HF Radar along the southern coast of Malta: Validation, Gap filling and Seasonality

Malcolm Refalo

Supervised by Dr Adam Gauci

Department of Geosciences

Faculty of Science

University of Malta

November, 2023

A dissertation submitted in partial fulfilment of the requirements for the degree of M.Sc. in Applied Oceanography.

Abstract

In an attempt to address the issue of coverage gaps in HF radar data obtained by the CALYPSO HF radar network, the performance of DINEOF (Data Interpolation Empirical Orthogonal Functions) as a tool is evaluated. The key objective within this study is the fine tuning of variables for DINEOF alongside the length of data used as input. The performance of DINEOF as a function of missing data in the timeframe to be gap filled is also explored, in the hopes of finding a clear relation.

The implementation details of Matlab and Python scripts to carry out the research are presented. This included a script to artificially introduce gaps to allow for easy evaluation of DINEOF's performance, another to automate the creation of DINEOF ".init" files which store all the variables for the gap filling, and a script designed to go through all the DINEOF results, evaluate the performance, and save all the results in a singular CSV file.

As a result of the methodology taken in this dissertation, it was found that DINEOF performs best when the length of data used as input is that of 72 hours or 3 days, and when the alpha parameter (a parameter controlling DINEOF filter strength) is set to a value of 0.1. The values found when fine tuning the execution of DINEOF, the length of data which provided the optimal results being 72 hours, suggests that providing additional input files might produce results that are less accurate. This indicated that using a continuous block of radar data spanning 72 hours as the input optimises DINEOF's gap-filling accuracy. When attempting to find the point where DINEOF's performance is hindered as a result of a lack of data in the time frame to be gap filled, no such clear relationship was found. The results obtained suggest that DINEOF has already been optimised to perform under extreme cases of data omission.

Contents

1 Introduction **1**
- 1.1 Research questions . . . 2
- 1.2 Aims and objectives . . . 2
- 1.3 Structure of the Dissertation . . . 3

2 Background & Literature Overview **4**
- 2.1 Mediterranean maritime landscape . . . 4
- 2.2 High Frequency radar . . . 5
  - 2.2.1 Physical principles of CODAR . . . 6
  - 2.2.2 CALYPSO South Project . . . 7
- 2.3 DINEOF . . . 7
  - 2.3.1 Applications of DINEOF . . . 8
  - 2.3.2 DINEOF as a benchmark . . . 8

3 Methodology **10**
- 3.1 Verifying the functionality of DINEOF . . . 10
- 3.2 Selecting the dates for DINEOF evaluation . . . 12
- 3.3 Main processing pipeline . . . 14
  - 3.3.1 Query Function . . . 15
  - 3.3.2 Introducing artificial gaps . . . 15
  - 3.3.3 Binary File Generation . . . 16
  - 3.3.4 Automation of the initialisation file . . . 17
  - 3.3.5 Error metrics . . . 18
- 3.4 CSV analysis . . . 19
  - 3.4.1 Finding the optimal parameters . . . 20
  - 3.4.2 Determining the effect of missing data . . . 21
- 3.5 Analysing DINEOF as a reanalysis tool . . . 21
  - 3.5.1 Visualising U , V, and *Totals* . . . 22
  - 3.5.2 Calculating statistics for each dataset . . . 23

4 Results **24**
4.1 DINEOF performance evaluation 24
4.1.1 Analysing parameter sensitivity & input data length 24
4.1.2 Analysing the effect of missing data on DINEOF 25
4.2 Different reconstruction methods 29
4.2.1 DINEOF as a reanalysis tool 29
4.2.2 Weekly statistics as quality control 35

5 Conclusions **38**
5.1 DINEOF parameter optimisation 38
5.2 Missing data and its effects on DINEOF's performance 39
5.3 Exploring DINEOF as a reanalysis tool 39
5.4 Weekly statistics for quality control 40
5.5 Areas for Improvement 41
5.6 Possible Future Research 41

References **43**

Code used **46**
dineof_prep_u.m 46
dineof_result_check.m 47
NAN_date_finder.m 49
Function: Query_dates.m 51
Function : Swap_Patches.m 53
Automatic DINEOF data file maker.m 54
Python_DINEOF_init_generator.py 57
DINEOF performance calculator.m 59
Powershell_auto_dineof 62
Python combination finder.py 62
Genetic Algorithm DINEOF optimisation.py 63
CSV file analysis.py 66
Final Jupyter notebook 67

List of Figures

2.1 Example received Doppler shifted wave echoes with and without sea surface currents from Evans & Georges (1979). 6

3.1 Comparing u and v components before and after being passed through DINEOF. . . 12
3.2 $Date\_finder$ flow chart 13
3.3 Main data processing pipeline. 14
3.4 Example of HF radar data and corresponding mask before (top) and after (bottom) adding artificial gaps. 16
3.5 Example data before and after DINEOF. 18
3.6 Example correlation plot used to calculate RMSE and R^2. 19
3.7 Example visualisation plot 22

4.1 Scatter plot of RMSE and R^2 against percentage missing data in gap filled time frame. 27
4.2 Distributions of percentage missing data in time frame, RMSE, and R^2. 28
4.3 Array of subplots containing U, V, and *Totals* for "3-day" and "9-day" datasets and their differences, March 27 2022 29
4.4 Array of subplots containing U, V, and *Totals* for "3-day" and "9-day" datasets and their differences, April 26 2022 30
4.5 Array of subplots containing U,V, and *Totals* for "3-day" and "9-day" datasets and their differences, May 15 2022 31
4.6 Array of subplots containing U,V, and *Totals* for "3-day" and "9-day" datasets and their differences, June 15 2022 32
4.7 Quiver plot for March 27 2022. 33
4.8 Quiver plot for April 26 2022. 34
4.9 Quiver plot for May 15 2022. 34
4.10 Quiver plot for June 15 2022. 35
4.11 Weekly Mean, Variance, Minimum, and Maximum statistics for the '3 day' and '9 day' datasets. 36
4.12 Weekly Mean, and Variance statistics for the '3 day' dataset, '9 day' dataset, and raw HF radar totals. 37

List of Tables

3.1 Dates to be used for analysis 14

4.1 Table showing best combination of parameters for DINEOF 25

1 Introduction

The Mediterranean Sea, renowned as a cradle of civilisation and a haven of biodiversity, has captivated the scientific community for centuries. The distinctive geographical, ecological, and climatological features of the region offer a diverse and complex area of study, particularly in oceanography. The Maltese archipelago, strategically positioned at the intersection of significant maritime routes, plays a crucial role within this marine domain. Comprehensive knowledge of the surrounding waters is imperative, impacting not only maritime safety but also the preservation of the ecological integrity of the region.

In response to this need, the CALYPSO project emerged as a collaborative initiative between Maltese and Sicilian entities. This venture led to the establishment of a network of High Frequency (HF) radars, designed to monitor surface current conditions in the Malta-Sicily channel continuously and in real-time. Despite the transformative insights provided by HF radar technology, it faces inherent challenges, particularly in ensuring the continuity and validity of the data collected. Recognising these challenges, this study, explores a gap filling algorithm as a possible solution.

Confronting the issue of data discontinuities in HF radar observations, this research introduces robust methodologies for data validation and sophisticated techniques for filling data gaps. Special attention is given to the utilisation of the DINEOF (Data Interpolating Empirical Orthogonal Functions) method, aiming to enhance the reliability and comprehensiveness of the data collected by the Calypso HF radar network. These hold significant implications for practical applications, including environmental conservation and the enhancement of maritime safety protocols.

The sea enveloping Malta is a bustling confluence of commercial activity and ecological significance. The CALYPSO project underscores the necessity of safeguarding this delicate balance, with a specific emphasis on the threat posed by maritime disasters such as oil spills. These incidents represent a profound risk to both the rich biodiversity of the Mediterranean and the economic stability of coastal communities. The quest for accurate, timely, and reliable data is a cornerstone in developing effective preventative and mitigate strategies against such environmental hazards.

This research connects the present capabilities of the Calypso HF radar network with the future potential unlocked by addressing and overcoming the challenges of data gaps. By enhancing the integrity of sea surface current data within the Malta-Sicily channel, the study marks a significant stride towards a future where the commercial vitality and natural beauty of the Mediterranean can thrive in harmony, safeguarded by informed understanding and innovative technology.

1.1 | Research questions

This study is primarily concerned with the development of a robust framework for addressing the significant challenge of data discontinuities in HF radar observations, particularly those collected by the CALYPSO network. The gap in data presents a critical barrier to accurate oceanographic analysis, necessitating the exploration of efficient and reliable gap-filling methodologies. Given this context, the research questions guiding this study are formulated as follows:

- How can the DINEOF method be optimally configured to address the coverage gaps in HF radar data collected by the CALYPSO network?
- In what ways do varying levels and patterns of missing data influence the performance of DINEOF?
- Can DINEOF be effectively utilised as a reanalysis tool for refining and improving the quality of HF radar datasets, and if so, how?
- What role does weekly statistical analysis play in ensuring the quality and reliability of gap-filled data, and how can these analyses be systematically integrated into the quality control process?

1.2 | Aims and objectives

The primary objective of this study revolves around addressing the coverage gaps present in the HF radar data collected by the CALYPSO network. The challenges posed by these data discontinuities necessitate the introduction of an efficient and reliable gap-filling method. To this end, DINEOF, a method renowned for its ability to handle large datasets, is explored in depth throughout this research.

To ensure the efficacy of this method the following rigorous experiments and tests were undertaken. The entire research process was facilitated and supported by computational scripts developed using MATLAB and Python.

- **Fine-tune DINEOF parameters:** The performance of DINEOF is heavily influenced by its configuration parameters. This study involved iterative processes to adjust and fine-tune these parameters, ensuring optimal gap-filling results that are both accurate and consistent with the nature of the original HF radar data.

- **Evaluate the effect of missing data on DINEOF performance:** Missing data can introduce biases and inaccuracies when using interpolation methods. This aspect of the study delved into understanding how varying levels and patterns of missing data affect the performance of DINEOF, establishing benchmarks for its reliability under different conditions.
- **Explore DINEOF as a reanalysis tool:** Beyond gap-filling, DINEOF possesses the capability to serve as a reanalysis tool, refining and improving the quality of datasets by identifying and rectifying anomalies. This research explored such capabilities, shedding light on the broader applications of DINEOF in oceanographic studies.
- **Weekly statistics for quality control:** To maintain the integrity and reliability of the data, weekly statistical analyses were performed. This systematic quality control measure ensured that the gap-filled data aligned with expected patterns and did not introduce any unintended biases or anomalies.

The aim was to develop a robust framework for addressing data discontinuities in HF radar observations, enhancing the value and utility of the data collected by the CALYPSO HF radar network.

1.3 | Structure of the Dissertation

This dissertation is structured into five main chapters, complemented by an appendix. After this introductory chapter, the subsequent four chapters delve into the details of the research alongside its findings. Chapter 2, Literature Review, provides a concise review of the relevant literature, focusing on key aspects related to the Mediterranean Sea, the CALYPSO project, high-frequency radars, and gap filling. Chapter 3, Methodology, offers a comprehensive description of the research methodology. This chapter elucidates the processes, rationale behind the code, and the practical implementation that drove the research forward. Chapter 4, Results and Analysis, closely examines the outcomes derived from the code and methodologies outlined in the previous chapter. A detailed analysis of the research findings are presented. Finally, Chapter 5, Conclusion, summarises the research results, highlighting key findings, and delves into a discussion of the limitations encountered during the research process. Additionally, it outlines potential directions for future research. The appendix provides supplementary materials, including code snippets and additional information to support the dissertation's content.

2

Background & Literature Overview

The following chapter delves into three main themes critical to the research. These three themes fundamentally describe the motivation behind the study, the technology and physics behind the measurements, and finally the tool being evaluated in the aim of gap filling.

The first section of the literature review will paint a picture of the Mediterranean as a landscape of great importance. This importance is both economical due to the strong presence of commercial shipping vessels, and ecological due to the diverse and borderless marine ecosystems. Thus the chapter starts off by examining the Mediterranean's dual role as a bustling maritime hub and a delicate ecological zone, highlighting the potential environmental consequences that heavy maritime traffic pose.

The second section explores the use of HF radar systems. These systems primarily CODAR will be discussed and the fundamental physical process behind the measurement of sea surface currents will be discussed. The impact and effectiveness of projects like the CALYPSO project in Malta, which pioneered the use of such technologies in the region will also be assessed.

Moreover, the persistent challenge of data gaps in remote sensing will be tackled. The DINEOF method, an algorithm developed to reconstruct missing information is introduced. The fundamental processing applied by DINEOF is explained also. Finally, research and use cases of DINEOF used to gap fill different datasets is discussed and explored, showing DINEOF's varying applications in many areas of remote sensing.

2.1 | Mediterranean maritime landscape

The Mediterranean Sea is a strategically placed area with easy access to the Atlantic Ocean, Red Sea, and Black Sea through ports with access to the Strait of Gibraltar, Suez Canal, and Bosporus Strait, as described by Tode et al. (2021). As described in Deidun et al. (2018), shipping accounts for 80% of all cargo and merchandise transport worldwide, with a 77% increase in the volume of cargo loaded and unloaded within Mediterranean ports between 1985 and 2001.

As noted by Tode et al. (2021), an increase in shipping and maritime traffic also results in increased anthropogenic pressures, such as the accidental and illicit discharge of oil and hazardous substances, marine litter, and water discharges. Despite the growth in the volume of oil and other cargo transported, global incident rates involving oil have decreased due to the adoption and implementation of international shipping conventions related to transportation safety. Nevertheless, with an estimated 200,000 commercial vessels crossing the Mediterranean Sea annually Deidun et al. (2018), limiting oily discharges from tank cleaning and other activities is essential for preserving marine environmental health.

Maritime transport has been an integral part of human activity throughout history, enabling global trade and contributing greatly to the global economy by reducing the unit transportation cost of goods when compared to road or rail transport (Buber et al., 2020). Although shipping is the most environmentally friendly mode of transportation based on emission per unit transported, ships are the only source of air pollution within the seas, emitting exhaust gases containing particulate matter, and volatile organic compounds.

2.2 | High Frequency radar

Recent advancements in High Frequency (HF) radar systems have made it easier to analyse complex coastal dynamics, which are known for their high variability. These radar systems continuously measure surface currents over a large coastal region, making it possible to remotely sense coastal and ocean currents. One such system is the U.S Integrated Ocean Observing system, which continuously monitors the surface coastal and ocean currents through the deployment of a hundred shore based HF radars (Liu et al., 2014). As and analogue to the system covering the majority or the U.S. eastern coast, European institutions are working together in order to build a similar large scale network (Rubio et al., 2017). Another system with a similar task but on a much smaller scale is the system deployed via the CALYPSO project in Malta and Sicily. During the CALYPSO project, a total of seven shore based HF radars were deployed, three of the southern coast of Sicily and four along the Maltese coastline. When optimally designed, these networks can be used for a variety of applications, including improving marine operations, mitigating hazards, and optimising ecosystem-based management (Soomere et al., 2012).

The magnitude and direction of sea surface currents is inferred from the back-scatter off the ocean surface over an area predetermined by the specifications and design of the HF radar. All measurements are made using the back-scatter power, which is normally dominated by two peaks resulting from Bragg scatter off of linear ocean waves (Wyatt, 2012). Such systems, like most others, are not 100% accurate. Radio interference, ionosphere interference, ship echoes, and uncertainty in the measurements themselves (Emery & Washburn, 2019) may result in sporadically non-realistic values in the measurements. This necessitates the use of quality control procedures, especially when non realistic values are observed at the edges of the radar's domain. In the case of WERA HF radar systems, they contain routines within the processing chain and is applied in near-real time. This is especially important when the data is being used for time sensitive applications (Heron et al., 2016).

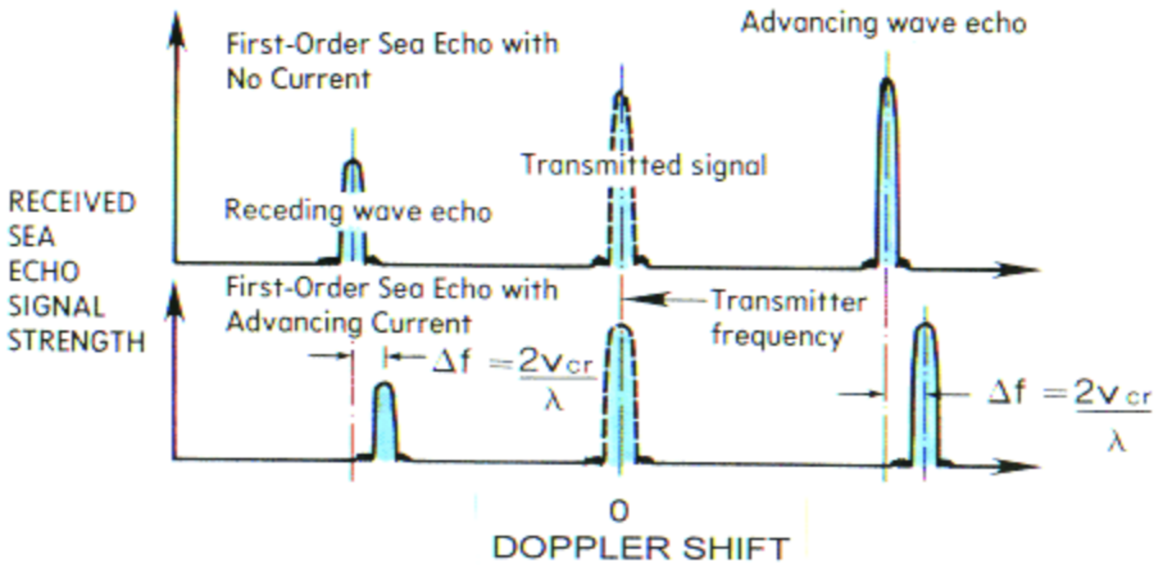


Figure 2.1: Example received Doppler shifted wave echoes with and without sea surface currents from Evans & Georges (1979).

2.2.1 | Physical principles of CODAR

In Crombie (1955), Bragg scattering was used to explain the measured Doppler shift of radio waves reflected from the sea surface. It was this principle which was exploited and expanded upon in Evans & Georges (1979), which allowed for the development of the CODAR (coastal Ocean Dynamics Radar) system.

As a consequence of Bragg scattering, the strongest received signals come from waves travelling radially toward or away from the radar (Crombie, 1955; Evans & Georges, 1979). Once the received signals are analysed, two peaks can be seen either side of the transmitted signal as shown in Figure 2.1. The frequency, ν_b, of which these two peaks will be found can be calculated as:

$$\nu_b = \pm\sqrt{\frac{g\nu_r}{\pi(c_0)}} \tag{2.1}$$

where g is the acceleration due to gravity, ν_r is the transmitted radar frequency, and c_0 is the vacuum speed of light (Martinez-Pedraja et al., 2013). In the presence of non-zero surface currents, both Bragg peaks will be shifted in the same direction by a frequency amount $\Delta\nu$ given by the following relation:

$$\Delta\nu = \frac{2V_{cr}\nu_r}{c_0} \tag{2.2}$$

where V_{cr} is the radial component of the current along the direction of the radar (Merz et al., 2021). Thus, the surface current velocity is simply the deviation from the expected Doppler shift mentioned previously, with the magnitude being proportional to the difference.

It is crucial to keep in mind that these measurements only measure the surface current component radially with respect to the installed Radar. Hence to map out a current vector map, at least two CODAR systems are required for the calculation of total current vectors (Evans & Georges, 1979).

2.2.2 | CALYPSO South Project

The CALYPSO South Project[1] served as an extension to its predecessors, the CALYPSO and CALYPSO Follow On projects, both of which laid the groundwork for HF radar systems in the Malta-Sicily Channel. Coordinated by the Oceanography Malta Research group at the University of Malta, CALYPSO was a two-year initiative that initially established a permanent HF radar observing system (Gauci et al., 2016; Roarty et al., 2019). CALYPSO Follow On, was a six-month intensive sequel, which enhanced the initial system with additional installation and expanded the partnership network to include both Maltese and Sicilian entities.

The CALYPSO South project addressed complex challenges in marine safety and environmental preservation by focusing on three core objectives, the expansion of the existing HF radar network, the development of advanced monitoring and forecasting tools, and the provision of specialised operational services. To achieve such goals, the project integrated cutting-edge IT solutions to expedite interventions and streamline search and rescue operations. Moreover, the project assisted maritime security agencies and environmental protection entities both in Malta and Sicily by installing additional HF radar and weather stations to improve data quality.

CALYPSO South uniquely emphasised transnational collaboration, pooling resources from academic institutions and governmental bodies across Malta and Sicily. This approach is imperative for tackling marine and environmental challenges such as marine litter and oil spills which are border-less events. The project served as a foundational step toward creating a comprehensive Marine Electronic Highway, aiming to improve marine transportation safety and enhance environmental protection across the region.

Given its multifaceted objectives and innovative approach, the CALYPSO South Project offers a rich scope for future research. Potential areas include the effectiveness of HF radar systems, the long-term impact of the project on marine pollution and safety, the economic valuation of ocean data collected. Thus the project provided a compelling case study on how technology and international collaboration can synergies to address urgent marine and environmental challenges.

2.3 | DINEOF

Remote sensing techniques are a crucial part of monitoring environmental conditions in both land and marine ecosystems, providing valuable data for understanding the effects of climate change (Konik et al., 2019). However, geophysical data sets from not only satellites often have gaps due to factors like clouds, rain, and the satellite's orbit (Álvarez et al., 2021). To address this issue, researchers have developed DINEOF (Data Interpolation Empirical Orthogonal Functions) the basis of which are outlined in Beckers & Rixen (2003), a software solution

[1] *https : //www.calypsosouth.eu*

that uses EOF (Empirical Orthogonal Functions) decomposition to iteratively calculate field values at missing positions. The number of EOFs is determined through cross-validation and the software has been updated to improve reconstruction speed and minimise errors (Beckers & Rixen, 2003).

As described in Alvera-Azcárate et al. (2011), DINEOF is based on an iterative method for calculating field values at missing positions. The method involves removing the spatial and temporal mean from the analysed data and initialising missing data to zero. Then, one EOF is calculated from the field, and the missing data is replaced with values obtained through EOF decomposition. This process repeats until a convergence criterion for the missing data values, specified by the user, is reached. After that, two EOFs are calculated, and the entire process is repeated. This continues for three EOFs, and so on, until the number of EOFs that minimises the difference between the calculated and actual values is determined through cross-validation.

2.3.1 | Applications of DINEOF

The method is widely used to reconstruct missing data in various fields, including sea surface current data, sea surface temperature fields, and daily sea surface salinity. The method involves using EOF decomposition to iteratively calculate field values at missing positions. The number of EOFs is determined through cross-validation, which ensures that the reconstructed values are as close to the actual values as possible.

Researchers have achieved good results using DINEOF to fill in gaps in geophysical data sets from satellites, despite factors like clouds and rain that can cause gaps in the data. For instance, Gauci et al. (2016) used DINEOF to reconstruct missing sea surface current data measured by shore-based HF radars. Nikolaidis et al. (2014) managed to recover a complete sea surface temperature field over the Eastern Mediterranean, Levantine sea, and the coasts of Cyprus using DINEOF. In another study, Alvera-Azcárate et al. (2016) obtained daily sea surface salinity from the Soil Moisture and Ocean Salinity (SMOS) satellite mission by using DINEOF to fill in the gaps. Similarly, Zhou et al. (2021) achieved very good results when using DINEOF to reconstruct sea surface temperature from the MODIS Aqua Level 3 SST data over an 8-day period.

In short, the DINEOF method has proven to be an effective tool when it comes to filling in gaps in remote sensed data, and researchers have achieved good results using this method in various fields. The method's ability to iteratively calculate field values at missing positions using EOF decomposition, with the number of EOFs determined through cross-validation, has made it a popular choice for researchers working with geophysical datasets.

2.3.2 | DINEOF as a benchmark

Recent studies, such as in Zheng & Li (2024), Hernández-Carrasco et al. (2018), and Kolukula et al. (2020), have employed DINEOF as both a foundational tool for new gap filling methodologies and a benchmark for comparing alternatives. Kolukula et al. (2020) tested the complex empirical orthogonal function (CEOF) method against DINEOF's EOF approach for filling gaps in HF radar data, demonstrating CEOF's effectiveness in replicating the original data's characteristics accurately.

In Hernández-Carrasco et al. (2018), the performance of three gap filling methods Self Organising Maps (SOMs), Open Boundary Method with Assimilation (OMA), and DINEOF was evaluated on HF radar data. The study revealed that OMA struggled with accurately capturing small scale features, contrasting with SOM and DINEOF methods, which provided precise reconstructions and effectively represented the dynamics of turbulent flows, thus proving reliable for coastal dynamics assessments.

Furthermore, Zheng & Li (2024) assessed the Empirical Function (EF) method against DINEOF, noting EF's flexibility in integrating prior knowledge and its applicability to a broader range of ocean data types beyond SST data. The EF method's innovation and adaptability present significant advancements over DINEOF, suggesting potential areas for future research to explore and benchmark against DINEOF's established performance.

3

Methodology

This chapter delves into the intricacies of the data processing pipeline and the subsequent analysis performed as part of the research methodology. The key functions and scripts developed in Matlab and Python, which were employed to semi-automatically process oceanographic data, are outlined. The primary objective of this data processing pipeline is to facilitate the execution of DINEOF and the evaluation of its performance in gap-filling and data reconstruction.

The processing pipeline encompasses a range of critical components, each dedicated to a specific task. These include data extraction, the introduction of artificial gaps, the generation of binary files for DINEOF, the creation of initialisation (".init") files, and the computation of performance metrics such as root mean square error (RMSE) and R-squared (R^2). This chapter provides an in-depth understanding of each component's functionality and its role in achieving the research goals.

The chapter also explores the analysis of the data, with a particular focus on understanding the impact of temporal parameters and gap sizes on the performance of DINEOF. Two distinct datasets are scrutinised to uncover insights into DINEOF's capabilities under varying conditions. Statistical measures and visualisations are used to gain a comprehensive perspective on the data and the performance of the gap-filling algorithm.

Through the discussions and analyses presented in this chapter, readers will gain a deeper insight into the methodologies employed in processing and analysing oceanographic data for the purpose of gap filling and reconstruction, as well as the insights obtained from these processes.

3.1 | Verifying the functionality of DINEOF

This section is dedicated to the verification of DINEOF's performance and functionality. Before delving into data manipulation and further analysis, it is crucial to ensure that DINEOF operates as intended. This verification process begins with a visual inspection of DINEOF's performance.

To achieve this, a set of hourly data files was selected, and combined into a single NetCDF file using the Climate Data Operators (CDO) tool developed by the Max Planck Institute for Meteorology[1]. The merging of files was accomplished using the following Unix-based terminal command, as shown in Listing 1:

```
cdo mergetime *.nc outfile
```

Listing 1: Example *CDO* code to merge multiple files by time.

Once the NetCDF files were successfully merged, Matlab was used to read various variables. The 'time' variable was stored as an array, while the 'U' and 'V' components of the totals were stored as matrices. From these matrices, a 'mask' was created by iteratively examining each time frame to identify locations with non-NaN values. This 'mask' delineated the areas where DINEOF would perform data reconstruction, filling in spaces with at least one measured value. The 'U,' 'V,' 'time,' and 'mask' were saved in a binary format compatible with DINEOF using a custom Matlab function called "uwrite" [2].

Subsequently, the 'DINEOF.init' file was modified to specify that DINEOF should read the files generated earlier by Matlab and save the results in the respective directory. Within this ".init" file, various parameters could be adjusted to optimise DINEOF for the specific dataset and analysis requirements. To execute DINEOF, the Windows terminal was utilised in conjunction with the command presented in Listing 2:

```
dineof.exe dineof.init
```

Listing 2: Windows terminal command to run DINEOF.

Once executed, and with the resultant output file successfully saved, a visual examination of the data was conducted. A Matlab script was implemented to read both the data before and after undergoing the DINEOF process. In Figure 3.1, a series of plots illustrate the 'U' and 'V' components of the measured values in the top row and the corresponding 'U' and 'V' components after being processed by DINEOF in the bottom row. Notably, DINEOF was configured to reconstruct only the points specified by the mask, rather than regenerating the entire domain from scratch.

[1]https://code.mpimet.mpg.de
[2]https://www.mathworks.com/matlabcentral/fileexchange/2055-uwrite-uread

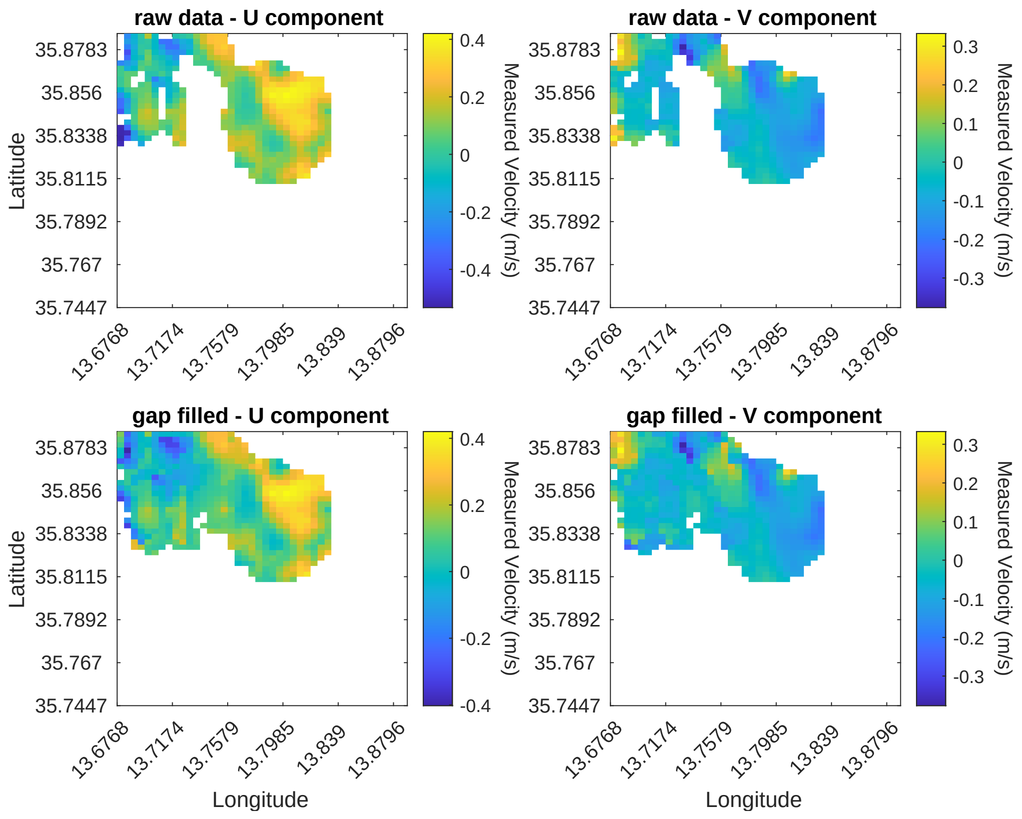


Figure 3.1: Comparing u and v components before and after being passed through DINEOF.

3.2 | Selecting the dates for DINEOF evaluation

This section of the methodology discusses the process of selecting 15 specific days for the evaluation of DINEOF's performance. The script employed for this task can be found in its entirety in Appendix 5.6. A simplified flowchart in Figure 3.2 outlines the steps involved in the process.

Initially, the entire 18-month-long dataset was loaded into Matlab, focusing on key variables, including the 'U,' 'V,' 'time,' and the 'base date.' With these variables loaded, the script proceeded to calculate the sum of NaN (Not-a-Number) values in each timestamp for the 'U' component. Subsequently, the script sorted the array of sum of NaN values in descending order, ensuring that timestamps with the least NaN values appeared first in the sorted list. The rearranged indices from the sorting operation were then applied to the 'time' array. Next, the 'base date' was added to each element in the rearranged 'time' array to calculate 'datenum,' which was subsequently converted into a date string. Finally, the script used the date shift function to eliminate duplicate dates. The resulting dates, as listed in Table 3.1, were chosen for evaluating DINEOF and its performance. These dates where chosen in this manner on the premise that they had the highest percentage of the domain measured, and hence the highest amount of data which could be removed for testing.

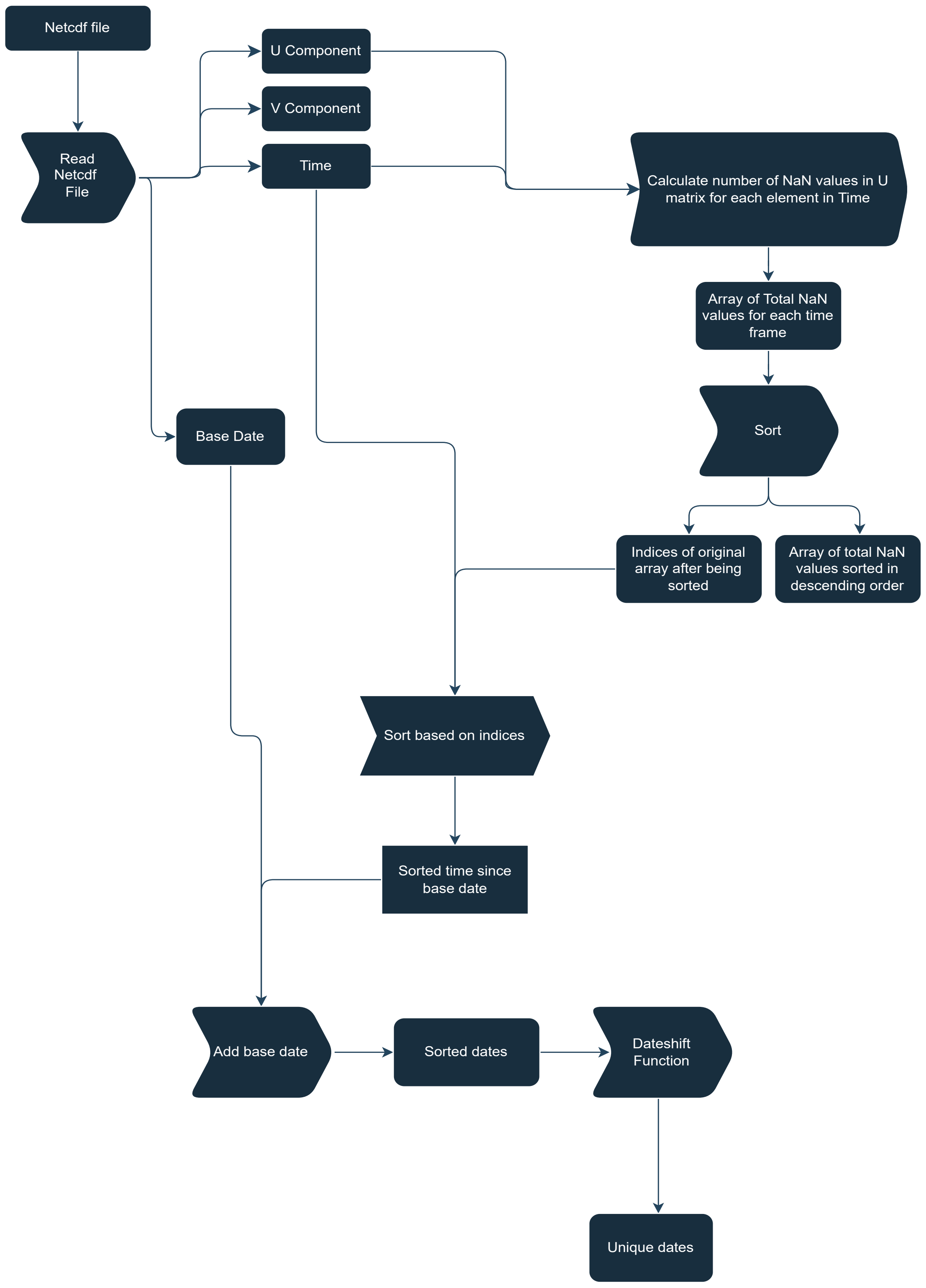

Netcdf file
Read Netcdf File
U Component
V Component
Time
Calculate number of NaN values in U matrix for each element in Time
Array of Total NaN values for each time frame
Sort
Indices of original array after being sorted
Array of total NaN values sorted in descending order
Base Date
Sort based on indices
Sorted time since base date
Add base date
Sorted dates
Dateshift Function
Unique dates

Figure 3.2: *Date_finder* flow chart

Table 3.1: Dates to be used for analysis

Dates for Analysis		
27-Mar-2022	26-Apr-2022	15-May-2022
23-May-2022	02-Jun-2022	15-Jun-2022
01-Aug-2022	07-Oct-2022	10-Nov-2022
20-Dec-2022	05-Jan-2023	10-Feb-2023
14-Mar-2023	01-Apr-2023	30-May-2023

3.3 | Main processing pipeline

This section provides details on a series of functions and scripts developed in both Matlab and Python, which were employed to semi-automatically process data, execute DINEOF, and assess its performance. A comprehensive flowchart illustrating the entire processing pipeline can be found in Figure 3.3.

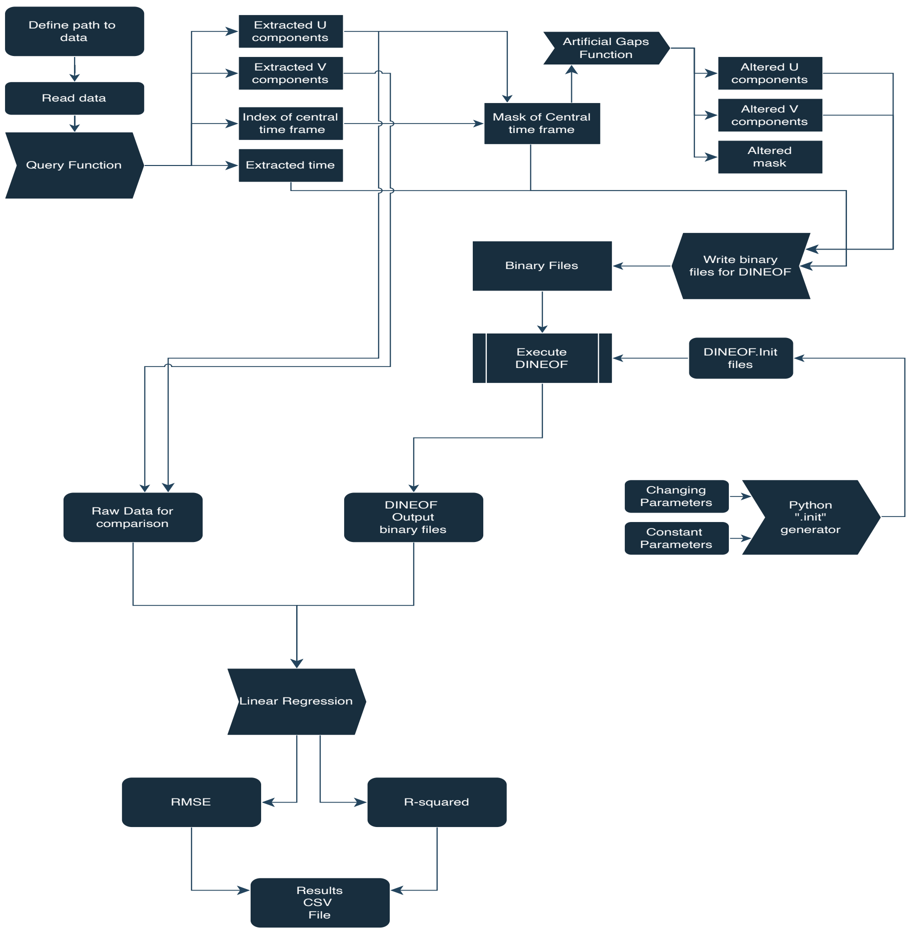


Figure 3.3: Main data processing pipeline.

Given the complexity of the numerous steps involved in generating the final CSV files containing essential information for subsequent analysis, this section is organised into several subsections. Each subsection delves into specific aspects of the processing pipeline, including the query function, the procedure for introducing artificial gaps, the creation of binary files for DINEOF, the automatic generation of ".init" files, and finally, the computation and storage of results in CSV format.

3.3.1 | Query Function

The "Query function", as presented in Appendix 5.6, has been specifically designed to extract a defined range of HF radar data from the extensive NetCDF file encompassing the entire dataset. This process is executed based on a provided central date and a user-specified number of days. The function takes six crucial parameters into account: the filename of the NetCDF file, global matrices encompassing the U and V components of the HF radar data, a global time array, the desired central date, and the number of days intended for extraction.

The operation of this function begins with the extraction of the *base date* attribute from the time variable within the NetCDF file, pinpointing the datasets' commencement date. Subsequently, both this base date and the specified central date are converted into Matlab datenums. This transformation enables the calculation of the total number of hours between the two dates. Utilising this information, the function determines the start and end indices within the global time array that correspond to the desired date range.

Following this initial stage, the function proceeds to generate an array comprising the dates to be extracted, converting these dates into date strings. It then proceeds to slice the U and V data arrays, based on the computed indices, effectively isolating the data relevant to the chosen date range. Simultaneously, the function also extracts the corresponding time data from the global time array.

In its final step, the function identifies the index of the central date within the array of extracted dates. As a result, the function returns this index, alongside the extracted U and V data arrays, the associated time values, and the array containing date strings. This meticulous methodology facilitates the precise extraction of data centred around the central date, enabling targeted analysis and evaluation.

3.3.2 | Introducing artificial gaps

The subsequent phase in the data processing pipeline, following the data query, involves the introduction of artificial gaps within the central time frame. This task was achieved by designing a dedicated function, which takes into account a 2D mask, user-specified parameters defining the dimensions of patches (i and j), a maximum allowable number of patches (k), and a seed for controlling random number generation and ensuring repeatability.

The initial step of this function involves rigorous validation of the input mask to ensure its compliance as a binary mask consisting solely of 0s and 1s. Subsequently, a *"modified mask"* is initialised as a replica of the input mask. The *"modified mask"* is then employed to identify all possible starting points for $i \times j$ patches. Within the main loop, the function randomly selects one of these starting points and proceeds to reset all values within the corresponding $i \times j$ patch in the *"modified mask"* to 0. This operation is iterated until all available starting points have been utilised or the number of applied patches reaches the predefined maximum k.

Following the loop, the function proceeds to identify the indices where disparities exist between the original mask and the *"modified mask"*. These distinct indices are subsequently converted into a single linear index and stored in an array. Thus, this function serves as a pivotal tool for the modification of a given 2D binary mask. An illustrative example depicting the data before and after the addition of artificial gaps can be observed in Figure 3.4.

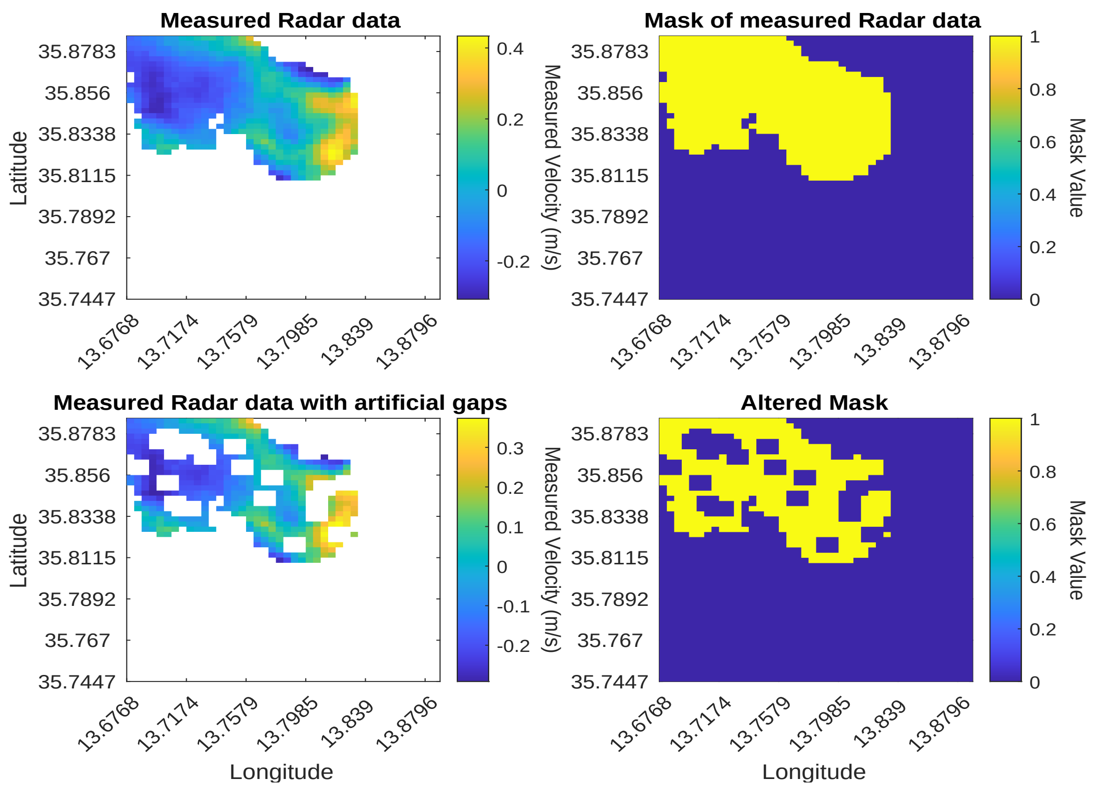


Figure 3.4: Example of HF radar data and corresponding mask before (top) and after (bottom) adding artificial gaps.

3.3.3 | Binary File Generation

This subsection focuses on the Matlab script detailed in Appendix 5.6, which served the purpose of automatically generating the essential binary files required for DINEOF processing. In Figure 3.3, this functionality is illustrated as the uppermost section leading to the component labelled "Binary Files." Having previously discussed both the data query and artificial gap functions, the primary emphasis within this subsection revolves around their integration to create the necessary binary files.

To commence this process, lists were created to define the parameters such as dates, time intervals, patch sizes, and the number of patches. The script employs nested loops to systematically iterate through every possible combination of these parameters. For each combination, the "query" function is called to extract the pertinent U and V data along with the corresponding time information. This information is subsequently used to initialise a 2D mask representing the central time frame.

Within these nested loops, the script executes various data manipulations. The "artificial gap" function comes into play to introduce artificial NaN values into a modified 2D mask. The indices derived from this altered mask are employed to selectively remove data from both the U and V matrices. Subsequently, the script prepares to save the manipulated data by constructing file names following a specific format. It proceeds to write the U, V, time, and mask data to the output files using a custom "uwrite()" function designed for creating binary files. This comprehensive script generates an equally comprehensive collection of output files for each unique combination of dates, time intervals, and removed data, facilitating in-depth data analysis.

3.3.4 | Automation of the initialisation file

The Python function detailed in Appendix 5.6 was developed to streamline the process of generating ".init" files essential for the operation of DINEOF. These ".init" files play a pivotal role in specifying data locations, calculation parameters, and output file paths, ensuring the proper execution of DINEOF. The function accommodates a range of parameters, encompassing dates, data duration (in days), data component types (U and V), alpha values (parameter specifying the strength of the filters), normalisation settings, EOF value outputs, patch sizes, and the number of patches.

Initially, the function establishes fundamental file paths for data storage, result tracking, and output destinations. Additionally, it defines constant values integral to multiple ".init" files. Within the function, nested loops systematically iterate through each conceivable combination of input parameters, thereby generating distinctive file names and paths for each parameter set. For each data type (U and V), the function generates a new initialisation file and populates it with imperative variable assignments. These assignments encompass file paths for data, mask, and time, as well as DINEOF-specific parameters. Subsequently, these initialised files are stored in designated directories, primed for collective use in processing data with DINEOF.

Through the invocation of this function with various sets of input parameters, it became possible to efficiently generate multiple ".init" files concurrently. For example, given the 15 distinct dates and diverse lists of parameter combinations, this script effortlessly produced over 3000 individual files. This capability proved immensely advantageous in facilitating large-scale data analysis via DINEOF.

3.3.5 | Error metrics

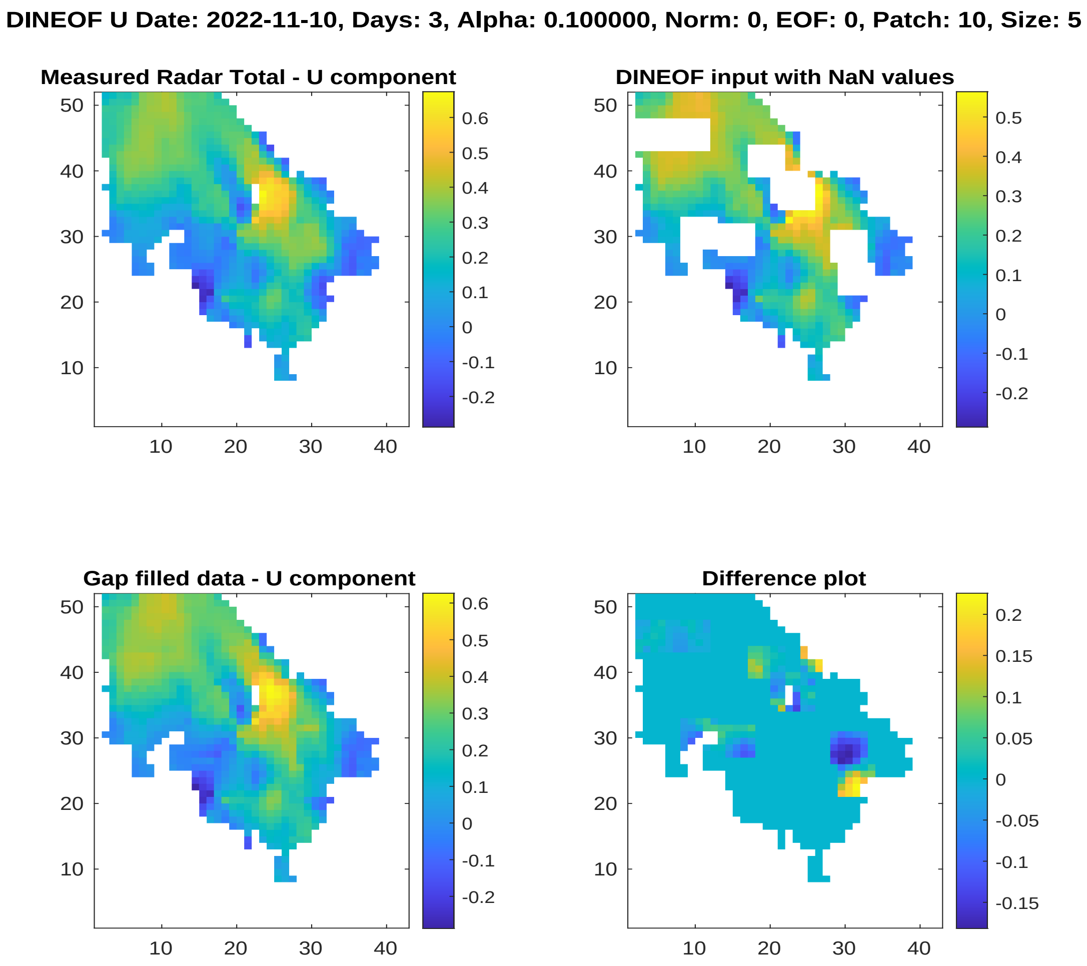


Figure 3.5: Example data before and after DINEOF.

In this subsection, the final script used as part of the main data processing pipeline, as depicted in Figure 3.3, and which is presented in Appendix 5.6. This script functions as an automated data analysis tool for assessing the performance of DINEOF.

The script commences by specifying the directory containing the ".data" files generated by DINEOF and initialises a table to store the results from linear regression. It then proceeds to read the year-long NetCDF file containing the U and V data.

Within the primary loop of the script, pairs of ".data" files containing the U and V velocity components are processed. Parameters such as the date of interest, length of data in days, alpha value, norm value, EOF value, number of patches, and patch size are extracted from the filenames. Subsequently, the script queries the year-long NetCDF file to obtain slices of data that match these parameters. After extracting the relevant slices, a mask is created and modified using the same seed to identify which indices within the ".data" files were gap-filled using DINEOF.

Linear regression models were fitted to the selected slices of data, such as can be seen in Figure 3.6, resulting in the calculation of both root mean square error (RMSE) and R^2 values, which serve as performance metrics. These metrics, along with the parameters extracted earlier, are stored in a results table for further analysis. The script also calculates the percentage of missing data that was handled by DINEOF. Consequently, by systematically recording performance metrics and parameters in a results table, the script constitutes the final component of the main processing pipeline. An illustrative example of the data before and after being processed by DINEOF can be observed in Figure 3.5. The difference plot clearly demonstrates that only the artificially introduced gaps were filled in by DINEOF.

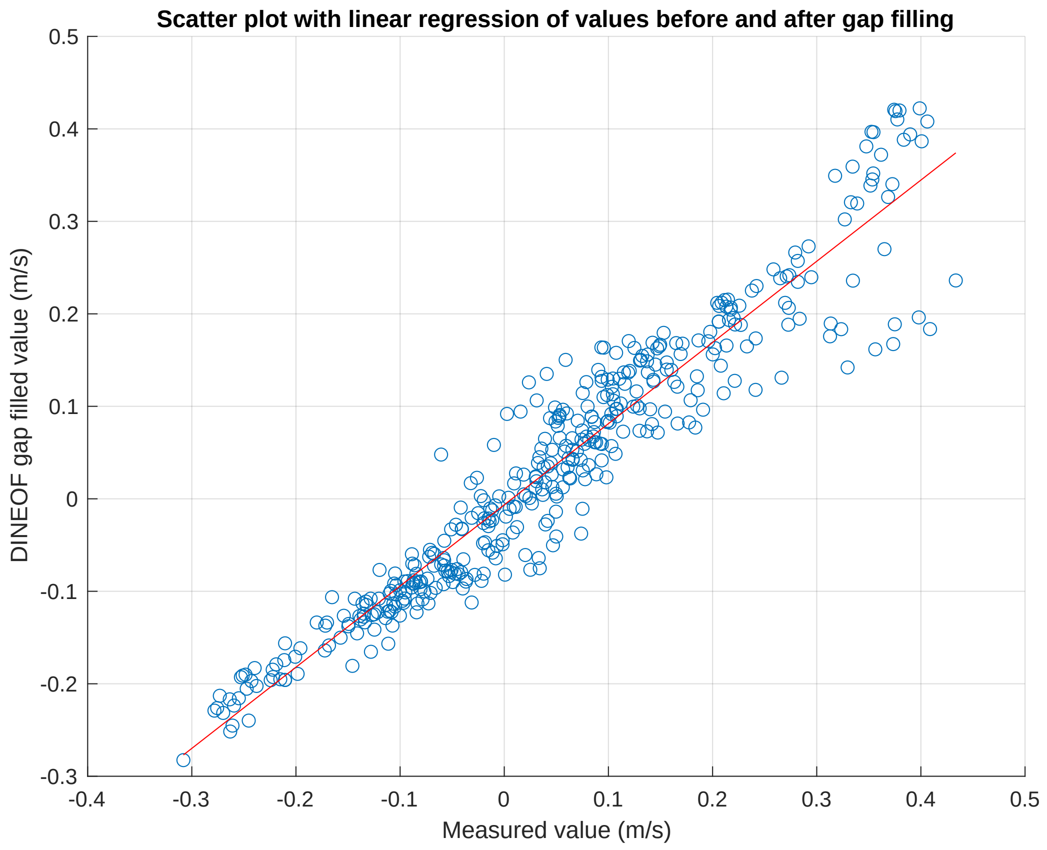


Figure 3.6: Example correlation plot used to calculate RMSE and R^2.

3.4 | CSV analysis

The processes and scripts employed to analyse the CSV files generated in Section 3.3.5 are now discussed. Two separate CSV files were created, each designed to address distinct research questions. The first CSV file was devised to investigate the impact of temporal data and the parameters in the ".init" file on the performance of DINEOF for a specific time frame, as mentioned in Section 3. The second CSV file was generated to explore any potential correlation between the extent of missing data and DINEOF's performance.

Both CSV files shared a common structure, consisting of nine columns. These columns were as follows:

1. **Date:** This column contained the date associated with the data.
2. **Days:** Indicating the total number of days' worth of data input into DINEOF.
3. **Alpha:** A numerical parameter representing the strength assigned to DINEOF filters.
4. **Norm:** A binary indicator denoting whether normalisation was enabled.
5. **EOF:** Another binary indicator used to enable or disable the writing of left and right modes of the input matrix.
6. **Patch Size:** Representing the size of each artificially added gap.
7. **Number of Patches:** Indicating the count of removed patches.
8. **RMSE:** Abbreviation for root mean square error, serving as a measure of the model's accuracy.
9. **R-squared:** Another result of the linear regression model, measuring the quality of fit for the gap-filled data.

3.4.1 | Finding the optimal parameters

A comprehensive dataset, comprised of 1,740 results, was generated and archived in a CSV file. To navigate through this extensive dataset and identify the parameter combinations that yielded superior performance, a specialised Python script was developed, as detailed in Appendix 5.6. The primary objective of this script was to query the data stored in the CSV file and pinpoint the optimal set of parameters—specifically, "days," "alpha," "norm," and "EOF." These parameters were selected to simultaneously minimise RMSE and maximise R^2 for each of the 15 specified dates.

In addition to the aforementioned script, an alternative approach was explored involving a genetic algorithm. This genetic algorithm, as outlined in Appendix 5.6, was designed to fine-tune the parameters specific to DINEOF.

The second Python script was developed using DEAP (Distributed Evolutionary Algorithms in Python) to optimise multi-parameter data through a genetic algorithm. The primary objective of the algorithm was to minimise both the RMSE and the deviation of R^2 from 1. In this genetic algorithm, individuals in the population were represented as lists of randomly selected parameters from the "days," "alpha," "norm," and "EOF" columns of the dataset. An evaluation function calculated the fitness of each individual based on these performance metrics.

The genetic algorithm incorporated several genetic operators: two-point crossover for mating, bit flip mutation, and tournament selection of size 3 for selecting individuals. The algorithm aimed to evolve a population of 200 individuals across 10 generations. During each generation, individuals were selected, mated, mutated, and evaluated, with the best-performing individual being recorded in the hall of fame. The algorithm's parameters, such as population size, mating and mutation probabilities, and the number of generations, tuned to optimise the DINEOF parameters via trial and error.

3.4.2 | Determining the effect of missing data

After determining the parameters for DINEOF, a second CSV file was generated, incorporating an additional column to track the percentage of remaining data after introducing artificial gaps into the time frame of interest. This secondary CSV file encompassed a total of 750 individual data points, stemming from 15 distinct dates, patch sizes ranging from 1×1 to 10×10, and the number of patches varying from 10 to 50 in increments of 10.

The code in Appendix 5.6 was developed to produce two sets of plots. The first set included histograms illustrating the distributions of three pertinent variables: the percentage of available data, RMSE, and R^2. The second set comprised scatter plots designed to explore the relationships between the percentage of available data for DINEOF and the two performance metrics.

In addition to these visualisations, regression analysis was conducted on the scatter plots to quantitatively assess potential relationships between the performance metrics and the quantity of available data. This statistical approach complemented the visual insights, offering a more rigorous evaluation to determine whether significant trends or patterns could be identified.

3.5 | Analysing DINEOF as a reanalysis tool

This section of the methodology delves into the evaluation of DINEOF's reanalysis capabilities by examining two distinct scenarios. The first scenario involves a 9-day dataset comprised of hourly files spanning nine days. Concurrent gap-filling is applied to showcase the available data for training the Empirical Orthogonal Functions (EOFs). In contrast, the second scenario entails a 3-day dataset, which consolidates hourly data from multiple 3-day files. The objective here is to reconstruct the middle seven days of the 9-day interval on an hourly basis. This comparison is designed to simulate and distinguish between different post-event analysis scenarios. To initiate this setup, the pertinent data is extracted from the annual NetCDF file, as described above. Additionally, ".init" files are generated to cater to the specific requirements of each dataset.

After executing DINEOF and completing the gap-filling process, the resulting files are processed and organised into a series of Matlab Structures based on their respective dates. These structures are then saved as ".mat" files for subsequent import into Python for in-depth analysis. The analysis comprises the following key components:

1. **Data Visualisation**: The U and V components are plotted alongside the reconstructed totals for each dataset, allowing for visual inspection and comparison.

2. **Weekly Statistics**: Weekly statistics are derived and presented for each dataset, providing insights into the temporal patterns and variations within the data.

This analysis was conducted using a single Python Jupyter notebook, which is available in its entirety in Appendix 5.6.

3.5.1 | Visualising *U* , *V*, and *Totals*

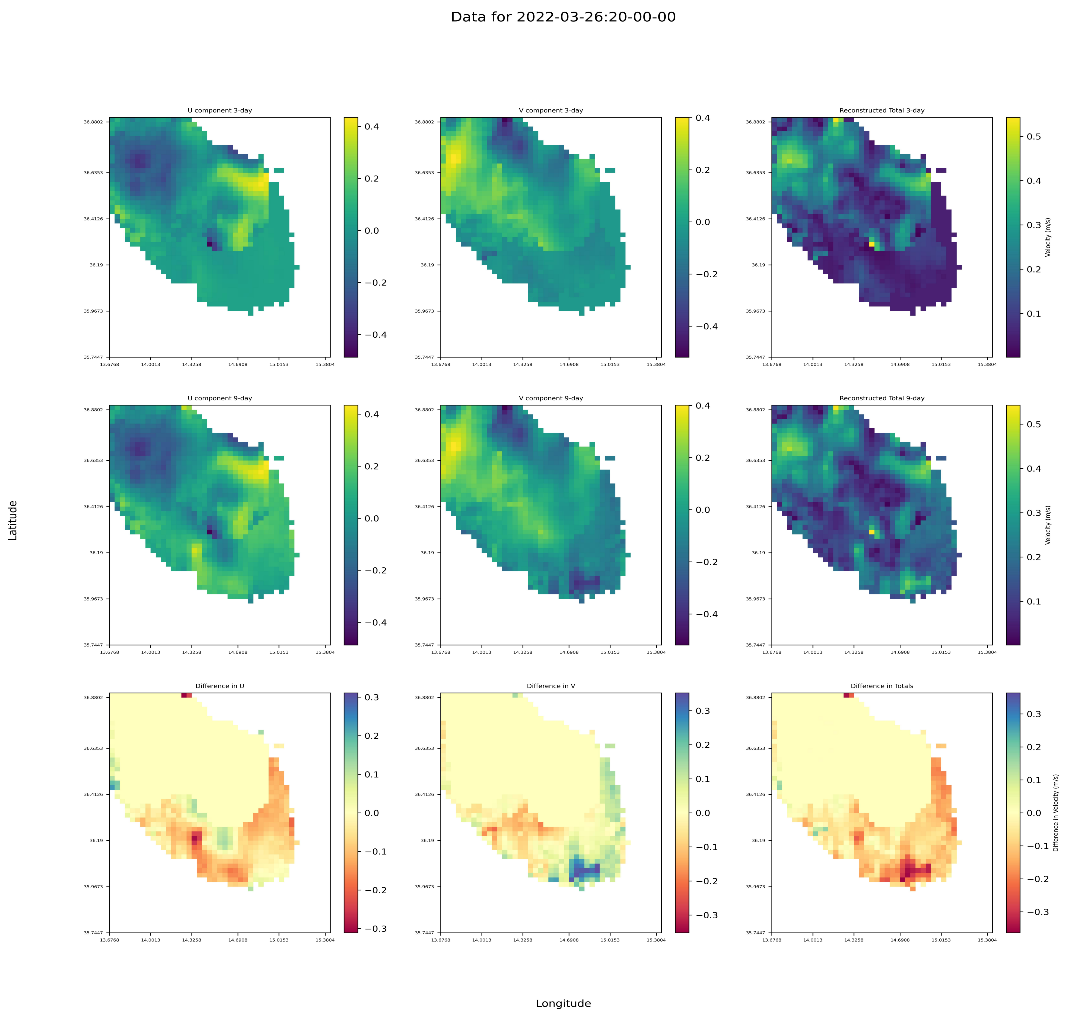


Figure 3.7: Example visualisation plot

One of the fundamental tasks within the scope of this analysis is data visualisation. This involves the creation of a series of figures, each comprising nine subplots. These subplots are structured to display the *U* and *V* components alongside the reconstructed *Total* for each of the two datasets, with the differences between them illustrated at the bottom. An illustrative example of such a visualisation is presented in Figure 3.7.

3.5.2 | Calculating statistics for each dataset

The second task involved the computation of weekly statistics, a task facilitated by the versatile capabilities of the NumPy package in Python. To accomplish this, a dedicated function was developed, primarily designed to calculate the mean, variance, minimum, and maximum values within the datasets. Careful attention was paid to ensure that these statistical calculations considered only the non-NaN values, ensuring the accuracy and relevance of the results. In the execution of this function, weekly statistics were systematically derived and securely stored. The exclusion of NaN values was a crucial step to maintain the robustness of the statistical analysis.

To organise and manage these statistics effectively, two dictionaries were initialised: "stats 3 day" and "stats 9 day." Each of these dictionaries corresponded to distinct datasets, specifically the reconstructed totals for hourly data and the 9-day batch dataset. These dictionaries were structured with keys representing different statistical measures and were initially populated with empty lists. These lists were progressively filled with the computed statistics for each week within the respective datasets.

Upon the completion of this iterative process, the dictionaries "stats 3 day" and "stats 9 day" held comprehensive statistical data, encompassing mean, variance, minimum, and maximum values. To present these statistics in an organised manner, a dedicated function was created. This function generated a figure composed of four distinct subplots, with each subplot dedicated to visualising a specific statistical measure. The function efficiently looped over each of the statistical dictionaries, enabling a straightforward comparative analysis of the two datasets.

4

Results

4.1 | DINEOF performance evaluation

The performance of DINEOF, like many algorithms, may be influenced by a number of factors. The intricate relationship between the quality of the output and the configuration of its parameters, length of data input, and the proportion of missing data was examined with intense interest. A systematic approach to deepen the understanding of these relationships was used. Hence, the following relationships were investigated.

1. **Parameter Sensitivity:** How were different configurations of DINEOF's parameters affecting its performance? Was an optimal parameter set identified under different scenarios?

2. **Input Data Length:** How did the length of the input data series influence the quality of the reconstructed data? Was there a minimum or maximum threshold for data length that ensured reliable outputs?

3. **Missing Data Proportion:** How did DINEOF perform under varying degrees of data sparsity? Was DINEOF able to maintain its reliability when faced with extremely sparse datasets?

4.1.1 | Analysing parameter sensitivity & input data length

As alluded to in Section 3.4.1, the initial investigation focused on four key parameters: the number of days used as input for DINEOF and three parameters found within the *".init"* file. These included alpha (which dictates DINEOF filter strength, norm (used for normalisation), and EOF (for computing left and right modes of the input matrix). After examining a total of 3480 files generated, a total of 1740 data points were compiled to identify any relationships between these parameters and performance metrics, root mean square error (RMSE) and the correlation coefficient (R^2) as calculated from scatter plots as can be seen in Figure 3.6.

The Python script described in Section 3.4.1 was used to sift through the comprehensive CSV file that recorded the results of each DINEOF run along with the associated dates and parameter settings. In Table 4.1, the best results for each of the 15 days can be viewed. The correlation values for all days were above 0.7, suggesting a strong congruence between original HF radar measurements and DINEOF's gap-filled outputs. Interestingly, April 26th 2022 stood out as it benefited from a 28-day data input, while the optimal data length for most days varied between 3 and 7 days.

Table 4.1: Table showing best combination of parameters for DINEOF

Date	#no of Days	Alpha	EOF	Norm	RMSE	R^2
2022-03-27	3	0.100	0	0	0.039230	0.945642
2022-04-26	28	0.010	0	0	0.042212	0.786014
2022-05-15	7	0.100	0	0	0.048773	0.876821
2022-05-23	7	0.005	0	0	0.042040	0.907875
2022-06-02	3	0.050	0	0	0.065584	0.776106
2022-06-15	7	0.100	0	0	0.042271	0.930268
2022-08-01	3	0.100	0	0	0.041977	0.890338
2022-10-07	3	0.100	0	0	0.041709	0.934338
2022-11-10	5	0.100	0	0	0.042341	0.972297
2022-12-20	5	0.005	0	0	0.043652	0.962006
2023-01-05	3	0.100	0	0	0.052625	0.928301
2023-02-10	3	0.100	0	0	0.083176	0.769230
2023-03-14	3	0.100	0	0	0.038099	0.937472
2023-04-01	3	0.100	0	0	0.022619	0.991248
2023-05-30	3	0.100	0	0	0.069253	0.889877

After scrutinising the entire dataset, it was found that neither normalisation nor EOF impacted DINEOF's performance. This leaves alpha as the sole parameter to fine-tine. The most frequently occurring optimal alpha value was found to be 0.1, with the full range spanning from 0.005 to 0.1.

The best-performing parameter combination was identified for April 1st 2023. This combination involved a three day input window, an alpha value of 0.1 and both EOF and normalisation set to zero. This configuration yielded and RMSE of less than 0.025 and a correlation coefficient greater than 0.99.

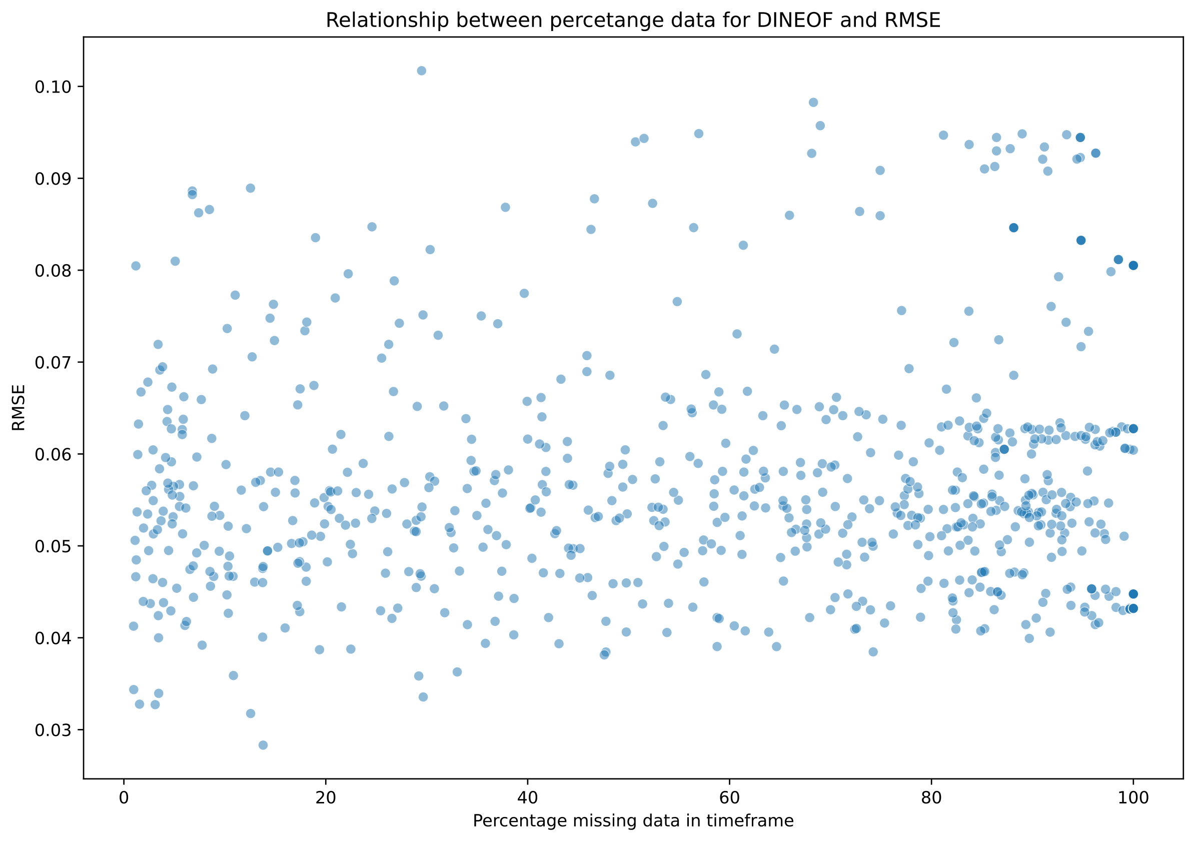
Additionally, a second python script was employed using the DEAP package as an alternative methodology as described in Section 3.4.1. This methodology was selected for its capacity to identify globally optimal parameter combinations in multivariate data sets. With this approach, the optimal parameters were found to be a three-day data input, an alpha value of 0.1, and both normalisation and EOF set to zero. According to the GA estimations, these settings should produce an RMSE in the range of 0.05 and an R^2 value exceeding 0.89 for DINEOF.

4.1.2 | Analysing the effect of missing data on DINEOF

After identifying the optimal parameters in the preceding experiment, the effect of missing data on DINEOF's performance was evaluated. As mentioned in Section 3.4.2, a second CSV file was created with an additional column dedicated to the percentage of data remaining after the addition of artificial gaps. A total of 750 unique data points were examined, which were derived from the 15 different dates, patch sizes ranging from 1×1 to 10×10 pixels, and varying numbers of patches from 10 to 50 in increments of 10.

In Figure 4.1, the relationship between the percentage missing data in the gap filled timeframe and the performance metrics, RMSE and R^2, are illustrated. From these two scatter plots, no conclusive impact on either RMSE or R^2 was observed as a result of varying the percentage of missing data in the gap filled timeframe. This suggests that DINEOF as an algorithm is already optimised to yield the best possible results, provided that sufficient prior data for the reconstruction of a specific time frame was available.

A more detailed analysis of the CSV file and the corresponding histograms are presented in Figure 4.2. A broad range of values for the performance metric was revealed by these histograms, confirming that DINEOF's effectiveness was not systematically compromised by missing data. Despite the considerable variations in the percentages of available data, a consistent pattern in either RMSE or R^2 was not displayed as evidenced by both the histograms and scatter plots. This consistency was strongly supportive of the notion that DINEOF had been effectively optimised for handling missing data, confirming its suitability for data reconstruction tasks when adequate prior data available.



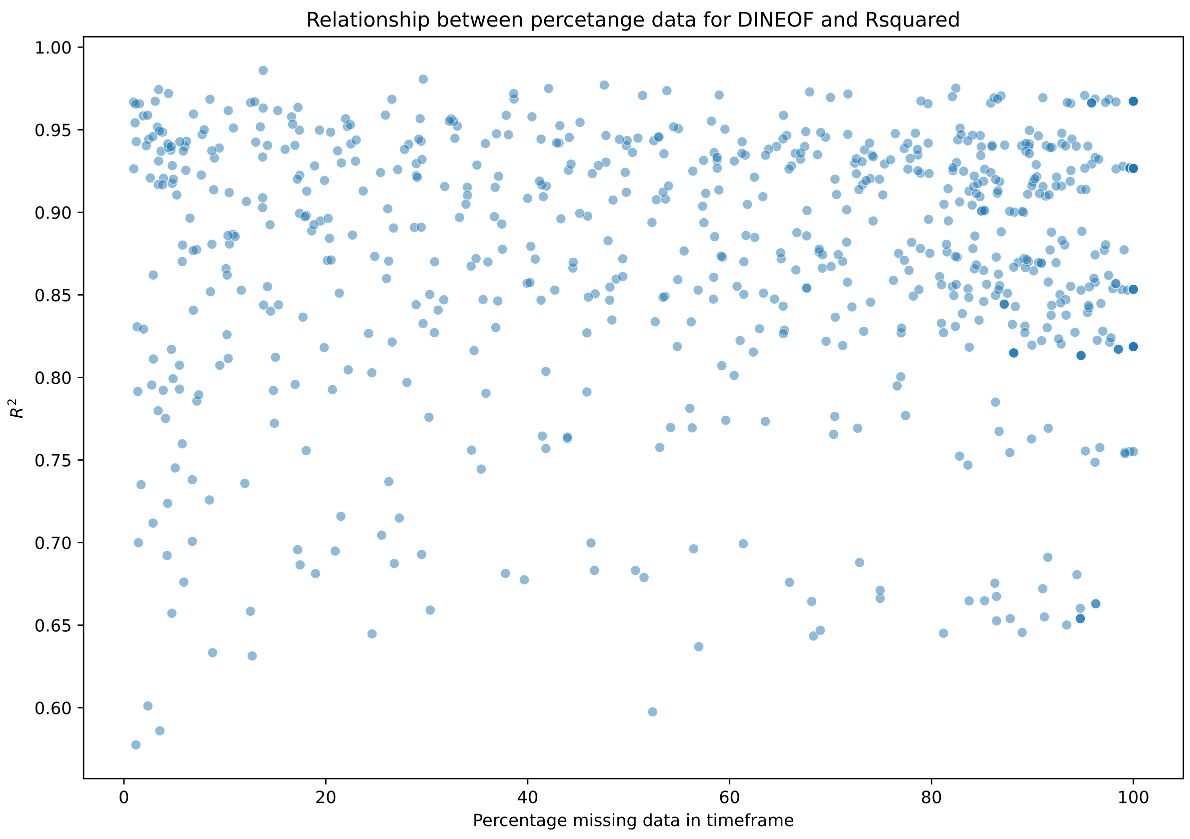


Figure 4.1: Scatter plot of RMSE and R^2 against percentage missing data in gap filled time frame.

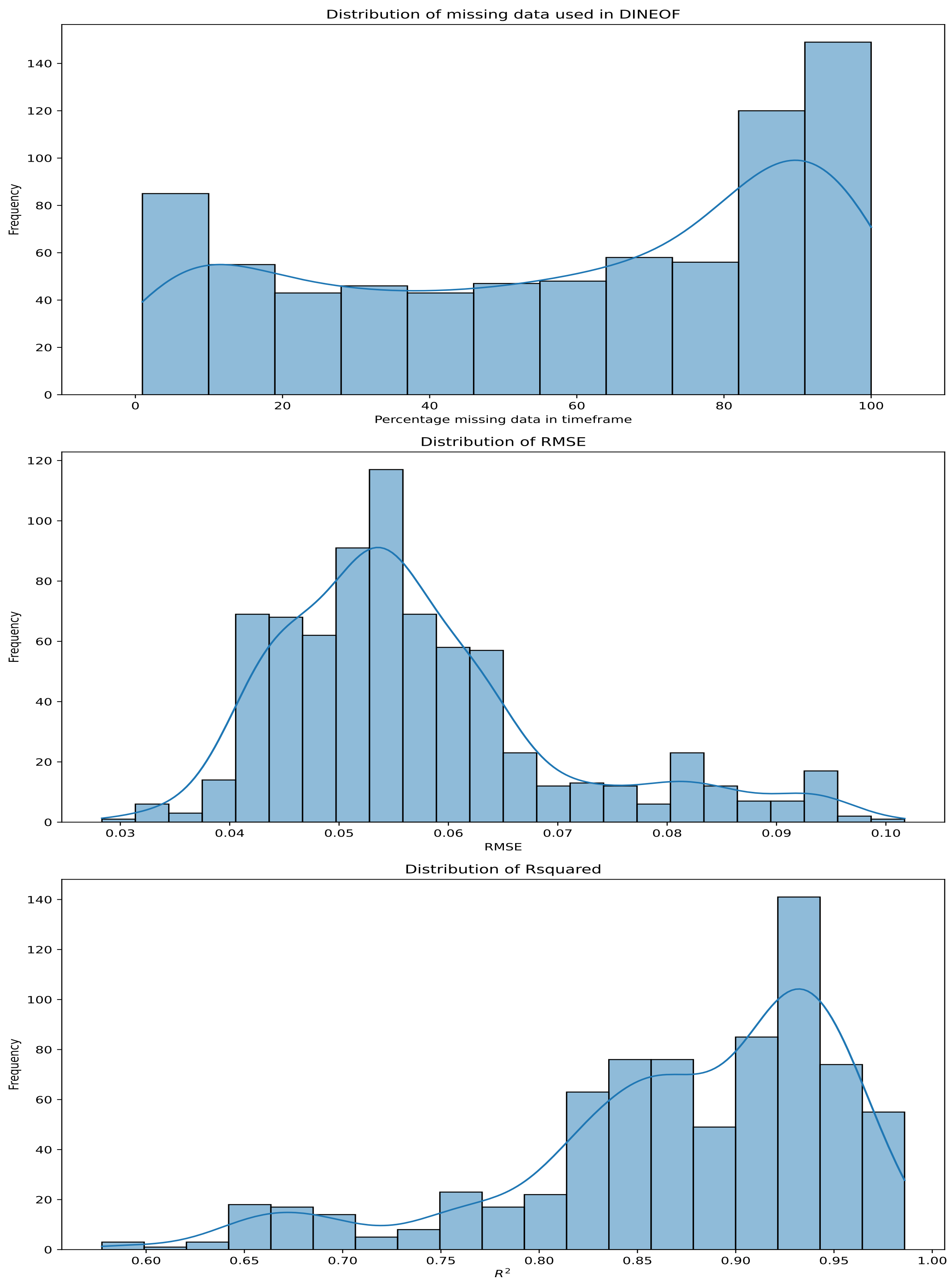


Figure 4.2: Distributions of percentage missing data in time frame, RMSE, and R^2.

4.2 | Different reconstruction methods

This section of the results will handle the insights obtained from Section 3.5. One should keep in mind that the mask for both datasets was identical. The mask contained all the points which had data at least 50% of the time over the week of interest. This should have left space for gap filling to take place, allowing for gap filling to be compared across the two methods of using DINEOF. The following concepts will be viewed and discussed in this section of the results:

1. Looking at the differences between the 9-day batch gap filling and the reconstructed hour by hour gap filling.
2. Looking at the statistics for the reconstructed totals of each scenario.

4.2.1 | DINEOF as a reanalysis tool

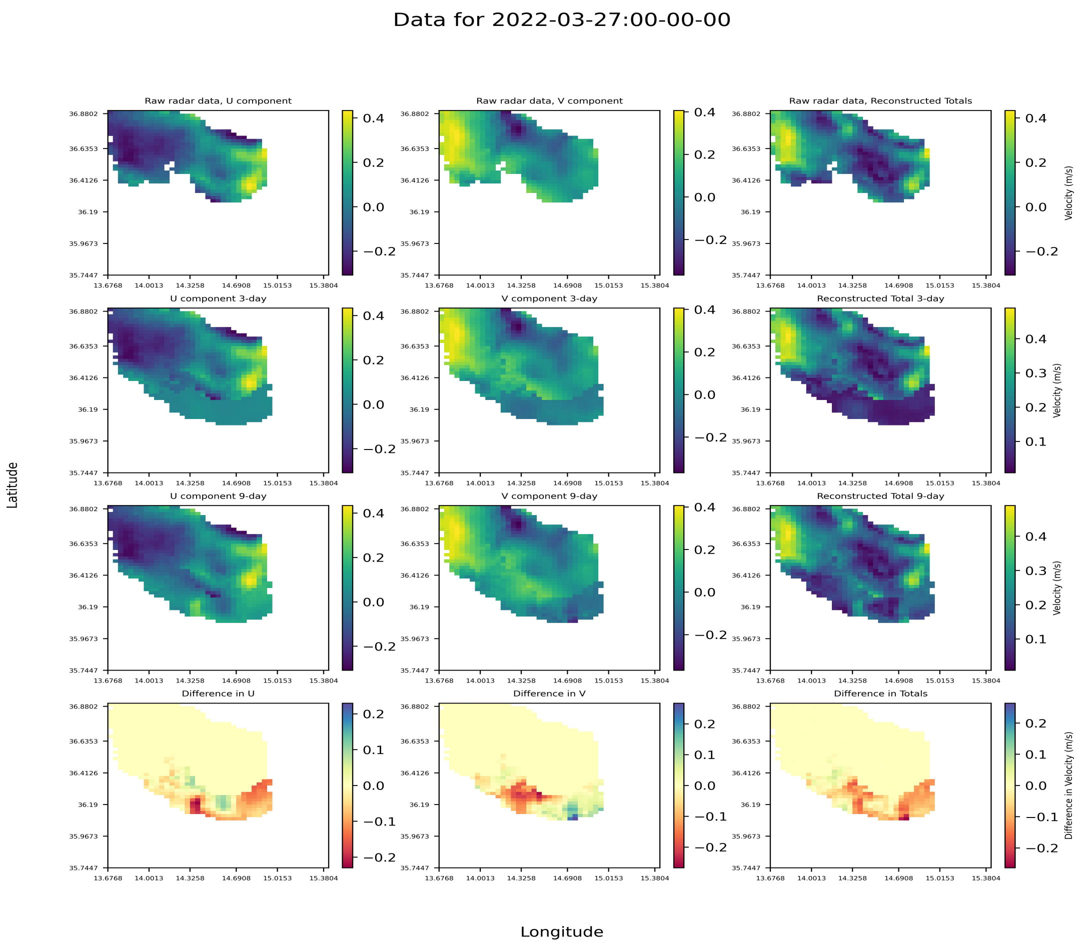

Data for 2022-03-27:00-00-00
Raw radar data, U component
Raw radar data, V component
Raw radar data, Reconstructed Totals
U component 3-day
V component 3-day
Reconstructed Total 3-day
U component 9-day
V component 9-day
Reconstructed Total 9-day
Difference in U
Difference in V
Difference in Totals
Velocity (m/s)
Difference in Velocity (m/s)
Latitude
Longitude

Figure 4.3: Array of subplots containing *U*, *V*, and *Totals* for "3-day" and "9-day" datasets and their differences, March 27 2022

Having ran DINEOF on both the ’3-day’ and ’9-day’ datasets, as described in Section 3.5, the performance of DINEOF and the effectiveness of the gap filling for these two datasets is now investigated. As detailed in Section 3.5, the initial observations were illustrated in figures containing a matrix of subplots. Each row of subplots showcases the *U*, *V*, and *Total* vectors: the raw data in row 1, the ’3-day’ dataset in row 2, the ’9-day’ dataset in row 3, and the difference between the two datasets in row 4. While we could generate such plots for each hourly interval, we confined the analysis to specific time frames for simplicity. These time frames correspond to the dates listed in Table 3.1 at midnight.

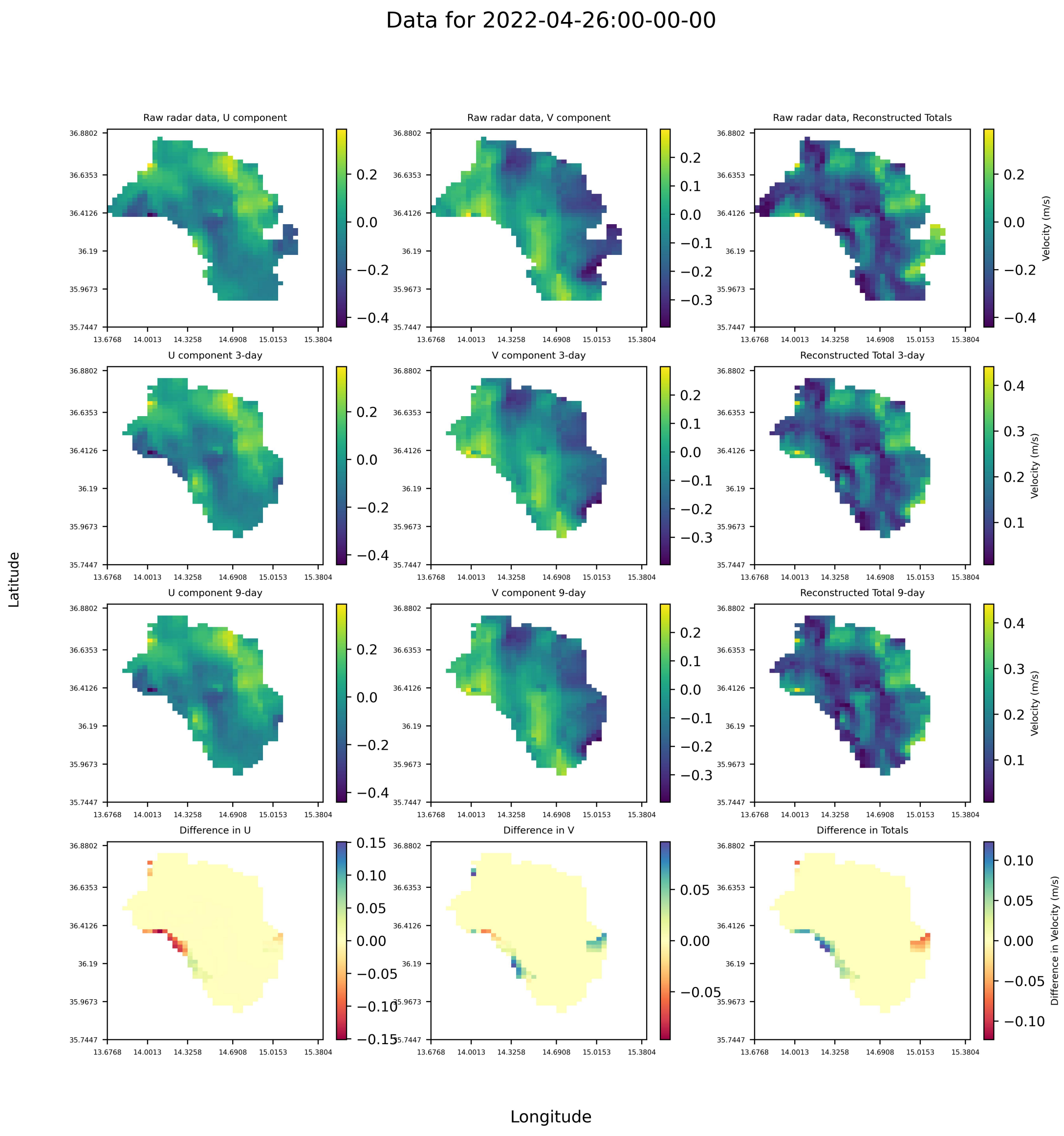


Figure 4.4: Array of subplots containing *U*, *V*, and *Totals* for "3-day" and "9-day" datasets and their differences, April 26 2022

The first date analysed is April 27th, 2022. The matrix of subplots for this date is presented in Figure 4.3. This figure highlights the differences in the *U*, *V*, and *Total* vectors between the two datasets. These variations arise from the differing lengths of data used to create the empirical orthogonal functions within DINEOF. In Figure 4.7, the raw data, alongside the two datasets, is shown on a singular quiver plot. This figure reveals differences in both the magnitude and direction of the reconstructed total surface current vectors between the ’3-day’ and ’9-day’ datasets.

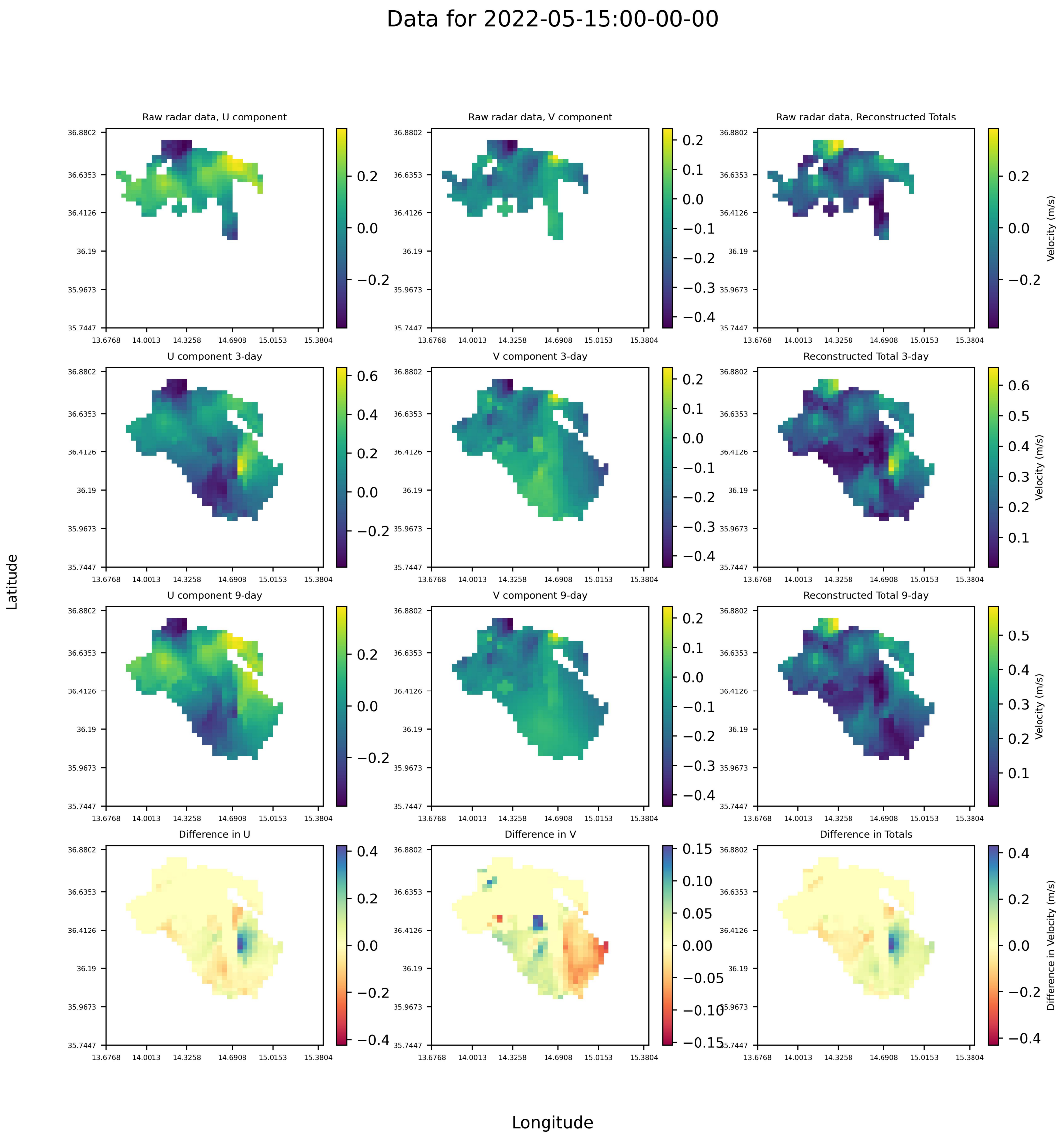

Data for 2022-05-15:00-00-00
Raw radar data, U component
Raw radar data, V component
Raw radar data, Reconstructed Totals
U component 3-day
V component 3-day
Reconstructed Total 3-day
U component 9-day
V component 9-day
Reconstructed Total 9-day
Difference in U
Difference in V
Difference in Totals
Velocity (m/s)
Difference in Velocity (m/s)
Latitude
Longitude

Figure 4.5: Array of subplots containing *U*,*V*, and *Totals* for "3-day" and "9-day" datasets and their differences, May 15 2022

The next date we examined is April 26th, 2022, with results depicted in Figure 4.4. A significant portion of the data aligned precisely, resulting in large sections where the differences were zero. Regions that DINEOF gap-filled are visible along the edges of the measured HF radar data. The corresponding quiver plot for this date is displayed in Figure 4.8. Here, discrepancies in both magnitude and direction between the two datasets were observed. However, at first glance, both datasets appear to offer realistic interpretations.

On May 15th, 2022, as shown in Figure 4.5, a substantial portion of the data was missing, providing ample opportunity for gap filling. The reconstructed total vectors for both datasets seem to follow a similar trend, indicating DINEOF's ability to consistently reconstruct missing data, regardless of input time length. The quiver plot for this date, in Figure 4.9, reveals a pronounced discrepancy in the magnitude of gap-filled vectors in the Eastern areas of the HF radar domain. While the vector magnitudes differ between the datasets, the directions are largely consistent. This suggests that both datasets are sufficient to determine vector directions, but the '9-day' dataset provides a more accurate estimation of missing values.

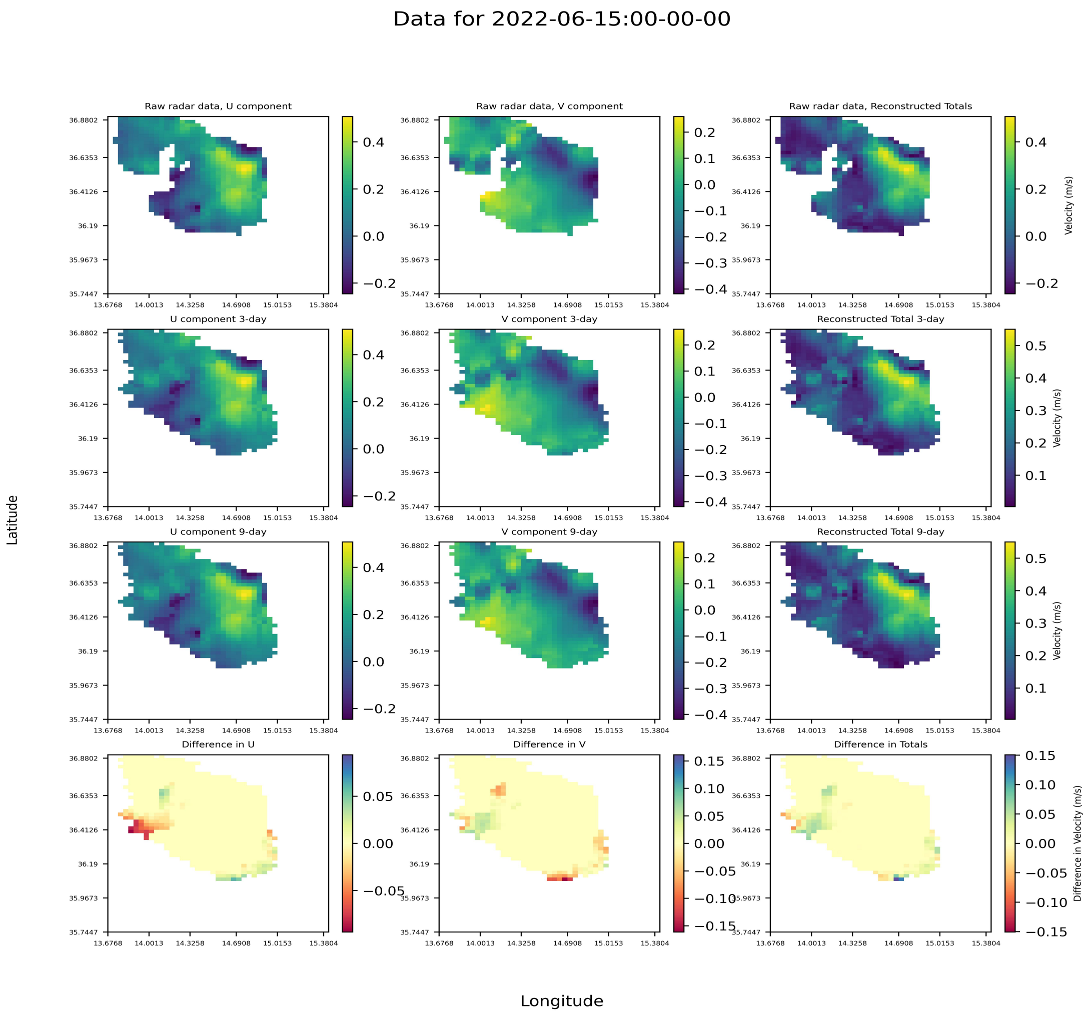


Figure 4.6: Array of subplots containing *U*,*V*, and *Totals* for "3-day" and "9-day" datasets and their differences, June 15 2022

The last date studied was June 15th, 2022. The data for this date is shown in Figure 4.6. It is evident which areas DINEOF gap-filled, especially on the southeastern edge of the raw HF radar measurements and a notable patch in the northwestern region. Based on the difference plots, vector magnitudes between the two datasets are quite similar. To further investigate these vectors, we referred to the quiver plot in Figure 4.10. This figure indicates that the most significant directional differences are along the edges. However, despite these differences, both datasets seem to provide plausible vectors via DINEOF's gap filling.

In conclusion, our analysis across different dates suggests that while there are discernible differences between the '3-day' and '9-day' datasets, both are effective in gap filling to varying degrees. The choice between them would likely depend on the specific requirements of a study or application.

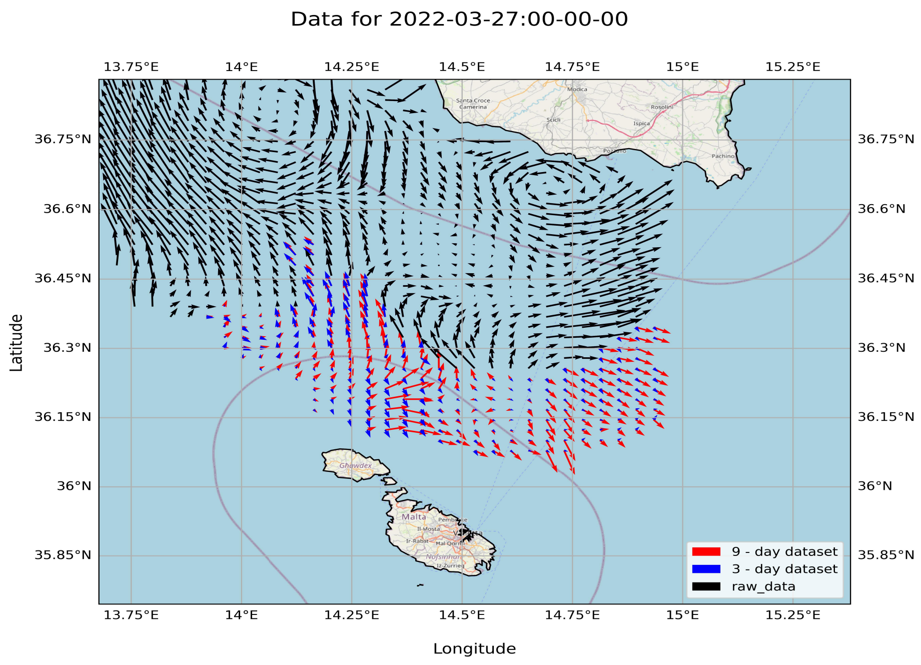


Figure 4.7: Quiver plot for March 27 2022.

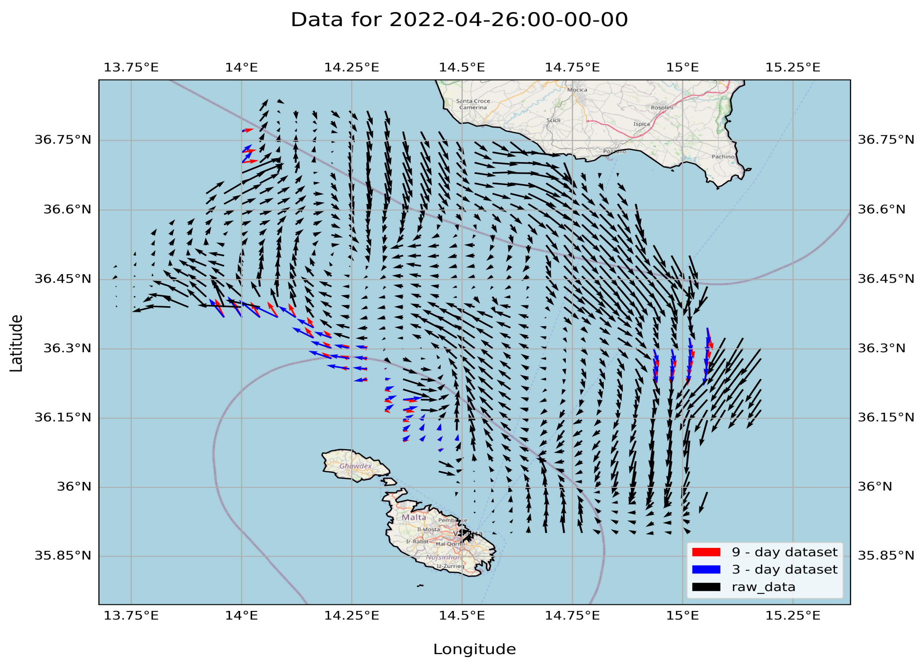


Figure 4.8: Quiver plot for April 26 2022.

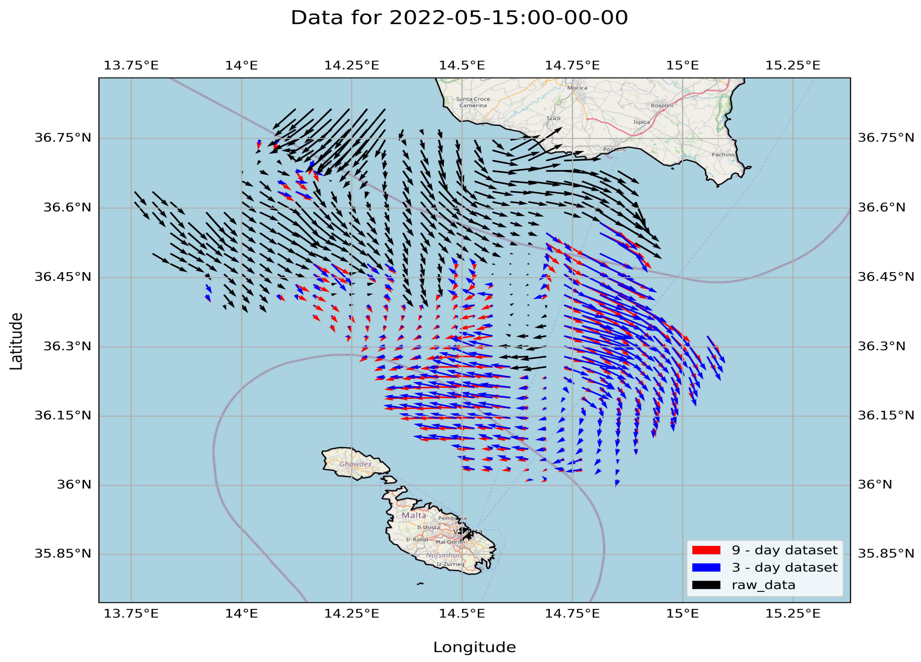


Figure 4.9: Quiver plot for May 15 2022.

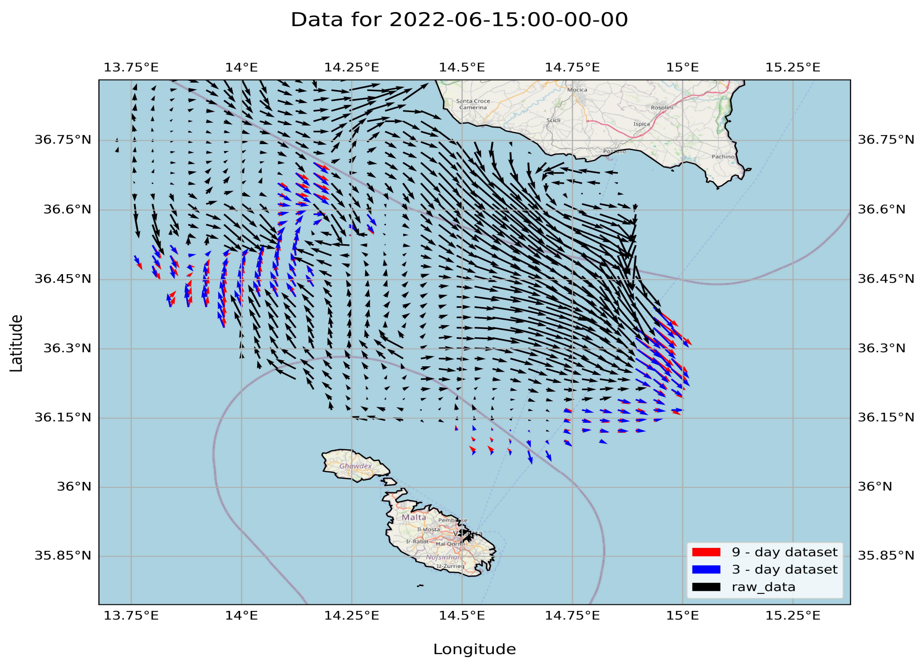


Figure 4.10: Quiver plot for June 15 2022.

4.2.2 | Weekly statistics as quality control

Weekly statistics often offer critical insights into the behaviour and consistency of time dependent data. For each set of reconstructed totals, the mean, variance, minimum and maximum were calculated as described in Section 3.5.2. The mean was calculated to reflect the central tendency of the weekly data, indicating stable conditions when consistency is present. The variance was calculated as a measure of the spread within the data, highlighting weeks with high and low amounts of volatility within the reconstructed data. Finally the min and max were calculated to extract each data's range for the week, as a means of outlier detection.

The metrics calculated for each data set were visualised within a series of subplots which can be seen in Figure 4.11. As can be seen in this figure, there is little difference in the weekly mean between the '3-day' and '9-day' datasets. From the mean time-series, an increase in the overall surface current velocities, peaking above 0.30 cm/s from December 2022 till April 2023, whereas the average current velocity did not exceed an average of 0.25 cm/s for the remainder of the time.

The variance over time as a measure of the variability of the surface currents within the datasets. One is able to notice the increase in variance within the datasets from December 2022 till April 2023. This increase in variance was to be expected as the weather in Malta tends to be most unpredictable in the winter. Discrepancies between the two data sets were noticed within the weekly maximum values. These discrepancies were a result of the differences in reconstruction outcomes of DINEOF.

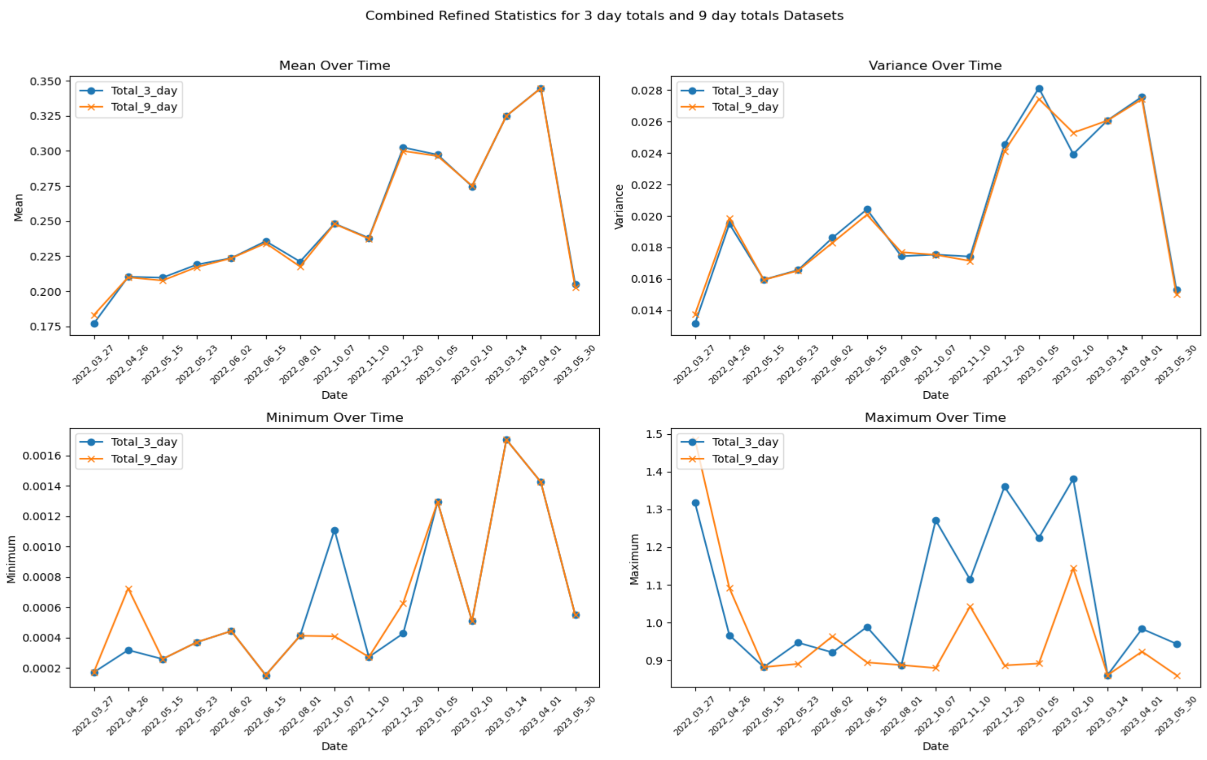


Figure 4.11: Weekly Mean, Variance, Minimum, and Maximum statistics for the ’3 day’ and ’9 day’ datasets.

In Figure 4.12, the mean and variance for the raw HF radar data can be seen along side the two datasets. Whilst the mean over time for the raw data was very similar to that of the two datasets, the variance was significantly greater than that of the reconstructed data. This suggests that the mask which only allowed data to be reconstructed where data was present at least 50% of the time played a large role in the minimisation of variance within the datasets.

In the realm of quality control, these weekly statistics play an indispensable role. The calculation of mean, variance, min, and max serves not just for analytical purposes but also as QC checkpoints. A consistent mean across datasets can affirm the reliability of data acquisition processes. The comparison with raw HF radar data further underscores this point. The significant reduction in variance after limiting the reconstruction to occur within the mask proves the efficacy of these statistics as QC checks. Such measures ultimately ensure that the data being worked upon is both representative and reliable, paving the way for credible analyses and conclusions.

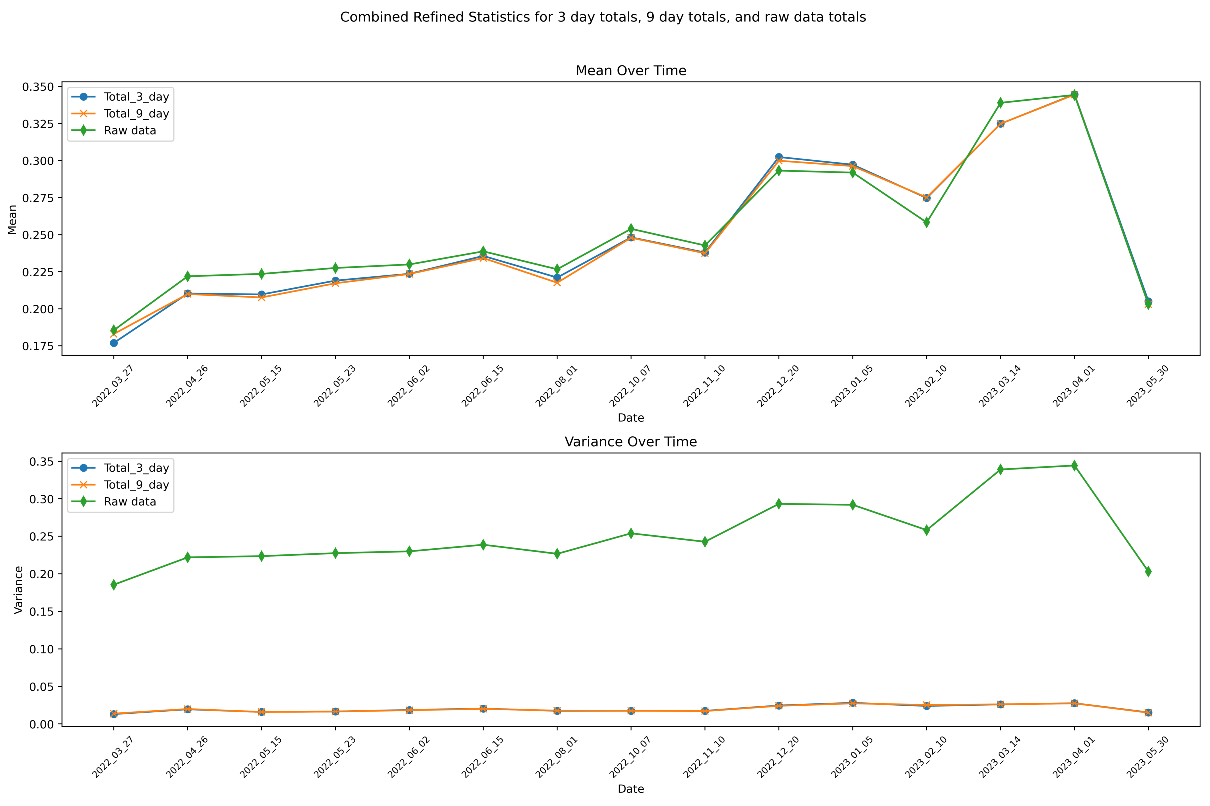


Figure 4.12: Weekly Mean, and Variance statistics for the ’3 day’ dataset, ’9 day’ dataset, and raw HF radar totals.

5

Conclusions

In this final chapter, a distilled discussion of the findings presented in the results chapter are presented, focusing primarily on the evaluation of the DINEOF method on data obtained by the CALYPSO HF radar network. Key parameters were fine tuned, to determine the optimal input data length and alpha value. Drawing an analogy to adjusting learning rates in neural networks, the optimisation process revealed a critical threshold beyond which additional data does not equate to improved performance, echoing the concept of overfitting in machine learning.

The analysis further scrutinised the robustness of DINEOF when confronted with incomplete data. Despite increasing amounts of missing data in the time frames to be gap filled, no significant impact on the performance metrics was observed. This resilience positions DINEOF as a steadfast tool in the field of oceanographic data reconstruction, even in less than ideal conditions.

Concluding this chapter, and dissertation, an overview of the potential enhancements and future direction are acknowledged. The study suggest the need for more sophisticated data querying and a more nuanced approach to parameter selection. Moving forward, comparative studies with alternative algorithms, the integration of external validation, and the exploration of advanced AI techniques are recommended to further advance the field. This discussion not only encapsulated the essence of the findings presented in this dissertation, but also paces the way for future endeavours that will continue to push the boundaries of our understanding of marine environments.

5.1 | DINEOF parameter optimisation

The parameter optimisation conducted within this study focused on two main parameters, these being the length of data used as input for EOF construction and the alpha parameter which can be thought of as the learning rate in an artificial neural network (ANN). The length of time was set at increments of 3, 5, 7, 14, and 28 days which equated to a total of 72, 120, 168, 336, and 672 hours worth of information for EOF reconstruction within a central time frame. The values of alpha chosen for the experiment were 0.005, 0.01, 0.02, 0.05, and 0.1.

The overall combination which simultaneously lowered the root mean square error and maximised the correlation coefficient between the measured values and the values obtained by DINEOF were found on the 1^{st} April 2023. This included an input data length of 3 days or 72 hours, and an alpha value of 0.1. These findings were also obtained when a genetic algorithm was trained and allowed to run for 100 generations, with a projected RMSE of 0.05 and R^2 of 0.89. Thus suggesting that too much information for EOF construction might be a detriment to gap filling, similar to over training in an ANN.

5.2 | Missing data and its effects on DINEOF's performance

The analysis of DINEOF's performance in the context of missing data underlines its robustness and adaptability in reconstructing HF radar data and oceanographic data as a whole. The histograms in Figure 4.2 shed light on crucial performance indicators, with the diversity in the percentage of missing data showcasing an extensive assessment over a range of data omission conditions. The RMSE distribution, while showcasing a breadth of error magnitudes, hints at a concentration toward lower values, signalling DINEOF's general reliability in reconstruction accuracy. More compelling is the R^2 histogram that , despite the data's incompleteness, suggest a consistent ability of the model to account for a significant proportion of variance in the data, a critical indicator of its performance.

The scatter plots in Figure 4.1 present a complete view of the relationship between data completeness and DINEOF's performance. The RMSE shows a marginal increase as data sparsity increases, indicating a slight but not steep decline in performance with escalating data omission. This subtle trend suggests that while DINEOF is generally resilient to data gaps, there is a threshold beyond which its accuracy falters. Conversely, the R^2 values remain largely unaffected by the percentage of missing data, highlighting DINEOF ability in recreating data irrespective of its completeness. This dichotomy in performance metrics highlights DINEOF's complex interplay with data sparsity, trading accuracy for unwavering consistency when accounting for data variability.

DINEOF demonstrated notable resilience against data incompleteness, maintaining reliable performance despite varying levels of data omission. Although a slight increase in RMSE suggests a minor reduction in accuracy as more data is missing, the consistent R^2 values highlight DINEOF's stable variance explanation, regardless of data sparsity. These findings thus validated DINEOF's utility for data reconstruction in scenarios with limited data.

5.3 | Exploring DINEOF as a reanalysis tool

The next idea explored in this study was to evaluate the effectiveness of DINEOF as a potential software tool for reanalysis, with a particular focus on how differences in how the data is input into DINEOF, hence the '3-day' and '9-day' datasets. Throughout the study, DINEOF showcased consistency across varied time frames, spanning from April to June 2022. The matrix subplots and quiver plots serves as invaluable visual aids, offering nuanced perspectives into the reconstructed data. While both datasets adequately captured overarching directionality, the '9-day' dataset exhibited a distinct edge in determining precise magnitudes, especially when larger segments of data were absent.

However, it is crucial to approach the results presented with a degree of caution. The reconstructed data, as presented, cannot be accepted as definitive. While DINEOF's capabilities in handling and filling data are commendable, the outcomes presented earlier require further scrutiny. Validating the reconstructed data against another instrument is essential to ensure its reliability. Instruments such as a Lagrangian drifter could server as an effective benchmark, offering a comparative lens to assess the accuracy and fidelity of DINEOF's reconstruction.

Thus, while DINEOF emerged as a promising candidate for reanalysis endeavours, further validation is required. The '9-day' data seemed to be better suited in producing surface currents which realistic magnitudes overall. This further highlights the importance of integrating additional validation mechanisms, such as the Lagrangian drifter, solidifying the credibility and applicability of these findings.

5.4 | Weekly statistics for quality control

The analysis of weekly statistics has proven invaluable in understanding and interpreting the behaviour and consistency DINEOF's outputs. The calculated metrics - mean, variance, minimum, and maximum - served dual functions: they provided insight into the central tendency and volatility of the data, and also acted as quality control checkpoints. Particularly, the stability reflected in the weekly mean across both the '3-day' and '9-day' datasets, while the identification of discrepancies in the maximum values highlighted the subtle intricacies involved in DINEOF's reconstruction process.

A pivotal observation was the significant escalation in mean surface current velocities, particularly from December 2022 to April 2023. This period coincided with Malta's unpredictable winter, leading to an anticipated increase in variance. However, it is the comparison with the raw HF radar data that provided a critical revelation, the variance within the reconstructed data was considerably lower than that of the original data. This phenomenon could be attributed to the strategic implementation of a mask, which significantly minimised variability by restricted reconstruction to location with data presence of at least 50%.

The role of weekly statistics extends beyond simple numerical representations; they are instrumental in quality assurance and the reliability of data processing methodologies. The consistency in the mean and the controlled variance are testimonies to the robustness of the data acquisition and reconstruction process. Furthermore, the effective use of a masking strategy has demonstrated its effectiveness in reducing data variability, ensuring that the data utilised for subsequent analyses is both representative and holds a high standard of credibility.

5.5 | Areas for Improvement

While the research conducted has provided valuable insights, it is essential to acknowledge that there were certain shortcomings and opportunities for improvement. Hindsight reveals areas where more efficient approaches could have allowed for additional experiments and deeper analysis. Three notable areas for improvement are highlighted below:

1. **Enhancing Data Querying:** The data querying function used in this research exhibited limitations. It did not effectively flag ranges of dates with incomplete data or entire missing time frames. Implementing a more robust data querying method could improve the identification of such data gaps and enhance the overall data analysis process.

2. **Optimising Parameter Analysis:** The analysis of DINEOF's parameters revealed intrinsic flaws. Specifically, two of the parameters selected had no discernible impact on DINEOF's ability to reconstruct missing data in our particular case. Streamlining the process of parameter selection and evaluation could lead to more meaningful insights into the method's performance.

3. **Validation with External Data Sources:** While the research focused on gap filling using DINEOF, validating the reconstructed data against external data sources, such as surface drifters or other instruments, could provide a more comprehensive assessment of gap-filling performance under real-world conditions. Integrating external validation could offer valuable insights into the accuracy and reliability of the gap-filled data.

Recognising these areas for improvement can guide future research endeavours towards more efficient and robust methodologies.

5.6 | Possible Future Research

While this study primarily focused on evaluating the performance of the DINEOF method in gap filling, there remains areas of potential research to explore in the future. Building upon the foundational work presented in this research, future studies could venture into various avenues of oceanographic data analysis. Some potential directions for future research include:

1. **Algorithmic Comparisons:** Conducting comparative studies to assess the effectiveness of the DINEOF method in comparison to other existing or emerging gap-filling algorithms. Tailoring algorithms to suit the unique characteristics of the Mediterranean Sea and addressing specific challenges encountered in regions like the Malta-Sicily channel could yield valuable insights.

2. **Ecosystem Impact Studies:** Delving deeper into the impact of enhanced accuracy and reliability in HF radar data on our understanding of local ecosystems. Research in this area could involve investigating the dispersion patterns of natural and anthropogenic

substances and their influence on marine biodiversity. It could also entail tracking the migration patterns of various marine species and assessing their ecological significance.

3. **Long-Term Climate Studies:** Leveraging improved data quality to explore long-term changes in the marine environment of the Mediterranean Sea. This avenue of research could provide valuable insights into the effects of global climate change on sea surface currents, temperatures, and sea levels. Such findings would contribute significant data to global climate models.

4. **Advanced AI and Machine Learning Techniques:** Investigating advanced artificial intelligence (AI) and machine learning (ML) techniques to enhance gap-filling performance beyond the capabilities of the DINEOF method in HF radar data. Exploring the development and application of cutting-edge AI and ML models could lead to superior results in data reconstruction and analysis.

These potential future research areas hold promise for expanding our understanding of oceanographic data analysis and its applications in addressing pressing environmental and climate-related challenges.

References

Alvera-Azcárate, A., Barth, A., Parard, G., & Beckers, J.-M. (2016). Analysis of SMOS sea surface salinity data using DINEOF. *Remote Sensing of Environment*, 180, 137–145. https://doi.org/10.1016/j.rse.2016.02.044

Alvera-Azcárate, A., Barth, A., Sirjacobs, D., Lenartz, F., & Beckers, J. M. (2011). Data Interpolating Empirical Orthogonal Functions (DINEOF): a tool for geophysical data analyses. *Mediterranean Marine Science*, 12(3), 5. https://doi.org/10.12681/mms.64

Beckers, J. M. & Rixen, M. (2003). EOF Calculations and Data Filling from Incomplete Oceanographic Datasets*. *Journal of Atmospheric and Oceanic Technology*, 20(12), 1839–1856. https://doi.org/10.1175/1520-0426(2003)020<1839:ECADFF>2.0.CO;2

Buber, M., Toz, A. C., Sakar, C., & Koseoglu, B. (2020). Mapping the spatial distribution of emissions from domestic shipping in Izmir Bay. *Ocean Engineering*, 210, 107576. https://doi.org/10.1016/j.oceaneng.2020.107576

Crombie, D. D. (1955). Doppler Spectrum of Sea Echo at 13.56 Mc./s. *Nature*, 175(4459), 681–682. https://doi.org/10.1038/175681a0

Deidun, A., Gauci, A., Azzopardi, J., Camilleri, C., Cutajar, D., Chalabreysse, M., & Trinquard, F. (2018). Development of a Novel Tool for the Monitoring of Shipping Traffic Within the Strait of Sicily (Central Mediterranean): the BIODIVALUE AIS Vessel Tracker. *Journal of Coastal Research*, 85, 1356–1360. https://doi.org/10.2112/SI85-272.1

Emery, B. & Washburn, L. (2019). Uncertainty Estimates for SeaSonde HF Radar Ocean Current Observations. *Journal of Atmospheric and Oceanic Technology*, 36(2), 231–247. https://doi.org/10.1175/JTECH-D-18-0104.1

Evans, M. & Georges, T. (1979). Coastal Ocean Dynamics Radar (CODAR): NOAA's Surface Current Mapping System. *OCEANS '79*, 379–384. https://doi.org/10.1109/OCEANS.1979.1151297

Gauci, A., Drago, A., & Abela, J. (2016). Gap Filling of the CALYPSO HF Radar Sea Surface Current Data through Past Measurements and Satellite Wind Observations. *International Journal of Navigation and Observation*, 2016, 1–9. `https://doi.org/10.1155/2016/2605198`

Hernández-Carrasco, I., Solabarrieta, L., Rubio, A., Esnaola, G., Reyes, E., & Orfila, A. (2018). Impact of HF radar current gap-filling methodologies on the Lagrangian assessment of coastal dynamics. *Ocean Science*, 14(4), 827–847. `https://doi.org/10.5194/os-14-827-2018`

Heron, M., Gomez, R., Weber, B., Dzvonkovskaya, A., Helzel, T., Thomas, N., & Wyatt, L. (2016). Application of HF Radar in Hazard Management. *International Journal of Antennas and Propagation*, 2016, 1–14. `https://doi.org/10.1155/2016/4725407`

Kolukula, S. S., Baduru, B., Murty, P. L. N., Kumar, J. P., Rao, E. P. R., & Shenoi, S. S. C. (2020). Gaps Filling in HF Radar Sea Surface Current Data Using Complex Empirical Orthogonal Functions. *Pure and Applied Geophysics*, 177(12), 5969–5992. `https://doi.org/10.1007/s00024-020-02613-x`

Konik, M., Kowalewski, M., Bradtke, K., & Darecki, M. (2019). The operational method of filling information gaps in satellite imagery using numerical models. *International Journal of Applied Earth Observation and Geoinformation*, 75, 68–82. `https://doi.org/10.1016/j.jag.2018.09.002`

Liu, Y., Weisberg, R. H., & Merz, C. R. (2014). Assessment of CODAR SeaSonde and WERA HF Radars in Mapping Surface Currents on the West Florida Shelf*. *Journal of Atmospheric and Oceanic Technology*, 31(6), 1363–1382. `https://doi.org/10.1175/JTECH-D-13-00107.1`

Martinez-Pedraja, J., Shay, L. K., Haus, B. K., & Whelan, C. (2013). Interoperability of SeaSondes and Wellen Radars in Mapping Radial Surface Currents. *Journal of Atmospheric and Oceanic Technology*, 30(11), 2662–2675. `https://doi.org/10.1175/JTECH-D-13-00022.1`

Merz, C. R., Liu, Y., & Weisberg, R. H. (2021). Sea surface current mapping with HF radar – a primer. *Ocean Remote Sensing Technologies: High frequency, marine and GNSS-based radar*, 95–116. Institution of Engineering and Technology. `https://doi.org/10.1049/SBRA537E_ch4`

Nikolaidis, A., Georgiou, G., Hadjimitsis, D., & Akylas, E. (2014). Filling in missing sea-surface temperature satellite data over the Eastern Mediterranean Sea using the DINEOF algorithm. *Open Geosciences*, 6(1). `https://doi.org/10.2478/s13533-012-0148-1`

Roarty, H., Cook, T., Hazard, L., George, D., Harlan, J., Cosoli, S., Wyatt, L., Alvarez Fanjul, E., Terrill, E., Otero, M., Largier, J., Glenn, S., Ebuchi, N., Whitehouse, B., Bartlett, K., Mader, J., Rubio, A., Corgnati, L., Mantovani, C., Griffa, A., Reyes, E., Lorente, P., Flores-Vidal, X., Saavedra-Matta, K. J., Rogowski, P., Prukpitikul, S., Lee, S.-H., Lai, J.-W., Guerin, C.-A., Sanchez, J., Hansen, B., & Grilli, S. (2019). The Global High Frequency Radar Network. *Frontiers in Marine Science*, 6, 164. `https://doi.org/10.3389/fmars.2019.00164`

Rubio, A., Mader, J., Corgnati, L., Mantovani, C., Griffa, A., Novellino, A., Quentin, C., Wyatt, L., Schulz-Stellenfleth, J., Horstmann, J., Lorente, P., Zambianchi, E., Hartnett, M., Fernandes, C., Zervakis, V., Gorringe, P., Melet, A., & Puillat, I. (2017). HF Radar Activity in European Coastal Seas: Next Steps toward a Pan-European HF Radar Network. *Frontiers in Marine Science*, 4. https://doi.org/10.3389/fmars.2017.00008

Soomere, T., Andrejev, O., & Myrberg, K. (2012). A future technology of environmental management of semi-enclosed seas. *2012 IEEE/OES Baltic International Symposium (BALTIC)*, 1–7. https://doi.org/10.1109/BALTIC.2012.6249172

Tode, L., Klarwein, S., & Khodjet, L. (2021). *Plan Bleu Notes #42: Maritime transport in the Mediterranean: Status and challenges*. https://planbleu.org/en/publications/note42-maritime-transport-in-the-mediterranean-status-and-challenges/

Wyatt, L. R. (2012). Shortwave Direction and Spreading Measured with HF Radar. *Journal of Atmospheric and Oceanic Technology*, 29(2), 286–299. https://doi.org/10.1175/JTECH-D-11-00096.1

Zheng, G. & Li, X. (2024). Empirical Function Method: A Precise Approach for Filling Data Gaps in Satellite Sea Surface Temperature Imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 62, 1–14. https://doi.org/10.1109/TGRS.2023.3335940

Zhou, Y., Gai, Y., & Li, J. (2021). Research on Sea Surface temperature Reconstruction from long-term MODIS data. *IOP Conference Series: Earth and Environmental Science*, 631(1), 012032. https://doi.org/10.1088/1755-1315/631/1/012032

Álvarez, N. G., Adenso-Díaz, B., & Calzada-Infante, L. (2021). Maritime Traffic as a Complex Network: a Systematic Review. *Networks and Spatial Economics*, 21(2), 387–417. https://doi.org/10.1007/s11067-021-09528-7

Code used

dineof_prep_u.m

```
% Read netcdf file containing data
filelist = fuf('/Volumes/Dissertation
↪  work/my_dineof/RadarExample/RadarData2/testfile.nc', 1, 'detail');
% Initialise empty arrays and counter
datauGlob = [];
timeGlob = [];
cnt = 1;
% For each element in the netcdf file (each time)
for f  = 1:1:numel(filelist)

    filename = filelist(f);
    filename = filename{1};
    % Read the U component of the measured radials
    dataU = ncread(filename, 'u');
    % Read the time of the measured radials
    time = ncread(filename, 'time');

    % Merge data into a arrays
    for t = 1:1:numel(time)
        datauGlob(:,:,cnt) = dataU(:,:,t);
        timeGlob(cnt) = time(t);
        cnt = cnt + 1;
    end
end
% Create a mask for the usefull data
mask = zeros(52, 43);
for t = 1:1:numel(time)
```

```
    idx = find(datauGlob(:,:,t) > -9000);
    mask (idx) = 1;
end
%Show mask
pcolor(mask)
%Transpose the time array
timeGlob = timeGlob';

% Use uwrite function to write the data into a format which is understood
% by DINEOF
nbmots = 10485760;
uwrite('/Volumes/Dissertation
↪   work/my_dineof/RadarExample/radar_u.data',datauGlob,52,43,10,-9999,nbmots);
uwrite('/Volumes/Dissertation
↪   work/my_dineof/RadarExample/radardates_u.data',[1:1:10],10,1,1,-9999,nbmots);
uwrite('/Volumes/Dissertation
↪   work/my_dineof/RadarExample/radardates_u.mask',mask,52,43,1,-9999,nbmots);
```

dineof_result_check.m

```
%% Read Data Before DINEOF
filelist =
↪   fuf('/Users/mrekxa/Desktop/dineof/RadarExample/RadarData2/testfile.nc', 1,
↪   'detail');

datauGlob = [];
timeGlob = [];
cnt = 1;
for f  = 1:1:numel(filelist)

    filename = filelist(f);
    filename = filename{1};
    dataU = ncread(filename, 'u');
    dataV = ncread(filename, 'v');
    time = ncread(filename, 'time');

    for t = 1:1:numel(time)
        datauGlob(:,:,cnt) = dataU(:,:,t);
        datavGlob(:,:,cnt) = dataV(:,:,t);
        timeGlob(cnt) = time(t);
        cnt = cnt + 1;
    end
end

%% Read Data After DINEOF
[flag,outdata_u,imax,jmax,kmax,valex,nbmots] = uread('/Volumes/Dissertation
↪   work/my_dineof/RadarExample/radar_test_u_out.data');
outdata_u = reshape(outdata_u, imax, jmax, kmax);

[flag,outdata_v,imax,jmax,kmax,valex,nbmots] = uread('/Volumes/Dissertation
↪   work/my_dineof/RadarExample/radar_test_v_out.data');
outdata_v = reshape(outdata_v, imax, jmax, kmax);

%% Show before and after
for t = 1:1:numel(timeGlob)
    subplot(2,2,1);
    dataU = datauGlob(:,:,t);
    idx = find(dataU == -9999);
    dataU(idx) = nan;
    pcolor(dataU); shading flat;
    caxis([-0.6 0.6]);
    colorbar;

    subplot(2,2,2);
    dataV = datavGlob(:,:,t);
    idx = find(dataV == -9999);
    dataV(idx) = nan;
    pcolor(dataV); shading flat;
    caxis([-0.6 0.6]);
    colorbar;

    subplot(2,2,3);
    dataUout = outdata_u(:,:,t);
    idx = find(dataUout == 9999);
    dataUout(idx) = nan;
    pcolor(dataUout); shading flat;
    caxis([-0.6 0.6]);
    colorbar;

    subplot(2,2,4);
    dataVout = outdata_v(:,:,t);
    idx = find(dataVout == 9999);
    dataVout(idx) = nan;
    pcolor(dataVout); shading flat;
    caxis([-0.6 0.6]);
    colorbar;

   %{
    set(gcf, 'PaperUnits', 'centimeters');
    set(gcf, 'PaperSize', [40, 15], 'PaperPosition',[0,0,40,15])
    print('-dpng', ['C:\dineof\RadarExample\Plots\', num2str(t, '%03d'),
↪   '.png']);
    %}
    waitforbuttonpress;
end

close all
```

NAN_date_finder.m

```
%% Clear Previous Data
clear all
close all
clc

%% Read Before DINEOF

% Define the path to the NetCDF file
input_file = '/Users/mrekxa/Desktop/dineof/merged_monthlies/whole_year.nc';

% Obtain a list of NetCDF files in the specified directory
filelist = fuf(input_file, 1, 'detail');
filename = input_file;

% Initialize empty arrays to store data
datauGlob = [];
datavGlob = [];
timeGlob = [];
cnt = 1;

% Read 'u' and 'v' data from the NetCDF file
```

```matlab
dataU = ncread(filename, 'u');
dataV = ncread(filename, 'v');

% Read the time variable from the NetCDF file
time = ncread(filename, 'time');

% Read the 'base_date' attribute from the time variable
base_date = ncreadatt(filename, 'time', 'units');

% Split the input text by spaces
splitText = strsplit(base_date, ' ');

% Extract the date part from the attribute
datePart = splitText{end - 1};

% Extract the time part from the attribute
timePart = splitText{end};

% Combine the date and hour parts to form a datetime string
dateTimePart = [datePart, ' ', timePart(1:2)];

% Convert the datetime string to a MATLAB datenum
dateNum = datenum(dateTimePart, 'yyyy-mm-dd HH');

% Organize the data into arrays for further processing
for t = 1:numel(time)
    datauGlob(:, :, cnt) = dataU(:, :, t);
    datavGlob(:, :, cnt) = dataV(:, :, t);
    timeGlob(cnt) = time(t);
    cnt = cnt + 1;
end

% Count the number of NaNs in each time step
nanCount = zeros(1, size(datauGlob, 3));
for i = 1:size(datauGlob, 3)
    nanCount(i) = sum(sum(isnan(datauGlob(:, :, i))));
end

% Sort the data based on the number of NaNs in each time step
[sorted_NaN_Count, sorted_Indices] = sort(nanCount, 'descend');
sorted_U_data = datauGlob(:, :, sorted_Indices);
```

```
sorted_V_data = datavGlob(:, :, sorted_Indices);
sorted_time = timeGlob(sorted_Indices);

% Select the first n time steps for further analysis
time_to_add = sorted_time(1:100);

% Define the number of hours to add to the initial datenum
hoursToAdd = time_to_add;

% Add the hours to the dateNum to get new dateNum values
dateNum = dateNum + hoursToAdd / 24;

% Convert the updated dateNum back to date strings
dateStr = datestr(dateNum, 'yyyy-mm-dd HH:MM:SS');

% Convert the date strings to a datetime array
dtArray = datetime(dateStr, 'InputFormat', 'yyyy-MM-dd HH:mm:SS');

% Set the time to the start of each day in the datetime array
dtArray = dateshift(dtArray, 'start', 'day');

% Find the unique dates in the datetime array
uniqueDates = unique(dtArray);

% Display the unique dates to focus the research on
disp(uniqueDates(:))
```

Function: Query_dates.m

```
function [Extracted_U, Extracted_V, Extracted_time,
↪   date_of_interest_index,date_str_array] = query_dates(filename, datauGlob,
↪   datavGlob, timeGlob, centre_date, num_days)
    % Read the 'base_date' attribute from the time variable
    base_date = ncreadatt(filename, 'time', 'units');

    % Split the input text by spaces
    splitText = strsplit(base_date, ' ');

    % Extract the date part from the attribute
    datePart = splitText{end - 1};
    % Extract the time part from the attribute
    timePart = splitText{end};

    % Combine the date and hour parts to form a datetime string
    dateTimePart = [datePart, ' ', timePart(1:2)];

    % Convert the datetime string to a MATLAB datenum
    Base_dateNum = datenum(dateTimePart, 'yyyy-mm-dd HH');

    % convert date of interest into total hours since start of dataset
    hours_since_basedate=(datenum(centre_date,'yyyy-mm-dd')-Base_dateNum)*24;

    %calculate the start and end indices of the slices to be extracted
    hours_since_basedate_start=hours_since_basedate-24*(num_days/2);
    hours_since_basedate_end=hours_since_basedate+24*(num_days/2);

    %find the indices which match the calculated datenums
    date_index_start= find(timeGlob(:) == hours_since_basedate_start);
    date_index_end = find(timeGlob(:) == hours_since_basedate_end);

    %create array with datenums to be extracted
    date_arrays=date_index_start:date_index_end;

    %create array of dates which are extracted and converted to date strings
    date_str_array=datestr(Base_dateNum+timeGlob(date_arrays(:))/24,'yyyy-mm-dd
↪   HH:MM:SS');

    %extract data
    Extracted_U=datauGlob(:,:,date_index_start:date_index_end);
    Extracted_V=datavGlob(:,:,date_index_start:date_index_end);
    Extracted_time=timeGlob(date_index_start:date_index_end);

    % find index of centre date
    centre_index = find(timeGlob(:) == hours_since_basedate);
    date_of_interest_index=find(date_arrays == centre_index);
end
```

Function : Swap_Patches.m

```
function [modifiedMask, swappedIndices] = swap_patches(mask, k, i, j, seed)

    rng(seed);

    if ~ismatrix(mask)
        error('Input mask must be a 2D matrix');
    end

    if ~all(ismember(mask(:), [0, 1]))
        error('Input mask contains values other than 0s and 1s');
    end

    [rows, cols] = size(mask);
    modifiedMask = mask;

    % Only consider rows and columns that have space for an i by j patch
    [availableRows, availableCols] = find(modifiedMask(1:rows-i+1, 1:cols-j+1)
↪   == 1);

    if numel(availableRows) < k
        error('There are not enough available data areas');
    end

    % Initialize swappedIndices as an empty matrix
    swappedIndices = [];

    for area = 1:k
        idx = randi(numel(availableRows));
        row = availableRows(idx);
        col = availableCols(idx);

        modifiedMask(row:row+i-1, col:col+j-1) = 0;

        availableRows(idx) = [];
        availableCols(idx) = [];
    end

    % Get indices of the elements that differ in original and modified mask
    [changedRows, changedCols] = find(mask - modifiedMask);
    for idx = 1:numel(changedRows)
        swappedIndices(end+1) = sub2ind(size(mask), changedRows(idx),
↪   changedCols(idx));
    end
end
```

Automatic DINEOF data file maker.m

```
%% Clear Previous Data
clear all % Clear all variables from the workspace
close all % Close all figure windows
clc % Clear Command Window

%% Read Before DINEOF

% Define the path to the NetCDF file
input_file = '/Users/mrekxa/Desktop/dineof/merged_monthlies/whole_year.nc';

% Read data from the NetCDF file
[datauGlob,datavGlob,timeGlob]=readData(input_file);

% Define the dates and number of days for which data will be extracted
centre_date = {'2022-03-27', '2022-04-26', '2022-05-15','2022-05-23',...
                '2022-06-02','2022-06-15','2022-08-01','2022-10-07',...
                '2022-11-10','2022-12-20','2023-01-05','2023-02-10',...
                '2023-03-14','2023-04-01','2023-05-30'}; % example dates
%centre_date = {'2023-04-01'}; % example dates

num_days = [3,5,7,14,28]; % example number of days
patch_nums=[10,20,30,40,50];
patch_sizes=[1,2,3,4,5,6,7,8,9,10];
% Loop over each date and number of days

for n = 1:length(num_days)
    for c = 1:length(centre_date)
        for k = 1:length(patch_nums)
            for j= 1:length(patch_sizes)
                current_patch_size=patch_sizes(j);
                current_patch_num=patch_nums(k);
                current_centre_date = centre_date{c};
                current_num_days = num_days(n);

                % Query the data for the current date and number of days
 ↪   [Extracted_U,Extracted_V,Extracted_time,date_of_interest_index,dateStrs]=...
 ↪   query_dates(input_file,datauGlob,datavGlob,timeGlob,current_centre_date,...
                current_num_days);

                % Initialize a mask and find the indices where the data is valid
                mask=zeros(52,43);
                idx = find(Extracted_U(:,:,date_of_interest_index) > -9000);
                mask (idx) = 1;

                % Transpose the time data and calculate the total time
                Extracted_time=Extracted_time';
                total_time=numel(Extracted_time);


                % Swap patches in the mask
                [swapped_mask,swap_indices]=...
 ↪   swap_patches_2(mask,current_patch_num,current_patch_size,current_patch_size,100);

                swapped_mask(swap_indices)=0;

                % Swap patches in the U and V data
                Swap_u=Extracted_U(:,:,:);
                Swap_u_slice=Extracted_U(:,:,date_of_interest_index);
                Swap_u_slice(swap_indices)=NaN;
                Swap_u(:,:,date_of_interest_index)=Swap_u_slice;

                Swap_v=Extracted_V(:,:,:);
                Swap_v_slice=Extracted_V(:,:,date_of_interest_index);
                Swap_v_slice(swap_indices)=NaN;
                Swap_v(:,:,date_of_interest_index)=Swap_v_slice;

                % Define the parameter nbmots
                nbmots = 10485760;

                % Define the names of the output files
```

```
                radar_u_text='radar_u';
                radar_v_text='radar_v';
                radardates_text='radardates';
                mask_text='mask';
                data_text='data';

                % Define the format of the output filenames
                base_file='/Users/mrekxa/Desktop/dineof/ForDineof/';
                specific_date='extracted_%s_length_%d_days_%s_patch_%d_%d.%s';

                formatSpec=base_file+specific_date;
                % Generate the output filenames

                filename_U=...
                sprintf(formatSpec,current_centre_date,current_num_days...
                ,radar_u_text,current_patch_num,current_patch_size,data_text);

                filename_V=...
                sprintf(formatSpec,current_centre_date,current_num_days...
                ,radar_v_text,current_patch_num,current_patch_size,data_text);

                filename_dates=...
                sprintf(formatSpec,current_centre_date,current_num_days...

↪   ,radardates_text,current_patch_num,current_patch_size,data_text);

                filename_mask=...
                sprintf(formatSpec,current_centre_date,current_num_days...

↪   ,radardates_text,current_patch_num,current_patch_size,mask_text);

                % Write the data to the output files
                uwrite(filename_U,Swap_u,52,43,total_time,-9999,nbmots);
                uwrite(filename_V,Swap_v,52,43,total_time,-9999,nbmots);

↪   uwrite(filename_dates,[1:1:total_time],total_time,1,1,-9999,nbmots);
                uwrite(filename_mask,mask,52,43,1,-9999,nbmots);

                % Display a message indicating that the files have been written
                disp('Files written')
            end
```

```
        end
    end
end
disp('Finished')
```

Python_DINEOF_init_generator.py

```
from datetime import datetime

def create_files(dates, lengths_k_days, data_types, alphas, norms, eofs,
↪   patch_numbers, patch_sizes):
    base_path = r"Z:\Desktop\dineof"
    data_path = base_path + r"\ForDineof"
    results_path = base_path + r"\From_Dineof"
    output_path = base_path + r"\RadarExample\Output"

    numit, nev, neini, ncv = 3, 5, 1, 10
    tol, nitemax, toliter, rec = 1.0e-8, 300, 1.0e-3, 0
    seed, cloud_size = 243435, 500

    for date in dates:
    for length_k_days in lengths_k_days:
    for alpha in alphas:
    for norm in norms:
    for eof in eofs:
    for patch_number in patch_numbers:
    for patch_size in patch_sizes:
    date_string = date.strftime('%Y-%m-%d')
    base_file_name= f'extracted_{date_string}_length_{length_k_days}_days'
    file_name = f'extracted_{date_string}_length_{length_k_days}_days_alpha_
        {alpha}_norm_{norm}_eof_{eof}_patch_{path_number}_size_{patch_size}'

    for data_type in data_types:
    data = f'{data_path}\\{base_file_name}_radar_{data_type}_patch_
                {patch_number}_{patch_size}.data'
    mask = f'{data_path}\\{base_file_name}_radardates_patch_
                {patch_number}_{patch_size}.mask'
    time = f'{data_path}\\{base_file_name}_radardates_patch_
                {patch_number}_{patch_size}.data'
    results = f'{results_path}\\{file_name}_radar_{data_type}_
```

```
                patch_{patch_number}_{patch_size}_output.data'

    with open(f'output_{file_name}_type_{data_type}.init', 'w') as file:
    file.write(f'data = ["{data}"]\n')
    file.write(f'mask = ["{mask}"]\n')
    file.write(f'time = "{time}"\n\n')
    file.write(f'alpha = {alpha}\n')
    file.write(f'numit = {numit}\n')
    file.write(f'nev = {nev}\n')
    file.write(f'neini = {neini}\n')
    file.write(f'ncv = {ncv}\n')
    file.write(f'tol = {tol}\n')
    file.write(f'nitemax = {nitemax}\n\n')
    file.write(f'toliter = {toliter}\n')
    file.write(f'rec = {rec}\n\n')
    file.write(f'eof = {eof}\n')
    file.write(f'norm = {norm}\n')
    file.write(f"Output = '{output_path}'\n")
    file.write(f'results = ["{results}"]\n\n')
    file.write(f'seed = {seed}\n')
    file.write(f'cloud_size = {cloud_size}\n')


dates = [datetime(2022,3,27),datetime(2022,4,26),datetime(2022,5,15),
         datetime(2022,5,23),datetime(2022,6,2),datetime(2022,6,15),
         datetime(2022,8,10),datetime(2022,10,7),datetime(2022,11,10),
         datetime(2022,12,20),datetime(2023,1,5),datetime(2023,2,10),
         datetime(2023, 3, 14), datetime(2022, 4, 1),datetime(2023,5,30)]
lengths_k_days = [3,5,7,14,28,56,84]
data_types = ["u", "v"]
alphas = [0.005,0.01, 0.02,0.05,0.1]
norms = [0, 1]
eofs = [0,1]
patch_numbers = [10]
patch_sizes = [4]

create_files(dates, lengths_k_days, data_types, alphas, norms, eofs,
↪   patch_numbers, patch_sizes)
```

DINEOF performance calculator.m

```
% Define the directory containing the files
dir_path = '/Users/mrekxa/Desktop/dineof/From_Dineof_2';

% Get a list of all .data files in the directory
files = dir(fullfile(dir_path, '*.data'));

% Initialize a table to store the linear regression results
results_table = table();
% Define the path to the NetCDF file
input_file = '/Users/mrekxa/Desktop/dineof/merged_monthlies/whole_year.nc';

% Read data from the NetCDF file
[datauGlob,datavGlob,timeGlob]=readData(input_file);

% Create a directory for results_images if it doesn't exist
if ~exist(fullfile(dir_path, 'results_images'), 'dir')
    mkdir(fullfile(dir_path, 'results_images'));
end
% Loop over the files
for i = 1:2:length(files)
    % Extract the parameters from the filenames
    filename_u = files(i).name;
    filename_v = files(i+1).name;
    params = sscanf(filename_u,'extracted_%d-%d-%d_length_%d_days_alpha_%f_
        norm_%d_eof_%d_patch_%d_size_%d_radar_u__patch_%d_%d_output.data');
    centre_date = datestr(datenum(params(1), params(2), params(3)),...
        'yyyy-mm-dd');
    days = params(4);
    alpha=params(5);
    norm = params(6);
    eof = params(7);
    num_patches = params(8);
    patch_size = params(9);

    % Query the data for the specified date and number of days
    [Extracted_U,Extracted_V,Extracted_time,date_of_interest_index,dateStrs]=...
        query_dates(input_file,datauGlob,datavGlob,timeGlob,centre_date,days);

    % Read and reshape the U and V data from the DINEOF results
```

```
[flag,outdata_u,imax,jmax,kmax,valex,nbmots] = uread(filename_u);
outdata_u = reshape(outdata_u, imax, jmax, kmax);

[flag,outdata_v,imax,jmax,kmax,valex,nbmots] = uread(filename_v);
outdata_v = reshape(outdata_v, imax, jmax, kmax);

% Initialize a mask and find the indices where the data is valid
mask=zeros(52,43);
idx = find(Extracted_U(:,:,date_of_interest_index) > -9000);
mask (idx) = 1;

% Transpose the time data and calculate the total time
Extracted_time=Extracted_time';
total_time=numel(Extracted_time);

% Define parameters for swapping patches
k=num_patches;
i=patch_size;
j=patch_size;

% Swap patches in the mask
[swapped_mask,swap_indices]=swap_patches_2(mask,k,i,j,100);
swapped_mask(swap_indices)=0;

% Swap patches in the U and V data
Swap_u=Extracted_U(:,:,:);
Swap_u_slice=Extracted_U(:,:,date_of_interest_index);
% Count the number of non-NaN values in the original Swap_u_slice
count_before_swap = sum(~isnan(Swap_u_slice(:)));
Swap_u_slice(swap_indices)=NaN;
% Count the number of non-NaN values in the modified Swap_u_slice
count_after_swap = sum(~isnan(Swap_u_slice(:)));

Swap_u(:,:,date_of_interest_index)=Swap_u_slice;

Swap_v=Extracted_V(:,:,:);
Swap_v_slice=Extracted_V(:,:,date_of_interest_index);
Swap_v_slice(swap_indices)=NaN;
Swap_v(:,:,date_of_interest_index)=Swap_v_slice;

% Extract the slices of interest from the raw and DINEOF result data
```

```
    result_u_dineof_slice=outdata_u(:,:,date_of_interest_index);
    result_v_dineof_slice=outdata_v(:,:,date_of_interest_index);
    raw_u_slice=Extracted_U(:,:,date_of_interest_index);
    raw_v_slice=Extracted_V(:,:,date_of_interest_index);

    % Replace invalid values in the DINEOF result data with NaN
    idx=find(result_u_dineof_slice == 9999);
    result_u_dineof_slice(idx)=NaN;
    result_v_dineof_slice(idx)=NaN;

    % Calculate percentage missing data
    percentage_data_for_dineof=100*((count_after_swap)/count_before_swap);

    x=[raw_u_slice(swap_indices), raw_v_slice(swap_indices)];
    y=[result_u_dineof_slice(swap_indices),...
        result_v_dineof_slice(swap_indices)];
    % Fit a linear regression model to your data
    % Replace 'x' and 'y' with your actual data
    mdl = fitlm(x, y);

    % Calculate the RMSE
    rmse = sqrt(mean(mdl.Residuals.Raw.^2));

    % Get the R-squared value
    rsquared = mdl.Rsquared.Ordinary;

    % In your loop where you store results:
    results_table = [results_table; table(string(centre_date), days, alpha,
↪   norm, eof, num_patches, patch_size, rmse, rsquared,
↪   percentage_data_for_dineof)];
end

% Display the results table
disp(results_table);
% Export the results_table as a CSV file
writetable(results_table, 'results_table_2.csv');
```

Powershell_auto_dineof

```powershell
    # Get list of files in the directory
$files = Get-ChildItem "Z:\Desktop\dineof\for_test" -Filter *.init

# Load the log of processed files
$processed = @{}
if (Test-Path "processed.log") {
    $processed = Get-Content "processed.log" | ConvertFrom-Json
}

# Iterate over each file in the directory
foreach ($file in $files) {
    # Check if the file has been processed
    if ($processed.ContainsKey($file.FullName)) {
        continue
    }

    # Call dineof.exe with the file as an argument
    & "Z:\Desktop\dineof\RadarExample\dineof.exe" $file.Fullname

    # Add the file to the log of processed files
    $processed[$file.FullName] = $true

    # Save the log of processed files
    $processed | ConvertTo-Json | Set-Content "processed.log"
}
```

Python combination finder.py

```python
import pandas as pd

# Load the data
data = pd.read_csv("results_table.csv")


# Function to find the optimal combination for each group (date)
def find_optimal_combination(group):
    return group.sort_values(by=["rmse", "rsquared"], ascending=[True,
    ↪  False]).iloc[0]
```

```
optimal_combinations = data.groupby("Date").apply(find_optimal_combination)

# Display the optimal combinations for each date
print(optimal_combinations[["days", "alpha", "eof", "norm", "rmse",
↪  "rsquared"]])


best_combination = find_optimal_combination(data)


print("Optimal Combination:")
print("-------------------")
print("Days:", best_combination["days"])
print("Alpha:", best_combination["alpha"])
print("EOF:", best_combination["eof"])
print("Norm:", best_combination["norm"])
print("RMSE:", best_combination["rmse"])
print("R^2:", best_combination["rsquared"])
```

Genetic Algorithm DINEOF optimisation.py

```
    # Import necessary libraries
from deap import base, creator, tools, algorithms
import random
import pandas as pd

# Load the optimization dataset
data = pd.read_csv("results_table.csv")

# Set a random seed to ensure reproducibility of results
random.seed(64)

# Define the optimization problem:
# - FitnessMulti: A multi-objective fitness (two objectives: RMSE and R^2
↪  deviation)
# - Individual: Represents a possible solution (combination of parameters)
creator.create("FitnessMulti", base.Fitness, weights=(-1.0, -1.0))
creator.create("Individual", list, fitness=creator.FitnessMulti)

# Create a toolbox to store various genetic algorithm operations
toolbox = base.Toolbox()

# Register functions to randomly select values for each parameter
toolbox.register("days", random.choice, data["days"].unique())
toolbox.register("alpha", random.choice, data["alpha"].unique())
toolbox.register("eof", random.choice, data["eof"].unique())
toolbox.register("norm", random.choice, data["norm"].unique())

# Define how an individual is constructed: A cyclic repetition of parameter
↪  values
toolbox.register(
    "individual",
    tools.initCycle,
    creator.Individual,
    (toolbox.days, toolbox.alpha, toolbox.eof, toolbox.norm),
    n=1,
)

# Define how a population is constructed: A list of individuals
toolbox.register("population", tools.initRepeat, list, toolbox.individual)


# Evaluation function: Computes fitness values (RMSE and R^2 deviation) for an
↪  individual
def evaluate(individual):
    # Filter dataset to match individual's parameter values
    subset = data[
        (data["days"] == individual[0])
        & (data["alpha"] == individual[1])
        & (data["eof"] == individual[2])
        & (data["norm"] == individual[3])
    ]

    # Compute the two objective values
    rmse_mean = subset["rmse"].mean()
    rsquared_diff_from_one = 1 - subset["rsquared"].mean()

    return rmse_mean, rsquared_diff_from_one
```

```
# Register genetic operators:
# - mate: Two-point crossover
# - mutate: Bit flip mutation with 5% probability
# - select: Tournament selection of size 3
# - evaluate: The previously defined evaluation function
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("evaluate", evaluate)

# Hall of Fame: Stores the best individual found across all generations
hof = tools.HallOfFame(1)

# Run the genetic algorithm:
# - population of 200
# - mu: Number of individuals to select for the next generation
# - lambda_: Number of children to produce
# - cxpb: Probability of mating two individuals
# - mutpb: Probability of mutating an individual
# - ngen: Number of generations
# - stats: Statistics to compute (None means no statistics)
# - halloffame: Our Hall of Fame
# - verbose: Display logs
population = toolbox.population(n=200)
algorithms.eaMuPlusLambda(
    population,
    toolbox,
    mu=200,
    lambda_=400,
    cxpb=0.7,
    mutpb=0.2,
    ngen=100,
    stats=None,
    halloffame=hof,
    verbose=True,
)

# Extract and print the best solution found
best_solution = hof[0]
```

```
print("\nOptimal Parameters:")
print("-------------------")
print("Optimal days:", best_solution[0])
print("Optimal alpha:", best_solution[1])
print("Optimal eof:", best_solution[2])
print("Optimal norm:", best_solution[3])
print("RMSE:", best_solution.fitness.values[0])
print("R^2:", 1 - best_solution.fitness.values[1])
```

CSV file analysis.py

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Reload the dataset
data = pd.read_csv("results_table_2.csv")

# Saving the distributions plot
fig1, axs = plt.subplots(nrows=3, figsize=(10, 18))

sns.histplot(
    data["percentage_data_for_dineof"],
    kde=True,
    ax=axs[0],
)
axs[0].set_title("Distribution of data available for DINEOF")
axs[0].set_xlabel("Percentage Data for DINEOF")
axs[0].set_ylabel("Frequency")
axs[0].set_xlim([-10, 110])

sns.histplot(data["rmse"], kde=True, ax=axs[1])
axs[1].set_title("Distribution of RMSE")
axs[1].set_xlabel("RMSE")
axs[1].set_ylabel("Frequency")

sns.histplot(data["rsquared"], kde=True, ax=axs[2])
axs[2].set_title("Distribution of Rsquared")
axs[2].set_xlabel("Rsquared")
```

```
axs[2].set_ylabel("Frequency")

plt.tight_layout()
pdf_path1 = "/Users/mrekxa/Desktop/dineof/results_images/distributions_plot.pdf"
fig1.savefig(pdf_path1)

# Saving the relationship scatter plots
fig2, axs = plt.subplots(nrows=2, figsize=(10, 14))

sns.scatterplot(
    x="percentage_data_for_dineof", y="rmse", data=data, ax=axs[0], alpha=0.5
)
axs[0].set_title("Relationship between percetange data for DINEOF and RMSE")
axs[0].set_xlabel("Percentage Data for DINEOF")
axs[0].set_ylabel("RMSE")

sns.scatterplot(
    x="percentage_data_for_dineof", y="rsquared", data=data, ax=axs[1],
    ↪  alpha=0.5
)
axs[1].set_title("Relationship between percetange data for DINEOF and Rsquared")
axs[1].set_xlabel("Percentage Data for DINEOF")
axs[1].set_ylabel("Rsquared")

plt.tight_layout()
pdf_path2 = "/Users/mrekxa/Desktop/dineof/results_images/relationship_plot.pdf"
fig2.savefig(pdf_path2)

pdf_path1, pdf_path2
```

Final Jupyter notebook

```
#!/usr/bin/env python
# coding: utf-8

# # Import all required packaged

# In[1]:
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.io import loadmat
import datetime
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from cartopy.mpl.gridliner import LONGITUDE_FORMATTER, LATITUDE_FORMATTER
import netCDF4 as nc
from datetime import datetime, timedelta
from netCDF4 import Dataset
import datetime
import cartopy.io.img_tiles as cimgt


# # Load all necessary files

# In[2]:


# Import all the data exported from Matlab
mat_data_1 = loadmat("/Users/mrekxa/Desktop/dineof/trimmed_3_day_u_4.mat")
mat_data_2 = loadmat("/Users/mrekxa/Desktop/dineof/trimmed_3_day_v_4.mat")
mat_data_3 = loadmat("/Users/mrekxa/Desktop/dineof/trimmed_9_day_u_4.mat")
mat_data_4 = loadmat("/Users/mrekxa/Desktop/dineof/trimmed_9_day_v_4.mat")
# Import the raw data from the netcdf file
filename = "/Users/mrekxa/Desktop/dineof/merged_monthlies/whole_year.nc"
raw_file = nc.Dataset(filename)
# Variable keys for the netcdf file
print(raw_file.variables.keys())


# In[3]:


mat_file_struct_names = [
    "concatenated_u_struct",
    "concatenated_v_struct",
    "trimmed_u_9_struct",
    "trimmed_v_9_struct",
]
```

```python
dates = [
    "2022_03_27",
    "2022_04_26",
    "2022_05_15",
    "2022_05_23",
    "2022_06_02",
    "2022_06_15",
    "2022_08_01",
    "2022_10_07",
    "2022_11_10",
    "2022_12_20",
    "2023_01_05",
    "2023_02_10",
    "2023_03_14",
    "2023_04_01",
    "2023_05_30",
]

dates2 = [
    "2022-03-27",
    "2022-04-26",
    "2022-05-15",
    "2022-05-23",
    "2022-06-02",
    "2022-06-15",
    "2022-08-01",
    "2022-10-07",
    "2022-11-10",
    "2022-12-20",
    "2023-01-05",
    "2023-02-10",
    "2023-03-14",
    "2023-04-01",
    "2023-05-30",
]


# In[4]:
```

```
# Create dictionary of hourly dates time strings for each of the days
def create_date_dictionary(dates, window_size):
    date_dict = {}
    for i in range(np.size(dates)):
        center_date = datetime.datetime.strptime(dates[i], "%Y_%m_%d")

        start_datetime = center_date - datetime.timedelta(days=window_size)
        end_datetime = center_date + datetime.timedelta(
            days=window_size, hours=23
        )  # Include the last hour of the last day

        # Generating hourly timestamps
        hours_diff = int((end_datetime - start_datetime).total_seconds() // 3600
         ↪  + 1)
        surrounding_dates = [
            start_datetime + datetime.timedelta(hours=i) for i in
             ↪  range(hours_diff)
        ]

        # Format the surrounding dates as string and store in dictionary
        date_dict[i] = [date.strftime("%Y-%m-%d %H") for date in
         ↪  surrounding_dates]

    return date_dict


window_size = 3  # To include the day before and the day after

date_dict = create_date_dictionary(dates, window_size)

# If you want to see the output
# for center_date, surrounding_dates in date_dict.items():
#     print(f"{center_date}: {surrounding_dates}")



# In[5]:



# Load all the data from matlab into their respective directories
U_3_day = {}
V_3_day = {}
```

```
U_9_day = {}
V_9_day = {}

for i in range(np.size(dates)):
    U_3_day[i] =
    ↪   np.array(mat_data_1[mat_file_struct_names[0]][f"d{dates[i]}"][0, 0])
    V_3_day[i] =
    ↪   np.array(mat_data_2[mat_file_struct_names[1]][f"d{dates[i]}"][0, 0])
    U_9_day[i] =
    ↪   np.array(mat_data_3[mat_file_struct_names[2]][f"d{dates[i]}"][0, 0])
    V_9_day[i] =
    ↪   np.array(mat_data_4[mat_file_struct_names[3]][f"d{dates[i]}"][0, 0])

# Calculate HF radar total vectors for each of the days
Total_3_day = {}
Total_9_day = {}

for i in range(np.size(dates)):
    Total_3_day[i] = np.sqrt((U_3_day[i] ** 2) + (V_3_day[i] ** 2))
    Total_9_day[i] = np.sqrt((U_9_day[i] ** 2) + (V_9_day[i] ** 2))

# Calculate the differnces in U, V and the calculated total vectors
Difference_U = {}
Difference_V = {}
Difference_Totals = {}

for i in range(np.size(dates)):
    Difference_U[i] = U_3_day[i] - U_9_day[i]
    Difference_V[i] = V_3_day[i] - V_9_day[i]
    Difference_Totals[i] = Total_3_day[i] - Total_9_day[i]


# Load the Longitude and Latitude of the data from the netcdf file
Lon = raw_file.variables["lon"]
Lat = raw_file.variables["lat"]
# Load the usefull data from the netcdf file
global_u = raw_file.variables["u"]
global_v = raw_file.variables["v"]
global_time = raw_file.variables["time"]
time_array = global_time[:]
np.shape(time_array)
```

```
# In[6]:


from netCDF4 import Dataset
from datetime import datetime, timedelta
import numpy as np


def query_dates_2(
    filename, datauGlob, datavGlob, timeGlob, centre_date, centre_hour, num_days
):
    """
    The function takes a filename of a NetCDF file, global data arrays, a centre
↪   date, centre hour,
    and number of days and extracts data for the specified number of days around
↪   the centre date.

    Args:
    filename (str): The name of the NetCDF file.
    datauGlob (np.ndarray): The global data array for U component.
    datavGlob (np.ndarray): The global data array for V component.
    timeGlob (np.ndarray): The global time array.
    centre_date (str): The centre date in 'yyyy-mm-dd' format.
    centre_hour (int): The centre hour.
    num_days (int): The number of days to extract data around the centre date.

    Returns:
    tuple: Extracted_U, Extracted_V, Extracted_time, date_of_interest_index,
↪   date_str_array
    """
    # Read the 'base_date' attribute from the time variable
    with Dataset(filename, "r") as nc:
        base_date = nc.variables["time"].getncattr("units")

    # Extract the date and time parts from the attribute
    date_part, time_part = " ".join(base_date.split(" ")[-2:]).split(" ")

    # Combine the date and hour parts to form a datetime string and convert it
    ↪   to a datetime object
```

```
base_datetime = datetime.strptime(f"{date_part} {time_part[:2]}", "%Y-%m-%d
↪  %H")

# Combine centre_date and centre_hour to form a complete datetime string and
↪  convert it to a datetime object
complete_centre_date = datetime.strptime(
    f"{centre_date} {centre_hour:02d}", "%Y-%m-%d %H"
)

# Convert date of interest into total hours since the start of the dataset
delta = complete_centre_date - base_datetime
hours_since_basedate = delta.total_seconds() // 3600  # 1 hour = 3600
↪  seconds

# Calculate the start and end indices of the slices to be extracted
hours_since_basedate_start = hours_since_basedate - 24 * (num_days // 2)
hours_since_basedate_end = hours_since_basedate + 24 * (num_days // 2)

# Find the indices which match the calculated hours
date_index_start = np.argmin(np.abs(timeGlob - hours_since_basedate_start))
date_index_end = np.argmin(np.abs(timeGlob - hours_since_basedate_end))

# Create an array of dates which are extracted and converted to date strings
date_str_array = [
    (base_datetime + timedelta(hours=int(hour))).strftime("%Y-%m-%d %H")
    for hour in timeGlob[date_index_start : date_index_end + 1]
]

# Extract data
Extracted_U = datauGlob[date_index_start : date_index_end + 1, :, :]
Extracted_V = datavGlob[date_index_start : date_index_end + 1, :, :]
Extracted_time = timeGlob[date_index_start : date_index_end + 1]

# Find the index of the center date
centre_index = np.where(timeGlob == hours_since_basedate)[0][0]
date_of_interest_index = np.where(
    np.arange(date_index_start, date_index_end + 1) == centre_index
)[0][0]

return (
    Extracted_U,
        Extracted_V,
        Extracted_time,
        date_of_interest_index,
        date_str_array,
    )


# In[7]:


Extracted_U = {}
Extracted_V = {}
Extracted_time = {}
idx = {}
date_str_array = {}

for i in range(np.size(dates)):

    Extracted_U[i],Extracted_V[i],Extracted_time[i],idx[i],date_str_array[i]  =
     ↪  query_dates_2( filename, global_u, global_v, time_array, dates2[i], 0,7)
# to show that function works
# date_str_array, date_of_interest_index


Extracted_totals={}
for i in range(np.size(dates)):
    Extracted_totals[i]=np.sqrt(Extracted_U[i] ** 2 + Extracted_V[i] ** 2)

Extracted_totals[0]


# In[8]:


import matplotlib.ticker as mticker
import cartopy.feature as cf  # Make sure to import cf

request = cimgt.OSM()
extent = []
LAND = cfeature.NaturalEarthFeature(
    "physical", "land", "10m", edgecolor="face", facecolor="tan"
```

```
)

# Set colors of the land.
edgecolor = "black"
landcolor = "tan"

state_lines = cfeature.NaturalEarthFeature(
    category="cultural",
    name="admin_1_states_provinces_lines",
    scale="50m",
    facecolor="none",
)


# Create a re-usable function for map features that we can pass an axes to.
def map_features(ax):
    # Axes properties and features
    ax.set_extent(extent)
#     ax.add_feature(LAND, edgecolor=edgecolor, facecolor=landcolor)
    ax.add_feature(cfeature.OCEAN)
    ax.add_feature(cfeature.RIVERS)
    ax.add_feature(cfeature.LAKES)
    ax.add_feature(cfeature.BORDERS)
    ax.add_feature(state_lines, zorder=11, edgecolor=edgecolor)

    coast = cf.GSHHSFeature(scale='h')  # Add this line to add the
    ↪  high-resolution coastline
    ax.add_feature(coast)  # Add this line to add the coastline feature to the
    ↪  axes


    # Gridlines and grid labels
    gl = ax.gridlines(
        draw_labels=True, linewidth=0.5, color="black", alpha=0.25,
        ↪  linestyle="--"
    )

    gl.top_labels = gl.right_labels =False
    gl.xlabel_style = {"size": 10, "color": "black"}
    gl.ylabel_style = {"size": 10, "color": "black"}
    gl.xlocator = mticker.MaxNLocator(integer=True)
    gl.ylocator = mticker.MaxNLocator(integer=True)
    gl.xformatter = LONGITUDE_FORMATTER
    gl.yformatter = LATITUDE_FORMATTER

    ax.tick_params(
        which="major",
        direction="out",
        bottom=True,
        top=False,
        labelbottom=True,
        labeltop=False,
        left=True,
        right=False,
        labelleft=True,
        labelright=False,
        length=5,
        width=2,
    )

    ax.tick_params(
        which="minor",
        direction="out",
        bottom=True,
        top=True,
        labelbottom=True,
        labeltop=False,
        left=True,
        right=False,
        labelleft=True,
        labelright=False,
        width=2,
    )


# In[29]:


i = 14
slice_1 = idx[i]
# slice_1 = 50
```

```
fig, ax = plt.subplots(figsize=(12, 8),dpi=800,
↪  subplot_kw=dict(projection=request.crs))
# ax = plt.axes(projection=request.crs)
ax.add_image(request, 10)  # If you want to use higher zoom level for OSM tiles.
# date_str_num=datetime (date_dict[i][slice_1],format = "%yyyy_%mm_%dd_%hh")
# print(date_str_num)
fig.suptitle(f"Data for {date_dict[i][slice_1]}", fontsize=16)
plt.quiver(
    Lon,
    Lat,
    U_9_day[i][:, :, slice_1],
    V_9_day[i][:, :, slice_1],
    scale=1, scale_units='inches',
    color="r",
    label="9 - day dataset",
    transform=ccrs.PlateCarree(),
)
plt.quiver(
    Lon,
    Lat,
    U_3_day[i][:, :, slice_1],
    V_3_day[i][:, :, slice_1],
    scale=1, scale_units='inches',
    color="b",

    label="3 - day dataset",
    transform=ccrs.PlateCarree(),
)

plt.quiver(
    Lon,
    Lat,
    Extracted_U[i][slice_1],
    Extracted_V[i][slice_1],
    scale=1, scale_units='inches',
    label="raw_data",
    transform=ccrs.PlateCarree(),
)

extent = [np.min(Lon), np.max(Lon), np.min(Lat), np.max(Lat)]
```

```
map_features(ax)
ax.legend(loc='lower right')

filename3 = f"quiver_plot_for_date_{i}.jpg"
plt.savefig(filename3, dpi=600 ,format='jpg')


# # Create subplots showing the difference DINEOF makes when gap filling from
↪  the raw data

# In[10]:



slice_1 = 72
for i in range(np.size(dates)):
    # Create the figure and subplots
    fig, axs = plt.subplots(4, 3, figsize=(12, 12),dpi=500)

    # Add title to the entire figure
    fig.suptitle(f"Data for {date_dict[i][slice_1]}", fontsize=16)

    # Plot the raw data in a pcolor plot
    c10 = axs[0, 0].pcolor(Extracted_U[i][idx[i]], cmap="viridis",
    ↪  shading="auto")
    fig.colorbar(c10, ax=axs[0, 0])
    axs[0, 0].set_title("Raw radar data, U component",fontsize=8)

    c11 = axs[0, 1].pcolor(Extracted_V[i][idx[i]], cmap="viridis",
    ↪  shading="auto")
    fig.colorbar(c11, ax=axs[0, 1])
    axs[0, 1].set_title("Raw radar data, V component",fontsize=8)

    totals = np.sqrt(Extracted_U[i] ** 2 + Extracted_V[i] ** 2)
    c12 = axs[0, 2].pcolor(totals[idx[i]], cmap="viridis", shading="auto")
    fig.colorbar(c12, ax=axs[0, 2])
    axs[0, 2].set_title("Raw radar data, Reconstructed Totals",fontsize=8)

    # Plot the first pcolor plot
    c1 = axs[1, 0].pcolor(U_3_day[i][:, :, slice_1], cmap="viridis",
    ↪  shading="auto")
    fig.colorbar(c1, ax=axs[1, 0])
```

```
axs[1, 0].set_title("U component 3-day",fontsize=8)

c2 = axs[1, 1].pcolor(V_3_day[i][:, :, slice_1], cmap="viridis",
↪  shading="auto")
fig.colorbar(c2, ax=axs[1, 1])
axs[1, 1].set_title("V component 3-day",fontsize=8)

c3 = axs[1, 2].pcolor(Total_3_day[i][:, :, slice_1], cmap="viridis",
↪  shading="auto")
fig.colorbar(c3, ax=axs[1, 2])
axs[1, 2].set_title("Reconstructed Total 3-day",fontsize=8)

# Plot the second row of pcolor plots
c4 = axs[2, 0].pcolor(U_9_day[i][:, :, slice_1], cmap="viridis",
↪  shading="auto")
fig.colorbar(c4, ax=axs[2, 0])
axs[2, 0].set_title("U component 9-day",fontsize=8)

c5 = axs[2, 1].pcolor(V_9_day[i][:, :, slice_1], cmap="viridis",
↪  shading="auto")
fig.colorbar(c5, ax=axs[2, 1])
axs[2, 1].set_title("V component 9-day",fontsize=8)

c6 = axs[2, 2].pcolor(Total_9_day[i][:, :, slice_1], cmap="viridis",
↪  shading="auto")
fig.colorbar(c6, ax=axs[2, 2])
axs[2, 2].set_title("Reconstructed Total 9-day",fontsize=8)

# Plot the third row of pcolor plots
max_abs_value = np.nanmax(np.abs(Difference_U[i][:, :, slice_1]))
c7 = axs[3, 0].pcolor(
    Difference_U[i][:, :, slice_1],
    cmap="Spectral",
    shading="auto",
    vmin=-max_abs_value,
    vmax=max_abs_value,
)
fig.colorbar(c7, ax=axs[3, 0])
axs[3, 0].set_title("Difference in U",fontsize=8)
max_abs_value = np.nanmax(np.abs(Difference_V[i][:, :, slice_1]))
c8 = axs[3, 1].pcolor(
            Difference_V[i][:, :, slice_1],
            cmap="Spectral",
            shading="auto",
            vmin=-max_abs_value,
            vmax=max_abs_value,
        )
        fig.colorbar(c8, ax=axs[3, 1])
        axs[3, 1].set_title("Difference in V",fontsize=8)
        max_abs_value = np.nanmax(np.abs(Difference_Totals[i][:, :, slice_1]))
        c9 = axs[3, 2].pcolor(
            Difference_Totals[i][:, :, slice_1],
            cmap="Spectral",
            shading="auto",
            vmin=-max_abs_value,
            vmax=max_abs_value,
        )
        fig.colorbar(c9, ax=axs[3, 2])
        axs[3, 2].set_title("Difference in Totals",fontsize=8)

        # Save the plot as a PDF
        filename2 = f"plot_for_date_{i}.jpg"
        plt.savefig(filename2, dpi=500 ,format='jpg',transparent=False)
        plt.show()
        # Close the figure to free up memory
        plt.close(fig)


# # Calculate weekly statistics per dataset and plot them within a subplot

# In[11]:


# Define a function to calculate statistics on non-NaN data
def calculate_non_nan_statistics(data):
    non_nan_data = data[np.isfinite(data)]
    mean_non_nan = np.mean(non_nan_data)
    variance_non_nan = np.var(non_nan_data)
    min_non_nan = np.min(non_nan_data)
    max_non_nan = np.max(non_nan_data)
    return mean_non_nan, variance_non_nan, min_non_nan, max_non_nan
```

```
# Initialize dictionaries to store the statistics for each day
statistics_3_day = {"mean": [], "variance": [], "min": [], "max": []}

statistics_9_day = {"mean": [], "variance": [], "min": [], "max": []}

statistics_raw_data = {"mean": [], "variance": [], "min": [], "max": []}

# Loop through each date and calculate the statistics for Total_3_day and
↪  Total_9_day data
for i in range(np.size(dates)):
    mean, variance, min_, max_ = calculate_non_nan_statistics(Total_3_day[i])
    statistics_3_day["mean"].append(mean)
    statistics_3_day["variance"].append(variance)
    statistics_3_day["min"].append(min_)
    statistics_3_day["max"].append(max_)

    mean, variance, min_, max_ = calculate_non_nan_statistics(Total_9_day[i])
    statistics_9_day["mean"].append(mean)
    statistics_9_day["variance"].append(variance)
    statistics_9_day["min"].append(min_)
    statistics_9_day["max"].append(max_)


    mean, variance, min_, max_ =
    ↪  calculate_non_nan_statistics(Extracted_totals[i])
    statistics_raw_data["mean"].append(mean)
    statistics_raw_data["variance"].append(variance)
    statistics_raw_data["min"].append(min_)
    statistics_raw_data["max"].append(max_)

statistics_3_day, statistics_9_day , statistics_raw_data


# In[12]:


# Function to plot refined statistics of Total_3_day and Total_9_day on the same
↪  figure
def plot_combined_statistics_over_time(dates, stats_3_day,
↪  stats_9_day,stats_raw):
    plt.figure(figsize=(15, 10),dpi=400)

    labels = ["Total_3_day", "Total_9_day","Raw data"]

    plt.subplot(2, 1, 1)
    plt.plot(dates, stats_3_day["mean"], marker="o", label=labels[0])
    plt.plot(dates, stats_9_day["mean"], marker="x", label=labels[1])
    plt.plot(dates, stats_raw ["mean"], marker="d", label=labels[2])
    plt.title("Mean Over Time")
    plt.xlabel("Date")
    plt.ylabel("Mean")
    plt.xticks(rotation=45)
    plt.tick_params(axis="x", labelsize=8)
    plt.legend(loc="upper left")

    plt.subplot(2, 1, 2)
    plt.plot(dates, stats_3_day["variance"], marker="o", label=labels[0])
    plt.plot(dates, stats_9_day["variance"], marker="x", label=labels[1])
    plt.plot(dates, stats_raw ["mean"], marker="d", label=labels[2])
    plt.title("Variance Over Time")
    plt.xlabel("Date")
    plt.ylabel("Variance")
    plt.xticks(rotation=45)
    plt.tick_params(axis="x", labelsize=8)
    plt.legend(loc="upper left")

#     plt.subplot(2, 2, 3)
#     plt.plot(dates, stats_3_day["min"], marker="o", label=labels[0])
#     plt.plot(dates, stats_9_day["min"], marker="x", label=labels[1])
#     plt.plot(dates, stats_raw ["mean"], marker="d", label=labels[2])
#     plt.title("Minimum Over Time")
#     plt.xlabel("Date")
#     plt.ylabel("Minimum")
#     plt.xticks(rotation=45)
#     plt.tick_params(axis="x", labelsize=8)
#     plt.legend(loc="upper left")

#     plt.subplot(2, 2, 4)
#     plt.plot(dates, stats_3_day["max"], marker="o", label=labels[0])
#     plt.plot(dates, stats_9_day["max"], marker="x", label=labels[1])
#     plt.plot(dates, stats_raw ["mean"], marker="d", label=labels[2])
#     plt.title("Maximum Over Time")
#     plt.xlabel("Date")
#     plt.ylabel("Maximum")
#     plt.xticks(rotation=45)
#     plt.tick_params(axis="x", labelsize=8)
#     plt.legend(loc="upper left")

    plt.suptitle(
        "Combined Refined Statistics for 3 day totals, 9 day totals, and raw
        ↪  data totals"
    )
    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()


# Plotting the combined refined statistics for Total_3_day and Total_9_day
↪  datasets
plot_combined_statistics_over_time(dates, statistics_3_day,
↪  statistics_9_day,statistics_raw_data)


# In[13]:


# Function to plot refined statistics of Total_3_day and Total_9_day on the same
↪  figure
def plot_combined_statistics_over_time(dates, stats_3_day, stats_9_day):
    plt.figure(figsize=(15, 10))

    labels = ["Total_3_day", "Total_9_day"]

    plt.subplot(2, 2, 1)
    plt.plot(dates, stats_3_day["mean"], marker="o", label=labels[0])
    plt.plot(dates, stats_9_day["mean"], marker="x", label=labels[1])
    plt.title("Mean Over Time")
    plt.xlabel("Date")
    plt.ylabel("Mean")
    plt.xticks(rotation=45)
    plt.tick_params(axis="x", labelsize=8)
    plt.legend(loc="upper left")
    plt.subplot(2, 2, 2)
    plt.plot(dates, stats_3_day["variance"], marker="o", label=labels[0])
    plt.plot(dates, stats_9_day["variance"], marker="x", label=labels[1])
    plt.title("Variance Over Time")
    plt.xlabel("Date")
    plt.ylabel("Variance")
    plt.xticks(rotation=45)
    plt.tick_params(axis="x", labelsize=8)
    plt.legend(loc="upper left")

    plt.subplot(2, 2, 3)
    plt.plot(dates, stats_3_day["min"], marker="o", label=labels[0])
    plt.plot(dates, stats_9_day["min"], marker="x", label=labels[1])
    plt.title("Minimum Over Time")
    plt.xlabel("Date")
    plt.ylabel("Minimum")
    plt.xticks(rotation=45)
    plt.tick_params(axis="x", labelsize=8)
    plt.legend(loc="upper left")

    plt.subplot(2, 2, 4)
    plt.plot(dates, stats_3_day["max"], marker="o", label=labels[0])
    plt.plot(dates, stats_9_day["max"], marker="x", label=labels[1])
    plt.title("Maximum Over Time")
    plt.xlabel("Date")
    plt.ylabel("Maximum")
    plt.xticks(rotation=45)
    plt.tick_params(axis="x", labelsize=8)
    plt.legend(loc="upper left")

    plt.suptitle(
        "Combined Refined Statistics for 3 day totals and 9 day totals Datasets"
    )
    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()


# Plotting the combined refined statistics for Total_3_day and Total_9_day
↪   datasets
plot_combined_statistics_over_time(dates, statistics_3_day, statistics_9_day)
```

```
# In[ ]:
```