Toward an improved method for the interpolation and differentiation of particle tracking velocimetry data

Peter Schilling

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics & Astronautics

University of Washington

2020

Committee:

Dana Dabiri

James Riley

Program authorized to offer degree:
Aeronautics & Astronautics

University of Washington

Abstract

Toward an improved method for the interpolation and differentiation of particle tracking velocimetry data

Peter Schilling

Chair of the supervisory committee:
Dana Dabiri
William E. Boeing Department of Aeronautics & Astronautics

An approach is presented in two and three dimensions for the employment of natural-neighbor techniques to improve the efficiency of interpolating scattered PTV data. Velocity and vorticity were interpolated for an artificial Lamb-Oseen vortex in two dimensions and an artificial Burgers vortex in three dimensions. Gradient calculations were performed using finite differences on the generated interpolants in two and three dimensions; additionally, the natural-neighbor shape functions were differentiated to compute gradients directly in two dimensions. Natural-neighbor interpolation was found to offer a significant savings in both setup and computational time when compared to RBF interpolation. Both methods offered low average relative error across the domains, with the relative accuracy between methods dependent on the problem. Neither technique was found to significantly amplify or filter noise in the data. Further investigation is necessary to refine the use of natural-neighbor methods for PTV, but such algorithms offer a promising direction for accurate rapid interpolation and differentiation of sparse and dense flow information.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 A brief overview of particle tracking velocimetry

The techniques of particle tracking velocimetry (PTV) seek to identify individual particles in a moving fluid over time and use the particle position data to determine, among other quantities, the velocity and vorticity parameters of the flow in question. Although qualitative flow analysis using PTV dates back over five hundred years,[19] it is only in the last few decades that methods have been developed for extracting quantitative information from the associated experimental observations.[1] Since the 1980s, the field has become rich with dozens—if not hundreds—of such methods, each readily taking advantage of the latest software and hardware innovations. A particular milestone in the processing of two-dimensional particle data was the 1996 development of a statistical technique requiring only the theoretical minimum of two sequential flow images to identify individual particles; until then, the state of the art required four frames.[3] Continuing developments into the twenty-first century have sought to combine multiple PTV algorithms into processes that can be tailored and optimized for different flow regimes and conditions.[7]

While traditionally limited to two dimensions, PTV has benefited from recent advancements using multiple high-speed digital cameras and computer processors to bring three-dimensional imaging into a more readily accessible realm.[26] In three dimensions, flows can be analyzed that are directly applicable to a variety of disciplines, including aeronautics, meteorology, and oceanography. A particular advantage of PTV—especially in the three-dimensional context—is that flows with any degree of turbulence, unsteadiness, shear, or complexity can be studied. As the associated technology and techniques have improved, it has in the last decade become possible to generate accurate position and velocity data for

tracked particles in three dimensions in real time using open-source software that can handle a wide range of experimental conditions.[23]

1.2 From particle data to flow fields

Although the development of techniques to identify individual particles and their properties has generated plentiful literature in the last thirty years, comparably less attention has been paid to the interpolation of that data to extract properties at arbitrary points in the flow regardless of whether trackers are present at those points. In that light, this thesis assumes that particles and their velocities have already been identified and focuses on the computational challenge of determining velocities and vorticities at locations away from these known data. For further exposition on the historical development of identification techniques presented in section 1.1, the reader is encouraged to consult Duncan.[13]

Of critical importance to the post-processing of PTV data is that the resulting particle locations are, for the purposes of a numerical differentiation scheme, random. Because the locations therefore do not *a priori* relate to a structured grid, canonical finite-difference methods are not directly usable to determine gradients or interpolants. However, the development of so-called meshfree or meshless methods in recent decades has resulted in several useful algorithms for the interpolation of scattered data.[15] While a variety of these methods can be applied to PTV information, the use of radial basis functions (RBFs) has become widespread in the field. The exceptional accuracy of RBF interpolation has been validated for estimating flow properties away from the measurement points within two-dimensional vortex and shear flows without noise.[13,25] For the calculation of gradients, a standard technique is to use an evenly spaced central difference scheme in which the values at the forward and backward locations are determined via the interpolation method of choice.[25] However, both of these processes have drawbacks that give rise to opportunities for improvement.

1.3 Desired improvements

Despite the demonstrated high accuracy of RBF methods for scattered PTV data interpolation, their computational overhead becomes concerning as the number of particles and the dimensions of the problem increase. Indeed, to construct a global RBF interpolant one must cross-correlate each known data point with every other known data point, creating exponentially more expensive scenarios. To then calculate gradients using a centered difference, the data must at least locally be interpolated onto a structured grid before the quotient can be taken, introducing another source of error on top of that already associated with finite differences. In addition, these techniques have previously been demonstrated to handle noise poorly in PTV applications,[13] which raises research questions about their appropriateness for processing real-world data.

Fortunately, some promising progress is offered by the use of so-called natural-neighbor (NN) methods, which interpolate at an unknown point based only on the nearest known data to that point, as determined by a Voronoi-based weighting scheme.[30] Such methods have gained widespread appreciation in the field of geographic information systems to, for example, construct a terrain map based on a sparse sampling of elevation data.[2] In fact, interpolation of scalar velocity values in a two-dimensional flow at an instant in time is an analogous mathematical problem. What's more, differentiation of the shape equations for natural-neighbor interpolation produces a method that can be used to directly calculate gradients without a gridding step.[31] Fundamentally, the locality and simplicity of natural-neighbor methods make them an attractive alternative in terms of accuracy and computational complexity. Additionally, their linearity and continuity properties imply that they will not add artifacts to the data that do not already exist. Finally, natural-neighbor interpolation has relatively straightforward theoretical extensibility into three dimensions and higher. With these advantages in mind, this thesis explores applications of such modern methods to artificial flows in two and three dimensions, both with and without noise, and compares the results to the standard RBF methods.

Chapter 2

INTERPOLATION METHODS

2.1 Grid-based methods

Grid-based interpolation methods are those that require predefined information about the relationship between the known data points. Techniques are available for interpolating a function and its derivatives on grids with regular or irregular spacing. For instructional simplicity, we focus here on rectilinear grids.

2.1.1 Linear interpolation

A simple approximation scheme for a function of one variable is linear interpolation. Consider a function whose value $f(x)$ is unknown at the point $x = \bar{x}$ but whose values are known at the forward point $\bar{x} + \Delta x$ and backward point $\bar{x} - \Delta x$ (see fig. 2.2). Then, selecting the backward point from which to calculate the slope,

$$\frac{f(\bar{x}) - f(\bar{x} - \Delta x)}{\Delta x} \approx \frac{f(\bar{x} + \Delta x) - f(\bar{x} - \Delta x)}{\Delta x + \Delta x}\,, \tag{2.1}$$

or

$$\tilde{f}(\bar{x}) = \frac{\Delta x f(\bar{x} - \Delta x) + \Delta x f(\bar{x} + \Delta x)}{\Delta x + \Delta x} \approx f(\bar{x})\,. \tag{2.2}$$

If $\Delta x = \Delta x = h$, as for a regularly spaced grid, then the method is just an average of the known data:

$$\tilde{f}(\bar{x}) = \frac{f(\bar{x} - h) + f(\bar{x} + h)}{2}\,. \tag{2.3}$$

Error in linear interpolation

It can be shown[34] that the error in a linear interpolation of $f(x)$ is bounded if $f''(x)$ exists on the domain $\Omega = [\bar{x} - \Delta x, \bar{x} + \Delta x]$. Because the proof involves numerical analysis beyond

the scope of this work, the result is only stated here:

$$\left|\tilde{f}(\bar{x}) - f(\bar{x})\right| \leq \frac{1}{8}(\text{Δ}x + \Delta x)^2 \max_{x\in\Omega} |f''(x)| \,. \tag{2.4}$$

For a regularly spaced grid, we therefore have

$$\left|\tilde{f}(\bar{x}) - f(\bar{x})\right| \leq \frac{1}{2}h^2 \max_{x\in\Omega} |f''(x)| \,. \tag{2.5}$$

Linear interpolation is thus said to be second-order accurate; that is, the magnitude of the difference between the approximation and the true value grows on the order of the square of the grid spacing, assuming that spacing is small.

2.1.2 Bilinear interpolation

Linear interpolation can be extended to arbitrary dimensions (multilinear interpolation). Here, we consider two-dimensional (bilinear) interpolation as an important example. For the function $f(x, y)$ whose value is unknown at $(x, y) = (\bar{x}, \bar{y})$, we require four known data points as presented in fig. 2.1: $(\bar{x} - \Delta x, \bar{y} - \Delta y)$, $(\bar{x} + \text{Δ}x, \bar{y} - \Delta y)$, $(\bar{x} - \Delta x, \bar{y} + \text{Δ}y)$, and $(\bar{x} + \text{Δ}x, \bar{y} + \text{Δ}y)$. The general scheme for multilinear interpolation is to linearly interpolate in each dimension one at a time. For example,

$$\tilde{f}(\bar{x}, \bar{y} - \Delta y) = \frac{\text{Δ}x}{\text{Δ}x + \Delta x} f(\bar{x} - \Delta x, \bar{y} - \Delta y) + \frac{\Delta x}{\text{Δ}x + \Delta x} f(\bar{x} + \text{Δ}x, \bar{y} - \Delta y) \tag{2.6}$$

and

$$\tilde{f}(\bar{x}, \bar{y} + \text{Δ}y) = \frac{\text{Δ}x}{\text{Δ}x + \Delta x} f(\bar{x} - \Delta x, \bar{y} + \text{Δ}y) + \frac{\Delta x}{\text{Δ}x + \Delta x} f(\bar{x} + \text{Δ}x, \bar{y} + \text{Δ}y) \,. \tag{2.7}$$

Then

$$\tilde{f}(\bar{x}, \bar{y}) = \frac{\text{Δ}y}{\text{Δ}y + \Delta y} \tilde{f}(\bar{x}, \bar{y} - \Delta y) + \frac{\Delta y}{\text{Δ}y + \Delta y} \tilde{f}(\bar{x}, \bar{y} + \text{Δ}y) \,. \tag{2.8}$$

In dimensions higher than one, it is useful to construct the problem in a matrix form. Consider that the bilinear approximation can be written

$$\tilde{f}(\bar{x}, \bar{y}) = A + B\bar{x} + C\bar{y} + D\bar{x}\bar{y} \,, \tag{2.9}$$

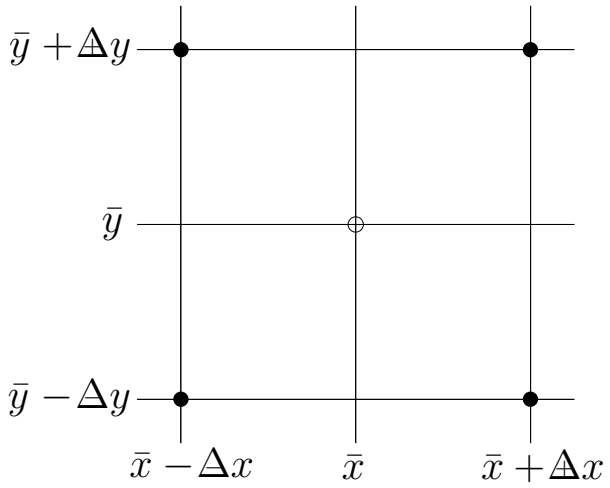


Figure 2.1: Data points for bilinear interpolation. Graphic adapted from LeVeque.[24]

where

$$\begin{bmatrix} 1 & \bar{x}-\Delta x & \bar{y}-\Delta y & (\bar{x}-\Delta x)(\bar{y}-\Delta y) \\ 1 & \bar{x}-\Delta x & \bar{y}+\Delta y & (\bar{x}-\Delta x)(\bar{y}+\Delta y) \\ 1 & \bar{x}+\Delta x & \bar{y}-\Delta y & (\bar{x}+\Delta x)(\bar{y}-\Delta y) \\ 1 & \bar{x}+\Delta x & \bar{y}+\Delta y & (\bar{x}+\Delta x)(\bar{y}+\Delta y) \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} f(\bar{x}-\Delta x, \bar{y}-\Delta y) \\ f(\bar{x}-\Delta x, \bar{y}+\Delta y) \\ f(\bar{x}+\Delta x, \bar{y}-\Delta y) \\ f(\bar{x}+\Delta x, \bar{y}+\Delta y) \end{bmatrix}. \tag{2.10}$$

Error in bilinear interpolation

Similarly to linear interpolation, the error for a bilinear interpolation is written[34]

$$\left|\tilde{f}(\bar{x}) - f(\bar{x})\right| \leq E(\Delta x + \Delta x)^2 \left\|\nabla^2 f(\bar{x}, \bar{y})\right\|, \tag{2.11}$$

where the norm is the Frobenius norm and E is a constant. For a uniform grid spacing, we have

$$\left|\tilde{f}(\bar{x}) - f(\bar{x})\right| \leq Eh^2 \left\|\nabla^2 f(\bar{x}, \bar{y})\right\|. \tag{2.12}$$

Bilinear interpolation is second-order accurate, which continues to hold into higher dimensions.

2.1.3 Finite-difference approximation

We now turn to approximating derivatives at an unknown point based on known function values at other points. To accomplish this analysis, we employ Taylor series. Again following fig. 2.2, we expand the values at the known points centered on the unknown point:

$$f(\bar{x}+\triangleleft\!\!\triangleleft x) = f(\bar{x}) + (\triangleleft\!\!\triangleleft x)f'(\bar{x}) + \frac{1}{2}(\triangleleft\!\!\triangleleft x)^2 f''(\bar{x}) + \frac{1}{6}(\triangleleft\!\!\triangleleft x)^3 f'''(\bar{x}) + \dots , \tag{2.13}$$

$$f(\bar{x}-\Delta x) = f(\bar{x}) - (\Delta x)f'(\bar{x}) + \frac{1}{2}(\Delta x)^2 f''(\bar{x}) - \frac{1}{6}(\Delta x)^3 f'''(\bar{x}) + \dots . \tag{2.14}$$

Now we can add eq. (2.13) to eq. (2.14) and solve for any derivative we desire. Selecting the first derivative, we obtain

$$f'(\bar{x}) = \frac{f(\bar{x}+\triangleleft\!\!\triangleleft x) - f(\bar{x}-\Delta x)}{\triangleleft\!\!\triangleleft x + \Delta x} + \frac{1}{2}(\triangleleft\!\!\triangleleft x - \Delta x)f''(\bar{x}) + \dots \tag{2.15}$$

$$\tilde{f}'(\bar{x}) = \frac{f(\bar{x}+\triangleleft\!\!\triangleleft x) - f(\bar{x}-\Delta x)}{\triangleleft\!\!\triangleleft x + \Delta x} . \tag{2.16}$$

This form is known as the centered-difference approximation, and we see that it is first-order accurate as written. However, on a uniformly spaced grid, where $\triangleleft\!\!\triangleleft x = \Delta x = h$, we recover

$$f'(\bar{x}) = \frac{f(\bar{x}+h) - f(\bar{x}-h)}{2h} + \frac{1}{6}h^2 f'''(\bar{x}) + \dots \tag{2.17}$$

$$\tilde{f}'(\bar{x}) = \frac{f(\bar{x}+h) - f(\bar{x}-h)}{2h} , \tag{2.18}$$

which is second-order accurate. Greater accuracy can be obtained by incorporating expansions at points farther from that under analysis, such as $f(x \pm 2h)$, $f(x \pm 3h)$, etc. It should be noted that while the error in multilinear interpolation arises from an appeal to integral calculus, the error in a finite-difference approximation is simply the result of truncating terms of the associated Taylor series.

2.2 Meshfree methods

In a meshfree method, no predefined information is required about the connections between known data points. This freedom allows the modeling of complex processes requiring moving nodes—such as turbulent fluid flows—without the error effected by deforming the grid or

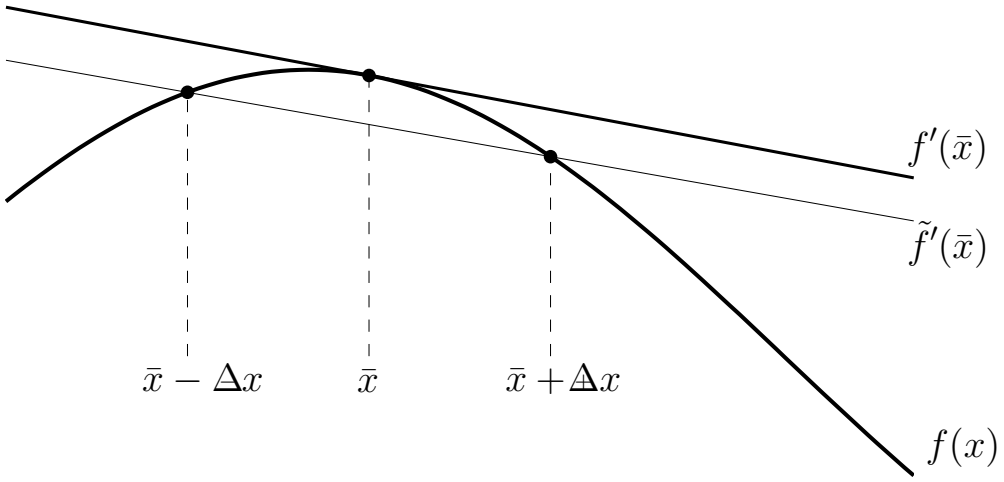


Figure 2.2: Various important characteristics of linear interpolation and finite-difference approximation. Graphic adapted from LeVeque.[24]

having to rebuild it altogether. However, meshfree methods applied to data on a grid cannot take advantage of that grid, which notably leads to error analyses that are less straightforward than those of grid-based methods.

2.2.1 Radial basis functions

Simply defined, a radial basis function φ satisfies the property

$$\varphi(\vec{x}) = \varphi(\|\vec{x}\|)\,, \tag{2.19}$$

where $\|\vec{x}\|$ is usually taken to be the Euclidean norm of the vector $\vec{x}$. The associated interpolation scheme is formed for N known points as trial functions of the form

$$\tilde{f}(\vec{x}) = \sum_{i=1}^{N} \alpha_i \varphi(\|\vec{x} - \vec{x}_i\|)\,, \tag{2.20}$$

where $\vec{x}$ is the point of interest and $\vec{x}_i$ is one of the known data points. The weights α are determined by solving the matrix equation

$$f(\vec{x}_j) = \sum_{i=1}^{N} \alpha_i \varphi(\|\vec{x}_j - \vec{x}_i\|)\,, \quad j = 1, 2 \ldots N\,, \tag{2.21}$$

where $\vec{x}_j$ is also a known point.

The benefits of RBF interpolation are severalfold, and include advantages in existence and uniqueness of solutions, convergence rate, accuracy, simplicity, and general solvability. Also important is the ease of extending the method to arbitrary dimensions. For references to and discussion of the theoretical rigor behind these advantages, the reader would find Chen, et. al.,[9] a good start.

Function definitions

For PTV applications, Casa and Krueger[8] demonstrated good performance using a Gaussian RBF, which is of the form

$$\varphi(\vec{x}) = e^{-\sigma^2\|\vec{x}-\vec{x}_i\|^2}, \tag{2.22}$$

where σ is a measure of the spread of the basis function: increasing its value results in a smoother but possibly less accurate or extensible approximation. Selection of the σ parameter is generally heuristic, based upon the particularities of the problem at hand.[16] Other RBF functions are available, such as multiquadric:

$$\varphi(\vec{x}) = \sqrt{1+\sigma^2\|\vec{x}-\vec{x}_i\|^2} \tag{2.23}$$

and thin-plate spline:

$$\varphi(\vec{x}) = \|\vec{x}-\vec{x}_i\|^2 \ln\left(\|\vec{x}-\vec{x}_i\|\right). \tag{2.24}$$

RBF error analysis

One cannot construct a universal, analytical error scheme for RBF interpolation (or any meshfree method in general) that allows for a global analysis like that of the finite-difference method, as there is no grid upon which to expand. Indeed, error estimates for RBF schemes remain highly dependent on the particularities of their implementation and the problem to which they are applied,[33] and in practical settings an empirical approach is more instructive.[8] In broad terms, the error in an RBF method decreases as the largest gap in the data decreases, but smaller gaps can also lead to poorly conditioned matrices that can cause stability issues.[9]

2.2.2 Natural-neighbor interpolation

In natural-neighbor interpolation, each point to be interpolated is estimated by a weighting scheme based on only the immediately surrounding known data.

The Voronoi diagram

A convenient geometrical representation underlies the gridless method we consider here. The Voronoi diagram in two dimensions partitions the plane into cells whose boundaries surround only that region the plane closer to a given node than to any other. Formally, the Voronoi polygon A around a node x is defined as

$$A(x) = \left\{ x \in \mathbb{R}^2 : \ d(x, x_i) < d(x, x_j) \ \forall j \neq i \right\}, \tag{2.25}$$

where $d(a, b)$ is the Euclidean distance between the points a and b. An example of the resulting diagram is in fig. 2.3.

For application to PTV, we are further interested in the extension of the Voronoi diagram to three dimensions. The higher-dimensional analogue to the Voronoi polygon is the Voronoi cell, which can be equivalently defined:

$$V(\vec{x}) = \left\{ \vec{x} \in \mathbb{R}^3 : \ d(\vec{x}, \vec{x}_i) < d(\vec{x}, \vec{x}_j) \ \forall j \neq i \right\}. \tag{2.26}$$

Here, vector notation is used to emphasize that each point exists in a multi-dimensional space. The three-dimensional Voronoi diagram is a collection of polyhedra whose volumes enclose only those regions closer to a given point than to any other.

In addition to extensions into multiple dimensions, the Voronoi diagram has extensions of order. That is, by incorporating the neighbors of the neighbors and so forth up to the nth neighbor, one can construct Voronoi cells of order n. Of particular importance for the interpolation schemes to be discussed is the second-order Voronoi polygon and polyhedron, respectively defined as

$$A'(x) = \left\{ \vec{x} \in \mathbb{R}^2 : \ d(x, x_i) < d(x, x_j) < d(x, x_k) \ \forall k \neq i, j \right\}. \tag{2.27}$$

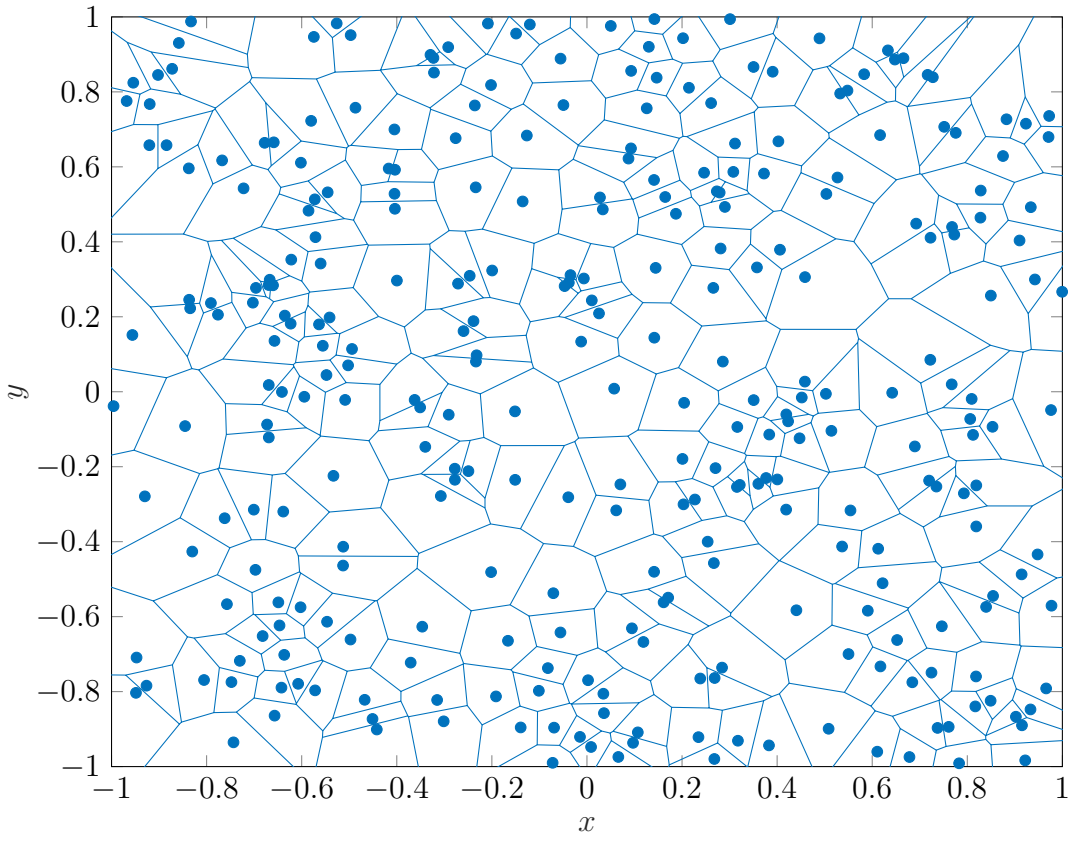


Figure 2.3: Sample two-dimensional Voronoi diagram with randomly located particles.

$$V'(\vec{x}) = \left\{ \vec{x} \in \mathbb{R}^3 : \; d(\vec{x}, \vec{x}_i) < d(\vec{x}, \vec{x}_j) < d(\vec{x}, \vec{x}_k) \; \forall k \neq i, j \right\}. \tag{2.28}$$

This cell is constructed based on the nearest and second-nearest neighbors of the point $\vec{x}$.

Sibsonian interpolation

In Sibsonian interpolation,[29] the point p to be interpolated is placed onto the Voronoi diagram and a series of weights w are determined by overlaying a new Voronoi cell around p and determining the areas of overlap between the new cell and the existing ones. Using the definition of the first- and second-order Voronoi cells,[11]

$$w_i(p) = \frac{A_i'(p)}{A(p)} \tag{2.29}$$

in two dimensions and

$$w_i(\vec{p}) = \frac{V_i'(\vec{p})}{V(\vec{p})}, \tag{2.30}$$

in three dimensions. The subscript indicates that this weight is with respect to a particular preexisting node x_i or $\vec{x}_i$. The interpolated value $\tilde{f}$ is generated by summing the weights over the n natural neighbors of p:

$$\tilde{f}(p) = \sum_{i=1}^{n} w_i(p) f(x_i). \tag{2.31}$$

in two dimensions and

$$\tilde{f}(\vec{p}) = \sum_{i=1}^{n} w_i(\vec{p}) f(\vec{x}_i). \tag{2.32}$$

in three dimensions. Figure 2.4 provides a graphical example of the two-dimensional case.

Non-Sibsonian interpolation

A more recent discovery is termed non-Sibsonian,[4] or Laplacian,[21] interpolation. In the two-dimensional implementation of this scheme, the weights for n natural neighbors are calculated as

$$w_i(p) = \frac{s_i(p)}{h_i(p)} \left(\sum_{j=1}^{n} \frac{s_j(p)}{h_j(p)} \right)^{-1}, \tag{2.33}$$

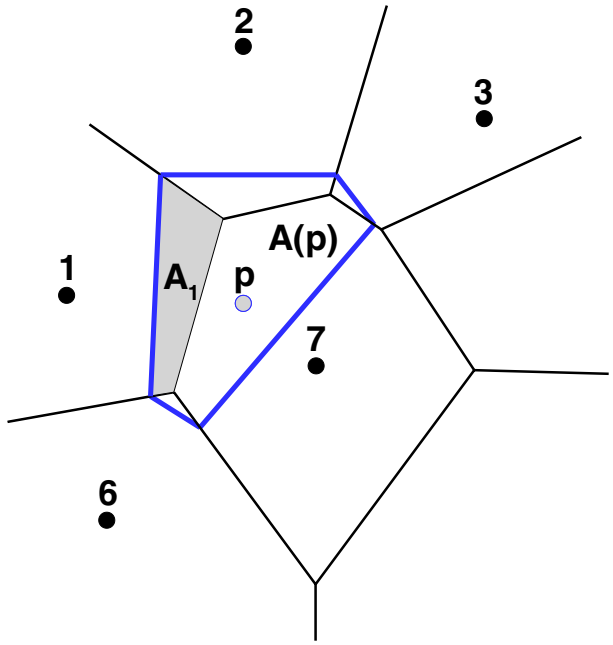


Figure 2.4: Schematic of two-dimensional Sibsonian interpolation at a point p, focused on the first natural neighbor. The shaded area A_1 is that of the associated second-order Voronoi cell. Graphic from Cueto, et. al.[11]

where $s_j(p)$ is the Voronoi polygon side length associated with the jth natural-neighbor node and $h_j(p)$ is the Euclidean distance between p and the jth natural-neighbor node. The interpolated value is then constructed as eq. (2.31). In the three-dimensional implementation of this scheme, the weights for n natural neighbors are calculated as

$$w_i(\vec{p}) = \frac{A_i(\vec{p})}{h_i(\vec{p})} \left(\sum_{j=1}^{n} \frac{A_j(\vec{p})}{h_j(\vec{p})} \right)^{-1}, \tag{2.34}$$

where $A_j(\vec{p})$ is the area of the planar intersection between the overlaid volumetric cell associated with $\vec{p}$ and the cell associated with the jth natural-neighbor node. Figure 2.5 provides a graphical example of the two-dimensional case.

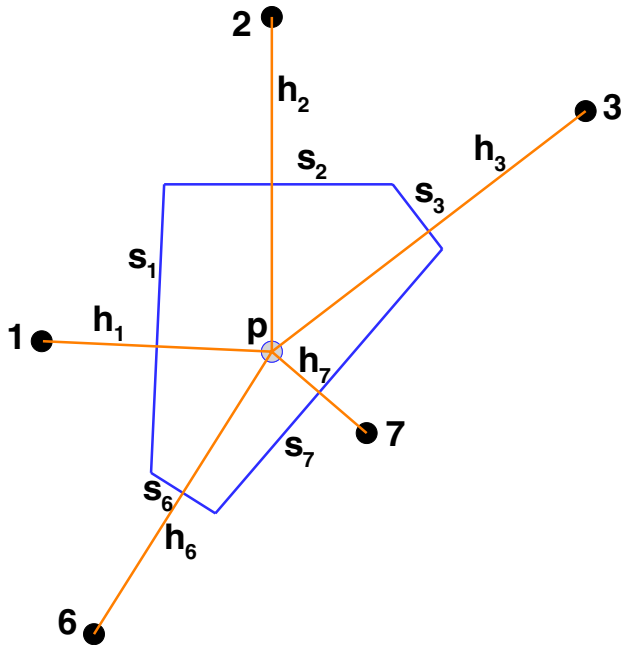


Figure 2.5: Schematic of two-dimensional non-Sibsonian interpolation at a point p analogous to that of fig. 2.4. Graphic from Cueto, et. al.[11]

The primary computational advantage of non-Sibsonian over Sibsonian interpolation is the reduction by one of the maximum dimension required for the computation. That is, in three dimensions, we need only calculate the areas of intersection instead of the volumes of

intersection. In two dimensions, the reduction is from the areas of intersection to the vertices thereof. Although less relevant for the purposes of this thesis, it is also worth noting that the Sibsonian interpolant is $\mathcal{C}^0$ continuous at the data sites and $\mathcal{C}^1$ continuous elsewhere, while the Laplacian interpolant is $\mathcal{C}^0$ continuous everywhere.[22] Also, unlike the Sibsonian interpolant, the Laplacian interpolant is linearly complete, which has important implications for its use in solving partial differential equations.[32] In our context, however, it is sufficient to observe that the same numerical results are obtained by either method, but with significantly less computational expense when using non-Sibsonian interpolation.

Determination of shape functions

Because non-Sibsonian interpolation does not require the computation of areas in two dimensions, a simple geometrically based formula is available to determine s/h as it appears in eq. (2.33).[32] The natural-neighbor nodes must first be numbered clockwise. Then, if $i+1$ is the index of the next clockwise node and $i-1$ the previous, we have

$$\alpha_i = \frac{s_i}{h_i} = |r_i - l_i| \,, \tag{2.35}$$

where

$$r_i = \frac{(x_i - x_{i+1})(x_{i+1} - x) + (y_i - y_{i+1})(y_{i+1} - y)}{(x_i - x)(y_{i+1} - y) - (x_{i+1} - x)(y_i - y)} \tag{2.36}$$

and

$$l_i = \frac{(x_i - x_{i-1})(x_{i-1} - x) + (y_i - y_{i-1})(y_{i-1} - y)}{(x_i - x)(y_{i-1} - y) - (x_{i-1} - x)(y_i - y)} \,. \tag{2.37}$$

It is instructive to note that for the case of four natural neighbors, the above equations reduce to a standard bilinear interpolation.

Derivatives using natural neighbors

If we desire to interpolate a derivative at point p, we begin by differentiating eq. (2.31) to obtain

$$\frac{\partial \tilde{f}(p)}{\partial p_d} = \sum_{i=1}^{n} \frac{\partial w_i(p)}{\partial p_d} f(x_i) \,, \quad d = 1, 2 \,, \tag{2.38}$$

where $p_1 = x$ and $p_2 = y$ are the spacial coordinates of point p. Then substituting $\alpha = s/h$ and applying the product rule to eq. (2.33) we have

$$\frac{\partial w_i(p)}{\partial p_d} = \frac{\left(\sum_{j=1}^{n} \alpha_j\right) \frac{\partial \alpha_i}{\partial p_d} - \alpha_i \frac{\partial}{\partial p_d} \sum_{j=1}^{n} \alpha_j}{\left(\sum_{j=1}^{n} \alpha_j\right)^2}, \quad d = 1, 2 \tag{2.39}$$

$$= \frac{\frac{\partial \alpha_i}{\partial p_d} - w_i(p) \sum_{j=1}^{n} \frac{\partial \alpha_j}{\partial p_d}}{\sum_{j=1}^{n} \alpha_j}, \quad d = 1, 2\,. \tag{2.40}$$

Now, because α_i and α_j are formulated equivalently, it is sufficient to find an expression for $\partial \alpha_i / \partial p_d$, which is the only term we do not already calculate when determining the shape function w_i in eq. (2.33). Differentiating eq. (2.35) yields

$$\frac{\partial \alpha_i}{\partial p_d} = \frac{(r_i - l_i)\left(\frac{\partial r_i}{\partial p_d} - \frac{\partial l_i}{\partial p_d}\right)}{|r_i - l_i|}, \quad d = 1, 2\,, \tag{2.41}$$

where

$$\frac{\partial r_i}{\partial x} = \frac{(y - y_{i+1})((x_i - x_{i+1})^2 + (y_i - y_{i+1})^2)}{((x_i - x)(y_{i+1} - y) - (x_{i+1} - x)(y_i - y))^2}, \tag{2.42}$$

$$\frac{\partial r_i}{\partial y} = -\frac{(x - x_{i+1})((x_i - x_{i+1})^2 + (y_i - y_{i+1})^2)}{((x_i - x)(y_{i+1} - y) - (x_{i+1} - x)(y_i - y))^2}, \tag{2.43}$$

$$\frac{\partial l_i}{\partial x} = \frac{(y - y_{i-1})((x_i - x_{i-1})^2 + (y_i - y_{i-1})^2)}{((x_i - x)(y_{i-1} - y) - (x_{i-1} - x)(y_i - y))^2}, \tag{2.44}$$

and

$$\frac{\partial l_i}{\partial y} = -\frac{(x - x_{i-1})((x_i - x_{i-1})^2 + (y_i - y_{i-1})^2)}{((x_i - x)(y_{i-1} - y) - (x_{i-1} - x)(y_i - y))^2}. \tag{2.45}$$

Errors in natural-neighbor interpolation

As previously mentioned, there is in general no analytical way to determine error bounds for meshfree methods such as natural-neighbor interpolation. However, there are two useful properties inherent to this method that can give some insight. The first is that natural-neighbor interpolant has no error at the known points. Because there is no smoothing applied as in an RBF method, the shape functions at each node collapse to the value at that node. The second is that a bilinear interpolation is recovered in the case of four neighbors. As we

saw previously, multilinear interpolation has second-order accuracy for sufficiently smooth functions. But these are special cases that do not apply in general. For global estimates of error, the modern approach is to use external validation data when available[20] or to use the nodes as a benchmark by evaluating the interpolant successively at each of those points as if the data were not known there.[14] This thesis adopts the former approach, as all the data considered is based on an analytic formula that can produce the actual value at any given point.

A final note should be made regarding error estimates for the natural-neighbor derivative calculation. We have seen that a finite-difference derivative estimate has an inherent truncation error resulting from neglecting higher-order terms of the relevant Taylor series. When calculating derivatives without using a grid, there is nothing meaningful over which to expand a Taylor series. So, perhaps unfortunately, we cannot develop a general order of accuracy for such a method.

Chapter 3

FLOWS INVESTIGATED

3.1 Lamb-Oseen vortex

The Lamb-Oseen vortex[27] is a two-dimensional, viscous, unsteady, axially symmetric flow described by an exact solution to the Navier-Stokes equations. If we assume a velocity field in cylindrical coordinates of the form[12]

$$\vec{v}(r,\theta,z) = \left(0, \frac{\Gamma}{2\pi r} g(r,t), 0\right), \tag{3.1}$$

then the Navier-Stokes equations reduce to

$$\frac{\partial g}{\partial t} = \nu\left(\frac{\partial^2 g}{\partial r^2} - \frac{1}{r}\frac{\partial g}{\partial r}\right), \tag{3.2}$$

where Γ is the circulation and ν is the kinematic viscosity. Using the Laplace transform (or another suitable method) we can find

$$g(r,t) = 1 - e^{\frac{-r^2}{4\nu t}}. \tag{3.3}$$

In time, the flow velocity decays, with the rate of decay increasing with higher viscosity. Upon taking the curl, the vorticity of the Lamb-Oseen vortex is obtained as

$$\vec{\omega}(r,\theta,z) = \left(0, 0, \frac{\Gamma}{4\pi\nu t} e^{\frac{-r^2}{4\nu t}}\right). \tag{3.4}$$

For our analysis, we are interested in a single moment in time, and so set $t = 1$ for convenience. Further, we are free to choose $\Gamma = 2\pi$ and $\nu = 1$, after which the velocity becomes

$$\vec{v}(r,\theta,z) = \left(0, \frac{1}{r}\left(1 - e^{\frac{-r^2}{4}}\right), 0\right) \tag{3.5}$$

and the vorticity becomes

$$\vec{\omega}(r,\theta,z) = \left(0, 0, \frac{1}{2} e^{\frac{-r^2}{4}}\right). \tag{3.6}$$

A graphical representation of such a vortex is in fig. 3.1.

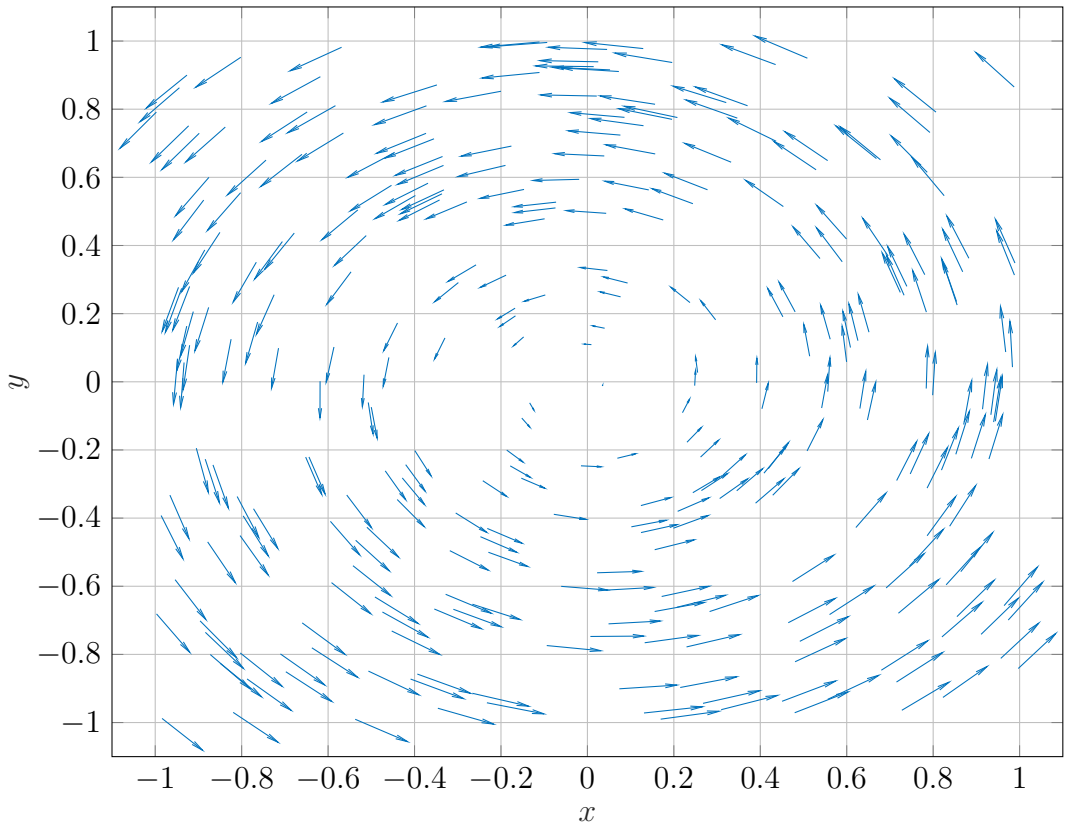


Figure 3.1: Sample Lamb-Oseen vortex from eq. (3.5) with randomly located particles on the domain $x = y = [-1, 1]$.

3.2 Burgers vortex

The Burgers vortex[12] is a three-dimensional, viscous, steady, axially symmetric flow described by an exact solution to the Navier-Stokes equations. If we assume a velocity field in cylindrical coordinates of the form

$$\vec{v}(r,\theta,z) = \left(-kr, \frac{\Gamma}{2\pi r}g(r), 2kz\right), \tag{3.7}$$

then the Navier-Stokes equations reduce to

$$r\frac{\mathrm{d}^2 g}{\mathrm{d}r^2} = -\left(\frac{kr^2}{\nu} - 1\right)\frac{\mathrm{d}g}{\mathrm{d}r}, \tag{3.8}$$

where Γ is the circulation, k is the strain rate, and ν is the kinematic viscosity. The relevant solution is

$$g(r) = 1 - e^{\frac{-kr^2}{2\nu}}. \tag{3.9}$$

The primary characteristic of this flow is vortex stretching, a result of the z velocity being dependent on the z location. Upon taking the curl, the vorticity of the Burgers vortex is obtained as

$$\vec{\omega}(r,\theta,z) = \left(0, 0, \frac{\Gamma}{2\pi\nu}e^{\frac{-kr^2}{2\nu}}\right). \tag{3.10}$$

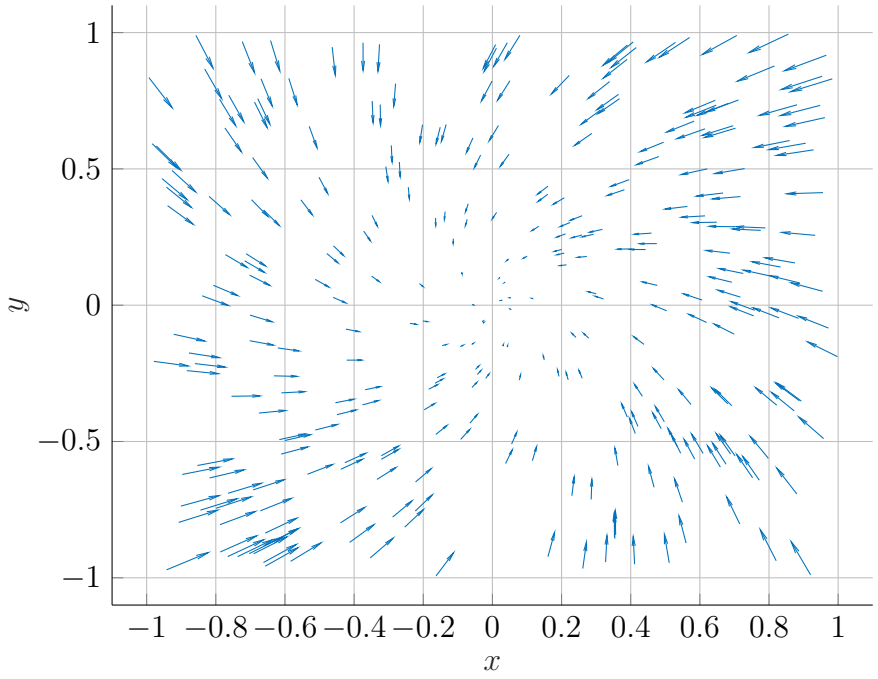
As with the Lamb-Oseen vortex, we are free to choose $\Gamma = 2\pi$ and $\nu = 1$. Additionally, we can choose $k = 1$, after which the velocity becomes

$$\vec{v}(r,\theta,z) = \left(-r, \frac{1}{r}\left(1 - e^{\frac{-r^2}{2}}\right), 2z\right) \tag{3.11}$$

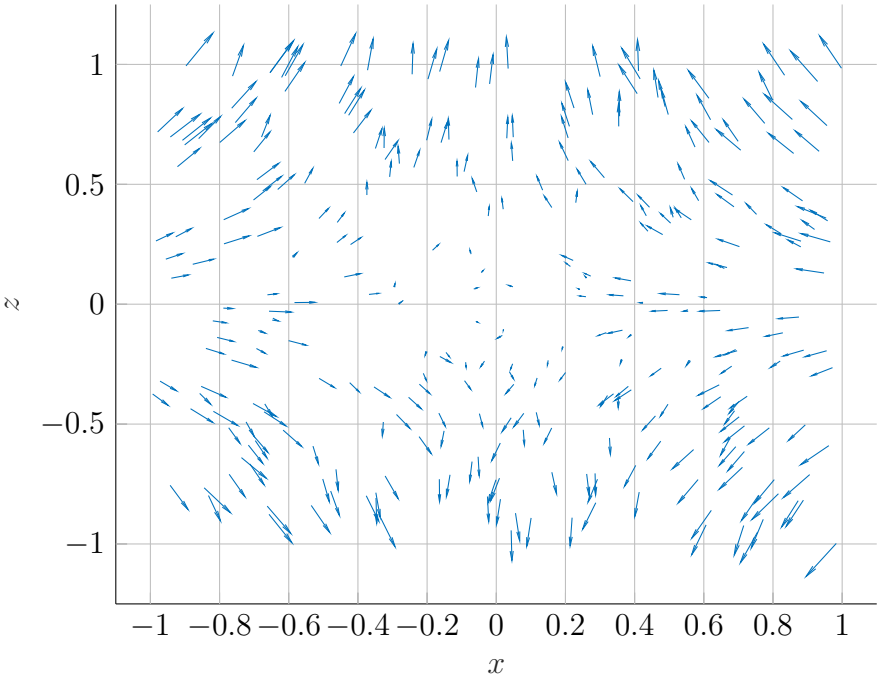
and the vorticity becomes

$$\vec{\omega}(r,\theta,z) = \left(0, 0, e^{\frac{-r^2}{2}}\right). \tag{3.12}$$

A graphical representation of such a vortex is in fig. 3.2.



(a) $x - y$ plane



(b) $x - z$ plane

Figure 3.2: Sample Burgers vortex from eq. (3.11) with randomly located particles on the domain $x = y = z = [-1, 1]$.

Chapter 4
METHODS AND PROCEDURES

The simulation of flows and PTV parameters was performed in MATLAB using a combination of built-in tools, third-party libraries, and scripts written specifically for this research effort. Conditions for each simulation were replicated and processed on a personal computer with 16 gigabytes of DDR3 random-access memory and an Intel Core i7-4700MQ quad-core processor rated at 2.40 GHz.

4.1 Construction of flows

4.1.1 Creation of domains

The simulated domain in two dimensions was a square field with coordinates $x = [-1, 1]$ and $y = [-1, 1]$. In three dimensions, this field was extended to a cube with $z = [-1, 1]$. The axes were divided into P segments apiece, with each segment representing the edge of a single pixel or voxel. For all simulations, $P = 100$ was used as a representative resolution; therefore, the 2D domain comprised 10,000 pixels and the 3D domain 1,000,000 voxels.

4.1.2 Generation and placement of particles

The number of particles per pixel or voxel—the particle density, denoted ρ—was varied across four values representative of typical PTV applications. Table 4.1 lists the densities considered. Each particle was given a random x, y, and z coordinate (as applicable) and placed into the domain with velocity data calculated using the appropriate equation for the given flow: eq. (3.5) for the two-dimensional Lamb-Oseen vortex and eq. (3.11) for the three-dimensional Burgers vortex.

Table 4.1: Number of particles N for each simulated particle density in two and three dimensions.

ρ	N (2D)	N (3D)
0.03	300	30 000
0.05	500	50 000
0.07	700	70 000
0.1	1000	100 000

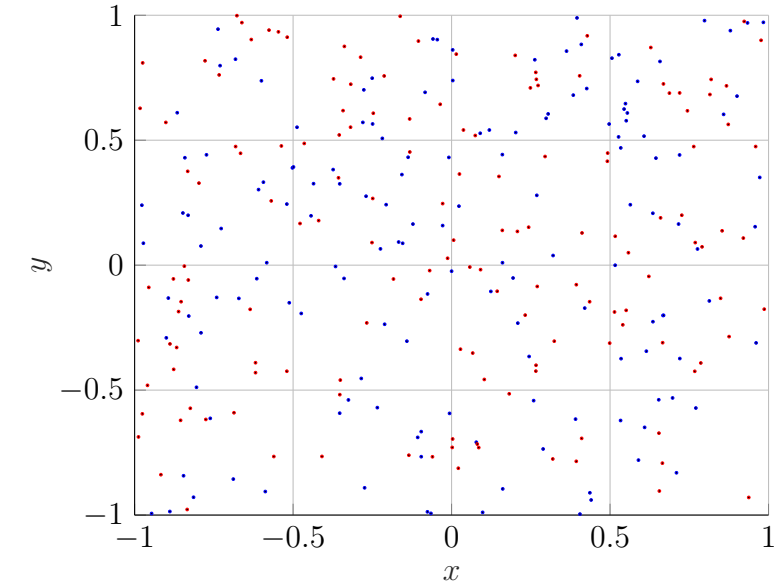
Two sets of N randomized particles were created: the interpolation basis and the interpolation points. The interpolation basis is the simulation of the particles captured during a PTV process; i.e., the measured ("known") data from images. The respective interpolants are built from these points. The interpolation points are those points at which we desire velocity data but do not have it. The respective interpolants are evaluated at these points and the accuracy of the interpolation is ultimately measured from this evaluation. An illustration of each domain is presented in fig. 4.1.

4.2 Construction of interpolants

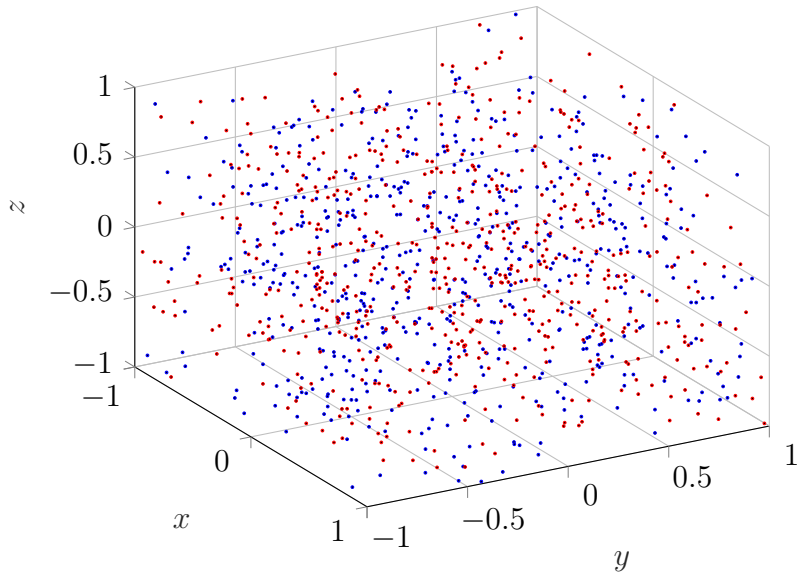
Both RBF and natural-neighbor interpolants were constructed in two and three dimensions using built-in MATLAB functions.

4.2.1 RBF

A Gaussian RBF in the form of eq. (2.22) was employed as the primary baseline. A particular challenge of any RBF application is optimizing the spread constant σ. In our case, a higher particle density inherently creates a shape closer to the desired interpolant, so the spread constant can be relaxed for greater accuracy with minimal loss of extensibility. Using multiple iterations with different values of σ for each particle density and observing qualitatively the shape of the interpolants led to selection of the values of σ in table 4.2. It is important to



(a) Two dimensions



(b) Three dimensions

Figure 4.1: Sample experimental domains with randomly located interpolation basis in blue and interpolation points in red. For the purposes of illustration, particle densities have been reduced from those used in the analysis.

note that these values are only applicable to noiseless data, where each point is guaranteed to lie on the analytical flow solution. Figure 4.2 illustrates the qualitative differences that come from varying the spread constant. The Gaussian RBF is built into MATLAB as a neural

Table 4.2: Spread constant σ used for each particle density ρ in noiseless RBF interpolation.

ρ	σ
0.03	0.9
0.05	0.5
0.07	0.1
0.1	0.09

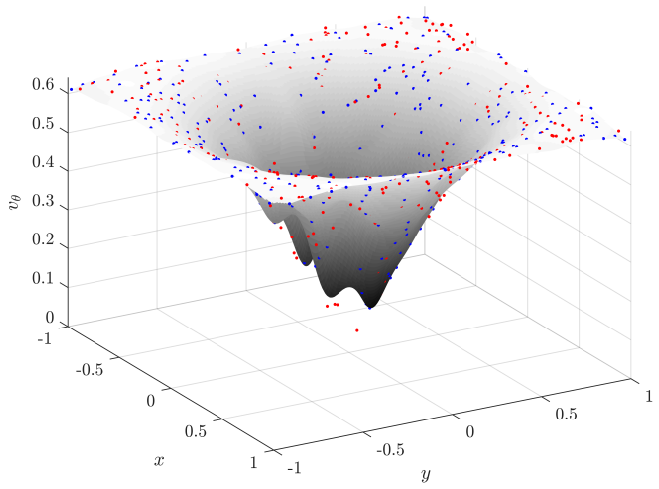
network with the `newrb` function. Other formulations of RBF interpolation are available in the library by Chirokov.[10]

4.2.2 Natural neighbor

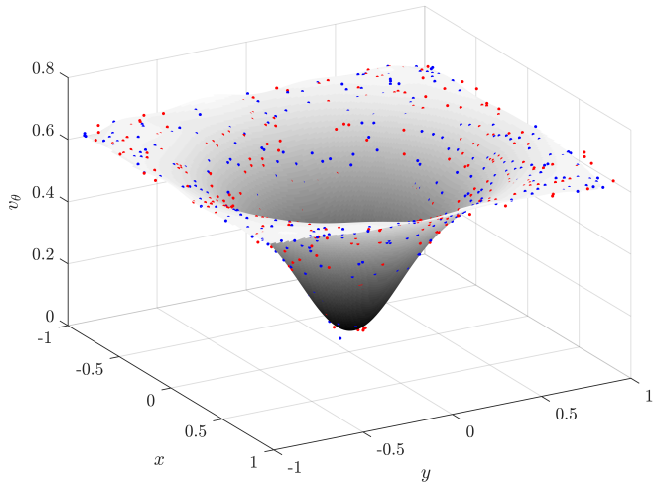
An implementation of natural-neighbor interpolation in two and three dimensions is built into MATLAB's `scatteredInterpolant` function; however, the associated documentation reveals little about the underlying algorithms. Comparison of performance and repeatability to Sakov's open-source natural-neighbor interpolation library[28] as extended for MATLAB by Foster[18] provided confidence that the MATLAB implementation follows the theory presented in this thesis. When possible, the built-in MATLAB function was used for the present research due to its better flexibility and extensibility.

4.3 Interpolation of velocity

After construction of the interpolants using the interpolation basis, velocities were estimated for the interpolation points by inserting their coordinates wholesale into the interpolants,



(a) RBF interpolant with $\rho = 0.03$ and $\sigma = 0.1$.



(b) RBF interpolant with $\rho = 0.03$ and $\sigma = 0.9$.

Figure 4.2: Influence of the spread constant σ on RBF interpolants of sparse data for a Lamb-Oseen vortex, with the interpolation basis in blue and interpolation points in red. A lower σ better captures the exact shape of the interpolation basis but fails to smoothly interpolate the underlying function. A higher σ provides a more extensible interpolant at the expense of some accuracy near the singularity at the origin.

resulting in vectors of interpolated values that could then be compared to the analytical solutions.

4.4 Interpolation of vorticity

4.4.1 Using finite differences

For each of our flows, the vorticity equation reduces to

$$\omega_z = \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y}\,. \tag{4.1}$$

Each partial derivative in eq. (4.1) can be approximated to second-order accuracy by the central difference of eq. (2.18) by using the interpolant to estimate the velocity at the $\pm h$ locations, with the subtraction term in the vorticity equation canceling out any local systematic errors. Theoretically, we should make h as small as possible to achieve the lowest error; in practice, too small a grid spacing can cause inaccuracies related to numerical precision, assuming perfect data. Functionally, selecting the h parameter is essentially a smoothing operation. For imperfect data such as ours, too small a value would be affected by waviness in the interpolant, while too large a value would be affected by the curve of the underlying function. For this analysis, it was found empirically that any value in the range $h = [0.001, 0.01]$ essentially minimized the error.

4.4.2 Using natural neighbors directly

As discussed in section 2.2.2, it is possible to estimate the derivative at a point using only the function values at the natural neighbors. This approach does not require the selection of an h and the associated extra error that arises from interpolating locally on a grid. To implement this method, the library of Sakov[28] was modified to calculate the shape-function derivatives.

4.5 Addition of noise

After validation of the approach on "perfect" data, noise was added by randomly perturbing the position vectors by 1, 2, 3, 4, 5, and 10 pixels in turn. Perturbations were applied along combinations of the x-, y-, and z-axis as applicable. Although all of the equations and functions discussed so far will happily accept noisy input, more analysis and optimization was necessary to make the results meaningful.

4.5.1 Influence on spread constant

The RBF spread constants in table 4.2 are no longer valid on noisy data. With noiseless data, the smoothing of the function occurs naturally at higher particle densities because each velocity lies on a smooth, analytically defined surface. With perturbed data, however, minor variations upset that smoothness. Thus, the heuristic optimization method described in section 4.2.1 was applied again. It was found that $\sigma = 0.7$ provided the optimum balance between smoothing and accuracy for all particle densities. Figure 4.3 presents a visualization of the phenomenon.

4.6 Calculation of error

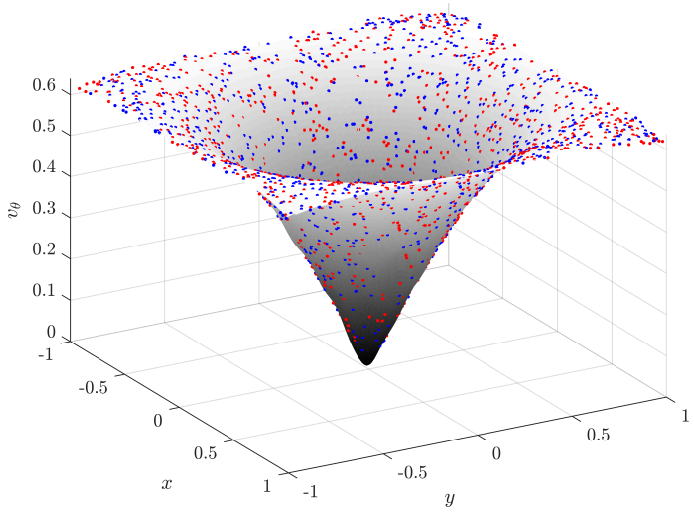
Due to the possibility of velocities extremely close to zero in our scenarios, a measure of relative accuracy is more instructive than one of absolute accuracy. The relative error in velocity is calculated as

$$E_v = \left| \frac{\tilde{v}_\theta - v_\theta}{v_\theta} \right|, \tag{4.2}$$

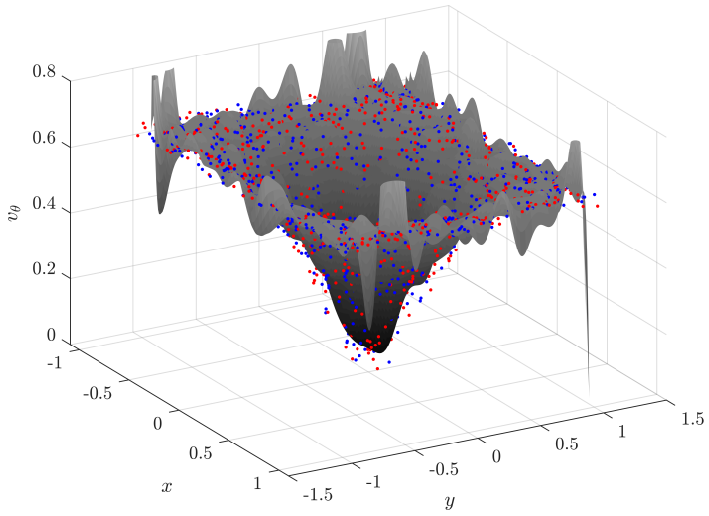
where $\tilde{v}_\theta$ is the velocity obtained via the interpolant and v_θ is the velocity calculated analytically via the appropriate equation from chapter 3. The relative error in vorticity is obtained the same way:

$$E_\omega = \left| \frac{\tilde{\omega}_z - \omega_z}{\omega_z} \right|. \tag{4.3}$$

The mean relative error was calculated across the domain for each case investigated, and the average of up to 100 trials' results was taken in order to provide a representative value.



(a) RBF interpolant with $\rho = 0.1$, $\sigma = 0.09$, and no noise.



(b) RBF interpolant with $\rho = 0.1$, $\sigma = 0.3$, and 10 pixels of random noise.

Figure 4.3: Influence of noise on RBF interpolants of dense data. The interpolation basis is in blue and the interpolation points are in red. With no noise, the smoothing can come from the data itself with a low spread constant, but with noise even a moderate spread constant is unable to map the underlying function.

Chapter 5
RESULTS AND ANALYSIS

This chapter presents numerical and graphical results for computation time and accuracy captured over multiple trials for each interpolation technique in each flow at each particle density. A discussion of the results' interpretation is undertaken, focusing on the relative merits of RBF versus natural-neighbor interpolation. Vorticity calculations in two dimensions are first presented using a finite-difference method and then compared to results for the direct method using natural neighbors, which was not undertaken in three dimensions. A descriptive summary of all results and analysis is presented in section 6.1.

5.1 Two dimensions

5.1.1 Noiseless

Time and accuracy results for the Lamb-Oseen vortex without added noise are tabulated in table 5.1 and compared in table 5.2. Plots of important trends are presented in figs. 5.1 and 5.2. The most striking result is the near-instantaneous construction of the natural-neighbor interpolant. Even for the lowest particle density, the approximately three-second time to build the RBF interpolant is three orders of magnitude greater than the millisecond required to perform the natural-neighbor interpolation. The trends in computation time are also as expected. The RBF computation time increases exponentially with an increase in the number of particles. Because RBF interpolation must solve an $N \times N$ linear system, such behavior matches the associated increase in complexity. The computation time for natural-neighbor interpolation increases linearly, which matches the associated linear theory. For the lower density case, the natural-neighbor method sometimes even outperforms RBF in terms of accuracy. However, for the higher density cases, RBF has clearly superior accuracy. This

Table 5.1: Results of 100 trials of RBF and natural-neighbor (NN) interpolation for a Lamb-Oseen vortex with no noise, with particle density ρ, time t in seconds, and mean percentage errors in velocity and vorticity $\%E_v$ and $\%E_\omega$, respectively. Vorticity evaluated using a finite-difference approach.

	RBF			NN		
ρ	t	$\%E_v$	$\%E_\omega$	t	$\%E_v$	$\%E_\omega$
0.03	2.84	1.48	1.78	0.0010	1.29	1.50
0.05	9.87	0.81	0.88	0.0016	1.02	1.02
0.07	25.65	0.34	0.55	0.0023	0.92	0.79
0.1	81.03	0.28	0.47	0.0034	0.52	0.60

Table 5.2: Comparison of results between RBF and NN methods as tabulated in table 5.1. Symbols are the same as in table 5.1 with the addition of subscripts identifying the methods.

ρ	$t_{\mathrm{RBF}}/t_{\mathrm{NN}}$	$E_{v\mathrm{RBF}}/E_{v\mathrm{NN}}$	$E_{\omega\mathrm{RBF}}/E_{\omega\mathrm{NN}}$
0.03	2.84×10^3	1.15	1.19
0.05	6.17×10^3	0.79	0.86
0.07	1.12×10^4	0.37	0.70
0.1	2.38×10^4	0.54	0.78

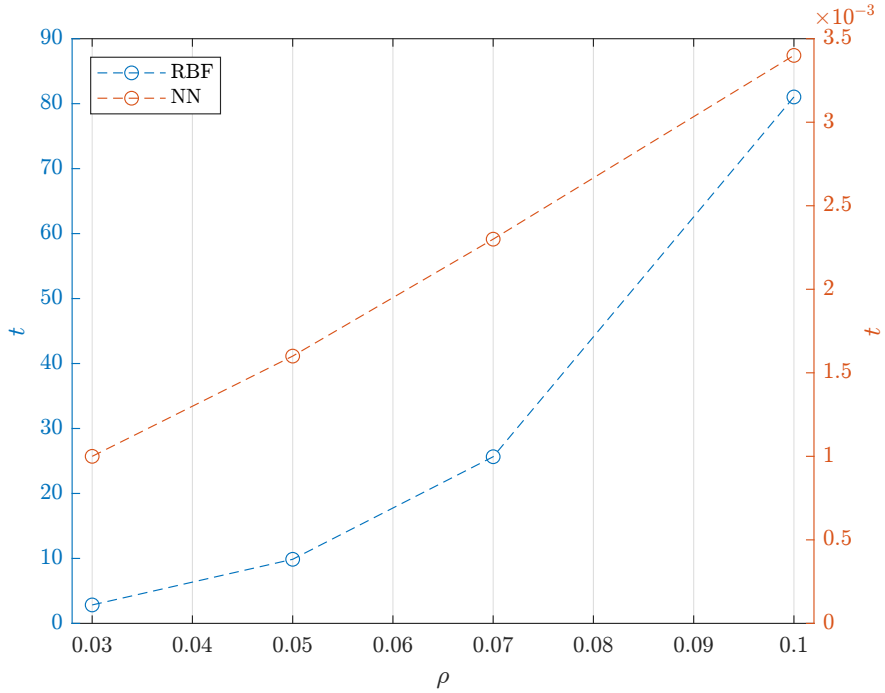


Figure 5.1: Plot of computation time (in seconds) for the Lamb-Oseen vortex. Data from table 5.1.

trend makes sense in that with more data to work with, the smoothing inherent in the RBF interpolant follows the smooth exponential in the underlying vortex function (see fig. 4.2b).

The story is altogether different for sparser data, as can be seen from fig. 5.3. The RBF smoothing handles large gaps in the data by introducing wavy features that depart from the underlying function, which are especially visible in the figure near the edges of the domain, although they exist throughout. Near the singularity, that same smoothing does not permit the interpolant to descend to $v_\theta = 0$ sharply enough to accurately capture the data there, even if there is a point of the interpolation basis very close to $x = y = 0$. On the other hand, the natural-neighbor interpolation provides essentially linear connections among the interpolation-basis points, and thus does not introduce any waviness. It also happily connects to data points close to the origin, allowing the sharpness there to be defined.

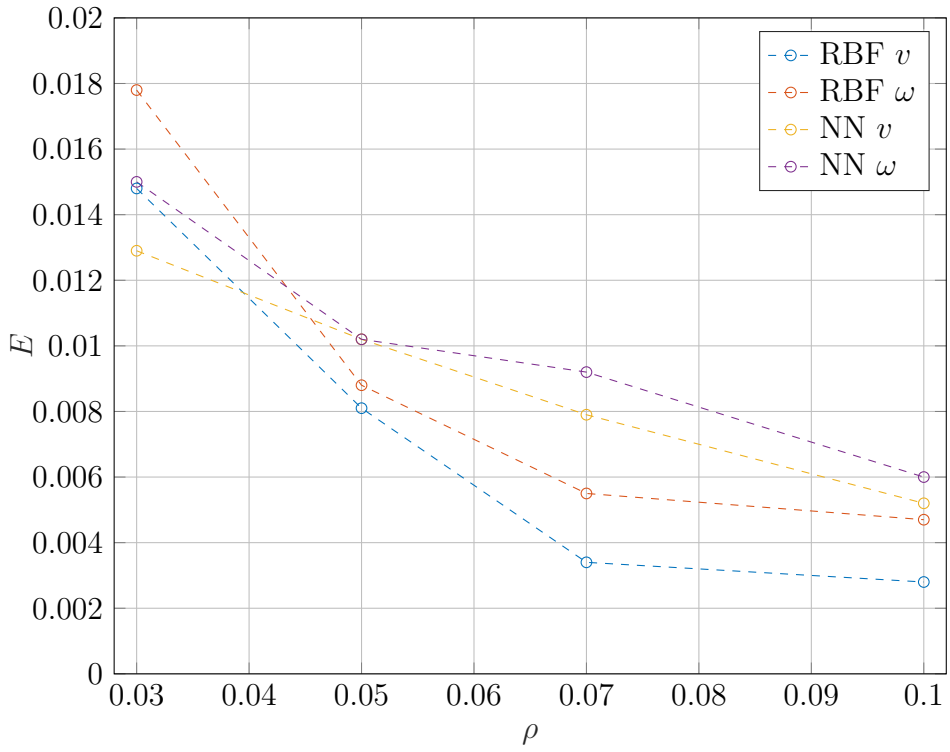
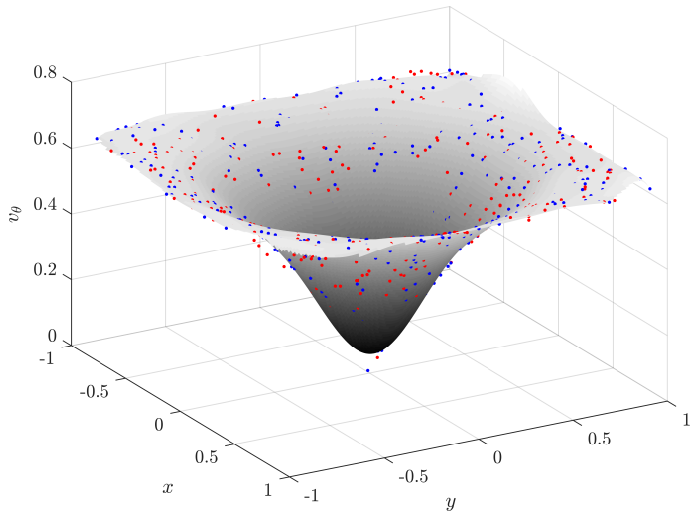
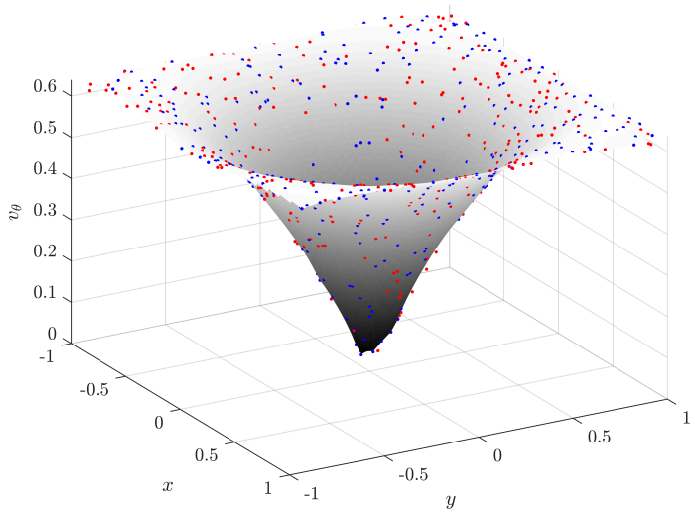


Figure 5.2: Plot of errors for the Lamb-Oseen vortex. Data from table 5.1.



(a) RBF interpolant with $\rho = 0.03$, $\sigma = 0.9$, and no noise.



(b) Natural-neighbor interpolant with $\rho = 0.03$ and no noise.

Figure 5.3: Comparison of RBF and natural-neighbor interpolant shapes at low particle density. The interpolation basis is in blue and interpolation points are in red.

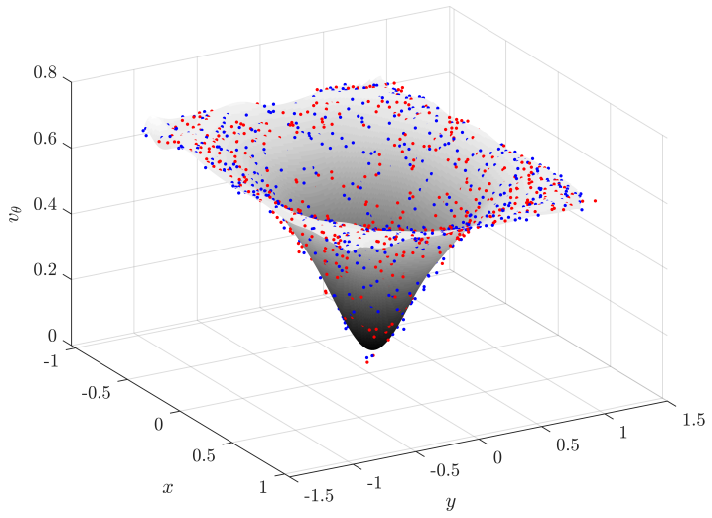
5.1.2 Noisy

Time and accuracy results for the Lamb-Oseen vortex with added noise are presented in table 5.3 and compared in table 5.4. Plots of important trends are presented in figs. 5.5 and 5.6. No pre- or post-processing was applied to the noisy data; any smoothing arose solely from the properties of the two interpolants. For each respective noise value and interpolant, we see the same general trends as for the noiseless data: greater processing times and accuracy for higher particle densities. However, there is more variability in the relative performance between the RBF and natural-neighbor methods. For the lowest noise value, NN outperforms RBF for velocity calculations at both the lowest and highest particle densities, with approximate parity at the middle densities. In all cases, NN outperforms RBF for both velocity and vorticity at $\rho = 0.03$ and $\rho = 0.05$.

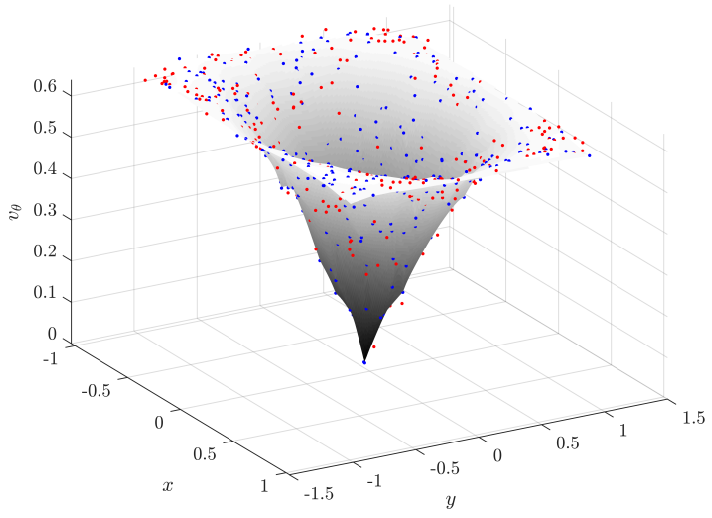
For vorticity computations, the finite-difference calculation based on the RBF interpolant generally outperforms that based on the NN interpolant, especially as the noise level increases. Again, however, we see that NN consistently outperforms RBF for the sparsest data. We can begin to explain the vorticity results by looking at fig. 5.4. The smoothing inherent to the RBF method tends to cancel out the random variations due to the noise on the local level over which the finite-difference method operates. The natural-neighbor interpolant, however, does not guarantee any such local (or global) smoothness. The jaggedness of the interpolant can cause very high gradients that are more likely to affect a derivative calculation than they are the simple interpolation of velocity.

5.1.3 Comparison of vorticity calculations

Up to this point, we have explored results for vorticity calculations undertaken using a finite-difference method on each interpolant. However, as discussed in section 2.2.2, it is possible to use the natural-neighbor method to calculate a derivative directly. Table 5.5 presents results for the clean case. The improvement is substantial, especially at the lower particle densities, although the RBF combined with a finite-difference still wins out for higher particle



(a) RBF interpolant with $\rho = 0.07$, $\sigma = 0.7$, and $p = 5$ pixels of noise.



(b) Natural-neighbor interpolant with $\sigma = 0.7$, and $p = 5$ pixels of noise.

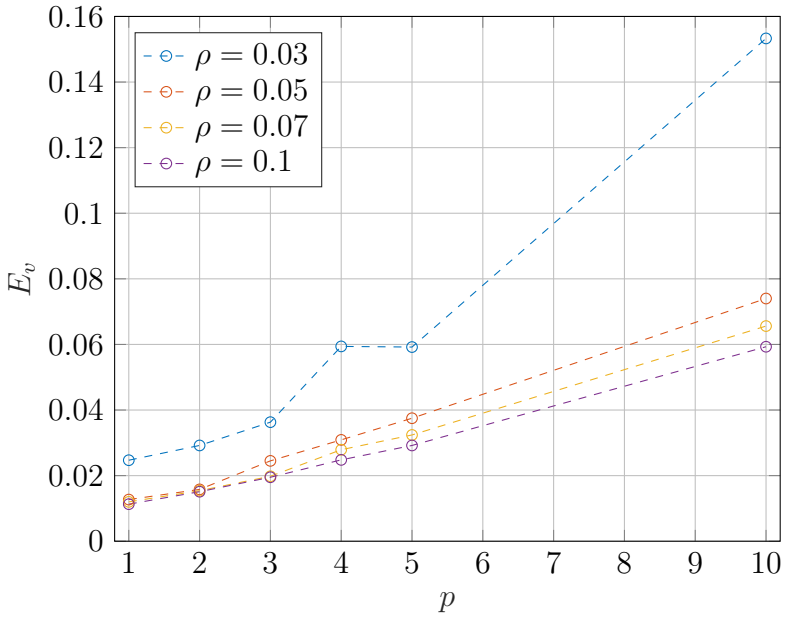
Figure 5.4: Comparison of RBF and natural-neighbor interpolant shapes at medium particle density and medium noise. The interpolation basis is in blue and interpolation points are in red.

Table 5.3: Results of 100 trials of RBF and natural-neighbor (NN) interpolation for a Lamb-Oseen vortex with random noise offset p pixels. Other symbols are the same as in table 5.1. Vorticity evaluated using a finite-difference approach.

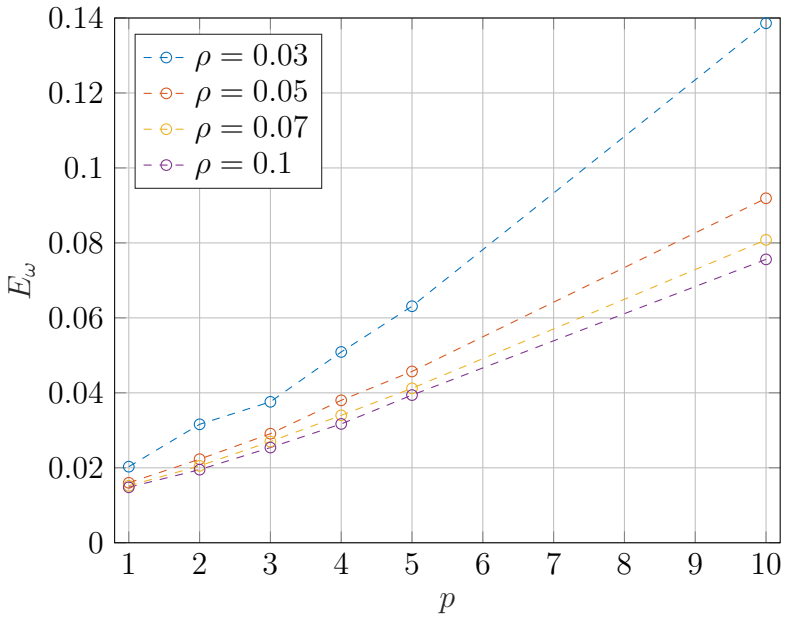
		RBF			NN		
p	ρ	t	$\%E_v$	$\%E_\omega$	t	$\%E_v$	$\%E_\omega$
1	0.03	3.02	2.47	2.03	0.0015	1.92	1.84
1	0.05	10.22	1.27	1.60	0.0016	1.35	1.45
1	0.07	26.83	1.13	1.52	0.0019	1.18	1.30
1	0.1	87.82	1.20	1.48	0.0044	0.93	1.21
2	0.03	2.94	2.92	3.16	0.0013	2.08	2.52
2	0.05	10.17	1.58	2.23	0.0016	1.79	2.30
2	0.07	26.73	1.54	2.05	0.0022	1.59	2.20
2	0.1	86.72	1.51	1.95	0.0032	1.49	2.10
3	0.03	3.04	3.63	3.76	0.0015	2.68	3.44
3	0.05	10.30	2.45	2.91	0.0023	2.31	3.19
3	0.07	27.01	1.98	2.70	0.0025	2.20	3.16
3	0.1	87.85	1.95	2.54	0.0038	2.18	3.09
4	0.03	2.93	5.94	5.09	0.0010	3.17	4.31
4	0.05	10.18	3.09	3.80	0.0019	2.93	4.19
4	0.07	26.65	2.79	3.40	0.0019	2.74	4.11
4	0.1	86.31	2.48	3.17	0.0037	2.70	4.07
5	0.03	2.95	5.92	6.31	0.0015	4.15	5.28
5	0.05	10.22	3.75	4.57	0.0020	3.48	5.17
5	0.07	26.53	3.24	4.12	0.0029	3.45	5.09
5	0.1	87.31	2.92	3.94	0.0038	3.29	5.05
10	0.03	3.00	15.33	13.86	0.0018	6.79	10.06
10	0.05	10.31	7.40	9.19	0.0020	6.46	10.18
10	0.07	26.69	6.56	8.08	0.0023	6.43	10.17
10	0.1	86.59	5.93	7.56	0.0040	6.48	9.66

Table 5.4: Comparison of results between RBF and NN methods as tabulated in table 5.3. Symbols are the same as in table 5.1 with the addition of subscripts identifying the methods.

p	ρ	$t_{\mathrm{RBF}}/t_{\mathrm{NN}}$	$E_{v\mathrm{RBF}}/E_{v\mathrm{NN}}$	$E_{\omega\mathrm{RBF}}/E_{\omega\mathrm{NN}}$
1	0.03	2.01×10^3	1.29	1.10
1	0.05	6.39×10^3	0.94	1.10
1	0.07	1.41×10^4	1.02	1.17
1	0.1	2.00×10^4	1.22	1.22
2	0.03	2.27×10^3	1.40	1.25
2	0.05	6.36×10^3	0.88	0.97
2	0.07	1.21×10^4	0.97	0.93
2	0.1	2.71×10^4	1.01	0.93
3	0.03	2.03×10^3	1.35	1.09
3	0.05	4.48×10^3	1.06	0.91
3	0.07	1.08×10^4	0.90	0.85
3	0.1	2.31×10^4	0.89	0.82
4	0.03	2.93×10^3	1.87	1.18
4	0.05	5.36×10^3	1.05	0.91
4	0.07	1.40×10^4	1.02	0.83
4	0.1	2.33×10^4	0.92	0.78
5	0.03	1.97×10^3	1.43	1.20
5	0.05	5.11×10^3	1.08	0.88
5	0.07	9.15×10^3	0.94	0.81
5	0.1	2.30×10^4	0.89	0.78
10	0.03	1.67×10^3	2.26	1.38
10	0.05	5.16×10^3	1.15	0.90
10	0.07	1.16×10^4	1.02	0.79
10	0.1	2.16×10^4	0.92	0.78

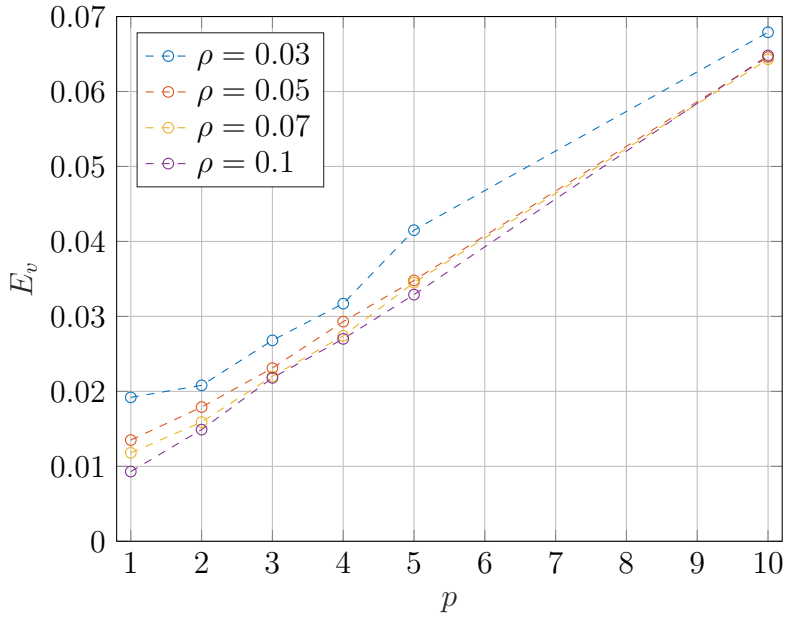


(a) velocity

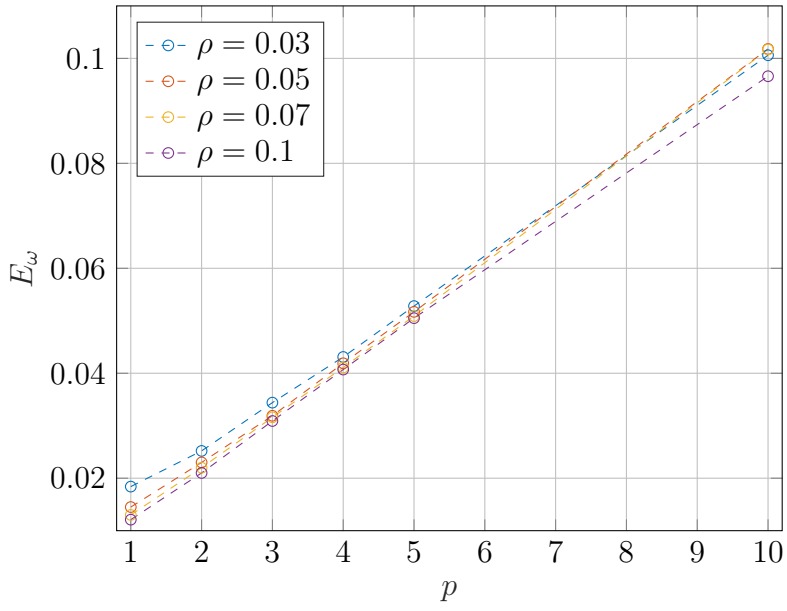


(b) vorticity

Figure 5.5: Trends in RBF interpolation error for a Lamb-Oseen vortex as a result of p pixels of random noise. Data from table 5.3.



(a) velocity



(b) vorticity

Figure 5.6: Trends in natural-neighbor interpolation error for a Lamb-Oseen vortex as a result of p pixels of random noise. Data from table 5.3.

Table 5.5: Comparison of natural-neighbor direct (dir) vorticity interpolation versus finite difference (FD) for a Lamb-Oseen vortex with no noise, with particle density ρ.

ρ	$\%E_{\omega\mathrm{dir}}$	% improvement over FD	$E_{\omega\mathrm{RBF}}/E_{\omega\mathrm{dir}}$
0.03	1.15	23.33	1.55
0.05	0.81	20.59	1.09
0.07	0.75	18.48	0.73
0.1	0.50	16.67	0.94

densities. It makes sense that the accuracy would increase: for the finite-difference method, the derivative is calculated using four points that are themselves interpolated, compounding potential sources of error. The direct method not only avoids this extra interpolation, but also can use information from as many natural neighbors as exist, which is likely more than four in our simulations.

Further, as listed in table 5.6, the extra accuracy that comes with the direct method is enough to beat out RBF for noisy data. The aforementioned use of more than four points is especially important here, as the Gaussian distribution of randomness for the offsets should tend to the mean with an increasing amount of data available.

5.2 Three dimensions

5.2.1 Noiseless

Time and accuracy results for the Burgers vortex without added noise are presented in table 5.7 and compared in table 5.8. The number of trials for each set of conditions was restricted due to computation time, and is listed in table 5.7. Plots of important trends are presented in figs. 5.7 and 5.8. In this situation, the natural-neighbor method clearly outperforms RBF for velocity interpolation in terms of both accuracy and computation time. While RBF takes over sixty hours to compute for the highest particle density, NN manages

Table 5.6: Comparison of natural-neighbor direct (dir) vorticity interpolation versus finite difference (FD) for a Lamb-Oseen vortex with a random noise offset of p pixels and particle density ρ.

p	ρ	$\%E_{\omega\mathrm{dir}}$	% improvement over FD	$E_{\omega\mathrm{RBF}}/E_{\omega\mathrm{dir}}$
1	0.03	1.57	22.66	1.29
1	0.05	1.25	21.88	1.28
1	0.07	1.22	19.74	1.25
1	0.1	1.20	18.92	1.23
2	0.03	2.43	23.10	1.30
2	0.05	1.76	21.08	1.27
2	0.07	1.65	19.51	1.24
2	0.1	1.59	18.46	1.23
3	0.03	2.91	22.61	1.29
3	0.05	2.29	21.31	1.27
3	0.07	2.17	19.63	1.24
3	0.1	2.06	18.90	1.23
4	0.03	4.01	21.22	1.27
4	0.05	3.01	20.79	1.26
4	0.07	2.73	19.71	1.25
4	0.1	2.59	18.30	1.22
5	0.03	4.98	21.08	1.27
5	0.05	3.62	20.79	1.26
5	0.07	3.31	19.66	1.24
5	0.1	3.21	18.53	1.23
10	0.03	10.95	21.00	1.27
10	0.05	7.34	20.13	1.25
10	0.07	6.48	19.80	1.25
10	0.1	6.13	18.92	1.23

Table 5.7: Results of n trials of RBF and natural-neighbor (NN) interpolation for a Burgers vortex with no noise, with particle density ρ, time t in seconds, and percentage errors in velocity and vorticity $\%E_v$ and $\%E_\omega$, respectively.

	RBF				NN			
ρ	n	t	$\%E_v$	$\%E_\omega$	n	t	$\%E_v$	$\%E_\omega$
0.03	25	8.43×10^3	1.10	1.40	100	0.5199	0.45	1.43
0.05	20	2.84×10^4	0.88	1.06	100	0.8521	0.32	1.18
0.07	10	7.29×10^4	0.69	0.82	100	1.2269	0.27	0.99
0.1	5	2.12×10^5	0.52	0.61	100	1.7935	0.21	0.85

Table 5.8: Comparison of results between RBF and NN methods as tabulated in table 5.1. Symbols are the same as in table 5.7 with the addition of subscripts identifying the methods.

ρ	$t_{\mathrm{RBF}}/t_{\mathrm{NN}}$	$E_{v\mathrm{RBF}}/E_{v\mathrm{NN}}$	$E_{\omega\mathrm{RBF}}/E_{\omega\mathrm{NN}}$
0.03	1.62×10^4	2.44	0.98
0.05	3.33×10^4	2.75	0.90
0.07	5.94×10^4	2.56	0.83
0.1	1.18×10^5	2.48	0.72

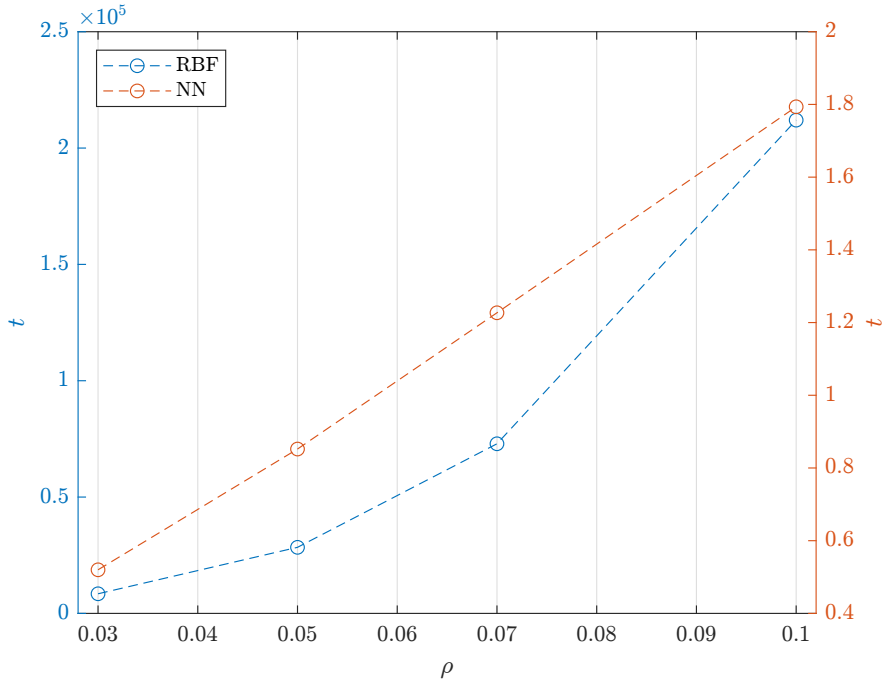


Figure 5.7: Plot of computation time (in seconds) for the Burgers vortex. Data from table 5.7.

the task in under two seconds. For all values of ρ, RBF exhibits over twice the error as NN. It is unfortunate for the three-dimensional case that we cannot visualize the four-dimensional interpolants to better understand the discrepancy in accuracy, but a modicum of trust can be established for the numerical accuracy results in that they are on the same order as those for two dimensions. For vorticity measurements, RBF holds a slight edge in all cases, indicating that NN likely establishes higher accuracy in velocity through increased gradients that cause inconsistencies when the finite-difference method is applied.

5.2.2 Noisy

Time and accuracy results for the Lamb-Oseen vortex with added noise are presented in table 5.9 and compared in table 5.10. Plots of important trends are presented in figs. 5.9 and 5.10.

There is a marked difference in three-dimensional results when noise is added. As pre-

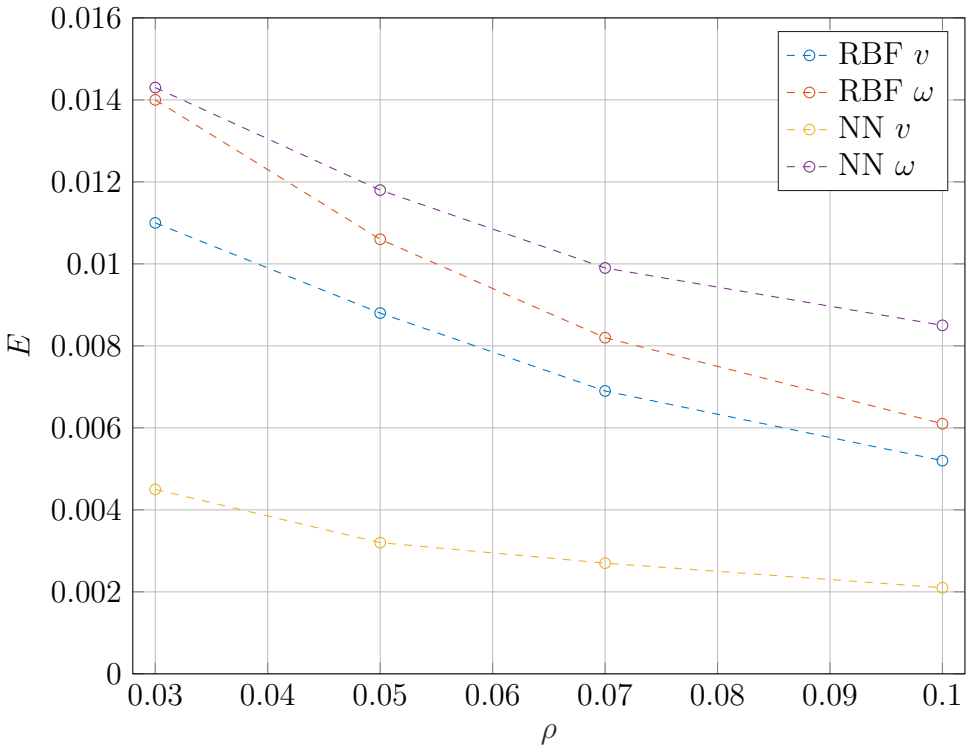


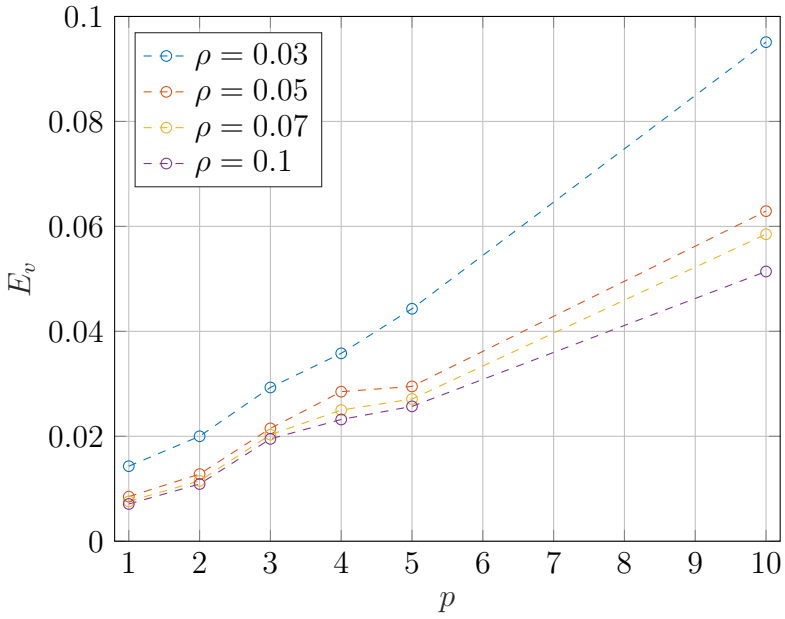
Figure 5.8: Plot of errors for the Burgers vortex. Data from table 5.7.

Table 5.9: Results of RBF and natural-neighbor (NN) interpolation for a Burgers vortex with random noise offset p pixels. Other symbols, as well as the number of trials for each condition, are the same as in table 5.7.

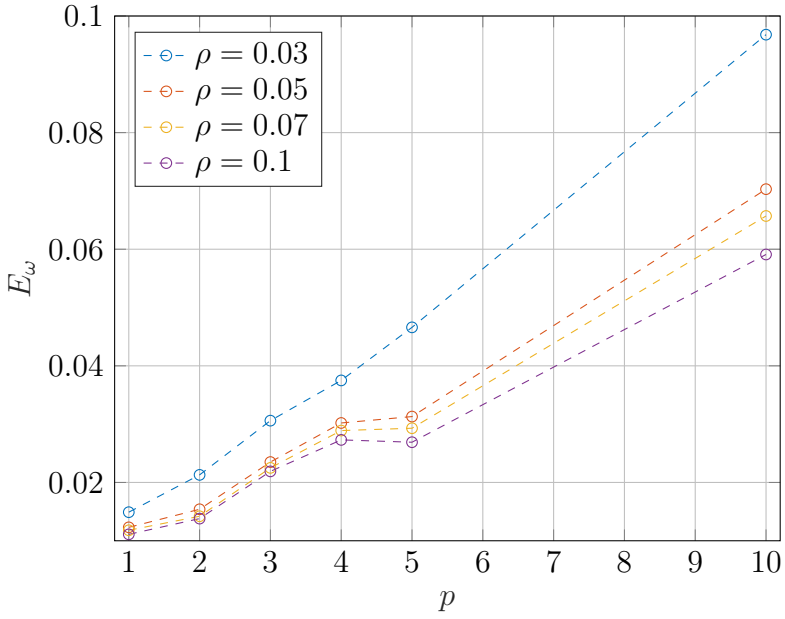
		RBF			NN		
p	ρ	t	$\%E_v$	$\%E_\omega$	t	$\%E_v$	$\%E_\omega$
1	0.03	8.38×10^3	1.43	1.49	0.52	0.99	0.90
1	0.05	2.85×10^4	0.85	1.23	0.88	0.93	0.86
1	0.07	7.32×10^4	0.76	1.18	1.31	0.88	0.84
1	0.1	2.12×10^5	0.71	1.11	1.90	0.85	0.83
2	0.03	8.43×10^3	2.00	2.13	0.58	1.74	1.63
2	0.05	2.84×10^4	1.28	1.54	0.87	1.66	1.60
2	0.07	7.34×10^4	1.15	1.42	1.25	1.62	1.59
2	0.1	2.13×10^5	1.09	1.38	1.84	1.56	1.57
3	0.03	8.42×10^3	2.93	3.06	0.51	2.39	2.39
3	0.05	2.87×10^4	2.15	2.35	0.86	2.33	2.36
3	0.07	7.30×10^4	2.03	2.25	1.24	2.34	2.35
3	0.1	2.12×10^5	1.95	2.19	1.85	2.27	2.35
4	0.03	8.35×10^3	3.58	3.75	0.53	3.17	3.15
4	0.05	2.88×10^4	2.85	3.02	0.88	3.13	3.16
4	0.07	7.28×10^4	2.50	2.89	1.23	3.13	3.14
4	0.1	2.12×10^4	2.32	2.73	1.81	3.03	3.14
5	0.03	8.43×10^3	4.43	4.66	0.51	3.90	3.96
5	0.05	2.84×10^4	2.95	3.13	0.92	3.82	3.95
5	0.07	7.29×10^4	2.71	2.93	1.25	3.80	3.94
5	0.1	2.14×10^5	2.57	2.69	1.81	3.84	3.94
10	0.03	8.39×10^3	9.51	9.68	0.55	7.67	8.09
10	0.05	2.85×10^4	6.29	7.03	0.86	7.51	8.08
10	0.07	7.34×10^4	5.85	6.57	1.25	7.48	8.12
10	0.1	2.13×10^5	5.14	5.91	1.82	7.51	8.16

Table 5.10: Comparison of results between RBF and NN methods as tabulated in table 5.9. Symbols are the same as in table 5.1 with the addition of subscripts identifying the methods.

p	ρ	$t_{\mathrm{RBF}}/t_{\mathrm{NN}}$	$E_{v\mathrm{RBF}}/E_{v\mathrm{NN}}$	$E_{\omega\mathrm{RBF}}/E_{\omega\mathrm{NN}}$
1	0.03	1.60×10^4	1.45	1.65
1	0.05	3.24×10^4	0.91	1.44
1	0.07	5.59×10^4	0.87	1.41
1	0.1	1.12×10^5	0.83	1.34
2	0.03	1.44×10^4	1.15	1.30
2	0.05	3.26×10^4	0.77	0.96
2	0.07	5.87×10^4	0.71	0.89
2	0.1	1.16×10^5	0.70	0.88
3	0.03	1.64×10^4	1.22	1.28
3	0.05	3.34×10^4	0.92	1.00
3	0.07	5.91×10^4	0.87	0.96
3	0.1	1.15×10^5	0.86	0.93
4	0.03	1.57×10^4	1.13	1.19
4	0.05	3.29×10^4	0.91	0.96
4	0.07	5.90×10^4	0.80	0.92
4	0.1	1.17×10^4	0.77	0.87
5	0.03	1.64×10^4	1.14	1.18
5	0.05	3.09×10^4	0.77	0.79
5	0.07	5.84×10^4	0.71	0.74
5	0.1	1.18×10^5	0.67	0.68
10	0.03	1.53×10^4	1.24	1.20
10	0.05	3.33×10^4	0.84	0.87
10	0.07	5.88×10^4	0.78	0.81
10	0.1	1.17×10^5	0.68	0.72

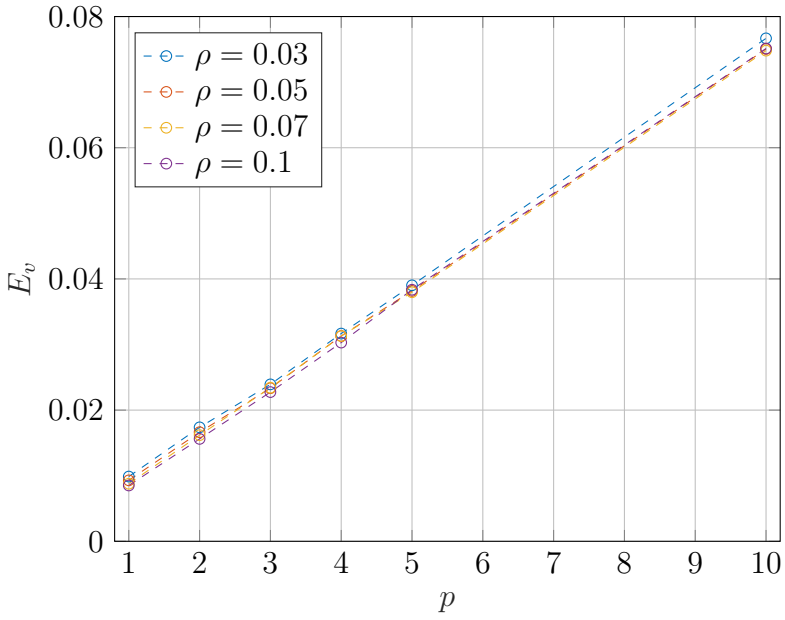


(a) velocity

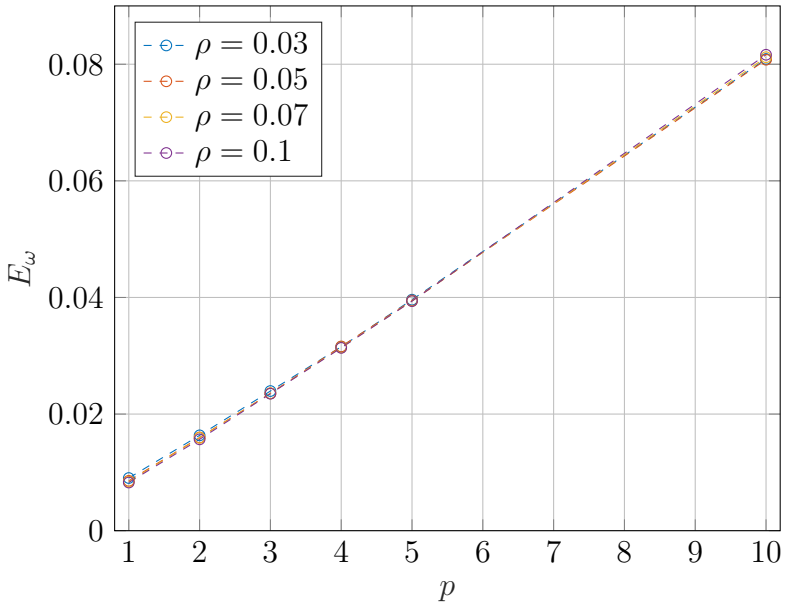


(b) vorticity

Figure 5.9: Trends in RBF interpolation error for a Burgers vortex as a result of p pixels of random noise. Data from table 5.9.



(a) velocity



(b) vorticity

Figure 5.10: Trends in natural-neighbor interpolation error for a Burgers vortex as a result of p pixels of random noise. Data from table 5.9.

viously discussed, the natural-neighbor method provides superior accuracy for the lowest particle density. However, RBF returns to higher accuracy for all cases of higher particle density. Although we again cannot visualize the interpolant to investigate this result, a plausible explanation is that the extra dimension increases the local noise offsets beyond the value where the natural-neighbor interpolant can recover the underlying function without any smoothing. More specifically, in two dimensions the maximum distance a point can be offset by noise applied to each Cartesian direction is $p\sqrt{2}$, while in three dimensions it is $p\sqrt{3}$, a 22% increase. RBF naturally smooths this difference as it operates over the solution space, while NN must provide the exact values at the interpolation-basis points.

It is also worth noting that the natural-neighbor method produces somewhat more uniform accuracy across the range of particle densities in three dimensions. The additional neighbors available in 3D as compared to 2D—even in the sparsest case investigated—provide the NN algorithm sufficient information to nearly maximize its accuracy. The RBF method, because it incorporates the entire data set and not just the local neighbors, can gain more accuracy with increasing N than NN is able to.

Chapter 6

CONCLUSIONS AND FUTURE WORK

6.1 Summary of results

Overall, the greatest advantage consistently demonstrated by natural-neighbor interpolation is time savings, both in terms of computation time and time spent optimizing the algorithms. Depending on the experimental conditions, the RBF interpolant takes three to five orders of magnitude longer to compute than the NN interpolant. Furthermore, there is no need to parameterize and tune the NN algorithm for each set of conditions as there is for the RBF method.

In terms of accuracy, the results offer a more mixed bag. It should be kept in mind that neither method demonstrates an error greater than about 2.5% for velocity or vorticity on clean data—and usually much less—so the comparison of relative results is contextualized in applications that require a high degree of relative accuracy. For interpolation of known velocity data, NN excels when that data is spread most sparsely, but RBF is capable of producing over twice the accuracy when there is more data to work with. For vorticity data, applying a finite difference on the RBF interpolant also consistently produces more accurate results with denser data sets than applying the same finite difference on the NN interpolant, but the margin is considerably less than twice as accurate.

However, major improvements are found when bypassing the finite-difference evaluation altogether and calculating vorticity directly using the natural neighbors. This method negates the extra error that comes with interpolating at the forward and backward points, while simultaneously providing more inputs to the gradient calculation. Furthermore, there is no need to parameterize with an h value to simulate a grid. The direct method produces a nearly 20% improvement in accuracy over the NN finite difference, and in every noisy case

tested that is enough to beat out the RBF-applied finite difference.

Both RBF and NN interpolation are shown to be able to handle a noisy input without amplifying that noise, except in a couple of the most sparse cases where the inherent smoothing of RBF is a detriment. In the three-dimensional case, the RBF smoothing is usually an asset, as it is able to deal with the relatively larger position errors more robustly than NN. In general, the accuracy of both interpolants decreases linearly with the amount of noise present.

6.2 Optimization and tuning of RBF methods

As described in section 2.2.1, there are a number of ways to construct an RBF interpolant, each of which require defining a spread constant that can take an infinite number of values. Further, there is no foolproof way to ensure that a particular function or spread constant is the best for the task at hand. Throughout the research conducted for this thesis, continual refinement of the RBF interpolation led to more and more accurate results; the only constraint for further tuning was time. Given that RBF interpolation in many situations can be made more accurate than NN interpolation, its primary disadvantage was shown to be computation time. Because RBF must cross-correlate each known data point with every other—unlike NN, which is completely local to the neighbors—the computational expense becomes exponential with more data and dimensions added. One method pursued for mitigating this consequence was discretizing the domain.

6.2.1 The local RBF

The radial symmetry of the Lamb-Oseen vortex naturally lends itself to breaking up the domain into rings around the origin. A preliminary result of such a scheme is presented in fig. 6.1. The most obvious feature is the discontinuities on the boundaries of the rings, which are the primary drawback to this method. On the other hand, it takes approximately 15% of the time to generate this interpolant as it does the global one with the same particle density, and the shape near the origin is more accurate. But the discontinuities still must

be dealt with. One option is to generate overlapping rings and average the areas of overlap, but that decreases the benefit in computation time. Furthermore, this method introduces yet more parameters to the optimization: in addition to selecting the number of rings and size of their overlap, each one must be individually optimized for spread constant. One could even try different basis functions on each ring. Overall, if the goal is extremely high accuracy, it may be possible to massage such a discretization into something useful, but for the purposes of this work the effort to optimize the method was excessively burdensome. There has been some research activity related to the reduction of computation time using partitioned RBF methods,[17] but whether the advantages are useful—especially in light of the low computational time required of natural-neighbor methods—is dependent on the specifics of the problem, especially regarding the effects on accuracy.

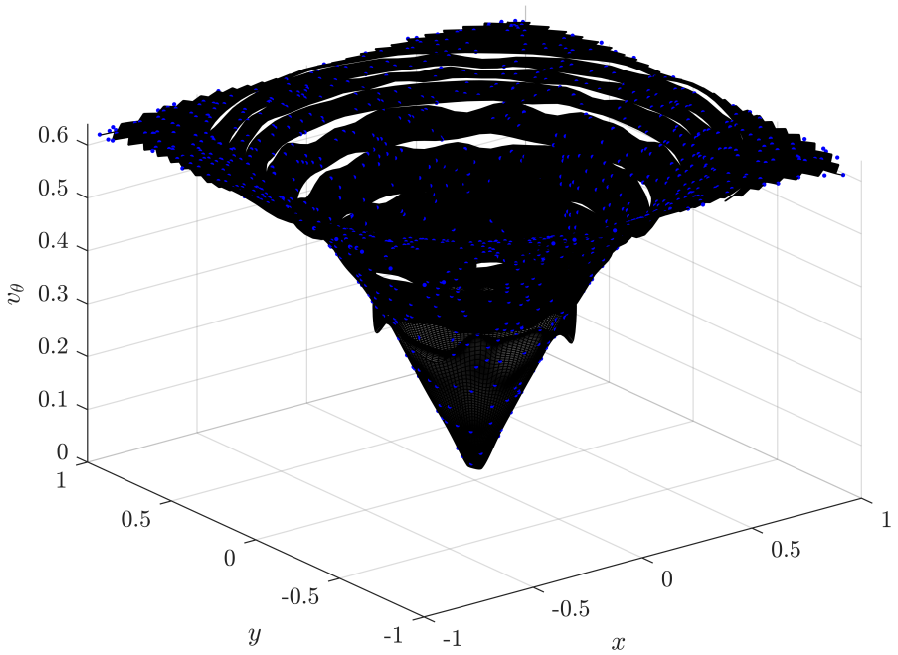


Figure 6.1: RBF interpolant of a Lamb-Oseen vortex with the domain divided into ten rings around the origin, $\rho = 0.1$, and $\sigma = 0.09$.

6.3 Optimization and tuning of natural-neighbor methods

Despite the natural-neighbor methods having no parameters to select, some discussion of their advanced use is warranted.

6.3.1 Higher-order continuity

As discussed in section 2.2.2, the Laplacian interpolant is $\mathcal{C}^0$ continuous everywhere. Because this work focuses on numerical differentiation of a well-behaved underlying analytical function, the continuity is of minimal consequence to the situations considered here. However, for more robust applications, it may be necessary to define a $\mathcal{C}^1$ interpolant, which is significantly more involved than defining the $\mathcal{C}^0$ interpolant but can be done so long as one has a way of estimating gradients at the interpolation basis points.[6] An even more complicated method exists for constructing a $\mathcal{C}^2$ interpolant using natural neighbors, and the implementation of that algorithm with any degree of accuracy is highly dependent on having good derivative estimates,[5] which implies an optimization process similar to that of RBF interpolation. A brief review of literature on the subject reveals little activity in recent years.

6.3.2 Higher-order data usage

Inherently, the natural-neighbor approach assigns no weights to known data beyond the immediate neighbors, whereas RBF interpolation assigns a weight to each known point regardless of its distance to the point of interest. Because the RBF approach is shown to be more accurate in certain situations, the question arises of whether it is possible to incorporate at least the next-nearest neighbors into the natural-neighbor method, and whether doing so could increase the accuracy of both function and gradient calculations. A higher-order weighting scheme would be required to incorporate such data, which may have a relationship to higher-order Voronoi diagrams. The computational expense of a higher-order scheme would be increased over the present methods, but not in a manner that would seriously challenge their advantage over RBF.

6.3.3 Derivatives in three dimensions

In order to appropriately scope this thesis, direct computation of derivatives using natural neighbors was not undertaken in three dimensions; however, such an extension is a natural next step to continue the present line of research.

6.4 Application to real-world data

The entirety of the research for this thesis was conducted on simulated flows, where the numerical results could be compared to a known analytical solution. While this approach is important for validating and verifying the methods, it reveals relatively little about their performance on actual experimental data. As has been emphasized throughout this chapter, RBF methods require optimization based on the specifics of the calculation to which they are applied, which puts them at a disadvantage in the real world where flow information may not be known *a priori*. Meanwhile, natural-neighbor methods have no parameters to optimize and thus provide an attractive option, especially when the data is sparse.

6.5 Final observations

As with most research, this work raised as many new questions as it answered existing ones. The phraseology "toward an improved method..." is chosen intentionally in the title of the thesis rather than something like "a definitely better method..." to emphasize that the results provide a promising direction, not a revolution. The hope is that the PTV community will benefit from this preliminary exposure to natural-neighbor methods and the general tenor of the results. Indeed, the use of widely available existing software libraries is intended to provide future researchers with a familiar and manageable baseline upon which to study the methods and their applications in more detail.

It is often frustrating when performing statistics-based analysis on randomly generated scenarios that there is usually not a rigorous analytical way to explain minor variations in the results. However, especially for work on basic computing hardware using existing

software libraries, the savings in computation time are not disputable when employing a natural-neighbor method over an RBF interpolation. The strongest conclusion that can be drawn from the data presented here is that if one is willing to accept the possibility of a small decrease in accuracy, functions and their gradients can be extracted rather quickly from flow data—bordering on real time even with standard consumer computing hardware—if the appropriate algorithms are applied to judiciously chosen domains.

Many of us find wonder and fascination in the mysteries of fluid dynamics, which after hundreds of years of study are still only fundamentally understood. But even small additions to our investigative toolkits can be revelatory when applied with the appropriate respect and care. Perhaps the discoveries expounded here will be just such an addition to a capability that ostensibly seems impossible—to see air.

BIBLIOGRAPHY

[1] Juan C. Agüí and J. Jiménez. "On the performance of particle tracking". In: *Journal of Fluid Mechanics* 185 (1987), pp. 447–468. DOI: 10.1017/S0022112087003252.

[2] P. V. Arun. "A terrain-based hybrid approach towards DEM interpolation". In: *Annals of GIS* 19.4 (2013), pp. 245–252. DOI: 10.1080/19475683.2013.843590. URL: https://doi.org/10.1080/19475683.2013.843590.

[3] S. J. Baek and S. J. Lee. "A new two-frame particle tracking algorithm using match probability". In: *Experiments in Fluids* 22.1 (Nov. 1996), pp. 23–32. ISSN: 1432-1114. DOI: 10.1007/BF01893303. URL: https://doi.org/10.1007/BF01893303.

[4] V. Belikov, V. Ivanov, V. Kontorovich, S. A. Korytnik, and A. Y. Semenov. "The non-Sibsonian interpolation: A new method of interpolation of the values of a function on an arbitrary set of points". In: *Computational Mathematics and Mathematical Physics* 37 (1997), pp. 9–15.

[5] T. Bobach, M. Bertram, and G. Umlauf. "Issues and Implementation of C^1 and C^2 Natural Neighbor Interpolation". In: *Advances in Visual Computing*. Ed. by George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Paolo Remagnino, Ara Nefian, Gopi Meenakshisundaram, Valerio Pascucci, Jiri Zara, Jose Molineros, Holger Theisel, and Tom Malzbender. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 186–195. URL: http://www-home.htwg-konstanz.de/~umlauf/Papers/isvc2006.pdf.

[6] Tom Bobach and Georg Umlauf. "Natural Neighbor Concepts in Scattered Data Interpolation and Discrete Function Approximation". In: *Visualization of Large and Unstructured Data Sets: Second workshop of the DFG's International Research Training Group "Visualization of Large and Unstructured Data Sets - Applications in Geospa-*

tial Planning, Modeling, and Engineering", September 9-11, 2007 Kaiserslautern, Germany. Ed. by Hans Hagen, Martin Hering-Bertram, and Christoph Garth. Vol. S-7. LNI. GI, 2007, pp. 23–35. URL: http://www-home.htwg-konstanz.de/~umlauf/Papers/Bobach.IRTG07.pdf.

[7] W. Brevis, Y. Niño, and G. H. Jirka. "Integrating cross-correlation and relaxation algorithms for particle tracking velocimetry". In: *Experiments in Fluids* 50.1 (Jan. 2011), pp. 135–147. ISSN: 1432-1114. DOI: 10.1007/s00348-010-0907-z. URL: https://doi.org/10.1007/s00348-010-0907-z.

[8] L.D.C. Casa and P.S. Krueger. "Radial basis function interpolation of unstructured, three-dimensional, volumetric particle tracking velocimetry data". In: *Measurement Science and Technology* 24.6 (May 2013). DOI: 10.1088/0957-0233/24/6/065304. URL: https://doi.org/10.1088/0957-0233/24/6/065304.

[9] C.S. Chen, Y.C. Hon, and R.A. Schaback. *Scientific Computing with Radial Basis Functions*. Hattiesburg, MS: Department of Mathematics, University of Southern Mississippi, 2007. URL: http://num.math.uni-goettingen.de/schaback/SCwRBF.pdf.

[10] Alex Chirokov. *Scattered Data Interpolation and Approximation using Radial Base Functions*. Version 1.0.0.0. Oct. 2006. URL: https://www.mathworks.com/matlabcentral/fileexchange/10056-scattered-data-interpolation-and-approximation-using-radial-base-functions.

[11] E. Cueto, N. Sukumar, B. Calvo, M. A. Martínez, J. Cegoñino, and M. Doblaré. "Overview and recent advances in natural neighbour galerkin methods". In: *Archives of Computational Methods in Engineering* 10.4 (Dec. 2003), pp. 307–384. ISSN: 1886-1784. DOI: 10.1007/BF02736253. URL: http://dilbert.engr.ucdavis.edu/~suku/nem/papers/nem_reviewproofs.pdf.

[12] P.G. Drazin and N. Riley. *The Navier-Stokes Equations. A Classification of Flows and Exact Solutions*. London Mathematical Society Lecture Note Series. Cambridge University Press, June 2006. ISBN: 9780521681629.

[13] Joseph Alan Duncan. "The development of advanced techniques for particle tracking velocimetry". Master's thesis. University of Washington, 2009.

[14] T. R. Etherington. "Discrete natural neighbour interpolation with uncertainty using cross-validation error-distance fields". In: *PeerJ Computer Science* 6.282 (July 2020). DOI: 10.7717/peerj-cs.282. URL: https://peerj.com/articles/cs-282/.

[15] G. E. Fasshauer. "Meshfree methods". In: *Handbook of theoretical and computational nanotechnology*. Ed. by Michael Rieth and Wolfram Schommers. Vol. 27. 2006, pp. 33–97. ISBN: 1-58883-042-X. URL: http://www.math.iit.edu/~fass/MeshfreeNano.pdf.

[16] Gregory E. Fasshauer and Jack G. Zhang. "On choosing "optimal" shape parameters for RBF approximation". In: *Numerical Algorithms* 45.1 (Aug. 2007), pp. 345–368. DOI: 10.1007/s11075-007-9072-8. URL: https://doi.org/10.1007/s11075-007-9072-8.

[17] Michael S. Floater and Armin Iske. "Multistep scattered data interpolation using compactly supported radial basis functions". In: *Journal of Computational and Applied Mathematics* 73.1 (1996), pp. 65–78. ISSN: 0377-0427. DOI: https://doi.org/10.1016/0377-0427(96)00035-0. URL: http://www.sciencedirect.com/science/article/pii/0377042796000350.

[18] Matt Foster. *matlab-interpolation-toolkit. MATLAB Multivariate Interpolation Toolbox*. Jan. 2009. URL: https://github.com/mattfoster/matlab-interpolation-toolkit.

[19] M. Gharib, D. Kremers, M. Koochesfahani, and M. Kemp. "Leonardo's vision of flow visualization". In: *Experiments in Fluids* 33.1 (July 2002), pp. 219–223. ISSN: 1432-1114. DOI: 10.1007/s00348-002-0478-8. URL: https://doi.org/10.1007/s00348-002-0478-8.

[20] Souparno Ghosh, Alan E. Gelfand, and Thomas Mølhave. "Attaching uncertainty to deterministic spatial interpolations". In: *Statistical Methodology* 9.1 (2012), pp. 251–

264. DOI: 10.1016/j.stamet.2011.06.001. URL: http://www.sciencedirect.com/science/article/pii/S1572312711000529.

[21] Hisamoto Hiyoshi and Kokichi Sugihara. "Two generalizations of an interpolant based on Voronoi diagrams". In: *International Journal of Shape Modeling* 5.2 (1999), pp. 219–231. DOI: 10.1142/S0218654399000186. URL: https://doi.org/10.1142/S0218654399000186.

[22] Hisamoto Hiyoshi and Kokichi Sugihara. "Voronoi-based interpolation with higher continuity". In: Jan. 2000, pp. 242–250. DOI: 10.1145/336154.336210.

[23] Mark Kreizer and Alex Liberzon. "Three-dimensional particle tracking method using FPGA-based real-time image processing and four-view image splitter". In: *Experiments in Fluids* 50.3 (Mar. 2011), pp. 613–620. ISSN: 1432-1114. DOI: 10.1007/s00348-010-0964-3. URL: https://doi.org/10.1007/s00348-010-0964-3.

[24] Randall J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations. Steady State and Time Dependent Problems*. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 2007. ISBN: 978-0-898716-29-0.

[25] Micah Philip Paul. "Development of a vision-based particle tracking velocimetry method and post-processing of scattered velocity data". Master's thesis. University of Washington, 2012.

[26] N. Ponchaut, C. Mouton, H. Hornung, and D. Dabiri. "3D Particle Tracking Velocimetry Method: Advances and Error Analysis". Unpublished. GALCIT Report FM2005.004. California Institute of Technology, Pasadena, CA., Aug. 2005. URL: https://resolver.caltech.edu/CaltechGALCITFM:2005.004.

[27] P.G. Saffman. *Vortex Dynamics*. Cambridge Monographs on Mechanics. Cambridge University Press, 1993. DOI: 10.1017/CBO9780511624063.

[28] Pavel Sakov. *nn-c. (Natural Neighbours interpolation)*. Oct. 2019. URL: https://github.com/sakov/nn-c.

[29] R. Sibson. "A brief description of natural neighbor interpolation". In: *Interpreting Multivariate Data*. Ed. by V. Barnett. Chichester: John Wiley, 1981, pp. 21–36.

[30] N. Sukumar. "The natural element method in solid mechanics". PhD Thesis. Evanston, IL: Theoretical and Applied Mechanics, Northwestern University, June 1998. URL: http://dilbert.engr.ucdavis.edu/~suku/nem/thesis.html.

[31] N. Sukumar and J. E. Bolander. "Numerical computation of discrete differential operators on non-uniform grids". In: *CMES* 4.6 (2003), pp. 691–706. URL: http://dilbert.engr.ucdavis.edu/~suku/nem/papers/nem_fdcmes.pdf.

[32] N. Sukumar, B. Moran, A. Semenov, and V. Belikov. "Natural neighbour Galerkin methods". In: *International Journnal for Numerical Methods in Engineering* 50 (Jan. 2001), pp. 1–27. URL: http://dilbert.engr.ucdavis.edu/~suku/nem/papers/nem_nonsibson.pdf.

[33] Holger Wendland. "Solving partial differential equations with multiscale radial basis functions". In: *Contemporary Computational Mathematics - A Celebration of the 80th Birthday of Ian Sloan*. Ed. by Josef Dick, Frances Y. Kuo, and Henryk Woźniakowski. Cham, Switzerland: Springer International Publishing, 2018, pp. 1191–1213. ISBN: 978-3-319-72456-0. DOI: 10.1007/978-3-319-72456-0_55. URL: https://doi.org/10.1007/978-3-319-72456-0_55.

[34] Masayuki Yano, James Douglass Penn, George Konidaris, and Anthony T. Patera. *Math, Numerics, & Programming (for Mechanical Engineers)*. 2.1. MIT, Aug. 2013. URL: https://ocw.mit.edu/ans7870/2/2.086/S13/MIT2_086S13_Textbook.pdf.