angle θ is omitted. This means that visualising the whole 2D wavelet transform is quite challenging. It is therefore common to only inspect a certain scale and rotation angle at a time. See chapter 4 for how the surface transforms are presented.

3 Methods

3.1 DNS data

The data is based on a Direct Numerical Simulation (DNS) of the free surface with turbulence underneath. The simulation was executed by Guo and Shen at the University of Minnesota. Figure 3.1 illustrates the simulation setup they used. The dimensions of the simulation domain are $L_x \times L_y \times \overline{H} = 2\pi \times 2\pi \times 5\pi$. The grid is divided uniformly such that the number of cells is $N_x = 256$, $N_y = 256$ and $Nz = 660$ in the x, y and z directions respectively. In the region described as the "bulk region" in Figure 3.1, a forcing term was used to stir the flow and generate turbulence. In the "damping region" this forcing was gradually reduced. The turbulence in these regions was deemed to be isotropic. Close to the free surface, in the region label "free region", there was no induced forcing, so the turbulence naturally advected to this region from the "bulk region". The Reynolds number in the simulation was 2500, and the Froude number was 0.1. The surface tension was omitted in this simulation.

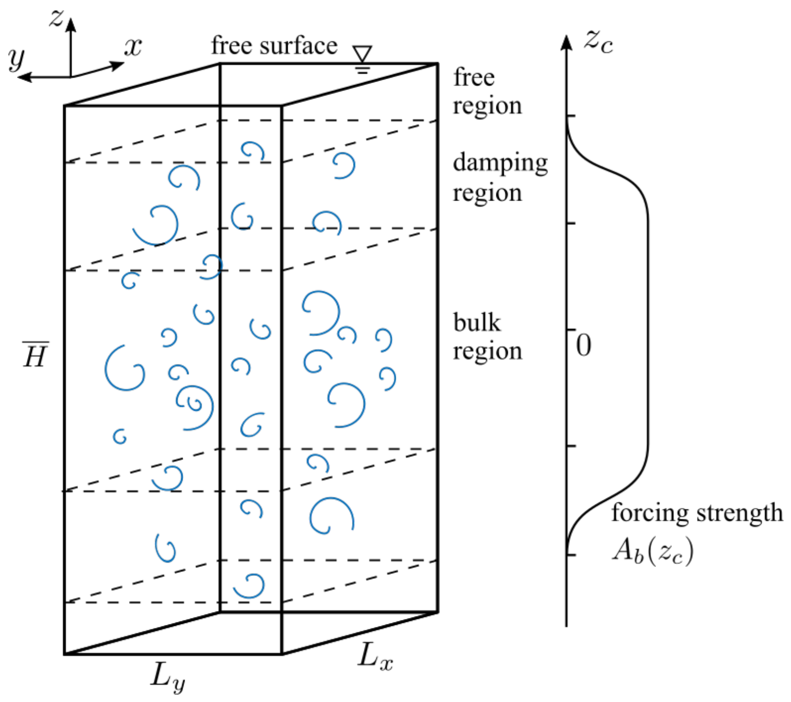


Figure 3.1: *The setup used in the simulation by Guo and Shen, 2010. They constructed this and equivalent figure is shown in the article from 2010.*

The data was given as `hdf5` files, which is a hierarchical data format. This data format saves the variables as a struct in a file. There is a struct for each timestep. It included various variables needed in the DNS. The surface elevation, and the velocities and pressure near the surface were extracted from the `hdf5` files using the build-in MATLAB functions `h5info` and `h5read`, (*h5info*, 2011, *h5read*, 2011). The scripts are included in the Appendix.

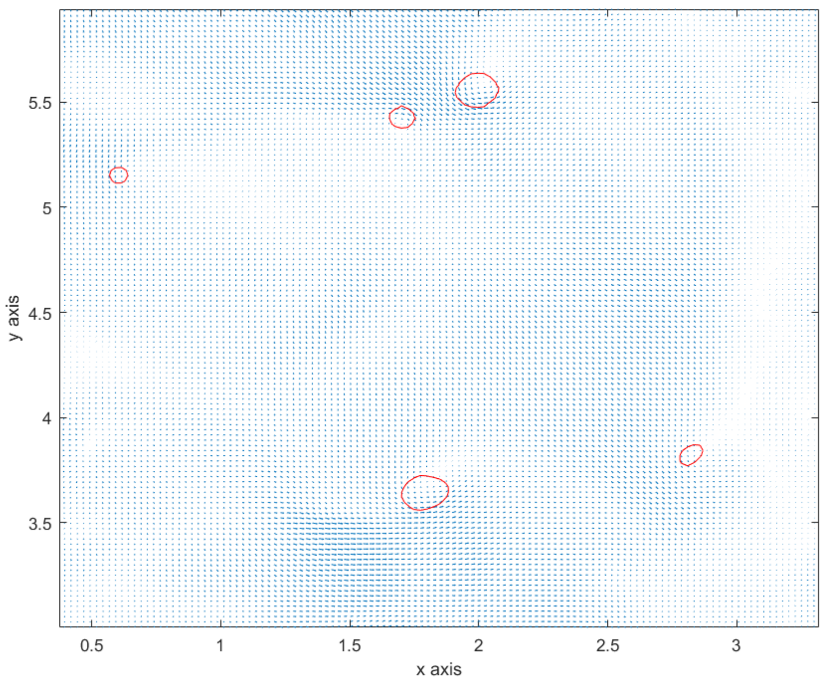


Figure 3.2: *Vortex cores at the surface given by the algorithm by Jeong and Hussain, 1995, the vortex cores are given by the red outlines.*

Since the whole velocity spectrum was known, it could be used to find the actual location of the various structures close to the surface. As mentioned in chapter 2 there are two main structures of interest, the attached vortices and the up- and downwellings. The vortices were found using the algorithm presented by Jeong and Hussain, 1995. This method finds the centres of vortices, and was used on the velocity field at the surface. If there is a vortex core of a certain strength at the surface it is then a attached vortex. Non-attached vortices are aligned horizontally, along the surface and their cores don't appear if we only look at the surface velocity (see Brocchini and Peregrine, 2001, Kumar et al., 1998).

Figure 3.2 gives an example of the method by Jeong and Hussain, 1995 on one of the timesteps in the DNS data. Here the red outlines are the vortices found by the method. As mention in chapter 2: Theory the attached vortices cause small circular dimples on the surface, looking at the surface and imposing these outlines, wee see which dimples are caused by an attached vortex. This can be seen in Figure 3.3.

For the up- and donwwellings, Guo and Shen give a method themselves. This is most thoroughly described in Shen et al., 2004, and is based on the surface divergence. They called the method the Variable Interval Space Averaging (VISA) method. The idea is to look at where the surface divergence changes the most. On top of the welling, the surface divergence is very large, but diminishes very quickly away from it. The variance of the surface divergence is calculated on a small interval around each point. If the variance is larger than a certain threshold, that point is part of an up- or downwelling, depending on whether the divergence is positive or negative at that point.

Since the variance is calculated in a small window around a point let us call the half side-length of this window W. The value of W should be around the width of these wellings. This length scale is hard to define since it is not the same as the Taylor microscale of the turbulent structures that cause these upwellings, as one would think, but much smaller (see Brocchini and Peregrine, 2001). No literature was found to estimate this length scale directly, so the value used for W came from qualitative observation of the structures on the surface on from comparison to the length scales of the wavelets used in the transformation. It was found that $W \approx 0.22$, captures the structures quite well.

So to use the VISA (Variable Interval Space Averaging) method, we need to compute what Shen et al.

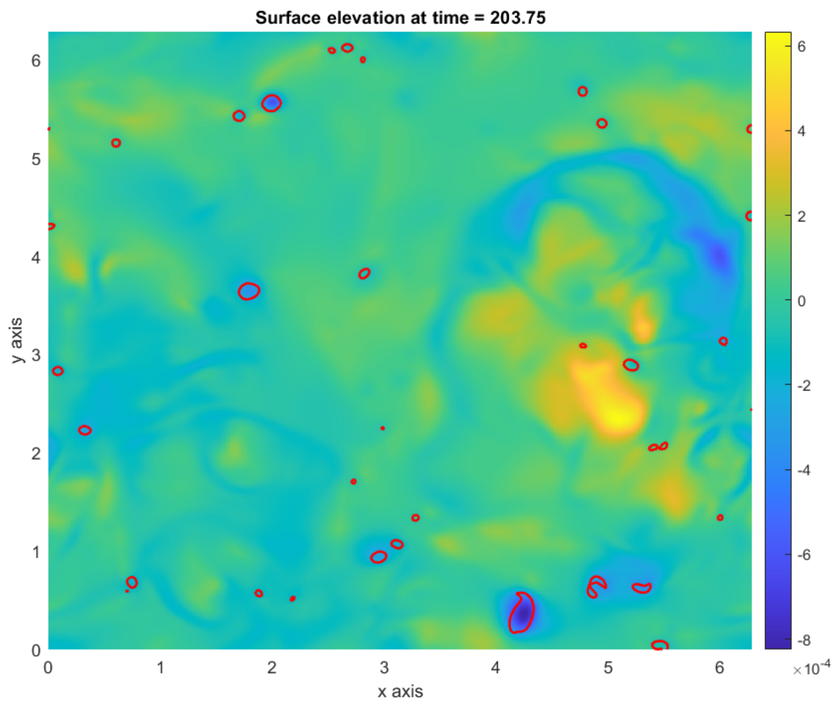


Figure 3.3: *Surface elevation with the vortex core given by the algorithm by Jeong and Hussain, 1995, the vortex cores are the red outlines.*

call a variable interval space-averaging quantity:

$$\underline{SD}(x, y, t, W) = \frac{1}{4W^2} \int_{x-W}^{x+W} \int_{y-W}^{y+W} SD(\xi, \eta, t) d\xi d\eta. \tag{3.1}$$

The local variance inside this window then becomes

$$SD_{var}(x, y, t, W) = SD^2(x, y, t) - \underline{SD}^2(x, y, t, W). \tag{3.2}$$

If at some point $(x, y), SD_{var}(x, y) > 5\, SD^2_{rms}$ (here SD_{rms} is the root mean square of the surface divergence) then it was deemed that this point is part of an up- or downwelling, depending on the sign of SD.

Figure 3.4 shows an example of the use of this method on one of the timesteps. As mentioned the upwellings are regions with large surface divergence, and the downwellings have small surface divergence. We see that the outlines in red and green capture these areas. Here the red outlines are the upwellings, and the green are downwellings.

3.2 The MATLAB Wavelet Toolbox

To perform the wavelet transformation the MATLAB Wavelet Toolbox was used. The source code for these functions is sometimes hard to find, but it was assumed that MATLAB would have better wavelet transformation functions that any that could be manually generated in the span of the work. Nonetheless, some signals in MATLAB were transformed and transformed back for comparison and testing.

Figure 3.5 and Figure 3.6 show examples of two built-in MATLAB data sets called “mtlb” and “noisdopp” significantly. Figure 3.5b and Figure 3.6b are the wavelet transforms, similar to the example in Figure 2.2. To get the 1D wavelet transforms in MATLAB, the function `wt` was used (*wt*, 2018). But this needed information about which wavelet to use and the scales in the transformation as a second input variable.

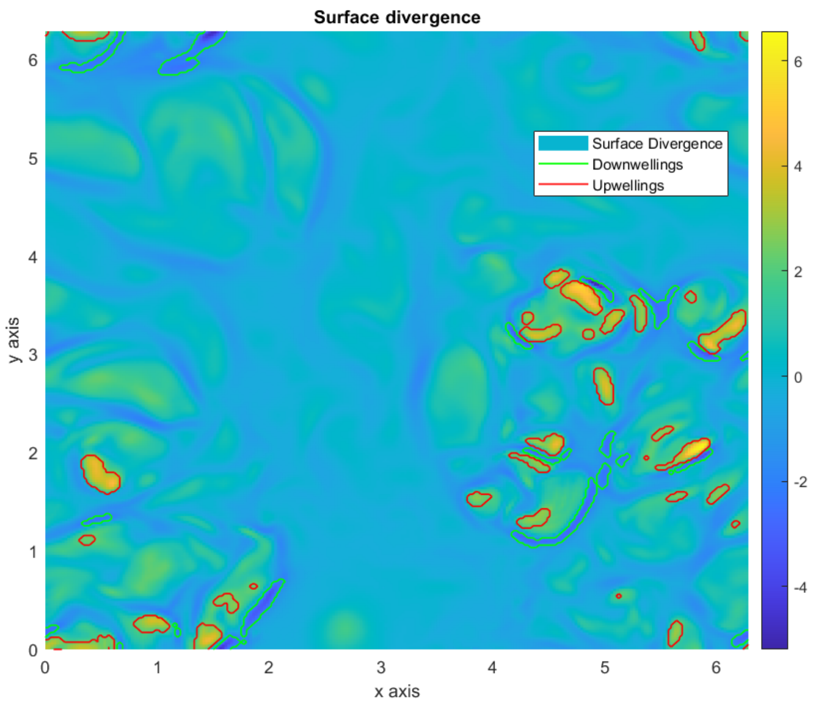


Figure 3.4: *Sample of the surface divergence at a timestep in the DNS data. The green outlines enclose a downwelling, while the red enclose an upwelling.*

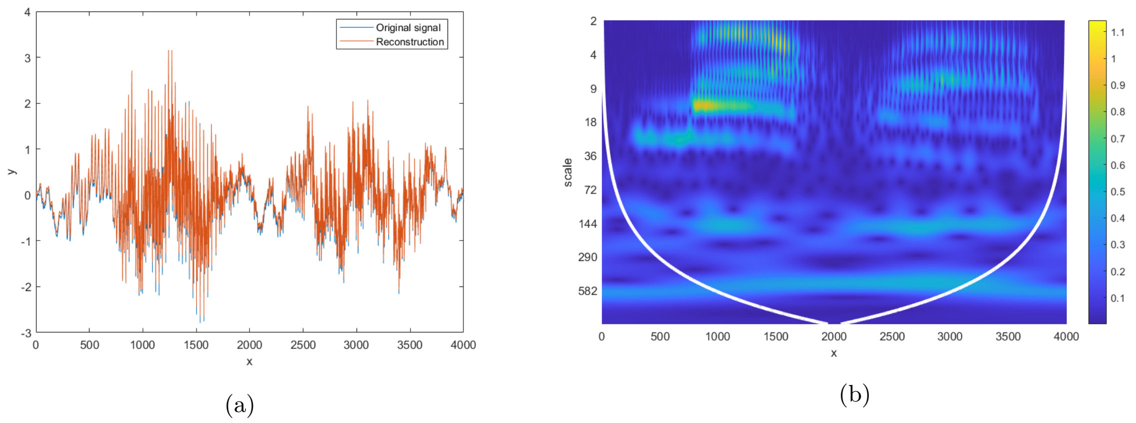
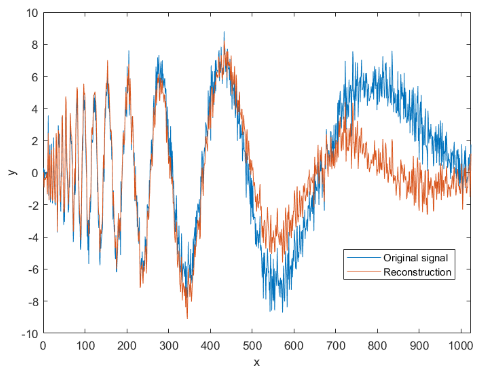


Figure 3.5: *Wavelet transform of the signal "mtlb" in MATLAB and comparison between original and inverse.*

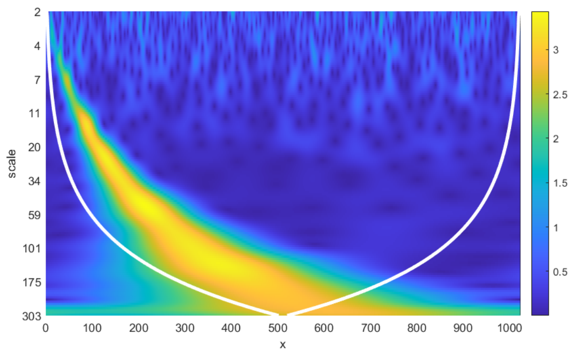
These were constructed using the function `cwtfilterbank` (*cwtfilterbank*, 2018). The Mexican hat wavelet was used in the example in Figure 2.2, but the Morlet wavelet, shown in Figure 3.7, was used for these examples.

The vertical axes are the scales in logarithmic spacing, and the horizontal the localisation, although another colour scheme was used and a colour bar was added to show the value of the wavelet coefficients (the coefficients are again the values for the fitting at each scale and localisation).

The signals were also reconstructed using the inverse wavelet transform function `icwt` (*icwt*, 2016) and we see that the "mtlb" signal had a good fit but "noisdopp" had some mismatch, as can be seen in Figure 3.6a. This is mainly at the end where the main oscillation in the signal is very wide, meaning that a wavelet of a very large scale is needed to capture this part of the signal. However, the transform goes up to a scale value of around 303, and it might be the case that a larger scale is needed to capture this



(a)



(b)

Figure 3.6: *Wavelet transform of the signal "noisdopp" in MATLAB and comparison between original and inverse.*

wave. Unfortunately, the scale of a wavelet of a signal can only be so large, in MATLAB this cutoff is such that the wavelet spans the whole signal at the largest scale. Meaning that a larger scale would involve a wavelet wider than the signal. As expected only including all possible scales correctly describes the signal, even though that would require an infinitely long signal (so that the scales can go to infinity). We see that since the "mtlb" has more data points, the largest scale present there is close to 1000.

Also note the white lines in the wavelet transform plots. They define what MATLAB calls the cone of influence (*wt*, 2018). All the values underneath these white lines represent areas where the wavelet, centred at that point, was partially beyond the signal. This meant that the coefficients at these scales and localisation cannot be fully trusted, as the wavelet tries to describe something that is beyond the signal. At the smallest scales, the wavelet is very squeezed and can go close to the boundary of the signal, without leaving the span of the signal. In contrary, for the larges scales (bottom of the plot) these lines almost seem to meet at the centre, meaning that the wavelet is so stretched out at this scale, that only when it is at the centre, it does not go outside the signal. Which means that The wavelet here is the same length as the signal. This is the cutoff scale mentioned in the previous paragraph.

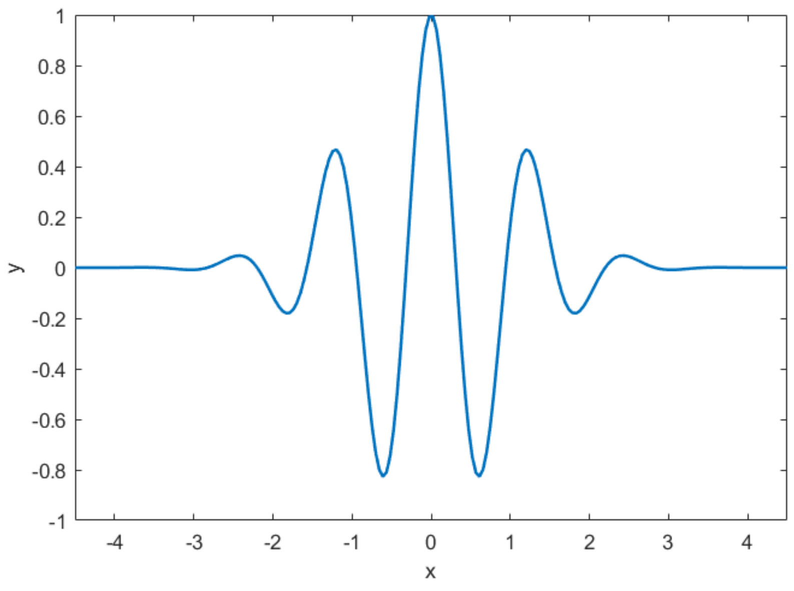


Figure 3.7: *The Morlet wavelet.*

Figure 3.7 shows the real part of the Morlet wavelet, since it is in general a complex valued function. This also means that the transform becomes complex, Figure 3.5b and Figure 3.6b show the absolute values of the wavelet coefficients. Since the wavelet toolbox is mostly used for signal processing the values on the vertical axis are returned as frequencies in MATLAB, not the wavelet scales. As mentions there is a very close relationship between scale s and frequency f in wavelet transform, they are inverse proportionate such that $f = \frac{c}{s} \implies s = \frac{c}{f}$ where c is a constant that depends on the wavelet. There is a function in MATLAB that interchanges between them depending on the wavelet called `scal2frq` which was used to find the scales in the figures (*scal2frq*, 2006).

For the two dimensional analysis of the whole surface, `cwtft2` calculates 2D wavelet transforms (*cwtft2*, 2013). As mentioned this function first transforms the signal with 2D fast Fourier transforms, then multiplies it with the Fourier transform of the wavelet at the desired scale to find the convolution. Here the 2D Mexican hat wavelet from Figure 3.8 was mostly used. This was also tested, but a function for inverting 2D wavelet transforms in MATLAB could not be found.

The two dimensional version of the Mexican hat wavelet is given by the function

$$\psi(\vec{x}) = (2 - |\vec{x}|^2)e^{-|\vec{x}|^2/2} = (2 - x_1^2 - x_2^2)e^{-(x_1^2+x_2^2)/2} \tag{3.3}$$

and can be seen in Figure 3.8.

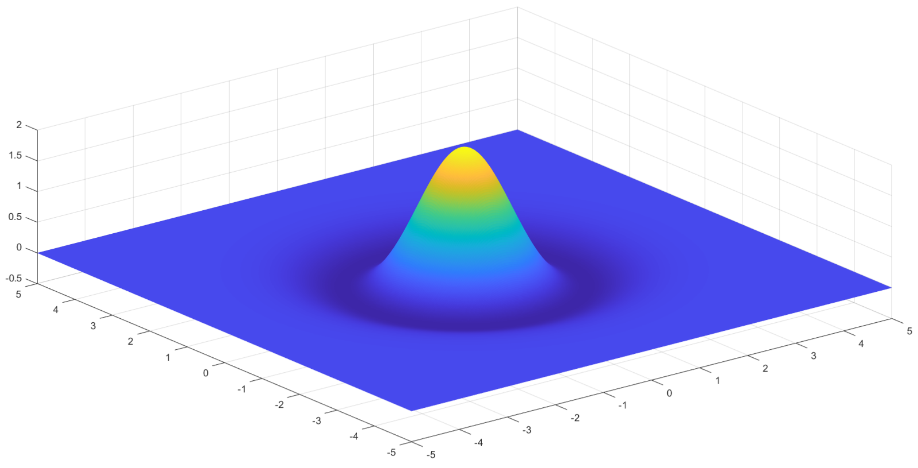


Figure 3.8: *The 2D Mexican hat wavelet.*

The functions `cwtft2` can be used with a variety of inputs, and one has a lot of choices in costuming the transform. Other than the input signal to be transformed, one can specify the wavelet used in the transformations. The function `cwtftinfo2` has a list of all the possible wavelets, and which parameters can be changed (*cwtftinfo2*, 2013). For this work, the Mexican hat was used in general. The wavelet in Figure 3.8 is isotropic in its basic version, but the parameters for the wavelets can be changes to stretch the wavelet in one direction or the other. The wavelet becomes elongated and is therefore no longer isotropic, this version of the wavelet proved very useful in some cases, as can be seen in Results. The function defaults to the Morlet wavelet if nothing is specified for the wavelet.

Another input parameter is the scale range for the transformation. For 1D transformation all plausible scales are used automatically based on the length of the signal, but for 2D transformations, not all scales are necessarily of interest. As before there are some conditions. The smallest scale has to be larger than one. There is not a largest scale condition built in, however the same problem arises if the scale is too large; the wavelet goes beyond the boundary of the signal. The documentation does not mention this, however the smallest scales is probably chosen to be the smallest amount of data points, but still retain the form of the original wavelet. For most of the work it was found that the smallest scales were preferable for this data set. The wavelet at this scale was reconstructed by looking into the code for `cwtft2` giving what we see in Figure 3.9. It looks jagged because at the smallest scale it is only a few data-points wide.

The last input parameter of interest is the set of angles. These are the angles used to rotate the wavelet relative to the signal. These are only used with anisotropic wavelets.

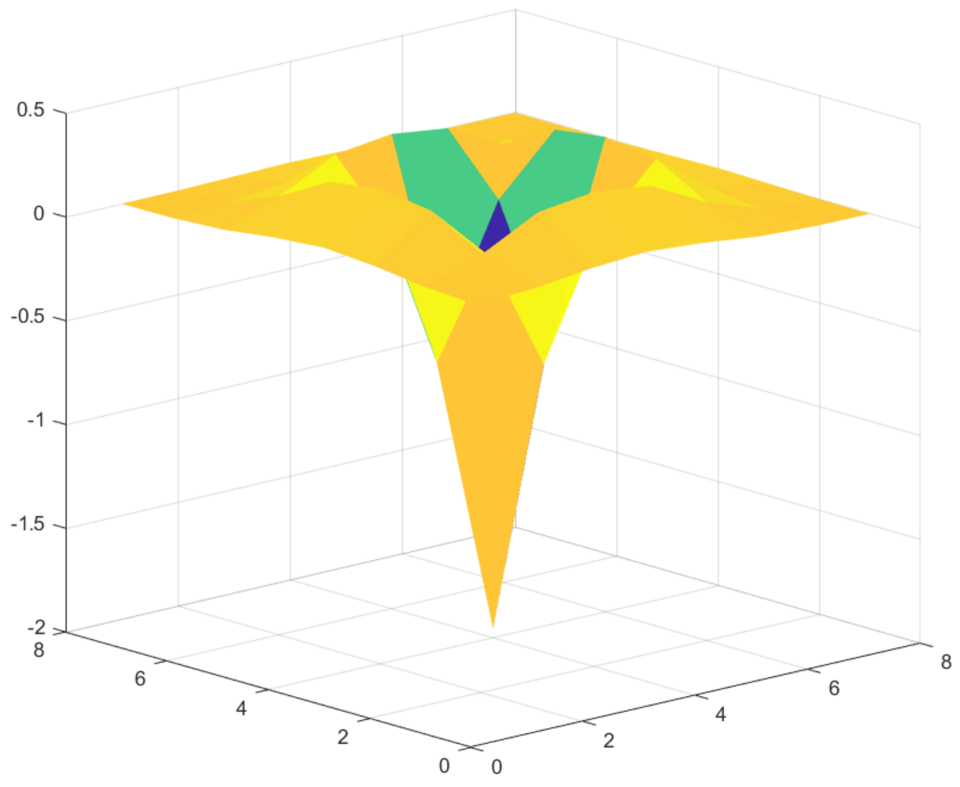


Figure 3.9: *Mexican hat wavelet at the smallest scales in MATLAB*

Most of the MATLAB scripts can be found in the chapter 5, even the ones used for just testing the wavelets, if they are of interest.

4 Results

4.1 First look at the transforms

Let us now look at some transforms of the surface elevation to get an idea of what responses we get and how they can be of use.

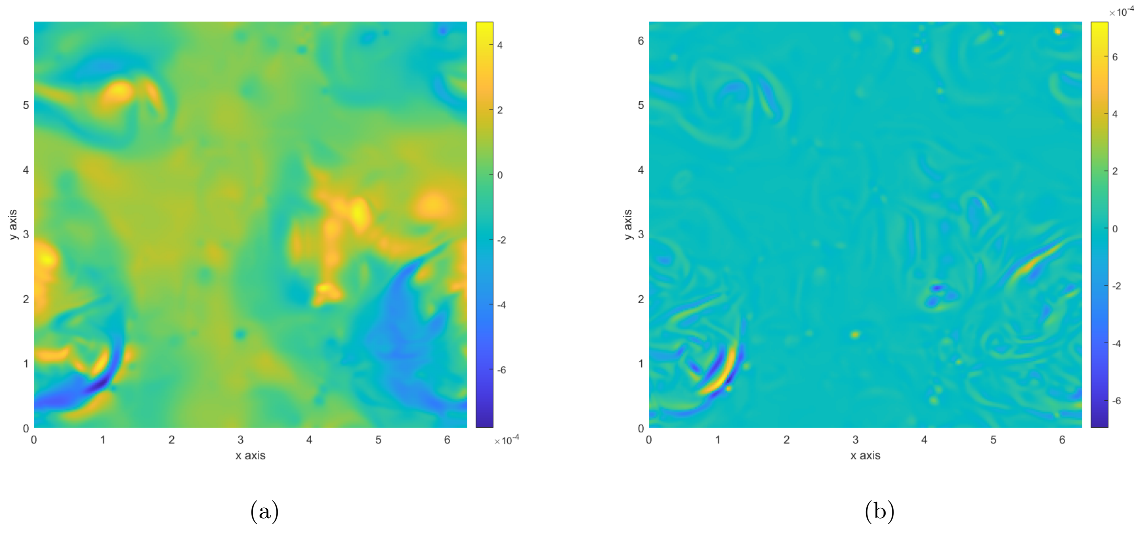


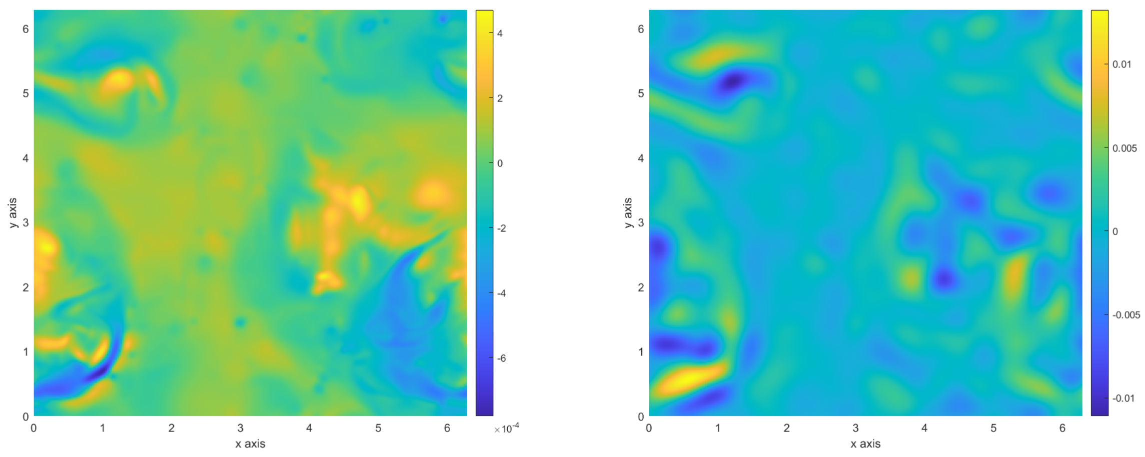
Figure 4.1: *Surface elevation at timestep 196.25 and it's wavelet transform with the Mexican hat at the smallest scale.*

Figure 4.1 shows the surface elevation at one of the timesteps in the DNS data on the left and it's wavelet transformation for the smallest scale on the right. The Mexican hat wavelet was used here (not the stretched version, but the unchanged wavelet). Since this wavelet is isotropic the angle is irrelevant.

We see how small or thin structures in the surface elevations give strong responses in the wavelet transform. However, distortions of surface at larges scales, for example in the middle of Figure 4.1a, give no response in wavelet space (the areas in light blue in Figure 4.1b are values close to 0).

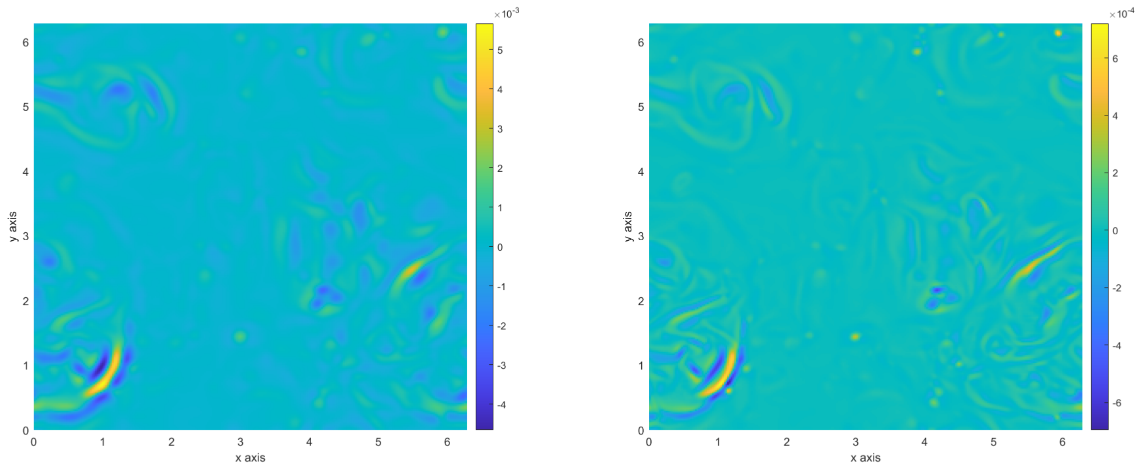
If we look at a larger scale for the Mexican hat we get Figure 4.2. Here the shapes are more diffused and the small or thin structures on the surface are not as apparent.

Since the structures we are looking for are quite small, using the small scale transform is preferable. As mentioned in section 2.2, at small scales the transform has high uncertainty in the scale. So the exact scale used is not important. In fact, for the scales from 1 to around 2.5 the transforms are really similar, and the structures become only slightly diffused. Figure 4.3 shows the transform the smallest scale of 1 (4.3b) and 2.6 (4.3a).



(a) (b)

Figure 4.2: *Surface elevation at timestep 196.25 and it's wavelet transform with the Mexican hat at the smallest scale*



(a) (b)

Figure 4.3: *Wavelet transform of the surface for the scales (a) 2.6, (b) 1*

4.2 Response from attached vortices

Let us then compare the wavelet transform for the smallest scale with the location of the various surface structures and see what we can find. The wavelet at the smallest scale was reconstructed in Figure 3.9. The top of the "hat" is pointed downwards, meaning that, when convolving with the surface elevation, we get positive positive values of the transform when the surface is bent downwards, and negative values when it is bent upwards. This means that the dimples from the attached vortices will give positive values in the wavelet transform. Now if we look at Figure 4.4, we have again included the outlines for the attached vortices. We see that, in general, these outlined regions coincide with where the transform is locally positive.

The colour scaling bar on the right side tells us that the coefficients of the wavelet transforms are between $-6.5 \cdot 10^{-4}$ and $6.5 \cdot 10^{-4}$. If we only look at the positive values for the wavelet, larger than $0.4 \cdot 10^{-4}$, we get what we can see in Figure 4.5. The yellow blobs are the areas in the transform larger than the threshold and the red outlines are, as before, the attached vortices given by the velocity field at the surface. We see that the dimples made by the attached vortices give strong responses in the wavelet transform. There are also some other structures appearing. These may be dips in the surface from up- or downwellings or other

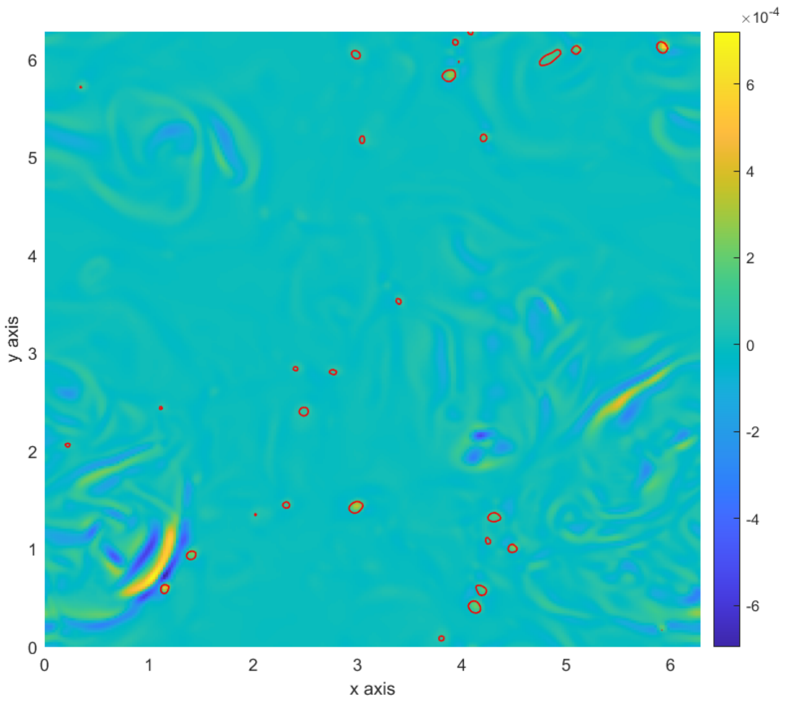


Figure 4.4: *Wavelet transform at the smallest scale, the red outlines represent the location of the attached vortices.*

structures on the surface.

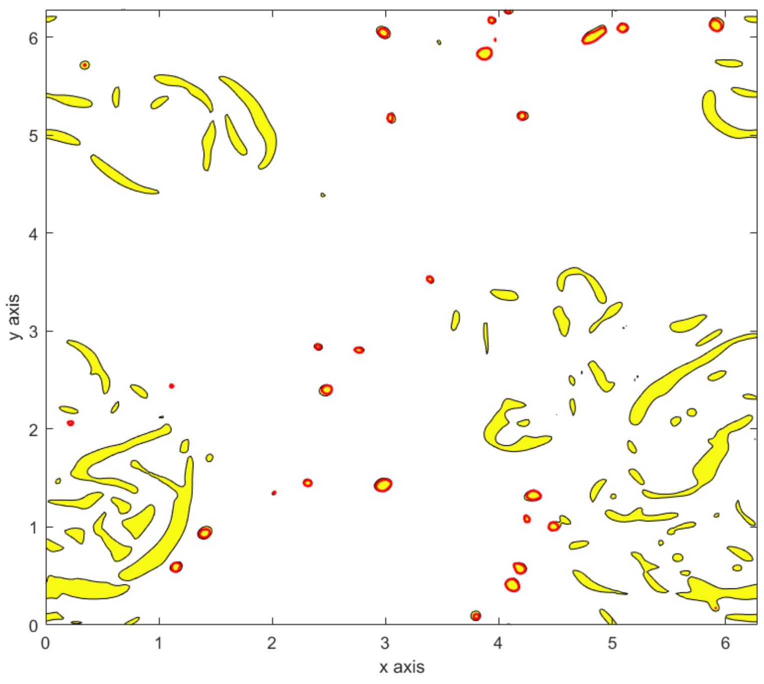


Figure 4.5: *Wavelet transform at the smallest scale. Only the positive value above the threshold of* $0.4 \cdot 10^{-4}$ *are shown. The red outlines represent the location of the attached vortices.*

From this point we try to look into two questions that come to mind when looking at Figure 4.5. One is if there is a way to isolate the dips on the surface caused by surface dimples. The other if the deformations on the surface are proportional to the strengths of the attached vortices, i.e. if the stronger the vortex, the more the deformed the surface is. This would mean that the wavelet transform may have information on the strength of the attached vortex causing the dimple it is detecting. Intuitively, the stronger the vortex the bigger the velocity change in it, and the more dipped the surface. But how does this translate in wavelet space?

4.2.1 On attached vortex strength

Getting an estimate of a vortex' actual strength from the velocity field was done in a simple manner. Using the areas designated as vortex cores by the algorithm mentioned in chapter 3, we integrate the enstrophy $\mathcal{E}$ using the velocity field at the surface. Since the attached vortices are mainly directed vertically, one could make an argument that only the vertical vorticity ω_z is required to make an estimate, but right under the surface the vortices are not necessarily vertically oriented. The enstrophy $\mathcal{E} = \omega_x^2 + \omega_y^2 + \omega_z^2$ was therefore used instead.

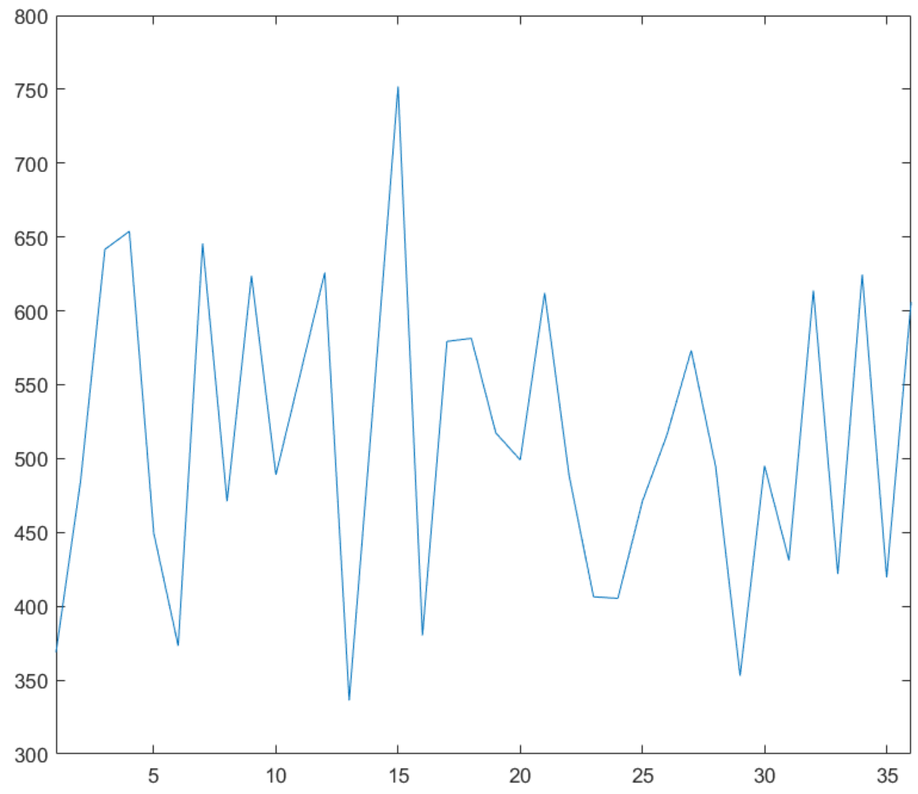


Figure 4.6: *Ratio between the integrated enstrophy and the wavelet transform over some observed attached vortices*

The vortices used in the comparison where chosen by looking at the ones detected by the vortex core algorithm. To see if the transform has information on the strength of the vortex, we integrate the transform on an area enclosing the vortex. The ratio between the two was then calculated and can be seen in Figure 4.6. Since the enstrophy is calculated using the derivatives of the velocity fields and the wavelet transform from the surface elevation, they are two orders of magnitude from each other. Even so, we are not interested in their values being close to each other, but that their ratio is consistent. It would mean that a strong vortex implies a strong dimple on the surface and a strong response in the transform. Looking at Figure 4.6 we see that their ratio varies between 350 and 750 for the 36 vortices that were identified and used in this comparison. This implies some dependency, and this assumption is fortified when instead of choosing the location of known attached vortices we look at random points in the same timesteps as these vortices.

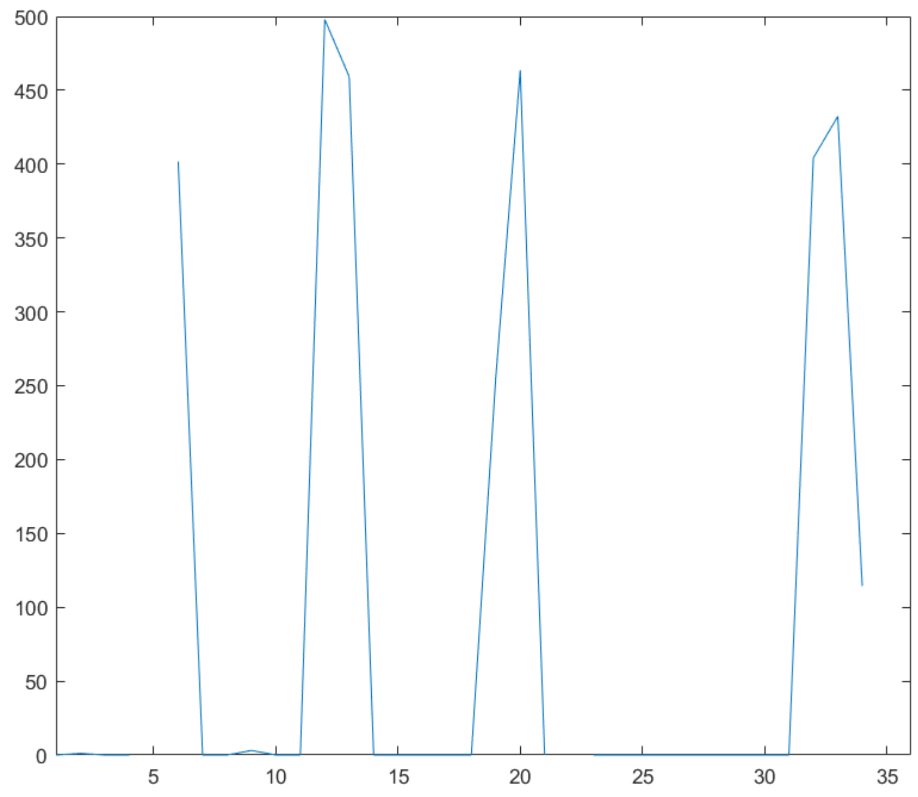


Figure 4.7: *Ratio between the integrated enstrophy and the wavelet transform over some random areas on the surface*

Figure 4.7 shows the result of using random points. Most of the values are 0, or close to 0, except for å few exceptions where they fall in the same range as before. The main reason for this is that the enstrophy is very close to 0 when we are not in a vortex or in a strong vortical structure. Since the enstrophy is the sum of the vorticities squared, when there is little to no vorticity, the enstrophy becomes really small. The wavelet transform, on the other hand, is much smaller when it is not close to a small scale structures, but the disparity between the values in areas with strong responses and areas with weak responses is not as large. Nevertheless, some dependency is present between the strengths of the vortices and the signals in the transforms, but is not investigated further here. Ideally one would like to compare all the vortices in the whole data set; these would then have to be identified manually, which was deemed unnecessary considering part of the reason for using these transform is also identifying these structures. This problem can be reopened once the identification of the attached vortices has improved.

4.2.2 On attached vortex isolation

Isolating the attached vortices means to filter out the other structures that appear in Figure 4.5. First notice how the dimples generated by the attached vortices are very circular, and mainly give a response in wavelet space in a small circular area. The other responses are, on the contrary, very elongated.

The method used here is based on exactly that. There is another tool in computer vision that is often used to find circles or circular shapes in images. It is called the circle Hough transform. The idea behind this method is very simple. For a given radius the Hough transform makes circles around contour points on the signal. If enough of them have a common intersection, that is probably the centre of a circle. The

MATLAB function `imfindcircles` from the Image Processing Toolbox was used. The documentation for this function (*imfindcircles*, 2012) or other sources on the Hough transformation can give more details on the circle Hough transform. Another criteria was used when using this method, the perimeter of the blobs could not be too much larger than the circumference of the circle with the radius of inspections. This filtered out long shapes that were semicircular at the ends and could give a response in the Hough transform. The radii for the attached vortices varied somewhat, so a range of radii was chosen based on the observed size of the dimples caused by the attached vortices. With the width of the domain being 2π nondimensional units, then the radii of the attached vortices where mostly between 0.022 and 0.123 units. The width of the whole vortices was then up to 0.24 units, close to the width W used in the VISA method mentions in chapter 3 to find the up- and downwellings.

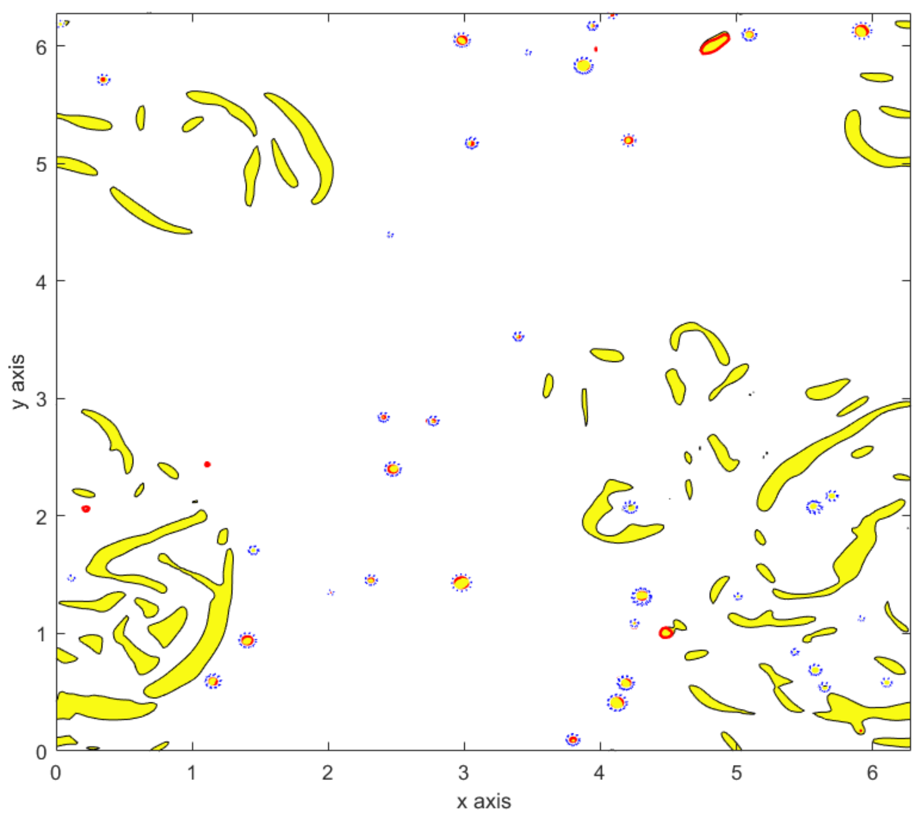
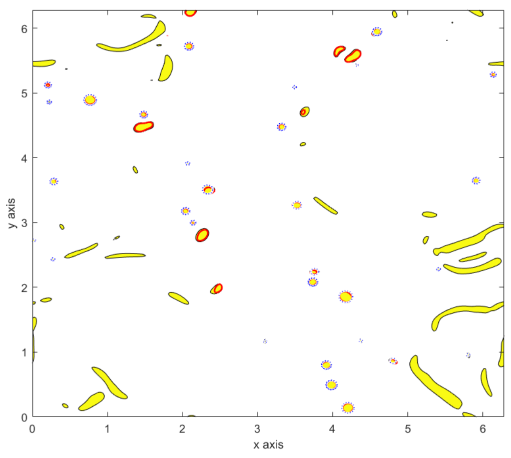


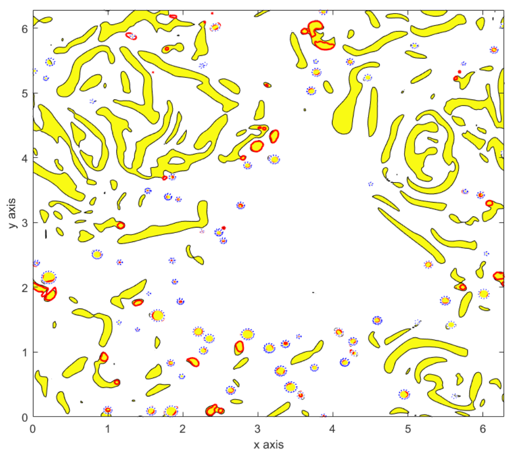
Figure 4.8: *Positive values of the wavelet transform at the smallest scale. The red outlines are the vortices found using the velocity field at the surface. The blue dotted circles are the attached vortices found by the Hough circle transform.*

The result of this method for the same timestep as Figure 4.4 can be seen in Figure 4.8. The blue dotted circles are the circles found by the Hough circle transform. We see that generally this isolates the circular structures made from the attached vortices quite well. It does not capture the vortical structures that are more elongated, like the one on the top right of the figure. This may be two attached vortices meeting at this timestep and fusing together, making this elongated shape. These types of structures can be hard to identify when we look at a single timestep. However, looking at several timesteps after one another could amend this problem. This was not implemented in this work. It could also be a vortex that is diagonally aligned with the surface, but that is unlikely as the presence of the surface realigns vortices to be either parallel with it or attached to it (Kumar et al., 1998, Shen et al., 1999, Dabiri, 2003).

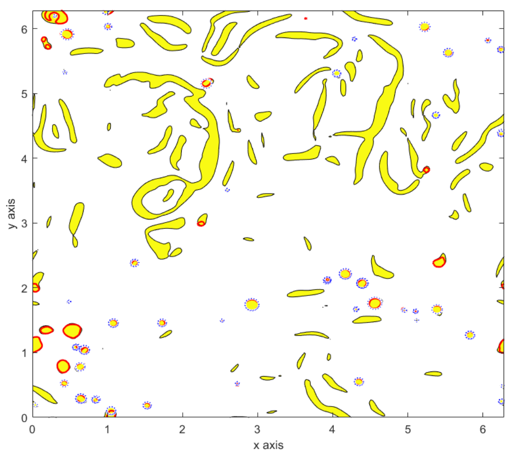
There are also some false positives, where the Hough algorithm finds circles in the response that are not necessarily caused by an attached vortex, but is just a region where the wavelet transform is above the threshold and has a circular contour. This are the areas around an up- or downwelling that give responses here as well.



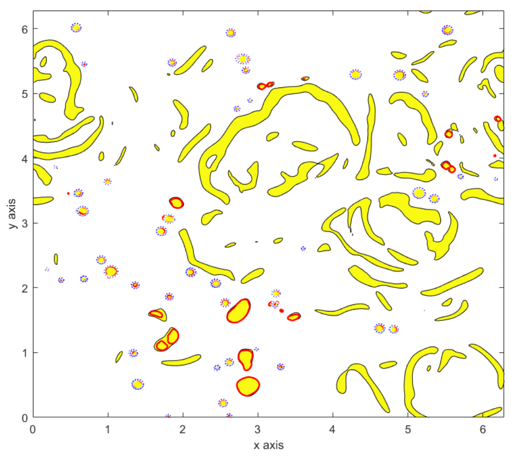
(a) *t = 215*



(b) *t = 227.5*



(c) *t = 252.5*



(d) *t = 340*

Figure 4.9: *Four different timesteps of the simulation data with the attached vortices outlined in red, and dotted blue circles are detected using the Hough circle transform.*

Figure 4.9 shows some additional timesteps in the simulation data, where this method was used. We see how the blue dotted lines capture a fair amount of the attached vortices (red outlines). However, when these vortices are large and somewhat deformed, they are badly captured. The Hough transform has a sensitivity variable that can be preset such that the blobs do not need a very clear circular shape to be included. Even so, setting this sensitivity too high sometimes meant that some circles where detected at the ends of some of the elongated shapes. In general the optimal sensitivity for the Hough transform depended on the timestep and on the cutoff threshold of the wavelet transform (the value $0.4 \cdot 10^{-4}$) mentioned in a previous section. It is therefore quite challenging to find threshold values that were appropriate for all the timesteps. Keeping in mind that this is all based on the same data set with the same Reynolds and Froude numbers, it means that for experimental data and real world implementation, there is still a lot to be improved.

One recommendation for future work is the to look at the time variation of the surface. The data in this work was inspected one timestep at the time, however a lot of information can be gathered from looking at the time variation. For instance these attached vortices are not stationary, but move slowly in a certain direction that is not the mean flow direction, however the up and downwellings do not, and they move along with the flow (Kumar et al., 1998); or, like in our case where there is no mean flow, they are stationary. The up- and downwellings do, however, expand slightly outward, as the fluid motion underneath causing them changes. Nonetheless, following these circular blobs in time should give an idea if it represents an attached vortex, or something else.

Another point to be made is if one should look at all the attached vortices, even the weaker ones. In Figure 4.8 and Figure 4.9 we see that there are a lot of very small vortices. Some of them cause very small disturbances on the surface that are still captured by the wavelet transform. We chose a very small threshold for the positive values in the wavelet transform, so all small scale structures that bent the surface slightly downwards are present. There is a case to be made if these small vortices are actually long lived attached vortices. They may be some very small turbulent structures that dissipate at the surface and do not renew the surface efficiently. Again an analysis in time will show if this is the case. The reason for mentioning this is that by increasing the threshold in the wavelet transform, a lot of the structures around the up- and downwellings that give false positives on the Hough transform are filtered out. If the deformation caused by the vortex is very small, and does not show sufficient values in wavelet space, that might also indicate that the vortex itself is very weak, and does not contribute to surface renewal in a significant way.

One could also be tempted to say that a way to filter out the small blobs around the up- an downwellings is to ignore them and only look at a certain distance away from them. This is, however, not a good proposition. In fact these attached vortices tend to appear at the ends of up- and downwellings as they die out. If we look at the bottom left of Figure 4.9b we see a semicircular yellow structure (around $(x, y) = (1, 1)$), with some others around it. This is probably a lingering structure after an up- or downwelling phenomena. We see it has two red outlines at is ends, meaning that two attached vortices have formed there, as the up- or downwelling died out. This has been postulated to be the main cause of these attached vortices in some flow by Kumar et al., 1998, and the blobs around the up- and downwellings cannot therefore be omitted.

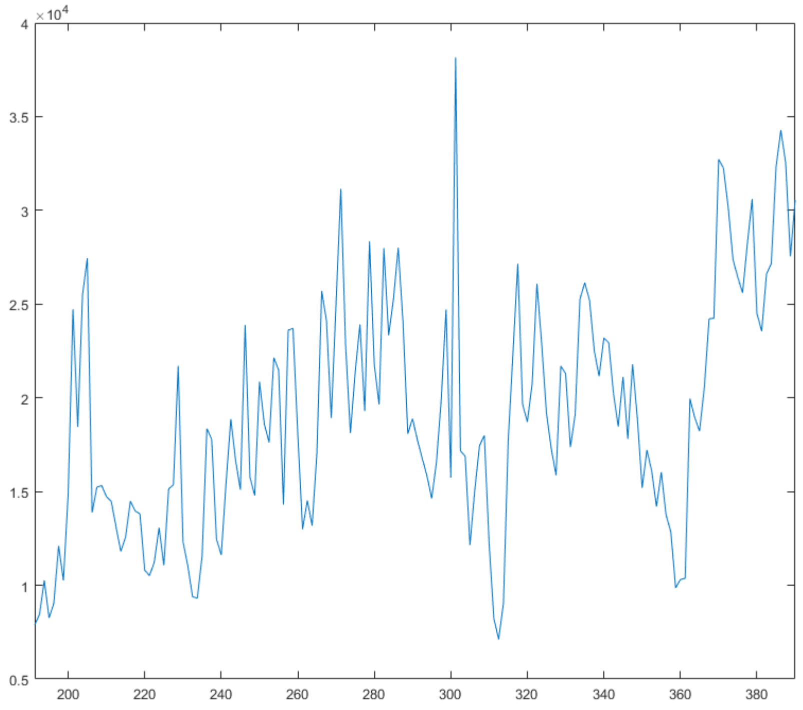


Figure 4.10: *Ratio between the sum of the vortex strength at each timestep calculated by summing the enstrophy and the estimated strength of the vortices detected using the wavelet and the Hough transforms*

The strength of the detected vortices using this method was also compared with the total enstrophy of all the vortices at each timestep detected with the algorithm by Jeong and Hussain mentioned in chapter 3. The idea was to see if the same correlation from the lest section could be found, but also to identify which timesteps gave a bad correlation relative to the others so that the identification could be improved overall. This gave the results in Figure 4.10. Since this represents the sum of all the vortices at each timestep, the ratio is in a different range than before. However, the spread is much larger, relative to the one in the previous section, going from $1 \cdot 10^{-4}$ to $3.5 \cdot 10^{-4}$. The clear correlation from before is not as present and reinforces the need to improve on the detection of attached vortices. This is the extent this work delves into attached vortex identification, now to look at up- and downwellings.

4.3 Response from up- and downwellings

Let us now draw the contours for the up and downwellings on the small scale wavelet transform and see what we can derive.

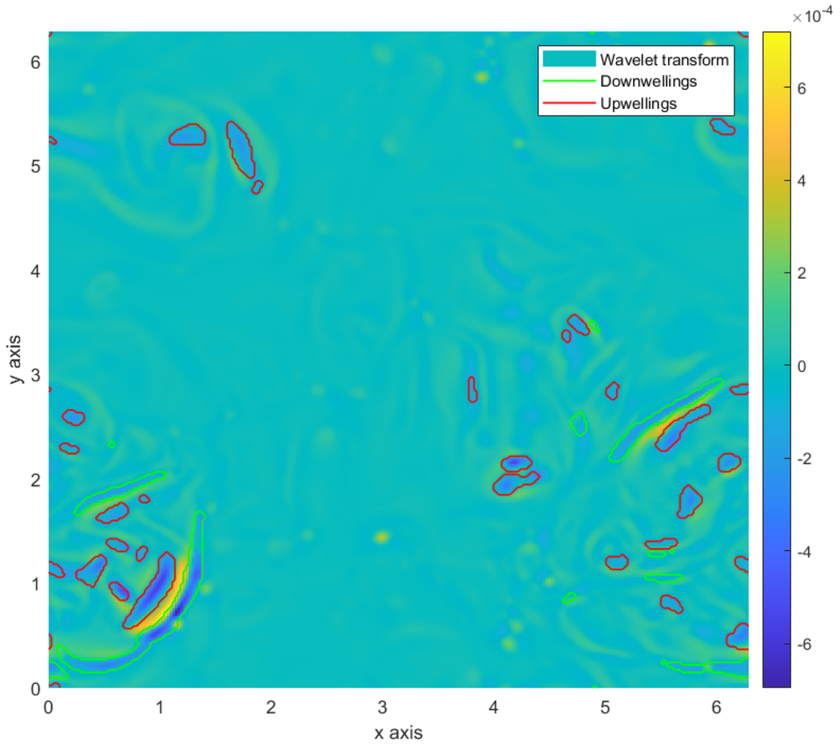


Figure 4.11: *Wavelet transform at the smallest scale at the timestep 196.25 with the green outlines representing the downwellings and the red outlines representing the upwellings.*

From Figure 4.11 we see that most of the up- and downwellings are present in areas of the transform with negative values. As mentioned earlier in section 4.2, this means that the surface is bent upward relative to it's surroundings in these regions. This is to be expected for upwellings, since they are formed when some fluid moving upwards meets the surface and causes a stagnation at the centre of the upwelling, where the fluid from the jet is dispersed to the sides. This stagnation, and the area around it have a higher pressure that the rest of the surface causing it to bend upward. But in most of the literature the presence of downwellings is argued with the presence of "scars" on the surface (Brocchini and Peregrine, 2001, Savelsberg and Van De Water, 2009). These are areas where the surface is bend sharply downward, like a long rift. So one would assume that the downwellings are right under these scars, however when we superimpose the outlines for the up and downwellings we see something different.

Figure 4.12 shows the same surface as before, but now the up- and downwellings are also included in green (downwellings) and red (upwellings) outlines. The white circles focuses on such a scar on the surface. We see that the downwelling does not coincide with it, but is slightly to the side. In fact it seems more like this scar is right in between an up- and a downwelling.

To explain this we need to review the theory of these surface deformations. It was mentioned in chapter 2 that the surface is deformed by velocity fluctuations that caused pressure fluctuations due to the conservation of energy law proposed by Bernoulli. As mentioned when there is a local increase in velocity the pressure drops and the surface bends downwards, while a decrease in velocity causes the pressure to

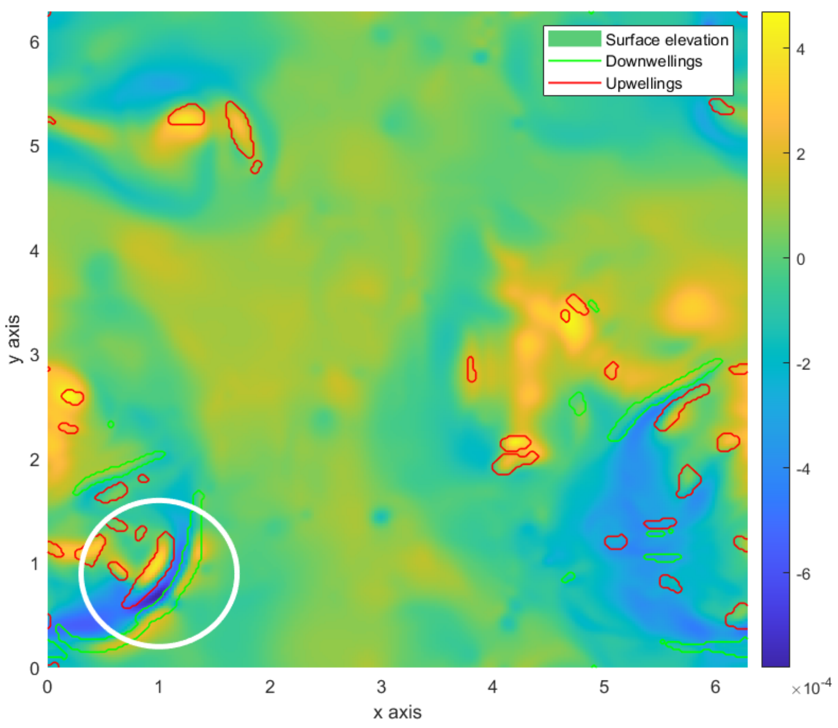


Figure 4.12: *Surface elevation at the timestep 196.25 with the green outlines representing the downwellings and the red outlines representing the upwellings. The white circle shows a scar on the surface.*

rise and the surface to bend upwards. We also mentioned that upwellings are generated when there is a jet of fluid that meets the surface, here it spreads to the side, causes a stagnation at the surface and the velocity to be very small close to the centre of the upwelling. This results in the surface being bent upwards. Downwelling phenomena occur close to upwellings, when the fluid that has been dispersed to the sides is jetted inwards and downwards again, because the mass needs to be conserved. This means that at the centre of the downwelling the fluid meets and causes a new stagnation point, where the fluid is decelerated close to it due to the change in velocity direction. A downwelling effect will, in fact, cause the surface to rise, not a fall. However, in the region between an upwelling and a downwelling the fluid is moving at a fairly high velocity, since it is being pushed by the upwelling and pulled by the downwelling. This causes a strong local decrease in pressure resulting in the scars we see on the surface.

With that cleared up let us again set a threshold for the wavelet transform, so that we are only looking at negative values. An appropriate threshold value in this case was found to be around $-1 \cdot 10^{-4}$. Figure 4.13 shows the wavelet transform again, but only taking into account the values below the threshold. Again the yellow blobs represent the areas in the transform where it is below the threshold, the red outlines are the upwellings and the green are the downwellings. Apparently there are very few other structures than downwellings and upwellings that cause a local increase in surface elevation and are therefore captured by the transform at this timestep. It is important to note that there may have been other areas where the surface was deformed to the same degree, but the wavelet transform gives large values only if this change happens one the same length scale as the scale of the wavelet used. So there may be other areas on the surface where it is bent upward in a large area, but since the scale of this structure was too big, it did not give a response in wavelet space that exceeded the set threshold.

As with the detection of attached vortices, the result of this varied from timestep to timestep in the simulation. Figure 4.14 shoes a few other timesteps in the simulation. In Figure 4.14b the up- and downwellings were not captured as well by only setting a threshold on the wavelet transform. This may have been because the up- and downwellings at this timesteps were weaker, at the end of their life and did not affect the surface as in other timesteps.

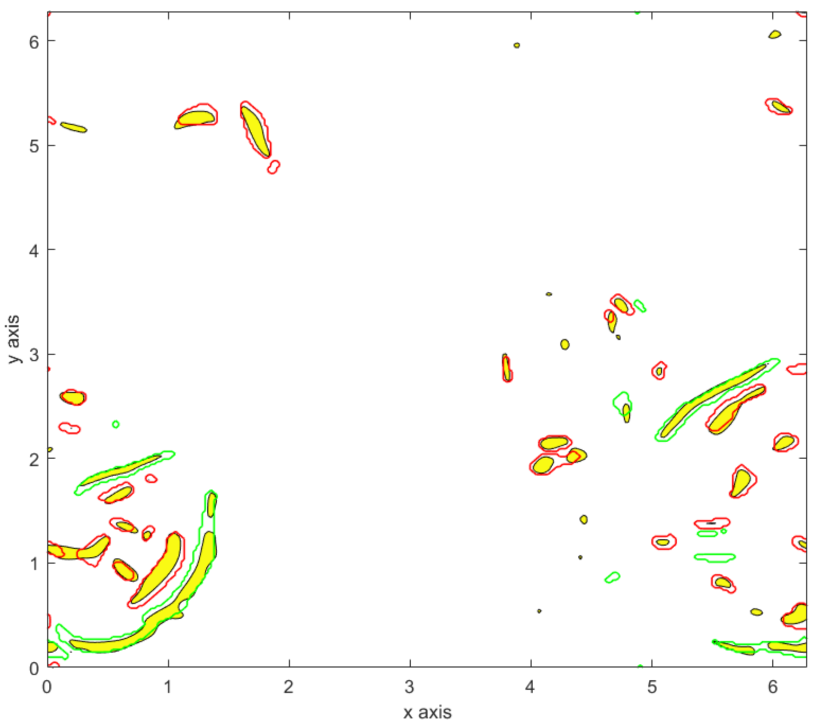


Figure 4.13: *Negative values of the wavelet transform at the smallest scale, the threshold value was around* $-1 \cdot 10^{-4}$. *The green outlines represent the downwellings and the red outlines represent the upwellings.*

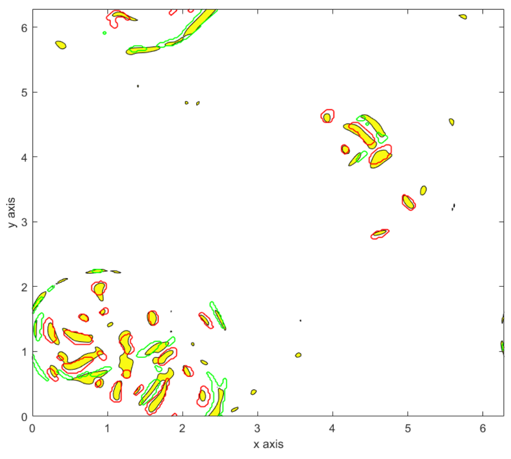
To improve on this we can use the fact that up- and downwellings tend to be elongated shapes, contrary to the attached vortices that were in general circular. This means that an elongated wavelet should be able to capture them better that the isotropic Mexican hat wavelet we have used thus far. As mentioned in chapter 3 the wavelet used in the transformations could be altered in some ways. The Mexican hat could, for example, be made anisotropic. Figure 4.15 shows a reconstruction of the smallest scale anisotropic Mexican hat wavelet used in this section. This was made by elongating the isotropic Mexican hat wavelet. The dips on the side of the hat we had all around the central top of the hat in the isotropic version are now only on two of the sides, and the wavelet is now much wider in one direction than the other.

As mentioned in subsection 2.2.1, with anisotropic wavelets the angle of rotation of the wavelet is also relevant for the transformations. In general the up- and downwelling can have any rotation, some do not follow a straight line, but can be bent. This means that several angles of the transform need to be considered. Figure 4.16 shows the transform of the surface at the timestep 195.25 for an angle of 0 and the wavelet is parallel with the x-axis. We see that all the structures in the transform are elongated and angled the same as the wavelet. This is the same timestep ad before. Again the areas where the wavelet has strong negative values are areas where the surface is locally bent upward, but since we are using an elongated Mexican hat, this also means that this elevated area is somewhat elongated. This means that following these should give good approximation of the location of the up- and downwellings.

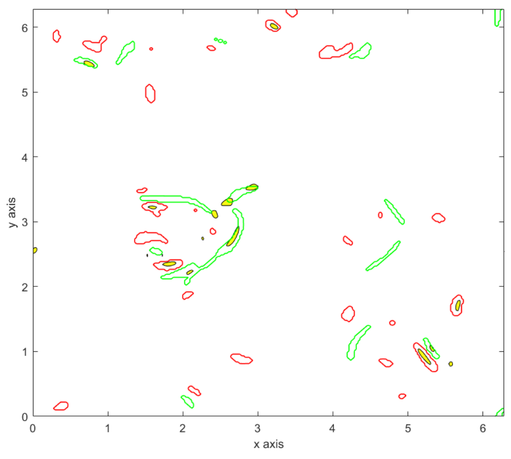
Let us then again set a threshold of around $0.9 \cdot 10^{-4}$ where we take values in the anisotropic transform that exceed it. We then do this for several angles such that the anisotropic Mexican hat is aligned with the up- or downwelling no matter the orientation. The angles chosen where all the angles from $-90°$ to $90°$ with a $15°$ interval, i.e.

$$[-90°, -75°, -60°, -45°, -30°, -15°, 0°, 15°, 30°, 45°, 60°, 75°, 90°].$$

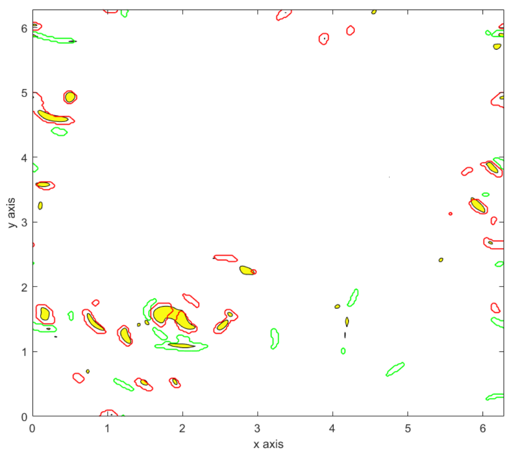
Figure 4.17 shows the result of using the anisotropic Mexican hat to detect the up- and downwellings. We see that there is no apparent improvement for this timestep than just using the isotropic Mexican hat, especially since that one captured them quite well. In fact, for most timesteps, using the anisotropic version of the wavelet was worse or only marginally better than the isotropic.



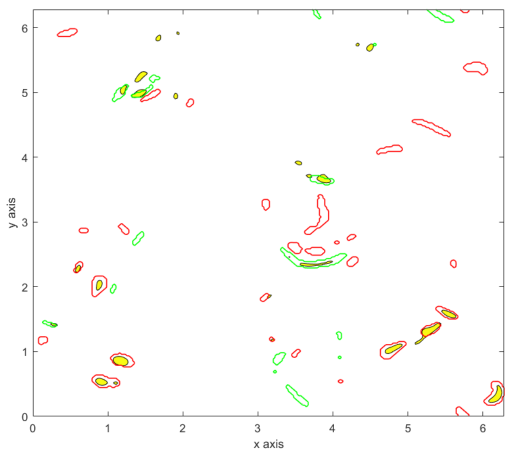
(a) t = *265*



(b) t = *301.25*



(c) t = *315*



(d) t = *352.5*

Figure 4.14: *Four other timesteps in the simulation with the outlines for the upwellings (red) and downwellings (green).*

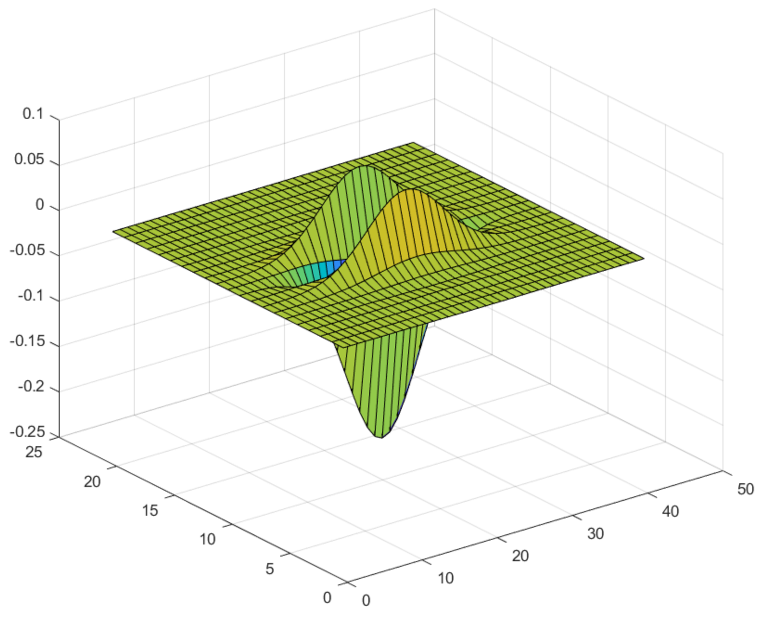


Figure 4.15: *Anisotropic Mexican hat wavelet for the smallest scale.*

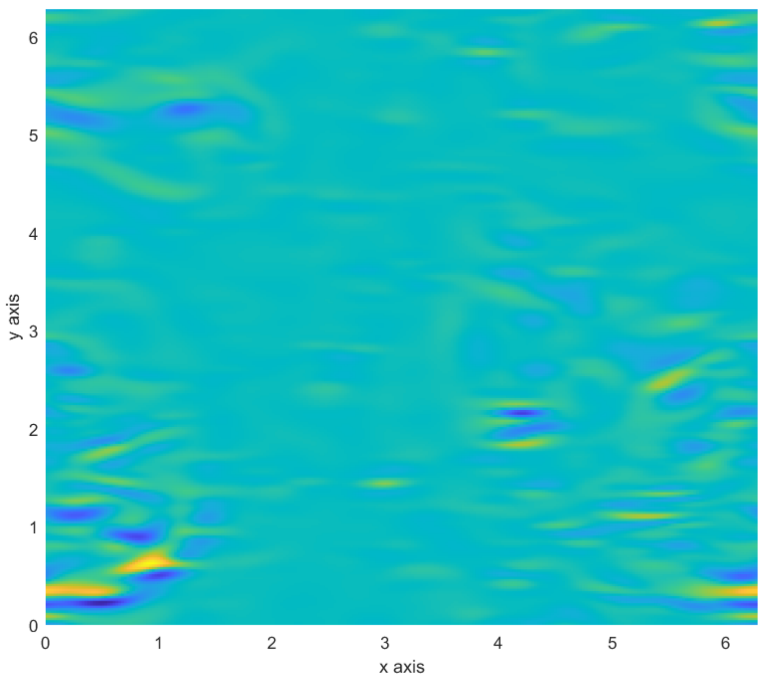


Figure 4.16: *Wavelet transform using the anisotropic Mexican hat wavelet at the time 196.25. The angle of the wavelet to the x-axis is 0.*

To see how well this transform captured the structures, a statistical correlation of the areas found using the wavelet transform and the ones found by looking at the surface divergence was calculated. Since the values at each point are not the same for the surface elevation and the surface divergence (they don't have the same units), the correlations was done purely on the areas where the up- and downwellings were found with both methods at each timestep. This could also be used to identify the timesteps with the best and worst correlation and see which aspects can be improved. The function `corr2` in MATLAB was used

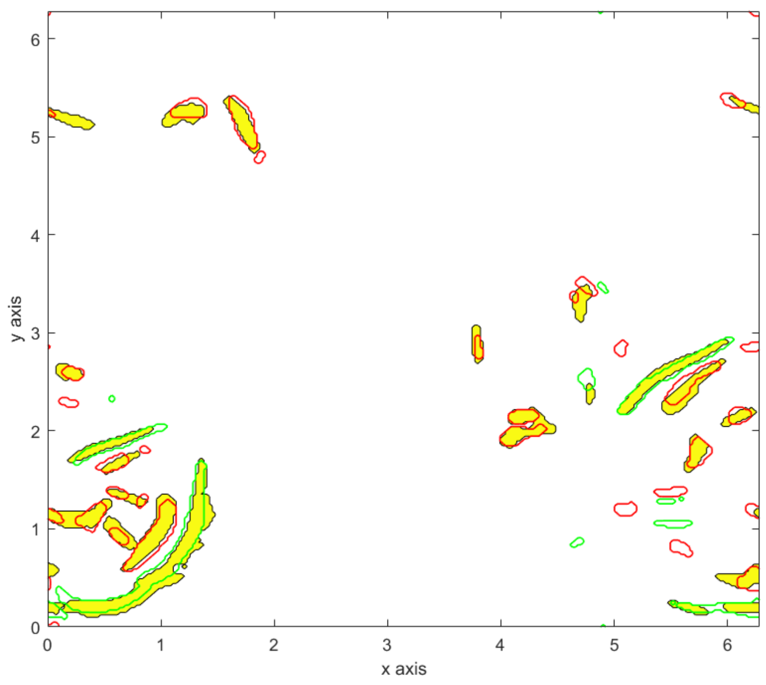


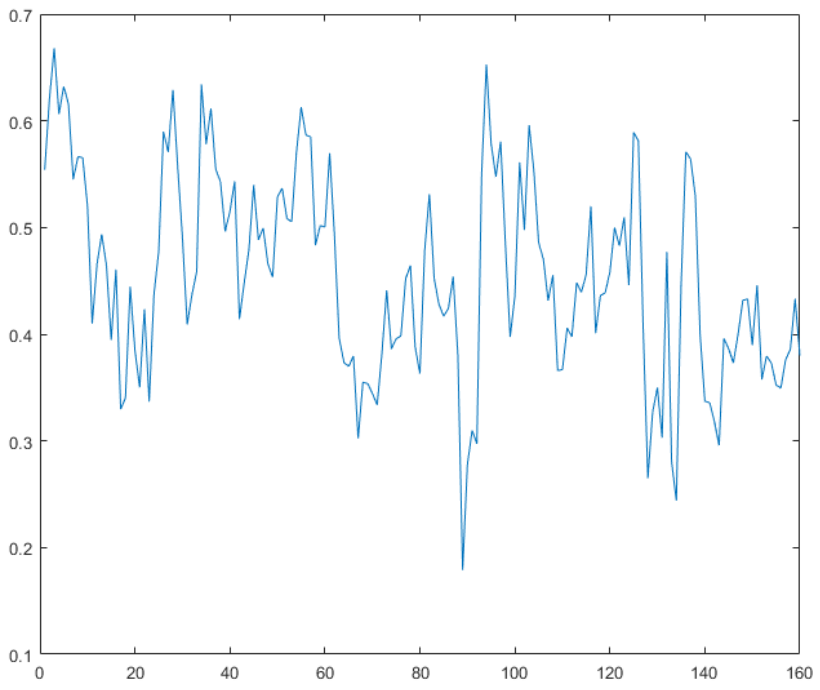
Figure 4.17: *Wavelet transform using the anisotropic Mexican hat wavelet at the time 196.25. This are all the values for all the angles that exceeded the threshold*

(*corr2*, 2006), and the result can be seen in Figure 4.18.

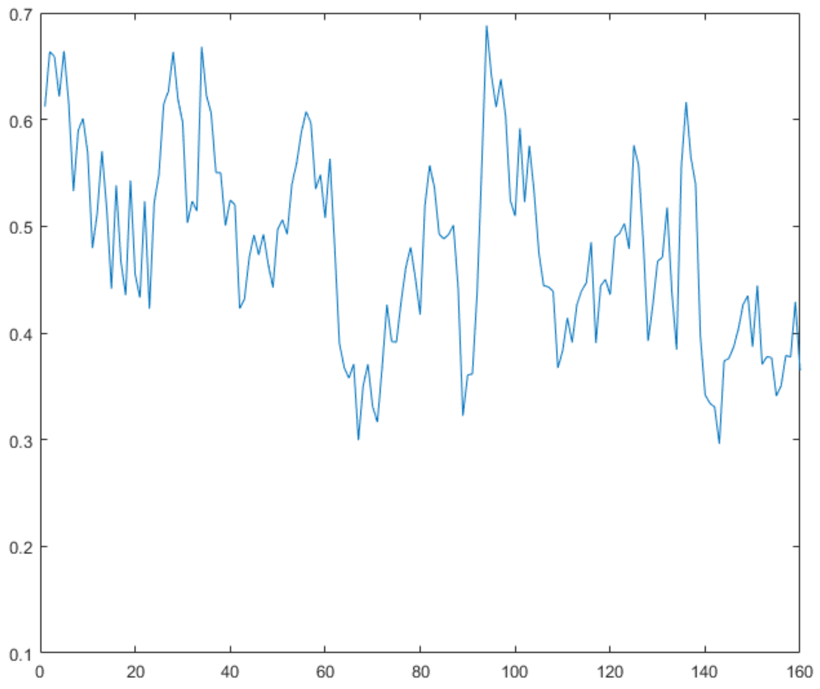
A correlation of 1 means that we have exactly the same picture. For both wavelet types the correlation was, for the most part, between 0.4 and 0.6, with some values outside this interval. We see that in general the isotropic Mexican hat has higher correlation values with the identified up- and downwellings.

There are two ways to alter the isotropic Mexican hat wavelet in MATLAB, Figure 4.15 represents stretching along one directions, but we can also squeeze the wavelet at the smallest scale. Using this squeezed wavelet gave the results in Figure 4.19. We see that this gave better correlations than the stretched wavelet.

When stretching the Mexican hat the general scale of the wavelet can be seen as increasing, the width of the wavelet is the same as before, but the stretching increases the scale in the other direction. While when squeezing it, the scale decreases. A small difference in scale should not affect the transform, as the wavelet transform has a large uncertainty when the scale is small, as was mentioned in 2.2. In fact the transform changed little when we increases the wavelet scale from 1 to 2.5. What happened instead was that the shapes in the transform became a little bit smeared out, like in 4.3. Nonetheless, the apparent change in scale was noticeable in this comparison. This is likely the result of the length scales of the structures we are detecting. No literature was found on how to estimate the length scales of the free surface structures for a general turbulence stirred flow, however, by inspecting the outputs form the VISA method presented in section 3.1, the width of the the up- and downwellings was around 0.15 nondimentional units across. The width of the isotropic Mexican hat and the elongated Mexican hat was around 0.2, this was the distance between the two "valleys" in the reconstructed wavelets. The width of the squeezed Mexican hat was around 0.14. So one explanation to why the elongated Mexican hat did not detect the up- and downwellings as well was because it was slightly too large, and even though it gave responses around these structures, they did not exceed the threshold for all timesteps. The problem with squeezing the wavelet is that we are already using the smallest possible scale, so squeezing it even more could loose some information about it. Using a finer grid should help with this problem as the wavelet with the same length scale as the smallest in this data set could be several datapoints wide, reducing the uncertainty. One way to do this is to interpolate the surface elevation, but this was not attempted in this work.



(a) *Anisotropic Mexican hat*



(b) *Isotropic Mexican hat*

Figure 4.18: *Statistical correlation of the location of the up- and downwellings between the surface divergence VISA method and (a) the anisotropic Mexican hat wavelet, (b) the isotropic Mexican hat wavelet.*

Another thing to note is that the timesteps with good or bad correlations matched across all three wavelet types, if we look at Figure 4.18a, Figure 4.18b and Figure 4.19. The explanation might simply be

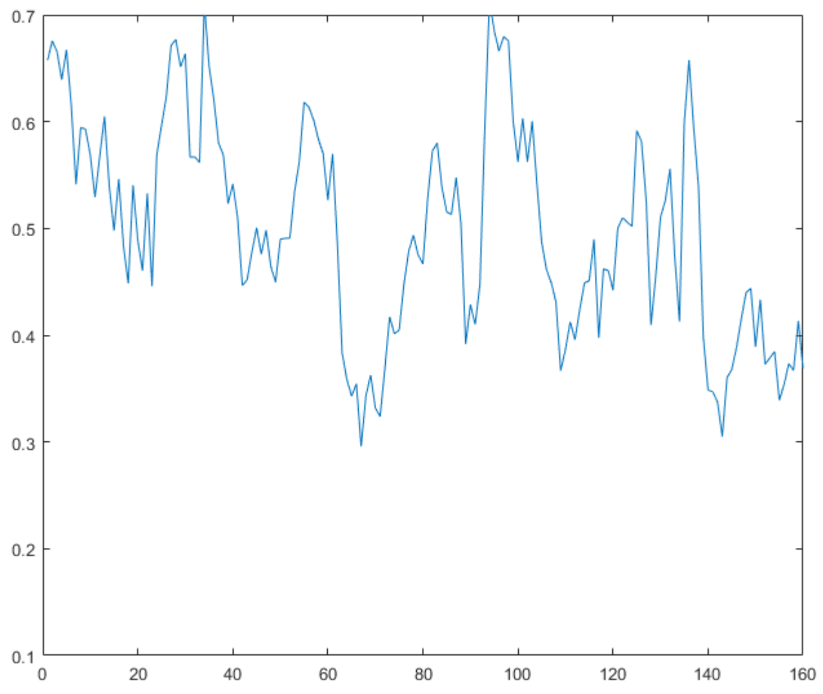


Figure 4.19: *Statistical correlation of the location of the up- and downwellings between the surface divergence VISA method and the squeezed anisotropic Mexican hat.*

that all three are a Mexican hat type wavelet, and some of these timesteps contained structures that were captured better or worse by this wavelet, however this was not investigated in detail.

In any case this indicates that we are at the extent of what can be detected by only using the Mexican hat wavelet. Other wavelets and tools used in computer vision might improve the process. There is also little progress in differentiating between up- and downwellings. Initially it was thought that the scars were in the same location as the downwellings, and that using the surface curvature could be used to differentiate between them, i.e. if the surface is bent upward it was an upwelling, if it was bent downwards it was a downwelling. However, as was mentioned earlier, both phenomena bend the surface upward, and the scars are located between them, where the fluid is moving from an upwelling to a downwelling. Looking into the surface curvature gave, nonetheless, some other interesting discoveries that are covered in the next section.

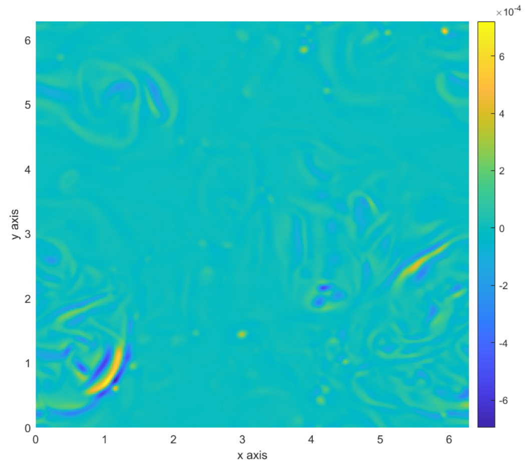
4.4 Looking at the surface curvature

During the course of the work it was thought the surface curvature could improve the isolation of some of the structures. For example the attached vortices form surface dimples that bend the surface downward, giving a strongly positive surface curvature. The surface curvature could also help in locating up- and downwellings, especially the sharp "scars" that form around them. The surface curvature was therefore calculated and compared with the wavelet transform, in the hope of improving the identification of these turbulent structures. Here we will look into the mean surface curvature and see how it relates to the transform.

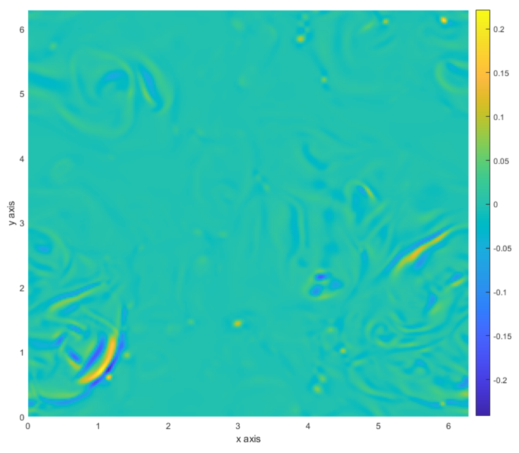
The mean surface curvature is the average of the two principal curvatures, which are the inverse of the principal radii of curvature (Pressley, 2010). Let us call the surface elevation η. The mean curvature of η becomes

$$H = \frac{1}{2}\left(\frac{1}{R_1} + \frac{1}{R_2}\right) = \frac{(1+\eta_x^2)\eta_{xx} + (1+\eta_y^2)\eta_{yy} - 2\eta_x\eta_y\eta_{xy}}{2(1+\eta_x^2+\eta_y^2)^{3/2}} \tag{4.1}$$

R_1 and R_2 are the two principal radii of curvature, the two principal curvatures are then $1/R_1$ and $1/R_2$, so the mean curvature is the mean of the two. η_x represents the derivative of η with respect to x, $\frac{\partial \eta}{\partial x}$ (see Pressley, 2010 chapter 8 for more). Let us now compare the mean surface curvature and the wavelet transform at the smallest scale.



(a)



(b)

Figure 4.20: *The wavelet transform for the smallest scale Mexican hat (a), and the surface curvature (b).*

Figure 4.20 shows the wavelet transform with the smallest scale Mexican hat wavelet and the surface mean curvature side by side. We see that they look really similar, even though the colour bar on the sides show they have different values. However it seems like they are a scaled versions of the other.

If we look at our surface elevation we see that most of it's values are really small, in the order of 10^{-4}. So the derivatives will also have really small values. We can then linearize Equation 4.1, since all the nonlinear terms are really close to 0 and we can simplify the equation for the mean curvature to

$$H = \frac{1}{2}(\eta_{xx} + \eta_{yy}). \tag{4.2}$$

In MATLAB these derivatives were calculated using the `gradient` function, that calculates the gradient numerically using central difference (*gradient*, 2006). This means that to find the derivative of f at a point n, this function calculates the difference in the values of the neighbouring points and divides it by the distance between them. Let's say that the data points are evenly space and that the spacing between them is Δx, the derivative becomes

$$\frac{1}{2\Delta x}(f_{n+1} - f_{n-1}).$$

This can also be represented as a numerical convolution, since we can think of this operation as taking the matrix containing the data to be differentiated and convulving with

$$\frac{1}{2\Delta x}\begin{bmatrix}-1 & 0 & 1\end{bmatrix}.$$

At the boundaries the derivatives are calculated using single-sided derivatives

$$\frac{\partial f}{\partial x}_{n=1} = \frac{1}{\Delta x}(f_2 - f_1)\ ,\ \frac{\partial f}{\partial x}_{n=N} = \frac{1}{\Delta x}(f_N - f_{N-1}) \tag{4.3}$$

Where N represents the last point.

Let us look at a quick example. We have a set of values with unit spacing, and the values

$$\begin{bmatrix}\dots & 4 & 5 & 6 & 9 & \dots\end{bmatrix}$$

are somewhere in the middle of this set of values. Let us look at the derivatives at second and third position in the section using the convolution with $1/2\begin{bmatrix}-1 & 0 & 1\end{bmatrix}$

$$\begin{gathered}\begin{bmatrix}\dots & 4 & 5 & 6 & 9 & \dots\end{bmatrix} * \frac{1}{2}\begin{bmatrix}-1 & 0 & 1\end{bmatrix} = \\ \begin{bmatrix}\dots & \frac{1}{2}(-1\cdot 4 + 0\cdot 5 + 1\cdot 6) & \frac{1}{2}(-1\cdot 5 + 0\cdot 6 + 1\cdot 9) & \dots\end{bmatrix} = \\ \begin{bmatrix}\dots & \frac{6-4}{2} & \frac{9-5}{2} & \dots\end{bmatrix} = \begin{bmatrix}\dots & 1 & 2 & \dots\end{bmatrix}\end{gathered}$$

which are the derivatives one gets by using central difference.

For the second derivative, the process is repeated, again using the `gradient` function. To find the second derivative of f at n we use the first derivatives at the neighbouring points again. If the derivative is $f_n^{'}$ at a point n, we get that the second derivatives are

$$\begin{aligned}\frac{\partial^2 f}{\partial x^2} &= \frac{1}{2\Delta x}(f_{n+1}^{'} - f_{n-1}^{'}) = \frac{1}{2\Delta x}\left(\frac{1}{2\Delta x}(f_{n+2} - f_n) - \frac{1}{2\Delta x}(f_n - f_{n-2})\right) = \\ &= \frac{1}{4\Delta x^2}(f_{n-2} - 2f_n + f_{n+2})\end{aligned} \tag{4.4}$$

which is equivalent to the convolution

$$\frac{\partial^2 f}{\partial x^2} = f * \frac{1}{4\Delta x^2}\begin{bmatrix}1 & 0 & -2 & 0 & 1\end{bmatrix}. \tag{4.5}$$

To find the derivatives in the x-direction we apply this along the rows of the matrix, and for the derivatives in the y-direction we apply this along the columns. Now, since convolution is a linear operation, finding the sum of the two derivatives is also quite simple

$$\begin{aligned}\frac{\partial^2 f}{\partial^2 x} + \frac{\partial^2 f}{\partial^2 y} &= f * \frac{1}{4\Delta x^2}\begin{bmatrix}1 & 0 & -2 & 0 & 1\end{bmatrix} + f * \frac{1}{4\Delta x^2}\begin{bmatrix}1 \\ 0 \\ -2 \\ 0 \\ 1\end{bmatrix} = \\ &= f * \frac{1}{4\Delta x^2}\begin{bmatrix}0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -4 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0\end{bmatrix}\end{aligned} \tag{4.6}$$

The matrix in 4.6 strongly resembles the isotropic Mexican hat wavelet at the smallest scale, even if it only applies for the centre points the signal we are calculating the second derivatives of. In fact the wavelet transform has some similar inconsistencies when it is close to the boundary of the signal, where part of the wavelet is outside the signal. Nonetheless the similarities in the matrix that we convolve with explains the similar responses from the wavelet transform and the mean surface curvature. This in turn means that looking at the surface curvature could not help with isolating the structures in this case. It is important to note that the formula for the surface curvature, Equation 4.1, can only be linearised when the values for the derivatives are much smaller then 1. Which means that data taken from experimental data, where the surface elevation and the derivatives are not necessarily of a small magnitude, the surface curvature might show something different.

This also prompted the idea that the large scale wavelet transforms with the Isotropic Mexican hat wavelet might show the surface curvature of a scaled down version of the surface, i.e. if the surface is minimised from 256×256 to, for example, 32×32, then surface curvature of this new surface could be similar to the wavelet transform at scale 8. This proved to be the case as can be seen in Figure 4.21.

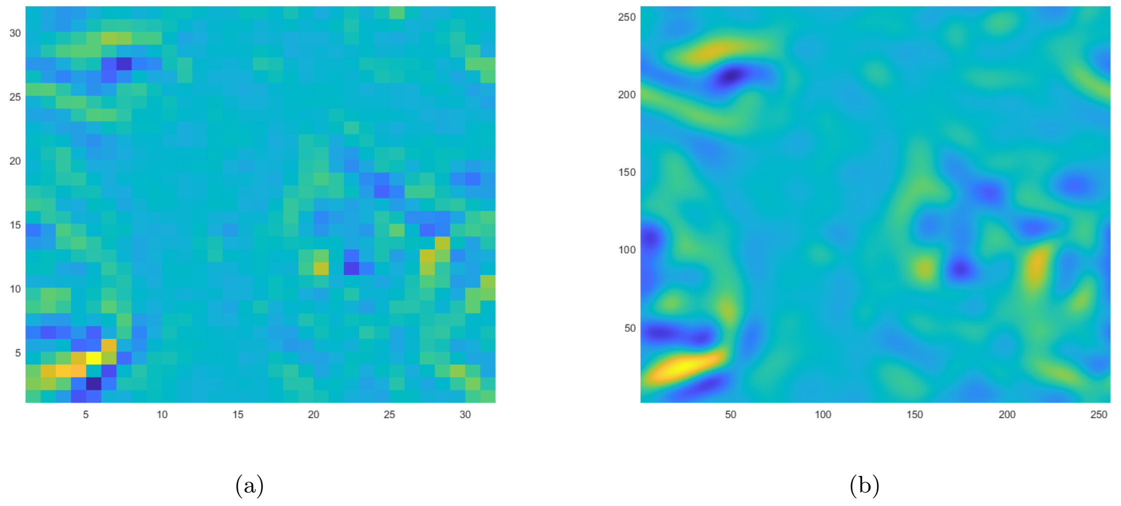


Figure 4.21: *The surface curvature of the resized surface (a), and the wavelet transform at scale 8 (b).*

The transform at the larger scale is then very similar to the surface curvature, or just the sum of the two second order derivatives. Wavelet transformations using the Mexican hat wavelet can then be seen as the second order derivatives of higher order than the central difference method used in `gradient`. The elongated Mexican hat could also represent a directional second derivative, but this was not tested here.

The principal curvatures, $\frac{1}{R_1}$, $\frac{1}{R_2}$, and the total, or Gaussian, curvature, $K = \frac{1}{R_1} \cdot \frac{1}{R_2}$ (Pressley, 2010), are also mainly derived from the second derivatives, and did not give informative results, but they where not investigated closely in this work.

5 Conclusions

The two dimensional wavelet transform proved to be an impressive tool in isolating structures on the free surface. When it comes to both attached vortices and up- and downwellings, the Mexican hat wavelet managed to filter out the structures of interest quite well. When detecting attached vortices, however, there where a lot of artefacts from structures near the up- and downwellings, and the detection of the surface wellings themselves can still be improved. It is important to note that in the data-set for this work, there where no waves. The surface was only agitated by the turbulence underneath, which may in some cases generate capillary waves, but these were not apparent in the given simulation data. In real world experiments, waves of different orders of magnitude will always be present, however, since these are moving oscillatory structures, they should be straight forward to filter out by looking into the time variation of the surface elevation.

Both the attached vortices and the up- and downwellings are very localised structures, the vortices move slowly in a certain directions, while the up- and downwellings are fairly stationary relative to the flow. This means that looking into the time variation of the structures on the surface can also improve in identifying these turbulent structures. As we mentioned in section 4.2, the time variation could drastically improve the detection of attached vortices by filtering out other unwanted structures.

Another factor to improve in future work is how the threshold values for the different detection methods are set. In this work, they were put in manually, using the ones which gave preferable result throughout the different timesteps. Still, one would like for these threshold values to be set automatically by the method depending on the input data.

And lastly the choice of wavelet is also important. Here the Mexican hat was used since it was simple and gave easy to read transforms. Some other wavelets may yet prove to be better suited, especially for detecting the up- and downwellings. The elongated anisotropic Mexican hat was not sufficient to cater to the long shapes of these phenomena, while the squeezed proved useful. Increasing the data points of the data could help understand why this is the case.

With all this in mind, one cannot overlook the fact that the wavelet transforms performed really well in this analysis, and that they can and should be used more in this field. Future work could also include other computer vision tools to improve detection and analysis of free surface structures.

References

Antoine, J., Murenzi, R., Vandergheynst, P., & Ali, S. (2008). *Two-dimensional wavelets and their relatives.* Cambridge University Press.

Brocchini, M., & Peregrine, D. H. (2001). The dynamics of strong turbulence at free surfaces. part 1. description. *J. Fluid Mech*, *449*, 225–254.

Çengel, Y. A., & Cimbala, J. M. (2014). *Fluid mechanics : Fundamentals and applications* (3rd in SI units.). McGraw-Hill.

Christopher, J. Z., Wade, R. M., Peter, A. R., James, B. E., Eric, J. H., Hendrik, J. Z., John, W. H. D., & David, T. H. (2007). Environmental turbulent mixing controls on air-water gas exchange in marine and aquatic systems. *Geophysical research letters*, *34*(10), L10601–n/a.

Corr2. (2006). https://se.mathworks.com/help/images/ref/corr2.html

Cwtfilterbank. (2018). https://se.mathworks.com/help/wavelet/ref/cwtfilterbank.html

Cwtft2. (2013). https://se.mathworks.com/help/wavelet/ref/cwtft2.html

Cwtftinfo2. (2013). https://se.mathworks.com/help/wavelet/ref/cwtftinfo2.html

Dabiri, D. (2003). On the interaction of a vertical shear layer with a free surface. *J. Fluid Mech*, *480*, 217–232.

Dolcetti, G., & García Nava, H. (2019). Wavelet spectral analysis of the free surface of turbulent flows. *Journal of Hydraulic Research*, *57*(2), 211–226.

Donelan, M. A., Drennan, W. M., & Magnusson, A. K. (1996). Nonstationary analysis of the directional properties of propagating waves. *Journal of physical oceanography.*, *26*(9), 1901–1914.

Frew, N. M., Bock, E. J., Schimpf, U., Hara, T., Hausecker, H., Edson, J. B., McGillis, W. R., Nelson, R. K., McKenna, S. P., Uz, B., & Jaehne, B. (2004). Air-sea gas transfer: Its dependence on wind stress, small-scale roughness, and surface films. *Journal of geophysical research. Oceans*, *109*(C8).

Gradient. (2006). https://se.mathworks.com/help/matlab/ref/gradient.html

Guo, X., & Shen, L. (2010). Interaction of a deformable free surface with statistically steady homogeneous turbulence. *J. Fluid Mech*, *658*, 33–62.

H5info. (2011). https://se.mathworks.com/help/matlab/ref/h5info.html

H5read. (2011). https://se.mathworks.com/help/matlab/ref/h5read.html

Icwt. (2016). https://se.mathworks.com/help/wavelet/ref/icwt.html

Imfindcircles. (2012). https://se.mathworks.com/help/images/ref/imfindcircles.html

Jähne, B., Münnich, K. O., Bösinger, R., Dutzi, A., Huber, W., & Libner, P. (1987). On the parameters influencing air-water gas exchange. *Journal of Geophysical Research: Oceans*, *92*(C2), 1937–1949.

Jeong, J., & Hussain, F. (1995). On the identification of a vortex. *Journal of fluid mechanics*, *285*(-1), 69.

Kermani, A., Khakpour, H. R., Shen, L., & Igusa, T. (2011). Statistics of surface renewal of passive scalars in free-surface turbulence. *J. Fluid Mech*, *678*, 379–416.

Khakpour, H. R., Igusa, T., & Shen, L. (2012). Coherent vortical structures responsible for strong flux of scalar at free surface. *International journal of heat and mass transfer*, *55*(19-20), 5157–5170.

Kumar, S., Gupta, R., & Banerjee, S. (1998). An experimental investigation of the characteristics of free-surface turbulence in channel flow. *Physics of fluids (1994)*, *10*(2), 437–456.

Lay, D. C., Lay, S. R., & McDonald, J. J. (2016). *Linear algebra and its applications* (5th). Pearson.

Li, Z., Yang, W., Peng, S., & Liu, F. (2020). A survey of convolutional neural networks: Analysis, applications, and prospects.

Mallat, S., & Peyre´, G. (2008). *A wavelet tour of signal processing: The sparse way.* San Diego: Elsevier Science & Technology.

Pan, Y., & Banerjee, S. (1995). A numerical study of free-surface turbulence in channel flow. *Physics of fluids (1994)*, *7*(7), 1649–1664.

Phillips, O. M. (1977). *The dynamics of the upper ocean* (2nd ed.). Cambridge University Press.

Pressley, A. N. (2010). *Elementary differential geometry*. London: Springer London, Limited.

Rottmann, K. (2019). *Matematisk formelsamling*. Universitetsforlaget.

Savelsberg, R., & Van De Water, W. (2009). Experiments on free-surface turbulence. *J. Fluid Mech*, *619*(1), 95–125.

Scal2frq. (2006). https://se.mathworks.com/help/wavelet/ref/scal2frq.html

Sen, D. (2014). The uncertainty relations in quantum mechanics. *Current Science*, *107*(2), 203–218. http://www.jstor.org/stable/24103129

Shen, L., Yue, D. K. P., & Triantafyllou, G. S. (2004). Effect of surfactants on free-surface turbulent flows. *J. Fluid Mech*, *506*, 79–115.

Shen, L., Zhang, X., Yue, D. K. P., & Triantafyllou, G. S. (1999). The surface layer for free-surface turbulent flows. *J. Fluid Mech*, *386*, 167–212.

Turney, D. E., & Banerjee, S. (2013). Air–water gas transfer and near-surface motions. *J. Fluid Mech*, *733*, 588–624.

Wang, N., & Lu, C. (2010). Two-dimensional continuous wavelet analysis and its application to meteorological data. *Journal of atmospheric and oceanic technology*, *27*(4), 652–666.

Wanninkhof, R., Asher, W. E., Ho, D. T., Sweeney, C., & McGillis, W. R. (2009). Advances in quantifying air-sea gas exchange and environmental forcing. *Annual Review of Marine Science*, *1*(1), 213–244.

Wt. (2018). https://se.mathworks.com/help/wavelet/ref/cwtfilterbank.wt.html

Appendix

Appendix A MATLAB scrips

Total inspection

```
%% Total inspection
clear

% Referance scales and flow variables
Re     = 2500;                  % Reynold's Number
Fr     = sqrt(0.01);            % Frode's Number
l_ref  = 2*pi;                  % Reference length
H_bar  = 5*pi;                  % Domain height
a_0    = 0.1;                   % Forcing parameter
u      = a_0*l_ref;             % Referance velocity
nu     = 1/Re;                  % Kinematic viscosity

% Loading surface elevation and timesteps
[eta0, timestamps] = load_surface_data();

% Square half side lenght
ds = 0.1;

% Cutoff value for veortex inspection
cutoff_lambda = 2;

% Wavelt scale index for inspection (i.e. the index in the scale vector for
% the scale of intereset)
wlt_scale = 1;

% Taylor microscale very close to the surface
[u_rms, taylor_scale] = taylor_length_scale();
% Taylor Reynlods number
Re_lambda = u_rms*taylor_scale/nu;

% Initializing vortex strength vector
vortex_strength = [];

% Testing variable
vortex_strength_2 = [];

% Initializing vortex strength vector
ds_ind = floor(ds*256/(2*pi))*2 + 1;
vortex_wlt_max = [];
vortex_wlt_int = [];

% Testing variable
vortex_wlt_max_2 = [];
```

```
vortex_wlt_int_2 = [];

% Differential area used for integration
dA = (2*pi/255).^2;

% Loading z coordinates
[~,~,~,~,zz] = load_near_surf_velocities(1);

% Important to note that zz is 1 at the suirface and decreas as we go
% further down in the domain
% Surface mesh
x_range = linspace(0, l_ref, 256);
y_range = linspace(0, l_ref, 256);
[x_2d, y_2d] = meshgrid(x_range, y_range);
[x_3d, y_3d, z_3d] = meshgrid(x_range, y_range, zz);
[x_ver, z_ver] = meshgrid(x_range, zz);
% [x_int, y_int, z_int] = meshgrid(x_range, y_range, zw);

% Index of horizontal slice for inspection
% insp_ind = 1:length(zz);
insp_ind = 1;
z_ind = 1;

% Initializing vortex strength variables
true_vortex_strength = zeros(1,length(timestamps));
vortex_wlt_strength  = zeros(1,length(timestamps));
vortex_elev_strength = zeros(1,length(timestamps));


%% Velocity field comparison at different timesteps

for data_index = 5
% Data index for the timestamp we are looking at

% Loading subsurface velocities (u,v,w) and the presure (p)
[v,u,w,p,zz] = load_near_surf_velocities(data_index);

% Surface elevation at this time index
surf_elev = reshape(eta0(data_index,:,:),[256 256]);


%% Finding the derivatives

% Differentiating the surface elevation
% Time step for differentiation
dt = (timestamps(2) - timestamps(1));

% Grid variables in the x ang y directions
xhi = linspace(0,l_ref,256);
psi = linspace(0,l_ref,256);
dx  = xhi(2)-xhi(1);

% The relevant derivatives
[deta_dt, ~, ~] = gradient(eta0, dt, dx, dx);

% Surface gradients at the investigated timestamp
[deta_dx, deta_dy] = gradient(surf_elev, xhi, psi);

% z_index = 1:length(zz);
% Calculating velocity derivatives
[du_dx, ~, ~] = vel_derivatives(u, zz, insp_ind,...
    surf_elev, deta_dx, deta_dy);
[~, dv_dy, ~] = vel_derivatives(v, zz, insp_ind,...
    surf_elev, deta_dx, deta_dy);

% The rest if needed
% [du_dx, du_dy, du_dz] = vel_derivatives(u, zz, insp_ind,...
%     surf_elev, deta_dx, deta_dy);
% [dv_dx, dv_dy, dv_dz] = vel_derivatives(v, zz, insp_ind,...
%     surf_elev, deta_dx, deta_dy);
% [dw_dx, dw_dy, dw_dz] = vel_derivatives(w, zz, insp_ind,...
%     surf_elev, deta_dx, deta_dy);
% [dp_dx, dp_dy, dp_dz] = vel_derivatives(p, zz, insp_ind,...
%     surf_elev, deta_dx, deta_dy);

% Surface gradient and surface curvature
% Finding the second order derivatives
[deta_dxx, deta_dxy] = gradient(deta_dx, xhi, psi);
[~, deta_dyy] = gradient(deta_dy, xhi, psi);

% Surface gaussian curvature, the product of the principal curvatures
surf_gaus_curv = (deta_dxx.*deta_dyy-deta_dxy.^2)./...
    (1 + deta_dx.^2 + deta_dy.^2).^2;

% Surface mean curvature, the mean of the principal curvatures
surf_mean_curv = ((1+deta_dy.^2).*deta_dxx + (1+deta_dx.^2).*deta_dyy ...
    - 2*deta_dx.*deta_dy.*deta_dxy)./(1 + deta_dx.^2 + deta_dy.^2).^(3/2);

% Inicializing principal curvatures
surf_prin_curv_1 = zeros(256,256);
surf_prin_curv_2 = zeros(256,256);

% Setting the principal curvatures
for i = 1:256
    for j = 1:256
        surf_II_form = [deta_dxx(i,j) deta_dxy(i,j);...
            deta_dxy(i,j) deta_dyy(i,j)];
        prin_curv = eig(surf_II_form);
        surf_prin_curv_1(i,j) = max(prin_curv);
        surf_prin_curv_2(i,j) = min(prin_curv);
    end
end

% Mean surface elevation
surf_mean = mean(surf_elev,'all');

%% Finding vortices

% Getting the middel eigenvalue at each point (lambda), the vorticity inn
% all trhee directions at each point (omega) and the sum og eignevalues Q
[lambda, omega, Q] = lambda_omega_Q(u, v, w, zz, insp_ind,...
    surf_elev, deta_dx, deta_dy);

% Checking for votices
vortex_pos = zeros(256,256); % Initializing variable
vortex_omega = zeros(256,256);
```

```
% When the second eigenvalue (lambda_2) is negative we have a vortex core
% When it is positive we don't, leaving that at 0
% Bathtub vortices will mainly have vorticity in the z directiions
if length(insp_ind) == 1
    vortex_pos = lambda(:,:,1).*(lambda(:,:,1)<0);

    % z-direction vorticity for bathtub vortices
    omega_z = reshape(omega(:,:,1,3),[256 256]);
    vortex_omega(lambda<-cutoff_lambda) = omega_z(lambda<-cutoff_lambda);
else
    vortex_pos = lambda(:,:,z_ind).*(lambda(:,:,z_ind)<0);

    % z-direction vorticity for bathtub vortices
    omega_z = reshape(omega(:,:,z_ind,3),[256 256]);
    vortex_omega(lambda(:,:,z_ind)<-cutoff_lambda) = ...
        omega_z(lambda(:,:,z_ind)<-cutoff_lambda);
end

% Calculating enstrophy
enstrophy = sum(omega.^2,4);

% Defining x and y coordinates for som bathtub vortices
[pos_x_vec, pos_y_vec] = bathtub_pos_vec(data_index);
pos_x_vec_2 = (2*pi-2*ds)*0.99*rand(1,10)+ds;
pos_y_vec_2 = (2*pi-2*ds)*0.99*rand(1,10)+ds;

% Calculating vortex strengths
if length(insp_ind) == 1
    for i = 1:length(pos_x_vec)
        vortex_strength(end+1) = vortex_strength_by_int(vortex_pos,...
            enstrophy, pos_x_vec(i), pos_y_vec(i));
        vortex_strength_2(end+1) = vortex_strength_by_int(vortex_pos,...
            enstrophy, pos_x_vec_2(i), pos_y_vec_2(i));
    end
else
    for i = 1:length(pos_x_vec)
        tmp_strength = [];
        tmp_strength_2 = [];
        for ind = z_ind
            tmp_strength(ind) = vortex_strength_by_int(vortex_pos,...
                enstrophy(:,:,ind), pos_x_vec(i), pos_y_vec(i));
            tmp_strength_2(ind) = vortex_strength_by_int(vortex_pos,...
                enstrophy(:,:,ind), pos_x_vec_2(i), pos_y_vec_2(i));
        end
        vortex_strength(end+1) = sum(tmp_strength);
        vortex_strength_2(end+1) = sum(tmp_strength_2);
    end
end

true_vortex_strength(data_index) = ...
    sum(enstrophy.*(vortex_pos<-1.5), 'all')*dA;
vortex_elev_strength(data_index) = ...
    sum(surf_elev.*(vortex_pos<-1.5), 'all')*dA;

%% Finding up and downwellings

% Surface divergence
surf_div = du_dx + dv_dy;
```

```
% Probable location of up and downwellings
[surf_upwelling, surf_downwelling] = surface_wellings_locations(surf_div);


%% Isotropic Wavelet transformation
% Wavelet name
wname_iso = "mexh";
wname_iso_full = "Mexican␣hat";

% Scale range for transform
scale_range = logspace(0, log10(2^3), 25);

% Wavelet transformation of the surface
cwt_elev_iso = cwtft2(surf_elev, 'wavelet', wname_iso, ...
    'scales', scale_range);

% Matrix the coefficients at each frequency
wlt_elev_iso = reshape(cwt_elev_iso.cfs(:,:,1,:), ...
    [256 256 length(scale_range)]);
angles_coeff_lev = angle(wlt_elev_iso); % Phase angle of the coefficients

% Calculating vortex strengths
for i = 1:length(pos_x_vec)
    [vortex_wlt_max(end+1), vortex_wlt_int(end+1)] = ...
        vortex_wlt_trans(vortex_pos, wlt_elev_iso, ...
        pos_x_vec(i), pos_y_vec(i));
    [vortex_wlt_max_2(end+1), vortex_wlt_int_2(end+1)] = ...
        vortex_wlt_trans(vortex_pos, wlt_elev_iso, ...
        pos_x_vec_2(i), pos_y_vec_2(i));
end

[wlt_iso, wlt_iso_len_scale] = ...
    wavelet_length_scale(scale_range(wlt_scale),surf_elev, wname_iso);

% Vortex position circles from wavelet transformation and vortex strength
% based on the transform of the elevation
% [vortex_wlt_strength(data_index), vor_circles, vor_radii] = ...
%     wlt_surface_vortices(wlt_elev_iso(:,:,1), surf_mean_curv);

% Calculating vortex strength based on the transform of the elevation
% vortex_wlt_strength(data_index) = vortex_wavelet_strength(vor_circles,...
%     vor_radii, wlt_elev_iso(:,:,1));

%% Nonisotropic wavelet transformation

wname_mexh_non = {"mexh", {2,1,0.2}}; % Mexican hat wvaelet 2D
wname_morl_non = {"morl", {6,1,1}}; % Morlet wavelet 2D

% Scale range for transform
scale_range = logspace(0, log10(2^3), 25);

% Angles used in the transform
angles = linspace(-pi/2, pi/2, 13);
% angles = [0 15 30 45 60 75 90 105]/180*pi;

cwt_elev_non = cwtft2(surf_elev, 'wavelet', wname_mexh_non, ...
    'angles', angles, 'scales', scale_range);

wlt_elev_non = reshape(cwt_elev_non.cfs, [256 256 ...
    length(scale_range) length(angles)]);

[wlt_non, wlt_non_len_scale] = ...
    wavelet_length_scale(scale_range(wlt_scale),surf_elev, wname_mexh_non);

wlt_scars = zeros(256, 256);

for i = 1:length(angles)
    wlt_scars = wlt_scars + (real(wlt_elev_non(:,:,wlt_scale,i))<-1.1*10^-4);
end


%% Stefan's idea: comparison with surface curvature

x_resize = 32;
xhi_resize = linspace(0,2*pi,x_resize);
surf_elev_resize = imresize(surf_elev, [x_resize x_resize]);

[deta_dx_resize, deta_dy_resize] = ...
    gradient(surf_elev_resize, xhi_resize, xhi_resize);

[deta_dxx_resize, deta_dxy_resize] = ...
    gradient(deta_dx_resize, xhi_resize, xhi_resize);
[~, deta_dyy_resize] = gradient(deta_dy_resize, xhi_resize, xhi_resize);

surf_mean_curv_resize = ((1+deta_dy_resize.^2).*deta_dxx_resize ...
    + (1+deta_dx_resize.^2).*deta_dyy_resize ...
    - 2*deta_dx_resize.*deta_dy_resize.*deta_dxy_resize)./...
    (1 + deta_dx_resize.^2 + deta_dy_resize.^2).^(3/2);

surf_mean_curv_resize_2 = imresize(surf_mean_curv, [x_resize x_resize]);

cwt_elev_iso_resize = cwtft2(surf_elev_resize, 'wavelet', wname_iso, ...
    'scales', 1 );

wlt_elev_iso_resize = reshape(cwt_elev_iso_resize.cfs(:,:,1,:), ...
    [x_resize x_resize]);

%% Timestep finished

fprintf("Timestep␣%5.2f␣done,␣nr␣%3d␣of␣%d␣\n",...
    timestamps(data_index), data_index, length(timestamps))

end


%% Ploting

if data_index == 160

    figure(1), clf
    plot(timestamps, true_vortex_strength./vortex_wlt_strength)
    xlim([timestamps(1) timestamps(end)])
    title("Comparison␣of␣vortex␣strength␣at␣each␣timestep")

else
    figure(1), clf
%     subplot(1,2,1)
    pcolor(x_2d, y_2d, surf_elev)
    hold on
%     contour(x_2d, y_2d, -vortex_pos*0.5*10^-4, [1.5 1.5]*0.5*10^-4,...
%         'r', 'linewidth', 1.5)
    contour(x_2d, y_2d, surf_downwelling*10^-4, [0.1 0.1]*10^-4,...
        'g', 'linewidth', 1)
    contour(x_2d, y_2d, surf_upwelling*10^-4, [0.1 0.1]*10^-4,...
        'r', 'linewidth', 1)
%     viscircles([1,0.9],0.7, 'color', 'w');
    shading flat
    colorbar
    xlim([0,2*pi])
    ylim([0,2*pi])
    xlabel("x␣axis")
    ylabel("y␣axis")
    legend("Surface␣elevation", "Downwellings", "Upwellings")
%     title("Surface elevation at time = " + timestamps(data_index))
    hold off

    figure(2), clf
    pcolor(x_2d, y_2d, wlt_elev_iso(:,:,wlt_scale))
    hold on
%     contour(x_2d, y_2d, -vortex_pos*0.5*10^-4, [1.5 1.5]*0.5*10^-4,...
%         'r', 'linewidth', 1)
%     contour(x_2d, y_2d, surf_downwelling*10^-4, [0.1 0.1]*10^-4,...
%         'g', 'linewidth', 1)
%     contour(x_2d, y_2d, surf_upwelling*10^-4, [0.1 0.1]*10^-4,...
%         'r', 'linewidth', 1)
    shading flat
    colorbar
    xlim([0,2*pi])
    ylim([0,2*pi])
    xlabel("x␣axis")
    ylabel("y␣axis")
%     legend("Wavelet transform", "Downwellings", "Upwellings")
%     title("Wavelet transform of surface elevation at scale = "...
%         + scale_range(1))
    hold off

    wlt_angle = 7;

    figure(3), clf
    pcolor(x_2d, y_2d, real(wlt_elev_non(:,:,wlt_scale, wlt_angle)))
    shading flat
    colorbar
    xlim([0,2*pi])
    ylim([0,2*pi])
    xlabel("x␣axis")
    ylabel("y␣axis")
    title("Anisotropic␣mexh␣transform␣with␣angle␣␣=␣" ...
        + angles(wlt_angle)/pi*180)
    hold off

    figure(4), clf
    % contour_levels = [cutoff_lambda cutoff_lambda];
%     contourf(x_2d, y_2d, wlt_elev_iso(:,:,wlt_scale), [0.4 0.4]*10^-4)
    pcolor(x_2d, y_2d, surf_mean_curv)
    hold on
%     contourf(x_2d, y_2d, real(wlt_elev_non(:,:,wlt_scale, 1)), ...
%         [1.2 1.2]*10^-4)
%     contour(x_2d, y_2d, -vortex_pos*10^-4, [0.7 0.7]*10^-4,...
%         'r', 'linewidth', 1)
%     viscircles(vor_circles,vor_radii);
    shading flat
    colorbar
    xlim([0,2*pi])
    ylim([0,2*pi])
    xlabel("x␣axis")
    ylabel("y␣axis")
    % legend("Wavelet transform", "Vorteces")
%     title("Wavelet transform positive values and attached vortices")
    hold off

    figure(5), clf
    % pcolor(x_2d, y_2d, wlt_coeff(:,:,1))
    contourf(x_2d, y_2d, -wlt_elev_iso(:,:,wlt_scale), [0.8 0.8]*10^-4)
%     contourf(x_2d, y_2d, real(-wlt_elev_non(:,:,wlt_scale, wlt_angle)),...
%         [1 1]*10^-4)
%     contourf(x_2d, y_2d, wlt_scars, [1 1]*10^-4 )
%     contourf(x_2d, y_2d, -surf_mean_curv, [0.03 0.03])
    hold on
    % contourf(x_2d, y_2d, -wlt_coeff(:,:,1), [1 1]*10^-4)
    contour(x_2d, y_2d, surf_downwelling*10^-4, [0.1 0.1]*10^-4,...
        'g', 'linewidth', 1)
    contour(x_2d, y_2d, surf_upwelling*10^-4, [0.1 0.1]*10^-4,...
        'r' ,'linewidth', 1)
    % contourf(x_2d, y_2d, 10*wlt_coeff(:,:,1), [1 1]*10^-3)
    shading flat
    % colorbar
    xlim([0,2*pi])
    ylim([0,2*pi])
    xlabel("x␣axis")
    ylabel("y␣axis")
%     legend("Wavelet transform", "Downwellings", "Upwellings")
%     title("Wavelet transform negative and posisive values comparison")
    hold off

    figure(6), clf
    pcolor(x_2d, y_2d, surf_mean_curv)
    hold on
    contour(x_2d, y_2d, surf_downwelling*10^-4, [0.1 0.1]*10^-4,...
        'g', 'linewidth', 1)
    contour(x_2d, y_2d, surf_upwelling*10^-4, [0.1 0.1]*10^-4,...
        'r', 'linewidth', 1)
    shading flat
    colorbar
    xlim([0,2*pi])
    ylim([0,2*pi])
    xlabel("x␣axis")
    ylabel("y␣axis")
    title("Surface␣mean␣curvature")
    hold off


    figure(7), clf
%     subplot(1,2,1)
    pcolor(surf_mean_curv_resize)
    shading flat
%     title("Mean curvature og the resized surface")
```

```
    figure(8), clf
%      subplot(1,2,2)
    pcolor(wlt_elev_iso(:,:,25))
    shading flat
%      title("Wavelet transform at scale = " + scale_range(25))
%
%      figure(8), clf
%      plot(vortex_strength./vortex_wlt_max)
%      xlim([1,length(vortex_strength)])
%
%      figure(9)
%      plot(vortex_strength_2./vortex_wlt_max_2)
%      xlim([1,length(vortex_strength_2)])
end
```

Attached vortices detection

```
%% Attached vorteces detection
clear

% Referance scales and flow variables
Re    = 2500;                   % Reynold's Number
Fr    = sqrt(0.01);             % Frode's Number
l_ref = 2*pi;                   % Reference length
H_bar = 5*pi;                   % Domain height
a_0   = 0.1;                    % Forcing parameter
u     = a_0*l_ref;              % Referance velocity
nu    = 1/Re;                   % Kinematic viscosity

% Loading surface elevation and timesteps
[eta0, timestamps] = load_surface_data();

% Square half side lenght
ds = 0.1;

% Cutoff value for veortex inspection
cutoff_lambda = 2;

% Wavelt scale index for inspection (i.e. the index in the scale vector for
% the scale of intereset)
wlt_scale = 1;

% Taylor microscale very close to the surface
[u_rms, taylor_scale] = taylor_length_scale();

% Initializing vortex strength vector
vortex_strength = [];

% Testing variable
vortex_strength_2 = [];

ds_ind = floor(ds*256/(2*pi))*2 + 1;

% Differential area used for integration
dA = (2*pi/255).^2;

% Loading z coordinates
[~,~,~,~,zz] = load_near_surf_velocities(1);

% Important to note that zz is 1 at the suirface and decreas as we go
% further down in the domain
% Surface mesh
x_range = linspace(0, l_ref, 256);
y_range = linspace(0, l_ref, 256);
[x_2d, y_2d] = meshgrid(x_range, y_range);
[x_ver, z_ver] = meshgrid(x_range, zz);

% Index of horizontal slice for inspection
% insp_ind = 1:length(zz);
insp_ind = 1;
z_ind = 1;

% Initializing vortex strength variables
```

```
true_vortex_strength = zeros(1,length(timestamps));
vortex_wlt_strength  = zeros(1,length(timestamps));
vortex_elev_strength = zeros(1,length(timestamps));


%% Velocity field comparison at different timesteps

for data_index = 5
% Data index for the timestamp we are looking at

% Loading subsurface velocities (u,v,w) and the presure (p)
[v,u,w,p,zz] = load_near_surf_velocities(data_index);

% Surface elevation at this time index
surf_elev = reshape(eta0(data_index,:,:),[256 256]);


%% Finding the derivatives

% Differentiating the surface elevation
% Time step for differentiation
dt = (timestamps(2) - timestamps(1));

% Grid variables in the x ang y directions
xhi = linspace(0,l_ref,256);
psi = linspace(0,l_ref,256);
dx  = xhi(2) - xhi(1);

% The relevant derivatives
[deta_dx, deta_dy] = gradient(surf_elev, xhi, psi);

% Surface gradient and surface curvature
% Finding the second order derivatives
[deta_dxx, deta_dxy] = gradient(deta_dx, xhi, psi);
[~, deta_dyy] = gradient(deta_dy, xhi, psi);

% Surface gaussian curvature, the product of the principal curvatures
surf_gaus_curv = (deta_dxx.*deta_dyy-deta_dxy.^2)./...
    (1 + deta_dx.^2 + deta_dy.^2).^2;

% Surface mean curvature, the mean of the principal curvatures
surf_mean_curv = ((1+deta_dy.^2).*deta_dxx + (1+deta_dx.^2).*deta_dyy ...
    - 2*deta_dx.*deta_dy.*deta_dxy)./(1 + deta_dx.^2 + deta_dy.^2).^(3/2);

% Inicializing principal curvatures
surf_prin_curv_1 = zeros(256,256);
surf_prin_curv_2 = zeros(256,256);

% Setting the principal curvatures
for i = 1:256
    for j = 1:256
        surf_II_form = [deta_dxx(i,j) deta_dxy(i,j); ...
            deta_dxy(i,j) deta_dyy(i,j)]...
            ./sqrt(1 + deta_dx(i,j)^2 + deta_dy(i,j)^2);
        prin_curv = eig(surf_II_form);
        surf_prin_curv_1(i,j) = max(prin_curv);
        surf_prin_curv_2(i,j) = min(prin_curv);
    end
end
```

```
% Mean surface elevation
surf_mean = mean(surf_elev,'all');


%% Finding vortices

% Getting the middel eigenvalue at each point (lambda), the vorticity inn
% all trhee directions at each point (omega) and the sum og eignevalues Q
[lambda, omega, Q] = lambda_omega_Q(u, v, w, zz, insp_ind,...
    surf_elev, deta_dx, deta_dy);

% Checking for votices
vortex_pos = zeros(256,256); % Initializing variable
vortex_omega = zeros(256,256);

% When the second eigenvalue (lambda_2) is negative we have a vortex core
% When it is positive we don't, leaving that at 0
% Bathtub vortices will mainly have vorticity in the z directiions
if length(insp_ind) == 1
    vortex_pos = lambda(:,:,1).*(lambda(:,:,1)<0);

    % z-direction vorticity for bathtub vortices
    omega_z = reshape(omega(:,:,1,3),[256 256]);
    vortex_omega(lambda<-cutoff_lambda) = omega_z(lambda<-cutoff_lambda);
else
    vortex_pos = lambda(:,:,z_ind).*(lambda(:,:,z_ind)<0);

    % z-direction vorticity for bathtub vortices
    omega_z = reshape(omega(:,:,z_ind,3),[256 256]);
    vortex_omega(lambda(:,:,z_ind)<-cutoff_lambda) = ...
        omega_z(lambda(:,:,z_ind)<-cutoff_lambda);
end

% Calculating enstrophy
enstrophy = sum(omega.^2,4);

% Defining x and y coordinates for som bathtub vortices
[pos_x_vec, pos_y_vec] = bathtub_pos_vec(data_index);

% Defining some random bathtub vortices locations

true_vortex_strength(data_index) = ...
    sum(enstrophy.*(vortex_pos<-1.5), 'all')*dA;
vortex_elev_strength(data_index) = ...
    sum(surf_elev.*(vortex_pos<-1.5), 'all')*dA;


%% Isotropic Wavelet transformation
% Wavelet name
wname_iso = "mexh";
wname_iso_full = "Mexican␣hat";

% Scale range for transform
scale_range = logspace(0, log10(2^3), 25);

% Wavelet transformation of the surface
cwt_elev_iso = cwtft2(surf_elev, 'wavelet', wname_iso, ...
    'scales', scale_range);
```

```
% Matrix the coefficients at each frequency
wlt_elev_iso = reshape(cwt_elev_iso.cfs(:,:,1,:), ...
    [256 256 length(scale_range)]);
angles_coeff_lev = angle(wlt_elev_iso); % Phase angle of the coefficients

[wlt_iso, wlt_iso_len_scale] = ...
    wavelet_length_scale(scale_range(wlt_scale), surf_elev, wname_iso);

% Vortex position circles from wavelet transformation and vortex strength
% based on the transform of the elevation
[vortex_wlt_strength(data_index), vor_circles, vor_radii] = ...
    wlt_surface_vortices(wlt_elev_iso(:,:,1), surf_mean_curv);

% Calculating vortex strength based on the transform of the elevation
% vortex_wlt_strength(data_index) = vortex_wavelet_strength(vor_circles,...
%     vor_radii, wlt_elev_iso(:,:,1));

fprintf("Timestep␣%5.2f␣done,␣nr␣%3d␣of␣%d␣\n",...
    timestamps(data_index), data_index, length(timestamps))

end


%% Ploting

if data_index == 160

    figure(1), clf
    plot(timestamps, true_vortex_strength./vortex_wlt_strength)
    xlim([timestamps(1) timestamps(end)])
    title("Comparison␣of␣vortex␣strength␣at␣each␣timestep")

else
    figure(1), clf
    pcolor(x_2d, y_2d, surf_elev)
    hold on
    shading flat
    colorbar
    xlim([0,2*pi])
    ylim([0,2*pi])
    xlabel("x␣axis")
    ylabel("y␣axis")
    title("Surface␣elevation␣at␣time␣=␣" + timestamps(data_index))
    hold off

    figure(2), clf
    pcolor(x_2d, y_2d, wlt_elev_iso(:,:,wlt_scale))
    hold on
    contour(x_2d, y_2d, -vortex_pos*0.5*10^-4, [0.9 0.9]*0.5*10^-4, 'r', ...
        'linewidth', 1)
    shading flat
    colorbar
    xlim([0,2*pi])
    ylim([0,2*pi])
    xlabel("x␣axis")
    ylabel("y␣axis")
%     title("Wavelet transform of surface elevation at scale = " ...
%         + scale_range(1))
```

```
    hold off

    figure(3), clf
%     contourf(x_2d, y_2d, surf_gaus_curv, [0 0]*10^-4)
    contour(x_2d, y_2d, wlt_elev_iso(:,:,wlt_scale).*(surf_gaus_curv>0), ...
        [0.5 0.5]*10^-4,'linewidth', 2)
    hold on
    shading flat
%     contour(x_2d, y_2d, wlt_elev_iso(:,:,wlt_scale).*(surf_gaus_curv>0), ...
%         [0.5 0.5]*10^-4,'linewidth', 2)
%     colorbar
    xlim([0,2*pi])
    ylim([0,2*pi])
    xlabel("x␣axis")
    ylabel("y␣axis")
    title("Surface␣gaussian␣curvature")
    hold off

    figure(4), clf
%     contour_levels = [cutoff_lambda cutoff_lambda];
    contourf(x_2d, y_2d, wlt_elev_iso(:,:,wlt_scale), [0.5 0.5]*10^-4)
%     contour(x_2d, y_2d, -vortex_pos*0.5*10^-4, [0.7 0.7]*0.5*10^-4, ...
%         'linewidth', 2)
    hold on
    contour(x_2d, y_2d, -vortex_pos*0.5*10^-4, [0.9 0.9]*0.5*10^-4, 'r', ...
        'linewidth', 1.5)
    viscircles(vor_circles, vor_radii, "LineWidth", 0.8, "Color", 'b', ...
        "LineStyle", ":");
    shading flat
    % colorbar
    xlim([0,2*pi])
    ylim([0,2*pi])
    xlabel("x␣axis")
    ylabel("y␣axis")
%     title("Wavelet transform positive values and attached vortices")
    hold off

    figure(5), clf
    pcolor(x_2d, y_2d, surf_mean_curv)
    hold on
    shading flat
    colorbar
    xlim([0,2*pi])
    ylim([0,2*pi])
    xlabel("x␣axis")
    ylabel("y␣axis")
    title("Surface␣mean␣curvature")
    hold off


%     figure(6), clf
%     contourf(x_2d, y_2d, wlt_elev_iso(:,:,wlt_scale), [1 1]*10^-4)
%     hold on
%     viscircles(vor_circles,vor_radii);
%     contour(x_2d, y_2d, -vortex_pos*0.5*10^-4, [1.5 1.5]*0.5*10^-4,...
%         'linewidth', 1.5)
%     shading flat
%     % colorbar
%     xlim([0,2*pi])
%      ylim([0,2*pi])
%      xlabel("x axis")
%      ylabel("y axis")
%      title("Attached vortex positions from wavelet transformation")
%      hold off

end
```

up- and downwellings detection

```
%% Surface upwelling and downdraghts detection
clear

% Referance scales and flow variables
Re     = 2500;                      % Reynold's Number
Fr     = sqrt(0.01);                % Frode's Number
l_ref  = 2*pi;                      % Reference length
H_bar  = 5*pi;                      % Domain height
a_0    = 0.1;                       % Forcing parameter
u      = a_0*l_ref;                 % Referance velocity
nu     = 1/Re;                      % Kinematic viscosity

% Loading surface elevation and timesteps
[eta0, timestamps] = load_surface_data();

% Cutoff value for veortex inspection
cutoff_lambda = 2;

% Wavelt scale index for inspection (i.e. the index in the scale vector for
% the scale of intereset)
wlt_scale = 1;

% Taylor microscale very close to the surface
[u_rms, taylor_scale] = taylor_length_scale();

% Differential area used for integration
dA = (2*pi/255).^2;

% Loading z coordinates
[~,~,~,~,zz] = load_near_surf_velocities(1);

% Important to note that zz is 1 at the suirface and decreas as we go
% further down in the domain
% Surface mesh
x_range = linspace(0, l_ref, 256);
y_range = linspace(0, l_ref, 256);
[x_2d, y_2d] = meshgrid(x_range, y_range);
[x_ver, z_ver] = meshgrid(x_range, zz);

% Index of horizontal slice for inspection
% insp_ind = 1:length(zz);
insp_ind = 1;
z_ind = 1;

% Preallocation some variables
scars_loc_corr = zeros(1,length(timestamps));
scars_loc_corr_iso = zeros(1,length(timestamps));

%% Velocity field comparison at different timesteps

for data_index = 5
% Data index for the timestamp we are looking at

% Loading subsurface velocities (u,v,w) and the presure (p)
[v,u,w,p,zz] = load_near_surf_velocities(data_index);
```

```
% Surface elevation at this time index
surf_elev = reshape(eta0(data_index,:,:),[256 256]);


%% Finding the derivatives

% Grid variables in the x ang y directions
xhi = linspace(0,l_ref,256);
psi = linspace(0,l_ref,256);

% The relevant derivatives
[deta_dx, deta_dy] = gradient(surf_elev, xhi, psi);

% Calculating velocity derivatives
[du_dx, ~, ~] = vel_derivatives(u, zz, insp_ind,...
    surf_elev, deta_dx, deta_dy);
[~, dv_dy, ~] = vel_derivatives(v, zz, insp_ind,...
    surf_elev, deta_dx, deta_dy);

% Surface gradient and surface curvature
% Finding the second order derivatives
[deta_dxx, deta_dxy] = gradient(deta_dx, xhi, psi);
[~, deta_dyy] = gradient(deta_dy, xhi, psi);

% Surface gaussian curvature, the product of the principal curvatures
surf_gaus_curv = (deta_dxx.*deta_dyy - deta_dxy.^2)./...
    (1 + deta_dx.^2 + deta_dy.^2).^2;

% Surface mean curvature, the mean of the principal curvatures
surf_mean_curv = 0.5*((1+deta_dy.^2).*deta_dxx + (1+deta_dx.^2).*deta_dyy ...
    - 2*deta_dx.*deta_dy.*deta_dxy)./(1 + deta_dx.^2 + deta_dy.^2).^(3/2);

% Inicializing principal curvatures
surf_prin_curv_1 = zeros(256,256);
surf_prin_curv_2 = zeros(256,256);

% Setting the principal curvatures
for i = 1:256
    for j = 1:256
        surf_II_form = [deta_dxx(i,j) deta_dxy(i,j); ...
            deta_dxy(i,j) deta_dyy(i,j)]...
            ./sqrt(1 + deta_dx(i,j)^2 + deta_dy(i,j)^2);
        prin_curv = eig(surf_II_form);
        surf_prin_curv_1(i,j) = max(prin_curv);
        surf_prin_curv_2(i,j) = min(prin_curv);
    end
end


%% Finding up and downwellings

% Surface divergence
surf_div = du_dx + dv_dy;

% Probable location of up and downwellings
[surf_upwelling, surf_downwelling] = surface_wellings_locations(surf_div);

%% Isotropic Wavelet transformation
% Wavelet name
```

```
wname_iso = "mexh";
wname_iso_full = "Mexican␣hat";

% Scale range for transform
scale_range = logspace(0, log10(2^3), 25);

% Wavelet transformation of the surface
cwt_elev_iso = cwtft2(surf_elev, 'wavelet', wname_iso, ...
    'scales', scale_range);

% Matrix the coefficients at each frequency
wlt_elev_iso = reshape(cwt_elev_iso.cfs(:,:,1,:), ...
    [256 256 length(scale_range)]);


%% Nonisotropic wavelet transformation

wname_mexh_non = {"mexh", {2,4,1}}; % Mexican hat wvaelet 2D
wname_morl_non = {"morl", {6,1,1}}; % Morlet wavelet 2D

% Scale range for transform
scale_range = logspace(0, log10(2^3), 25);

% Angles used in the transform
angles = linspace(-pi/2, pi/2, 13);
% angles = [0 15 30 45 60 75 90 105]/180*pi;

cwt_elev_non = cwtft2(surf_elev, 'wavelet', wname_mexh_non, ...
    'angles', angles, 'scales', scale_range);

wlt_elev_non = reshape(cwt_elev_non.cfs, [256 256 ...
    length(scale_range) length(angles)]);

[wlt_non, wlt_non_len_scale] = ...
    wavelet_length_scale(scale_range(wlt_scale),surf_elev,wname_mexh_non);

wlt_scars = zeros(256, 256);

for i = 1:length(angles)
    wlt_scars = wlt_scars + (real(wlt_elev_non(:,:,wlt_scale,i))<-0.9*10^-4);
end

wlt_scars = wlt_scars.*(surf_mean_curv<0);


%% Correltaion between the up and downwellings lovations

% Anisotropic wavelet
% Location matrix of the wellings found by the wavelet transform
wlt_scars_loc = ones(256).*(wlt_scars>1*10^-4);

% Location matrix of the wellings according to the surface divergense
surf_scars_loc = ones(256).*((surf_upwelling>0.1) + (surf_downwelling>0.1));

% Correlations between the two
scars_loc_corr(data_index) = corr2(wlt_scars_loc,surf_scars_loc);


% Isotropic wavelet
```

```
wlt_iso_scars = ones(256).*(wlt_elev_iso(:,:,wlt_scale)<-0.8*10^-4);

% Correlation with the isotropci wavelet
scars_loc_corr_iso(data_index) = corr2(wlt_iso_scars,surf_scars_loc);

fprintf("Timestep␣%5.2f␣done,␣nr␣%3d␣of␣%d␣\n", ...
    timestamps(data_index), data_index, length(timestamps))


end


%% Plotting

if data_index == 160

    figure(1), clf
    plot(scars_loc_corr)
    ylim([0.1 0.7])
%     title("Surface welling location correlation")

    figure(2), clf
    plot(scars_loc_corr_iso)
    ylim([0.1 0.7])
%     title("Surface welling location (isotropic wavelet)")

else

    figure(1), clf
    pcolor(x_2d, y_2d, surf_elev)
    hold on
    shading flat
    contour(x_2d, y_2d, surf_downwelling*10^-4, [0.1 0.1]*10^-4,...
        'g', 'linewidth', 1)
    contour(x_2d, y_2d, surf_upwelling*10^-4, [0.1 0.1]*10^-4,...
        'r', 'linewidth', 1)
%     colorbar
    xlim([0,2*pi])
    ylim([0,2*pi])
    xlabel("x␣axis")
    ylabel("y␣axis")
    title("Surface␣elevation␣at␣time␣=␣" + timestamps(data_index))
    hold off

    figure(2), clf
%     pcolor(x_2d, y_2d, wlt_coeff(:,:,1))
%     contourf(x_2d, y_2d, -wlt_elev_iso(:,:,wlt_scale), [1 1]*10^-4)
%     contourf(x_2d, y_2d, real(-wlt_elev_non(:,:,wlt_scale, wlt_angle)),...
%         [1 1]*10^-4)
    contourf(x_2d, y_2d, wlt_scars, [0.1 0.1]*10^-4)
%     contourf(x_2d, y_2d, -surf_mean_curv, [0.03 0.03])
    hold on
%     contourf(x_2d, y_2d, -wlt_coeff(:,:,1), [1 1]*10^-4)
%     contour(x_2d, y_2d, -surf_mean_curv*10^-4, [0.02 0.02]*10^-4,...
%         'linewidth', 1.5)
    contour(x_2d, y_2d, surf_downwelling*10^-5, [0.1 0.1]*10^-5,...
        'g', 'linewidth', 1)
    contour(x_2d, y_2d, surf_upwelling*10^-5, [0.1 0.1]*10^-5,...
        'r', 'linewidth', 1)
    % contourf(x_2d, y_2d, 10*wlt_coeff(:,:,1), [1 1]*10^-3)
    shading flat
    % colorbar
    xlim([0,2*pi])
    ylim([0,2*pi])
    xlabel("x␣axis")
    ylabel("y␣axis")
    % legend("Wavelet transform", "Downwellings", "Upwellings")
%     title("Wavelet transform negative and posisive values comparison")
    hold off

    figure(3), clf

    subplot(1,2,1)
    pcolor(x_2d, y_2d, surf_scars_loc)
    hold on
    shading flat
    xlim([0,2*pi])
    ylim([0,2*pi])
    xlabel("x␣axis")
    ylabel("y␣axis")
    title("Surface␣wellings␣according␣to␣surface␣divergence")
    hold off

    subplot(1,2,2)
    pcolor(x_2d, y_2d, wlt_scars)
    hold on
    shading flat
    xlim([0,2*pi])
    ylim([0,2*pi])
    xlabel("x␣axis")
    ylabel("y␣axis")
    title("Surface␣wellings␣fond␣from␣wlt␣tranform")
    hold off

    wlt_angle = 7;

    figure(4), clf
    pcolor(x_2d, y_2d, real(wlt_elev_non(:,:,wlt_scale, wlt_angle)))
    hold on
%     contour(x_2d, y_2d, -real(wlt_elev_non(:,:,wlt_scale, wlt_angle)),...
%          [0.9 0.9]*10^-4)
    shading flat
    xlim([0,2*pi])
    ylim([0,2*pi])
    xlabel("x␣axis")
    ylabel("y␣axis")
%     title("Anisotropic mexh transform with angle  = " ...
%         + angles(wlt_angle)/pi*180)
    hold off

    figure(5), clf
    pcolor(x_2d, y_2d, surf_div)
    hold on
    contour(x_2d, y_2d, surf_downwelling*10, [0.001 0.001]*10,...
        'g', 'linewidth', 1)
    contour(x_2d, y_2d, surf_upwelling*10, [0.001 0.001]*10,...
        'r', 'linewidth', 1)
    shading flat
    colorbar
    xlim([0,2*pi])
    ylim([0,2*pi])
    xlabel("x␣axis")
    ylabel("y␣axis")
    title("Surface␣divergence")
    legend("Surface␣Divergence", "Downwellings", "Upwellings",...
        "Location", "best")
    hold off

    figure(6)
    pcolor(x_2d,y_2d,surf_mean_curv)
    hold on
    shading flat
    colorbar
    xlim([0,2*pi])
    ylim([0,2*pi])
    xlabel("x␣axis")
    ylabel("y␣axis")
    hold off

%      % Line on the surface at
%      y = 1;
%      y_ind = round(y*256/(2*pi));
%      figure(6), clf
%      subplot(4,1,1)
%      quiver(reshape(u(:,y_ind,:),[256 59])',reshape(w(:,y_ind,:),[256 59])')
%      xlim([0,256])
%      ylim([45 60])
%
%      subplot(4,1,2)
%      plot(xhi, surf_elev(y_ind,:))
%      xlim([0,2*pi])
%
%      subplot(4,1,3)
%      pcolor(surf_elev)
%      hold on
%      shading flat
%      contour(surf_downwelling*10^-2, [0.1 0.1]*10^-2, 'g', 'linewidth', 1)
%      contour(surf_upwelling*10^-2, [0.1 0.1]*10^-2, 'r', 'linewidth', 1)
%      ylim([y_ind-5,y_ind+5])
%      hold off
%
%      subplot(4,1,4)
%      plot(xhi, surf_div(y_ind,:))
%      xlim([0,2*pi])
%
%      figure(7), clf
% %      quiver(x_2d, y_2d, u(:,:,1), v(:,:,1))
%      pcolor(x_2d, y_2d, w(:,:,1))
%      hold on
%      quiver(x_2d, y_2d, u(:,:,1), v(:,:,1), 'w')
%      contour(x_2d, y_2d, surf_downwelling*10^-2, [0.1 0.1]*10^-2,...
%          'g', 'linewidth', 1)
%      contour(x_2d, y_2d, surf_upwelling*10^-2, [0.1 0.1]*10^-2,...
%          'r', 'linewidth', 1)
%      shading flat
%      xlim([0,2*pi])
%      ylim([0,2*pi])
%      title("Horisontal velocity at the surface")
%      hold off

end
```

Load surface data from the hdf5 file

```
function[eta0, timestamps] = load_surface_data()
    % Scrip to load the surface data into a 3D Matrix containing all the
    % surface elevations extracted from the full data set in
    % get_surface_elevation

    surf_data_info = h5info("surf_data");
    num_of_timestamps = length(surf_data_info.Groups.Datasets);

    timestamps = h5read("surf_data", "/timestamps");
    eta0 = zeros(num_of_timestamps, 256, 256);

    for i = 1:num_of_timestamps
        path = "/eta0/" + num2str(timestamps(i));
        eta0(i,:,:) = h5read("surf_data", path);
    end
end
```

Load near surface velocities and pressure from the hdf5 file

```
function[u,v,w,p,zz] = load_near_surf_velocities(timeindex)
    % Scrip to load the veloviteis and the pressure near the surface into 4
    % 3D matrices containin the
    % surface data into a 3D Matrix containing all the
    % surface elevations extracted from the full data set in
    % get_surface_elevation

    timestamps = h5read("near_surf_velocities", "/timestamps");
    zz = h5read("near_surf_velocities", "/zz");

    height = length(zz);
    % Initializing matrices
    u = zeros(256, 256, height);
    v = u;
    w = u;
    p = u;

    % Path to the velocities' location
    u_path = "/u/" + timestamps(timeindex);
    v_path = "/v/" + timestamps(timeindex);
    w_path = "/w/" + timestamps(timeindex);
    p_path = "/p/" + timestamps(timeindex);

    % Getting near surface velocities from file
    u(:,:,:) = h5read("near_surf_velocities", u_path);
    v(:,:,:) = h5read("near_surf_velocities", v_path);
    w(:,:,:) = h5read("near_surf_velocities", w_path);
    p(:,:,:) = h5read("near_surf_velocities", p_path);

end
```

Calculate the derivatives of the velocities

```
function[du_dx, du_dy, du_dz] = vel_derivatives(u, zz, z_index,...
    surf_elev, deta_dx_2D, deta_dy_2D)

    % Function for finding all derivativatices at a certain elevation.
    % Saves computational saving space since the derivatives are not
    % calculated everywhere.

    H_bar = 5*pi; % initializing domain height
    l_ref = 2*pi; % reference length scale

    % Grid variables in the x ang y directions
    xhi = linspace(0, l_ref, 256);
    psi = linspace(0, l_ref, 256);

    % Differentioation with the grid tranformed variables
    [du_dxhi, du_dpsi, du_dzeta] = gradient(u,xhi,psi,zz);

    % Temporary variable for matrix calculations
    tmp = zeros(256, 256, length(zz));

    % Correction to the x derivative
    for i = z_index
        tmp(:,:,i) = zz(i)*deta_dx_2D; end
    du_dx = du_dxhi(:,:,z_index) - ...
        tmp(:,:,z_index).*du_dzeta(:,:,z_index)./(surf_elev + H_bar);

    % Correction to the y derivative
    for i = z_index
        tmp(:,:,i) = zz(i)*deta_dy_2D; end
    du_dy = du_dpsi(:,:,z_index) - ...
        tmp(:,:,z_index).*du_dzeta(:,:,z_index)./(surf_elev + H_bar);

    % Correction to the z derivative
    du_dz = du_dzeta(:,:,z_index)./(surf_elev + H_bar);
end
```

Vortex cores algorithm and vorticity

```
function[lambda, omega, Q] = lambda_omega_Q(u, v, w, zz, z_index,...
    surf_elev, deta_dx_2D, deta_dy_2D)

% Calculating velocity derivatives
[du_dx, du_dy, du_dz] = vel_derivatives(u, zz, z_index,...
    surf_elev, deta_dx_2D, deta_dy_2D);
[dv_dx, dv_dy, dv_dz] = vel_derivatives(v, zz, z_index,...
    surf_elev, deta_dx_2D, deta_dy_2D);
[dw_dx, dw_dy, dw_dz] = vel_derivatives(w, zz, z_index,...
    surf_elev, deta_dx_2D, deta_dy_2D);

% Initializeing the matrices. For each point in the grid we have a 3x3
% matrix for the strain and rotaion stresses. Giving these 5 dimensions
S = zeros(256,256,length(z_index),3,3);
O = S;

% Setting the values for the stress tensor at each point
S(:,:,:,1,1) = 0.5*(du_dx + du_dx);
S(:,:,:,1,2) = 0.5*(du_dy + dv_dx);
S(:,:,:,1,3) = 0.5*(du_dz + dw_dx);
S(:,:,:,2,1) = 0.5*(dv_dx + du_dy);
S(:,:,:,2,2) = 0.5*(dv_dy + dv_dy);
S(:,:,:,2,3) = 0.5*(dv_dz + dw_dy);
S(:,:,:,3,1) = 0.5*(dw_dx + du_dz);
S(:,:,:,3,2) = 0.5*(dw_dy + dv_dz);
S(:,:,:,3,3) = 0.5*(dw_dz + dw_dz);

% Setting the values for the rotation tensor at each point
O(:,:,:,1,1) = 0.5*(du_dx - du_dx);
O(:,:,:,1,2) = 0.5*(du_dy - dv_dx);
O(:,:,:,1,3) = 0.5*(du_dz - dw_dx);
O(:,:,:,2,1) = 0.5*(dv_dx - du_dy);
O(:,:,:,2,2) = 0.5*(dv_dy - dv_dy);
O(:,:,:,2,3) = 0.5*(dv_dz - dw_dy);
O(:,:,:,3,1) = 0.5*(dw_dx - du_dz);
O(:,:,:,3,2) = 0.5*(dw_dy - dv_dz);
O(:,:,:,3,3) = 0.5*(dw_dz - dw_dz);

% Initializing the eigenvalues matrix
lambda    = zeros(256,256,length(z_index));   % Median eigenvalues matrix
omega     = zeros(256,256,length(z_index),3); % Voticity matrix
Q = lambda;                         % Sum og eigenvalues matrix

% Retrieving second largest eigenvalue (lambda_2) and Q matrix values
for i = 1:256
   for j = 1:256
      for k = z_index
         % Temporary index to make the logic fit
         if length(z_index)==1
            tmp_ind = 1;
         else
            tmp_ind = z_index(k);
         end
         S_2D = reshape(S(i,j,tmp_ind,:,:),[3 3]);
         O_2D = reshape(O(i,j,tmp_ind,:,:),[3 3]);
         tmp = eig(S_2D^2 + O_2D^2);
         lambda(i,j,tmp_ind) = median(tmp);
```

```
            omega(i,j,tmp_ind,:)=[dw_dy(i,j,tmp_ind) - dv_dz(i,j,tmp_ind),...
                                  du_dz(i,j,tmp_ind) - dw_dx(i,j,tmp_ind),...
                                  dv_dx(i,j,tmp_ind) - du_dy(i,j,tmp_ind)];
            Q(i,j,tmp_ind) = -0.5*sum(tmp);
        end
    end
end

end
```

Known attached vortices positions

```
function [pos_x_vec, pos_y_vec] = bathtub_pos_vec(data_index)

% This function lists alle bathtub vortices x and y locations according to
% the timestep given. I had found these locations by hand and recorded them
% here.

switch data_index
    case 1 % 1st timestep in the data
        pos_x_vec = [4.16, 4.52, 1.87, 0.65, 2.93, 5.02];
        pos_y_vec = [6.15, 1.99, 2.31, 0.88, 1.85, 0.60];
    case 2 % 2nd timestep in the data
        pos_x_vec = [4.37, 3.02, 2.04, 4.41];
        pos_y_vec = [1.75, 1.66, 2.35, 1.62];
    case 3 % 3rd timestep in the data
        pos_x_vec = [4.25, 4.01, 5.34, 3.37, 4.29, 4.26];
        pos_y_vec = [1.58, 5.63, 0.15, 1.75, 0.81, 1.44];
    case 4 % 3rd timestep in the data
        pos_x_vec = [4.27, 3.95, 1.17, 4.20, 4.34];
        pos_y_vec = [1.49, 5.75, 0.96, 0.63, 1.34];
    case 5 % 5th timestep in the data
        pos_x_vec = [3.00, 1.20, 1.43, 3.90, 5.93, 4.15, 4.21, 5.12];
        pos_y_vec = [1.46, 0.62, 1.00, 5.85, 6.15, 0.46, 0.62, 6.12];
    case 6 % 6th timestep in the data
        pos_x_vec = [3.05, 1.52, 1.75, 3.83, 5.95, 4.12, 4.10];
        pos_y_vec = [1.40, 0.41, 1.32, 5.96, 6.00, 0.42, 0.25];
    otherwise
        pos_x_vec = [];
        pos_y_vec = [];
end

end
```

Vortex strength by integrating the enstrophy

```
function v_strength = vortex_strength_by_int(vortex_pos, enstrophy, pos_x, pos_y)

% Function to calculate the strenght of a vortex at a ceratin level near
% the surface by integrationg the enstrophy (magnitude of the vorticity
% squared, or omega^2)

% Making a square around the surface structure for inspection
ds = 0.09; % Square half side lenght
% Mesh points number for sqaure half side length
ds_ind = floor(ds*256/(2*pi));

% Center indices for the structure
x_ind = floor(pos_y/(2*pi)*256); % x index for structure
y_ind = floor(pos_x/(2*pi)*256); % y index for structure

% Differential area used for integration
% Here 2*pi/255 is the distance between two grid points in either the x or
% y direction since the grid is structured
dA = (2*pi/255).^2;

% Extracting index on the grid for the relevatn structue
% x interval for vortex structure
x_interval = [x_ind - ds_ind, x_ind + ds_ind];
% y interval for vortex structure
y_interval = [y_ind - ds_ind, y_ind + ds_ind];

% focusing on points close to the vortex core
vortex_pos = reshape(vortex_pos(x_interval(1):x_interval(2)...
    , y_interval(1):y_interval(2)), [2*ds_ind+1 2*ds_ind+1]);
enstrophy  = reshape(enstrophy(x_interval(1):x_interval(2)...
    , y_interval(1):y_interval(2)), [2*ds_ind+1 2*ds_ind+1]);

% Filtering out the enstrophy to only the areas around the vortex core,
% defined by when vortex_pos is bigger than 0
enstrophy_corr = enstrophy.*(vortex_pos<0);

% Integrating enstrophy to find estimate for vortex strength
v_strength = sum(enstrophy_corr,'all').*dA;

% surf(enstrophy_corr)
% shading flat
% colorbar
end
```

Locating up- and downwelling with the surface divergence

```
function [surf_upwelling, surf_downwelling ]= ...
    surface_wellings_locations(surf_div)

% Variable interval space averaging half square length
W = 0.22;

% Maximun indexing for the grid. Since the whole surface is 256 by 256,
% and half square length for the variable interval cannot excede the
% domain
W_ind = ceil(W*256/(2*pi));

% Differential area used for integration
dA = (2*pi/255).^2;

% Surface divergence root mean square
surf_div_rms = sqrt(sum(surf_div.^2, 'all')/(256^2));
surf_div_pos = surf_div.*(surf_div>0);
surf_div_neg = surf_div.*(surf_div<0);

% Initializing some variables
surf_div_mean_pos = zeros(256);
surf_div_mean_neg = zeros(256);
surf_div_var_pos  = zeros(256);
surf_div_var_neg  = zeros(256);

% Variabel interval space-averaging for surface divergence
for i = 1:256
    for j = 1:256
        x_int = max(i-W_ind,1):min(i+W_ind,256);
        y_int = max(j-W_ind,1):min(j+W_ind,256);
        % Variabel interval space-averaging quantity
        surf_div_mean_pos(i,j) = dA/(4*W^2).*...
            (sum(surf_div_pos(x_int,y_int), 'all'));
        surf_div_mean_neg(i,j) = dA/(4*W^2).*...
            (sum(surf_div_neg(x_int,y_int), 'all'));
        % Localized varience for the surface divergence
        surf_div_var_pos(i,j) = surf_div_pos(i,j).^2 - surf_div_mean_pos(i,j).^2;
        surf_div_var_neg(i,j) = surf_div_neg(i,j).^2 - surf_div_mean_pos(i,j).^2;
    end
end

% Criterion for a up or downwelling is that the surface local maximum
% variance is 5 times larger than the surface divergence rms value.
surf_upwelling   = surf_div_var_pos.*(surf_div_var_pos>5*surf_div_rms.^2);
surf_downwelling = surf_div_var_neg.*(surf_div_var_neg>5*surf_div_rms.^2);

end
```

Vortex response in wavelet space of known vortices

```
function [v_wlt_max, v_wlt_int] = vortex_wlt_trans(vortex_pos,...
    wlt_coeff, pos_x, pos_y)

% Function to calculate the strenght of a vortex at a ceratin level near
% the surface by integrationg the enstrophy (magnitude of the vorticity
% squared, or omega^2)

% pos_x and pos_y are coordinates for a surface dimple. Due to the finite
% domain size these values have limits. Minimum for both is ds, and
% maximum is 2*pi-ds since the domain is between 0 and 2*pi

% Making a square around the surface structure for inspection
ds = 0.09; % Square half side lenght
% Mesh points number for sqaure half side length
ds_ind = floor(ds*256/(2*pi));

% The scale chosen is defined by this index. So 1 is the smallest scale,
% and so on till the largest (which depends on the number of scales used in
% the tranformation in the main code
scale_index = 2;

% Center indices for the structure
x_ind = floor(pos_y/(2*pi)*256); % x index for structure
y_ind = floor(pos_x/(2*pi)*256); % y index for structure

% Differential area used for integration. Here 2*pi/255 is the distance
% between two grid points in either the x or y direction since the grid
% is structured
dA = (2*pi/255).^2;

% Extracting index on the grid for the relevatn structue
% x interval for vortex structure
x_interval = [x_ind - ds_ind, x_ind + ds_ind];
% y interval for vortex structure
y_interval = [y_ind - ds_ind, y_ind + ds_ind];

% focusing on points close to the vortex core
vortex_pos = reshape(vortex_pos(x_interval(1):x_interval(2)...
    , y_interval(1):y_interval(2)), [2*ds_ind+1 2*ds_ind+1]);
wlt_coeff  = reshape(wlt_coeff(x_interval(1):x_interval(2)...
    , y_interval(1):y_interval(2),scale_index), [2*ds_ind+1 2*ds_ind+1]);

% Filtering out the enstrophy to only the areas around the vortex core,
% defined by when vortex_pos is bigger than 0
% v_transform = wlt_coeff.*(vortex_pos<0);
v_transform = wlt_coeff.*(wlt_coeff>0);

% Integrating enstrophy to find estimate for vortex strength
v_wlt_int = sum(v_transform,'all').*dA;

v_wlt_max = max(v_transform,[],'all');

% surf(v_transform)
% shading flat
% colorbar
end
```

Length scale of a wavelet

```
function [wlt, wlt_length] = wavelet_length_scale(scale, surf_elev, wname)
% Function to fint the size of the wavelet used at each scale. This is
% based on the cwtft2 function in MATLAB. Since cwtft2 uses fft to
% calculated the fourier transfomr it multiplies the fft og the signal with
% the complex conjugate og the fft of the wavelet to calculate the
% convolution needed for the wavelet transform

if iscell(wname)
    name = wname{1};
    var  = [wname{2}{:}];
else
    name = wname;
    var  = [];
end

fimg    = fft2(surf_elev); % fft of the surface elevation
S       = size(fimg);      % Size of the fft
W       = S(1);            % Width of the fft
% We only use the widht here since our domain is square
W2      = floor((W-1)/2);  % Half width parameter used in cwtft2
W_puls  = 2*pi/W*[ 0:W2  (W2-W+1):-1 ]; % Frequency range in the transform
[xx,yy] = meshgrid(W_puls,W_puls); % Meshgrid of the frequencies
xx      = scale.*xx;       % Scaling of the frequencies
yy      = scale.*yy;       % Scaling of the frequencis

% fft of the wavelet using the scaled frequencies from the fft of the
% original surface elevation
switch name
    case "mexh"
        if isempty(var)
            var = [2,1,1];
            % For the Mexican hat wavelet the first value, here 2, is
            % called the order, it has the effect of adding valleys and
            % tops to the wavelet. I have not found a reason for it yet.
            % The second and third are the stretching if the x and y
            % direction respectively, used for creating asymmetry in the
            % wavelet making it anisotropic
        end
        fft_wlt = scale .* (-2*pi) * (xx.^2 + yy.^2).^(var(1)/2).*...
            exp(-((var(2)*xx).^2 + (var(3)*yy).^2)/ 2);
    case "morl"
        if isempty(var)
            var = [6,1,1];
            % For the non isotropic Morlet wavele the first values is used
            % to adjust the frequency of the sine wave in the gaussina
            % evenlope. The sencond os for stretching the envelope, and the
            % third for stretchin the envelope in just the y axis.
        end
        fft_wlt = exp( - var(2)^2*((xx - var(1)).^2 + (var(3)*yy).^2)/2 );
    otherwise
        error("Not␣implemented␣for␣this␣wavelet␣yet")
end

% Inverse fft of the wavelet to give us the actual wavelet
ifft_wlt = ifft2(conj(fft_wlt));

% Preallocating the wavelet
```

```
wlt = zeros(W,W);

% Rearragin the wadrants of the inverse to remake the wavelet
wlt(1:W/2,1:W/2) = ifft_wlt(end-W/2+1:end, end-W/2+1:end);
wlt(end-W/2+1:end,1:W/2) = ifft_wlt(1:W/2, end-W/2+1:end);
wlt(1:W/2,end-W/2+1:end) = ifft_wlt(end-W/2+1:end, 1:W/2);
wlt(end-W/2+1:end,end-W/2+1:end) = ifft_wlt(1:W/2, 1:W/2);

% Cut of the whole wavelet going throught the peak of the mexican hat or
% other wavelet
switch name
    case "mexh"
        if var(2)>var(3)
            wlt_cut = wlt(W/2+1:end, W/2+1);
        else
            wlt_cut = wlt(W/2+1, W/2+1:end);
        end
    case "morl"
        wlt_cut = real(wlt(W/2+1, W/2+1:end));
end

% Zooming in at the values around the center of the wavelet
% wlt = wlt(W/2-18:W/2+21,W/2-18:W/2+21);

% For the mexican hat wavelet the next part is to calculte the location of
% where the wavelet first crosses the x axis.

for i = 2:W/2-1
    if wlt_cut(i+1)<wlt_cut(i), break, end
end

% Where the wavelet crosses the x-axis
zero_loc = interp1(wlt_cut(1:i), 1:i, 0);

switch name
    case "mexh"
        % Then I used the fact that zero crossin is 1/sqrt(3) parts of the
        % legnth to the bottom of the valley in the mexican hat. I'm using
        % the distande to the walley as my wavelet scale. The lengthscale
        % for the wavelet is then the distance between the two  valleys in
        % the mexican hat wavelet
        wlt_length = 2*(zero_loc*sqrt(3))/W * (2*pi);
    case "morl"
        % The length scale here is the distance between the two closest
        % valleys in the morlet wavelet, since these are is the smalles
        % scale the wavelet can detect. For the whole wavelet, multiply by
        % about half the frequency (var(1)/2). That shoul give a good
        % estimate for the witdh of the whole wavelet
        wlt_length = 4*(zero_loc)/W * (2*pi);
end

end
```

Taylor length scale close to the surface

```
function [u_rms, taylor_scale] = taylor_length_scale()
% This is a funtion to fint the taylor microscale for the turbulence
% structures in the data given by Gou and Shen near the surface. It is
% based on function (36) & (38) from the 2009 paper from Gou and Shen "On
% the generation and measurement of waves and turbulence in simulations of
% the free surface turbulence". In essence we calculate the microscale in
% the bulk region using the forcing used and the Reynolds number. The use
% the conversion funtion in (38) to find the length scale in the free
% region above.

% Forcing term used in the bulk region (the same as the one used in the Gou
% and Shen article from 2010 "INteraction of a deformable free surface with
% statistically steady homogeneous turbulence". It is the same as the one
% in this simulation, but for the different viscosity giving a different
% Reynolds number.
a_0 = 0.1;

% Reynolds number for this simulation
Re = 2500;

% Constant used in the eq 38 in Gou & Shen 2009, for a damping region of
% pi/2 the values for these are listed in table 2 in the same article
C_l = -1.0124;
D_l = 0.2085;
C_u = 4.3016;
D_u = -0.9681;

% B_l = 0.4245;
% B_u = 0.3114;

% This is the distace from the center og the domain where we are tying to
% find the Taylor length scale
z_c = 3*pi/2 + pi/2 + 0.5*pi/2;

% Taylor length scale in the bulk region, see eq 36 in Gou & Shen 2009
bulk_scale = 2.24/sqrt(a_0*Re);
u_rms_bulk = 0.57*a_0*2*pi;

% The taylor scale wery close to the surface
taylor_scale = bulk_scale * exp(C_l + D_l*z_c);
u_rms = sqrt(u_rms_bulk^2 * exp(C_u + D_u*z_c));

% taylor_scale = bulk_scale * B_l * z_c;
% u_rms = u_rms_bulk * B_u / z_c;

end
```

Some 2D wavelet testing

```
% Wavelet transform responses
clear
W = 256;
x_range = linspace(-2*pi,2*pi,W);
y_range = x_range;
[x,y] = meshgrid(x_range,y_range);

func_i = 1;

switch func_i
    case 1
        func = sin(x);
    case 2
        func = sin(x)+sin(y);
    case 3
        func = exp(-(x.^2)/2);
    case 4
        func = exp(-(x.^2+y.^2)/2);
    case 5
        func = 0.5*exp(-((x-2.5).^2)/2)-exp(-((x+2.5).^2)/2);
    case 6
        func = exp(-((x-y.^2).^2)/2);
    otherwise
        disp("Fnnction␣number␣not␣specified")
        func = x;
end

func_scaling = 5;

func = func.*func_scaling;

wname = "mexh";

scale_range = logspace(0, log10(2^3), 25);
wlt_scale = 17;

[wlt, wlt_length_scale] = wavelet_length_scale(scale_range(wlt_scale), func, wname);

disp(wlt_length_scale)

cwt_1 = cwtft2(func, 'wavelet', wname, 'scales', scale_range);
wlt_coeff_1 =  reshape(cwt_1.cfs(:,:,1,:),[W W length(scale_range)]);

[df_dx,  df_dy]  = gradient(func,  x_range, y_range);
[df_dxx, df_dxy] = gradient(df_dx, x_range, y_range);
[~,      df_dyy] = gradient(df_dy, x_range, y_range);

mean_curv = ((1+df_dy.^2).*df_dxx + (1+df_dx.^2).*df_dyy ...
    - 2*df_dx.*df_dy.*df_dxy)./(1 + df_dx.^2 + df_dy.^2).^(3/2);

figure(1), clf
surf(x,y,func)
shading flat
colorbar

figure(2), clf
surf(x,y,wlt_coeff_1(:,:,wlt_scale))
```

```
shading flat
colorbar

figure(3), clf
surf(x,y,df_dxx+df_dyy)
shading flat
colorbar
```

Isolating attached vortices with Hough circle transform

```
function [vor_wlt_strength, vor_circles, vor_radii] = ...
    wlt_surface_vortices(wlt_elev_iso, surf_mean_curv)

% This function tries to isolate the attached vortices at the surface by
% using the wavelet tranform og the surface elevation. It uses Hough circle
% transfomrs to do so.

% Mesh for the original domain
x_orig = linspace(0,2*pi,256);
y_orig = x_orig;
[x_orig,y_orig] = meshgrid(x_orig,y_orig);

% Interpolation mesh
int_size = 2000;
x = linspace(0, 2*pi, int_size);
y = x;
[x,y] = meshgrid(x,y);


% Threshold for the contour
cont_thresh = 0.5*10^-4;

% Interpolating surface elevation for more points to use in later
% algorithms
wlt_elev_int = interp2(x_orig, y_orig, wlt_elev_iso, x, y);
wlt_elev_thresh = ones(int_size).*(wlt_elev_int>cont_thresh);
wlt_elev_pos = wlt_elev_int.*(wlt_elev_int>0);


% contour matrix with all the edges of the contoru isolines
cont_mat = contourc(wlt_elev_int, [cont_thresh cont_thresh]);
cont_loc = {};

% Since the matrix is set up in a wierd way I extract the edges of the
% contour line in a matrix.
i = 1;
cont_num = 1;
while i<length(cont_mat(1,:))
    current_cont = zeros(cont_mat(2,i),2);
    current_it = 1;
    for i = i+1:i+cont_mat(2,i)
        current_cont(current_it,:) = ...
            [round(cont_mat(2,i)),round(cont_mat(1,i))];
        current_it = current_it + 1;
    end
    cont_loc{cont_num} = current_cont;
    cont_num = cont_num + 1;
    i = i + 1;
end

tot_cont = zeros(int_size);
for i = 1:length(cont_loc)
    for j = 1:length(cont_loc{i})
        tot_cont(cont_loc{i}(j,1), cont_loc{i}(j,2)) = 1;
    end
end
```

```
circles = [];
radii = [];
chosen_cont = zeros(int_size);
for radius = 7:45
    current_cont = zeros(int_size);
    circum_up_thresh = radius*2*pi*1.3;
    circum_low_thresh = radius*2*pi*0.8;
    for i = 1:length(cont_loc)
        cont_circum = 0;
        query = cont_loc{i};
        for j = 1:length(cont_loc{i})-1
            cont_circum = cont_circum + sqrt(...
                (query(j+1,1) - query(j,1))^2 +...
                (query(j+1,2) - query(j,2))^2);
        end
        if cont_circum>circum_up_thresh || cont_circum<circum_low_thresh
            continue
        end
%          mean_x_ind = max(mean(query(:,1))*(256/int_size),1);
%          mean_y_ind = max(mean(query(:,2))*(256/int_size),1);
%          if surf_mean_curv(round(mean_x_ind),round(mean_y_ind)) < 0
%              continue
%          end
        for j = 1:length(cont_loc{i})
            current_cont(cont_loc{i}(j,1), cont_loc{i}(j,2)) = 1;
            chosen_cont(cont_loc{i}(j,1), cont_loc{i}(j,2)) = 1;
        end
    end


    method = "TwoStage";
    switch method
        case "TwoStage"
            [circ, rad] = imfindcircles(current_cont, [radius-1 radius+5],...
                "Sensitivity", 0.96, "Method", "TwoStage");
            % Sensitivity for cutoff = 0.83
            % Sensitivity for contour points = 0.92
        case "PhaseCode"
            [circ, rad] = imfindcircles(current_cont, [6, 30],...
                "Sensitivity", 0.95, "Method", "PhaseCode");
        otherwise
            circ = [];
            rad = [];
    end
    if ~isempty(circ)
        circles = [circles; circ];
        radii = [radii; rad];
    end
end

% Removing all circles not in an area of positive surface curvature (since
% the mean curvature has to be positive at an attached vortex)
i = 1;
while i < length(circles)
    x_ind = max(round(circles(i,2)*(256/int_size)),1);
    y_ind = max(round(circles(i,1)*(256/int_size)),1);
    if surf_mean_curv(x_ind,y_ind) < 0
        circles(i,:) = [];
        radii(i) = [];
    else
        i = i + 1;
    end
end

% Setting return coordinates and radii for the circles (2*pi x 2*pi grid)
vor_circles = circles.*(2*pi/int_size);
vor_radii = radii.*(2*pi/int_size);


figure(6), clf
pcolor(chosen_cont)
shading flat
viscircles(circles,radii);


% Preallocating total strength variable
vor_wlt_strength = 0;

% Differential area used in integration
dA = (2*pi/int_size).^2;

% ds = round(max(vor_radii)*2);

for i = 1:length(circles)
    % Side length for integration domain for each vortex
    ds = round(radii(i)*2.5);

    % Indeces for circle centers
    x_ind = round(circles(i,2));
    y_ind = round(circles(i,1));

    % Integration area indeces
    x_inter = [max(x_ind - ds, 1), min(x_ind + ds, int_size)];
    y_inter = [max(y_ind - ds, 1), min(y_ind + ds, int_size)];

    % Preallocating partition of the total area
    par_wlt_elev = wlt_elev_pos(x_inter(1):x_inter(2)...
        ,y_inter(1):y_inter(2));
    for x = 1:length(x_inter)
        for y = 1:length(y_inter)
            if sqrt((x_inter(x) - x_ind)^2 + (y_inter(y) - y_ind)^2) > ds
                par_wlt_elev(x,y) = 0;
            end
        end
    end

    % Integrating wavelet transform in the location of the found vortices
    vor_wlt_strength = vor_wlt_strength + dA*sum(par_wlt_elev,'all');
end

end
```

Initial 1D wavelet transform testing

```
clear
clc

load kobe
load mtlb
load noisdopp

data = noisdopp;
% data(end+1:end+1000) = 0;

%% Continous wavelet transform with filterbank
wname = 'amor';
fb = cwtfilterbank('Wavelet', wname, 'SignalLength',length(data), ...
    'VoicesPerOctave',20, 'Boundary','periodic');
[coeff, freq, cone_of_inf] = wt(fb,data);
% [coeff, freq, cone_of_inf] = cwt(data,wname, 'VoicesPerOctave',20,...
%    'NumOctaves', 7);
t = 0:length(data)-1;

% Plotting the original signal
figure(1), clf
plot(data)

% Plotting of the continous wavelet transform
figure(2), clf
pcolor(t, freq, abs(coeff))
hold on
plot(t, cone_of_inf,'w-','LineWidth',3) % cone of influence

set(gca, 'YScale', 'log')
shading flat
y_max_tick = log10(max(freq));
y_min_tick = log10(min(freq));
y_tick_vec = logspace(y_min_tick,y_max_tick,10);
yticks(y_tick_vec)
yticklabels(arrayfun(@num2str,round(scal2frq(y_tick_vec,'morl')),...
    'UniformOutput',0))
% color_lim = [min(min(abs(coeff))) max(max(abs(coeff)))];
colorbar
% caxis(color_lim)
xlabel("x")
ylabel("scale")
xlim([0 length(data)])

hold off

data_recon = icwt(coeff, wname, freq, [min(freq) max(freq)]);

figure(3), clf
plot(data), hold on
plot(data_recon)
xlabel('x')
ylabel('y')
legend("Original␣signal", "Reconstruction", "Location", "best")
xlim([0 length(data)])
```

```
%% Testing out descrete wavelet transform
wname = 'bior6.8';
[cA, cD] = dwt(data, wname);


% Plotting
figure(4), clf

plot(cA)
hold on
plot(cD)
hold off
%
% set(gca, 'YScale', 'log')
% shading flat
% colorbar
% xlim([0 length(data)])



%% 2D continous wavelet transform
data_woman = load('woman');
data_2D = data_woman.X;
wname = 'morl';
scales = 2.^(0:0.1:5);
angles = 0:pi/8:2*pi-pi/8;
cwtmorl = cwtft2(data_2D, 'wavelet', wname, ...
    'scales',scales, 'angles', angles, 'plot');

figure(5), clf
pcolor(data_2D(end:-1:1,:)), hold on
colormap('pink')
shading flat
axis off
hold off

figure(6), clf
pcolor(abs(cwtmorl.cfs(end:-1:1,:,1,1))), hold on
colormap('pink')
shading flat
hold off

% coeff_2D = [];
% coeff_2D(:,:,:) = cwtmorl.cfs(end:-1:1,:,:);
```

NTNU

Norwegian University of
Science and Technology