WEATHER FORECASTING USING DMD

MICHAEL CAMPIGLIA AND ANTON PALJUSEVIC

ABSTRACT. The aim of this paper is to use data reduction tools, more specifically Dynamic Mode Decomposition (DMD), to compress and analyze weather data with the intent of accurately forecasting future weather states in a new and unique way.

INTRODUCTION

Predicting or forecasting weather is done using various methods, most rely on either finding historical patterns similar to present day conditions or using numerical analysis with established weather equations. Historical pattern recognition may not provide accurate results and using the more precise numerical analysis method requires complex equations and large computers for calculations. A possible alternative method would be using Dynamic Mode Decomposition (DMD) as a forecasting tool to predict future days based on daily historical recorded data. The goal is to take daily historic weather recordings for a given location from NOAA.gov and run the time series data through DMD to produce a function of time that outputs weather conditions.

While Dynamic Mode Decomposition is a recently discovered mathematical tool originally created for the analysis of flow data in fluid mechanics, it has already shown extreme potential and has resulted in many useful applications in finances [2], video processing [3], etc. One useful reference for our study is a paper that focused on using a dynamic mode decomposition for short term electric load forecasting [1]. Key results of this paper showed that the DMD model was data-driven, meaning that relevant data can be extracted without knowing the physical model of the system, and it was adaptive to seasonal and cyclic patterns in the data. These key features translate over well and their successful results almost provide a proof of concept towards our study, which focuses on short term forecasting of weather data that is also seasonal and data-driven. There have also been many improvements towards the original model of Dynamic Mode Decomposition, such as one of the first analyses of DMD by Rowley et al.[4], which established the connection between DMD and the Koopman operator, helping to explain the output of DMD when applied to nonlinear systems, or a recent study exploring how centering data can improve DMD results [5]. We may look into implementing some of these enhancements to our model to optimize our results.

METHODS

Dynamic Mode Decomposition.

Dynamic Mode Decomposition, referred to as DMD, is a relatively new dimensionality reduction method that captures the spatio-temporal dynamics of a dynamical system and computes a set of modes for a time series of data, allowing us to predict how the system will change in time using the temporal evolution of the data. Considering our weather data set, the information for each day can be stored in a matrix $\vec{x}$. For a long period of time, we have the matrices

$$\mathbf{X} = [\vec{x}_{t_1} \quad \vec{x}_{t_2} \quad \vec{x}_{t_3} \quad \dots \quad \vec{x}_{t_{m-1}}]$$

$$\overline{\mathbf{X}} = [\vec{x}_{t_2} \quad \vec{x}_{t_3} \quad \vec{x}_{t_4} \quad \dots \quad \vec{x}_{t_m}]$$

Our goal is to find the optimal matrix $\mathbf{A}$ such that

$$\mathbf{A} \cdot \vec{x}_{k-1} = \vec{x}_k \text{ or } \overline{\mathbf{X}} = \mathbf{AX}$$

Finding A will actually lead to a formula for X(t) at any time t. To perform DMD, the steps below can be followed.

1. Perform the rank truncated Singular Value Decomposition (SVD) on $\mathbf{X}$

$$\mathbf{X} = U_r \Sigma_r V_r^* \text{ where } r \text{ is the rank of } \mathbf{X}$$

The Full SVD of a matrix decomposes it into three seperate matrices U, Σ, and V^* such that the columns of U are the left singular vectors, Σ is a diagonal of singular values, and the rows of V^* are the right singular vectors of the matrix. A visual example of SVD on a Matrix $\mathbf{M}$ is shown in the figure below.

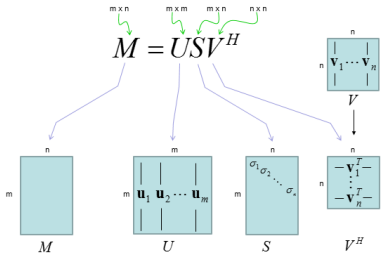

FIGURE 1. Visual Example of Singular Value Decomposition

In this case, however, we are computing the rank truncated SVD, where we take the r (rank) largest singular values as well as their corresponding left and right singular vectors, resulting in U to become U_r with size m x r, Σ to become Σ_r with size r x r, and V^* to become V_r^* with size r x n.

2. Using the property $\mathbf{A} = \overline{\mathbf{X}} \cdot \mathbf{X}^\dagger$ where $\mathbf{X}^\dagger$ is the Pseudo-inverse (generalization of inverse matrix) of $\mathbf{X}$, compute

$$A_r = U_r^* \overline{\mathbf{X}} V_r \Sigma_r^{-1}$$

3. Find the eigenvectors ($\vec{w}$) and eigenvalues (λ) of $\mathbf{A_r}$ and let

$$\mathbf{W} = [\vec{w}_1 \quad \vec{w}_2 \quad \dots \quad \vec{w}_r]$$

$$\Lambda = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_r \end{bmatrix}$$

4. Compute $\Phi = \overline{\mathbf{X}} V_r \Sigma_r^{-1} \mathbf{W}$ where the columns of Φ are called the DMD modes. There are two ways to use this result, either in discrete or continous form. We will be using the discrete form which follows,

$$X_k = \sum_{j=1}^{r} \phi_j \, \lambda_j^{k-1} \, b_j$$

We can solve for b_j by using

$$X(0) = \Phi B \rightarrow B = \Phi^{-1} \cdot X(0)$$

where

$$B = [b_1 \quad b_2 \quad \dots \quad b_r] \text{ and } \Phi = [\phi_1 \quad \phi_2 \quad \dots \quad \phi_r]$$

Implementation of Mathematical Tools.

Prior to using DMD, the dataset will need to be standardized and centered because this optimizes the results returned methods such as PCA and DMD. Once each measurement is standardized, the data will be transposed to properly format the data for DMD. In DMD the data must be presented in a way that each column vector is an instance and each row in the column is data at that instance. Since the data after standardization will still contain each daily recording stored by row, transposing this will arrange the data for DMD correctly. After programming a python function for the DMD method described above, the transposed matrix could be plugged in to return a matrix function of time. This function will then be tested to ensure that the output for a given day is similar to the original data. Once this is proven to replicate the input data, the forecasting model will be used to predict days outside of the input data range. The forecast will be tested to see how far the predictions can go before the information is dramatically different from what occurred on a given day.

Dataset Description.

All historical weather data was obtained from the National Oceanic and Atmospheric Administration website (NOAA.gov), a federal agency responsible for tracking, monitoring, and researching Earth systems. The dataset this project focused on was daily historical weather station readings for the Westchester County Airport, but other station locations should be equally viable sources. Raw data was obtained from available '.txt' files and imported into MS Excel for cleaning and preparation, a sample screenshot of the imported data is shown below. Types of data recorded include temperature readings, wind speeds, pressure readings, precipitation amounts, and observed weather conditions.

The process of cleaning the data started with removing all 'flags' appended to the end of recordings for certain fields. These flags exist to inform how the data was obtained; for example, the asterisk at the end of max temperature indicates that the temperature was an interpolation of data rather than an exact reading. The next step was to fill in any missing information for

Date	Mean Temp	Mean Temp Count	Mean Dew Point	Mean Dew Point Count	Mean Sea Lvl Pressure	Mean Sea Lvl Pressure Count	Mean Station Pressure	Mean Station Pressure Count	Mean Visibility	Mean Visibility Count	Mean Wind Speed	Mean Wind Speed Count	Max Wind Speed	Max Wind Gust	Max Temp	Min Temp	Total Precipitation	Snow Depth	FRSHTT
10/1/2019	66.3	24	58.2	24	1018.8	23	1004.7	24	9.9	24	4	24	8.9	999.9	79.0	54.0	0.01G	999.9	10000
9/30/2019	60.6	24	45.9	24	1025.8	22	1011.5	24	10	24	5.1	24	8.9	999.9	77.0	54.0	0.00G	999.9	0
9/29/2019	70.2	24	58.5	24	1021.5	24	1007.5	24	10	24	6.6	24	11.1	19	79.0	55.9	0.00G	999.9	0
9/28/2019	66.8	24	59.2	24	1019	24	1004.8	24	10	24	4.3	24	11.1	17.1	79.0	48.9	0.00G	999.9	0
9/27/2019	61.7	24	50.6	24	1017.8	21	1003.3	24	9.6	24	3.2	24	11.1	999.9	79.0	48.9	0.18G	999.9	100000
9/26/2019	64.4	24	58.3	24	1009.6	22	995.4	24	9.6	24	4.9	24	18.1	25.1	79.0	51.1	0.00G	999.9	10000
9/25/2019	63.1	24	49.4	24	1011.3	24	997.1	24	10	24	4	24	8.9	999.9	77.0	51.1	0.00G	999.9	0
9/24/2019	70	24	55.9	24	1007.7	24	993.9	24	9.8	24	9.3	24	15.9	22	88.0	64.0	0.14G	999.9	10000
9/23/2019	75.8	24	65	24	1011.7	24	998	24	10	24	4	24	8.9	999.9	88.0	55.9	0.00G	999.9	10000
9/22/2019	69	24	58.7	24	1020.1	24	1006	24	9.2	24	4.7	24	11.1	17.1	84.9	54.0	0.00G	999.9	0
9/21/2019	67.2	24	52.8	24	1022	24	1007.8	24	10	24	4.1	24	7	999.9	80.1	48.9	0.00G	999.9	0
9/20/2019	61	24	44.7	24	1023.5	24	1009	24	10	24	5.4	24	12	18.1	78.1	43.0	0.00G	999.9	0
9/19/2019	55.9	24	40.7	24	1026.8	24	1012.1	24	10	24	4.3	24	7	999.9	69.1	43.0	0.00G	999.9	0
9/18/2019	59.8	24	46.3	24	1023	24	1008.6	24	10	24	5	24	8	999.9	75.9	54.0	0.00G	999.9	0
9/17/2019	64.9	24	46.5	24	1019.2	24	1005	24	10	24	6.7	24	14	21	75.9	54.0	0.00G	999.9	0
9/16/2019	66.9	24	60.1	24	1017.4	22	1003.3	24	10	24	2.8	24	5.1	999.9	80.1	62.1	0.00G	999.9	10000
9/15/2019	70.3	24	60.5	24	1020.2	23	1006.2	24	8.9	24	3.4	24	9.9	999.9	80.1	55.0	0.02G	999.9	10000
9/14/2019	64.4	24	56.5	24	1026.4	23	1011.9	24	10	24	6	24	11.1	19	71.6*	55.9*	0.00G	999.9	10000
9/13/2019	61.3	24	50	24	1029.2	23	1014.5	24	10	24	7.8	24	13	19	78.1	54.0	0.03G	999.9	10000
9/12/2019	72.3	24	68	24	1017.1	20	1003.6	24	8.7	24	3.7	24	8	999.9	87.1	64.0	0.05G	999.9	10000
9/11/2019	74.1	24	65.9	24	1020.4	19	1006.9	24	10	24	4.9	24	8.9	999.9	87.1	63.0	0.02G	999.9	0
9/10/2019	66.9	24	62.1	24	1026.2	23	1012	24	9.5	24	3.1	24	9.9	999.9	75.0	60.1	0.00G	999.9	10000
9/9/2019	67.2	24	53.8	24	1022.6	24	1008.4	24	10	24	5.1	24	7	999.9	77.0	57.9	0.00G	999.9	0
9/8/2019	66.5	24	53.5	24	1015.2	24	1001.1	24	10	24	5.9	24	12	17.1	77.0	54.0	0.12G	999.9	0
9/7/2019	62.5	24	52.5	24	1008.5	22	994.3	24	9.6	24	10.3	24	15	24.1	73.0*	53.6*	0.14G	999.9	10000
9/6/2019	61	24	54.8	24	1013.8	24	999.6	24	9.4	24	6	24	12	25.1	72.0	57.9	0.00G	999.9	10000
9/5/2019	66.3	24	56.2	24	1017.1	24	1003.1	24	10	24	4	24	9.9	999.9	86.0	57.9	0.00G	999.9	0
9/4/2019	71.5	24	64.5	24	1013.8	21	1000	24	8.6	24	3.3	24	8.9	999.9	86.0	61.0	0.00G	999.9	10000

FIGURE 2. Original Data Table

unreported recordings, which appear as '999.9' in the dataset. Since this rarely occurred (only about 20-25 recordings), the missing information was manually entered from the historical report on WeatherUnderground.com. Next, the last column 'FRSHTT' was reformatted into a numerical range. This column was the observed conditions in a binary format (0-No / 1-Yes), each of the six digits indicate whether a condition was observed (F - Fog, R - Rain, S - Snow, H - Hail, T - Thunder, T - Tornado). The numerical scale consisted of numbers 0 - 3; with 0 being no conditions present, 1 for fog, 2 for rain, and 3 for snow (any combination would be a decimal between two values). Tornados weren't considered because they were never reported in the dataset. Thunder wasn't considered because it would be difficult to incorporate in the numerical scale since it can occur with any other weather condition, a redesigned scale to incorporate all six observations would be ideal in the future. The final step in cleaning the data was removing any column of recordings that are not recorded at the Westchester Airport weather station, for example the snowfall is not recorded and can be seen by all values being '999.9'.

Once all the data was cleaned, each column was normalized by subtracting each value by the mean of the column and dividing by the standard deviation of the column. This method has been tested to improve data compression methods such as PCA, because it scales the recordings to a have the same normal standard deviation and therefore equal weights. The new normalized data was then transposed and saved to a '.csv' file to prepare it in the proper DMD time series format where each column represents an equidistant timestamp for recordings. The final '.csv' file is shown below in a screenshot of the data along with a screenshot of the imported data into Python.

After analyzing the historical weather data using the developed DMD function, the function was tested on a second dataset. Instead of testing it on a different airport's weather station, the function was tested on a series of satellite images because the function should be versatile and work for different types of datasets. The satellite images chosen were a series of water vapor cloud coverage pictures of the north-western hemisphere (also obtained from NOAA.gov). A sample image can be seen in the figure below.

Each image is taken roughly 30 minutes apart; in order to analyze a full day, 48 images were downloaded for the day January 1st, 2019. A Python function was created to cycle through all

Date:	10/1/2009	10/2/2009	10/3/2009	10/4/2009	10/5/2009	10/6/2009	10/7/2009	10/8/2009
Mean Temp	-0.21467026	-0.13959846	0.6399933	0.65731756	0.22998578	0.1144907	0.46097593	0.172238
Mean Temp Count	0.0614444	0.0614444	0.0614444	0.0614444	0.0614444	0.0614444	0.0614444	0.06144
Mean Dew Point	-0.08086748	0.04974281	1.01103455	0.83340456	0.15945545	0.12810898	0.43634927	-0.059969
Mean Dew Point Count	0.06723224	0.06723224	0.06723224	0.06723224	0.06723224	0.06723224	0.06723224	0.067232
Mean Sea Lvl Pressure	-0.19330383	0.03434224	-0.59503218	-0.74233257	-0.51468651	-0.31382233	-1.84039006	-0.327213
Mean Sea Lvl Pressure Count	0.6956456	0.6956456	-1.87344411	-1.35962617	0.6956456	0.6956456	-1.61653514	0.69564
Mean Station Pressure	-0.18820322	0.03182358	-0.55949844	-0.66951184	-0.49074006	-0.28446494	-1.90716258	-0.298216
Mean Station Pressure Count	0.13535921	0.13535921	0.13535921	0.13535921	0.13535921	0.13535921	0.13535921	0.135359
Mean Visibility	0.6388811	0.6388811	-2.52196961	-2.06313645	0.6388811	0.6388811	-0.3297667	0.63888
Mean Visibility Count	0.07172117	0.07172117	0.07172117	0.07172117	0.07172117	0.07172117	0.07172117	0.071721
Mean Wind Speed	-0.20700923	-0.43601937	-0.89403965	-1.40931246	0.48002118	-0.52189817	1.02392026	1.33880
Mean Wind Speed Count	0.08796011	0.08796011	0.08796011	0.08796011	0.08796011	0.08796011	0.08796011	0.087960
Max Wind Speed	-0.18309376	0.00856726	-0.94973787	-1.33305993	0.39188932	-0.77724295	1.54185548	1.906011
Max Temp	-0.48600333	-0.04268702	0.28022239	0.6031318	0.11055812	0.17076157	0.28022239	-0.042687
Min Temp	-0.10515637	-0.35245163	0.88991268	0.88991268	0.35999425	0.17746679	0.54252171	0.424762
Total Precipitation	-0.36023793	-0.36023793	-0.21885198	0.45980059	-0.36023793	-0.36023793	0.54463216	-0.360237
Forecast	-0.87539601	-0.87539601	1.03299082	0.0787974	-0.87539601	-0.87539601	1.03299082	-0.875396

FIGURE 3. Cleaned, Normalized, and Transposed Data Table

```
In [2]: hist_data
Out[2]:
                          Date  10/1/2009  ...  9/30/2019  10/1/2019
0                   Mean Temp  -0.214670  ...   0.449426   0.778587
1              Mean Temp Count   0.061444  ...   0.061444   0.061444
2               Mean Dew Point  -0.080867  ...   0.258719   0.901322
3        Mean Dew Point Count    0.067232  ...   0.067232   0.067232
4        Mean Sea Lvl Pressure  -0.193304  ...   1.266309   0.328943
5   Mean Sea Lvl Pressure Count   0.695646  ...   0.181828   0.438737
6        Mean Station Pressure  -0.188203  ...   1.351984   0.416870
7   Mean Station Pressure Count   0.135359  ...   0.135359   0.135359
8              Mean Visibility   0.638881  ...   0.638881   0.587900
9        Mean Visibility Count   0.071721  ...   0.071721   0.071721
10             Mean Wind Speed  -0.207009  ...  -0.407393  -0.722282
11       Mean Wind Speed Count   0.087960  ...   0.087960   0.087960
12              Max Wind Speed  -0.183094  ...  -0.777243  -0.777243
13                    Max Temp  -0.486003  ...   0.772796   0.882257
14                    Min Temp  -0.105156  ...   0.595514   0.595514
15          Total Precipitation  -0.360238  ...  -0.360238  -0.331961
16                    Forecast  -0.875396  ...  -0.875396   1.032991

[17 rows x 3652 columns]
```

FIGURE 4. Python DataFrame for the dataset

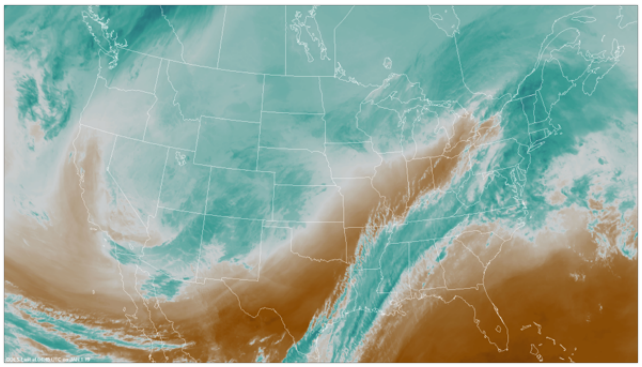

FIGURE 5. Original Water Vapor Image

the images, crop the image for north-east America, convert the image to an 'RGB' array of pixel information, and then append the array as a column vector to a data matrix. The final matrix will be in the proper DMD format, each column would be one image's 'RGB' information. In

this case, after cropping the images, the matrix was a 450,000 x 48 numpy array. A sample cropped image is shown in the figure below.

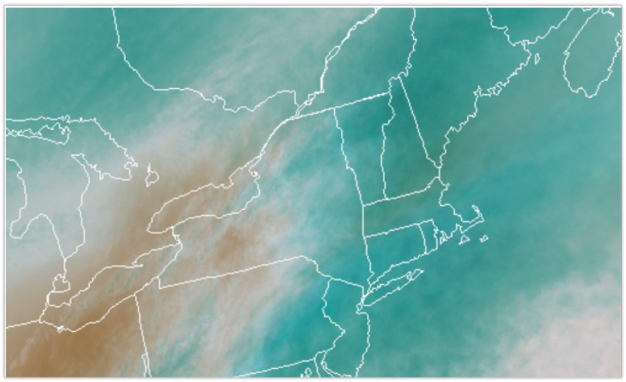

FIGURE 6. Cropped Water Vapor Image

Results.

Our first plan of action was to perform Dynamic Mode Decomposition onto the full dataset. We created a DMD reconstruction function on Python that accepted our dataset as the input parameter and returned a matrix of the reconstructed data. Seen in the figure below is the plot of the original data's mean temperature measurement vs the DMD reconstructed results.

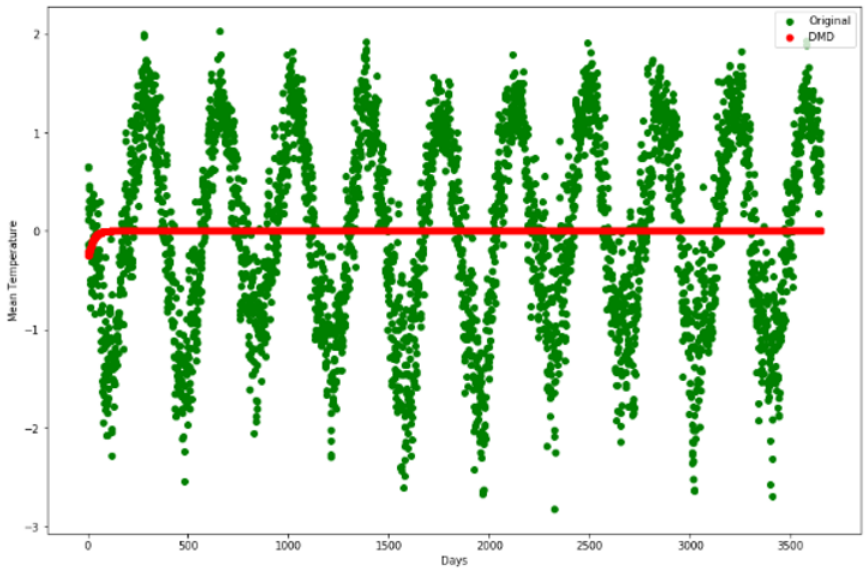

FIGURE 7. Original Mean Temperature Data vs DMD Reconstructed Data

These results were not what we were expecting, and we initially thought that there may have been a mistake in our algorithm. While looking further into why this was occurring, we discovered that because the A_r that we calculated was full rank, we could use the formula $\overline{\mathbf{X}} = A_r\mathbf{X}$ to get an accurate reconstruction of the data, which is displayed in the following figure.

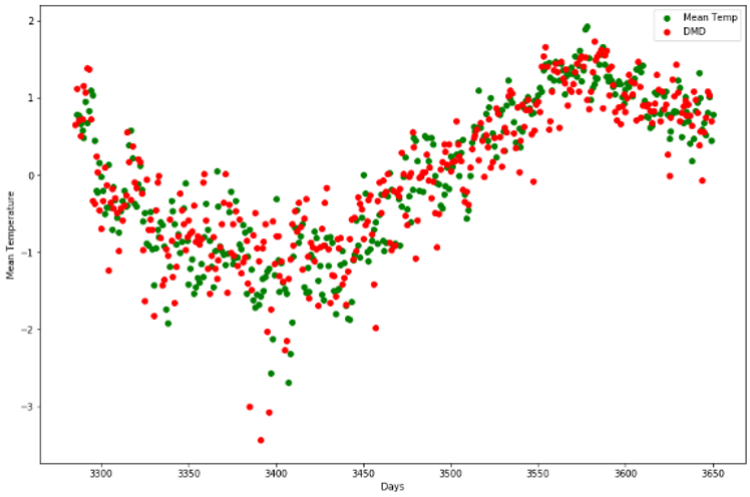

FIGURE 8. Original Mean Temperature Data vs A_r Reconstructed Data

We eventually discovered that our issue had occurred because of the properties of our dataset. More specifically, the number of snapshots was exceedingly too large compared to the number of snapshot attributes, causing DMD to fail, but this is further explained in the discussion section. A possible work around we found in our research was a method called 'Augmenting' to force a larger number of rows without collecting new data. This is done by copying the data from the second time recording to the last into a new set of rows and deleting the last time recording in the original rows. This can be done multiple times to as shown in the general equations below.

$$\mathbf{X}_{\text{aug}} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{m-s} \\ \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_{m-s+1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_s & \mathbf{x}_{s+1} & \cdots & \mathbf{x}_{m-1} \end{bmatrix},$$

$$\mathbf{X}'_{\text{aug}} = \begin{bmatrix} \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_{m-s+1} \\ \mathbf{x}_3 & \mathbf{x}_4 & \cdots & \mathbf{x}_{m-s+2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_{s+1} & \mathbf{x}_{s+2} & \cdots & \mathbf{x}_m \end{bmatrix}.$$

This essentially provides the data of future recordings in each column, allowing the algorithm to recognize the pattern better. Doing this especially helps with data such as a sinusoidal wave, where DMD would just predict a horizontal line as it did in our mean temperature prediction, this has been proven to work by augmenting a sin wave function. The results of testing this out on our data is shown below.

Testing out progressively larger amounts of augmentation created more accurate replication results to an extent. This was proven to not work with our data set when tested on a matrix with larger number of rows than columns (augmenting 250 times), the reproduced data again became meaningless as the values started to increase exponentially. The downside of this method is that we create a larger rank matrix by introducing new independent rows, which effectively takes more resources and again defeats the purpose of DMD. Running the larger augmentation matrices took 20+ minutes to run so testing this on our dataset was stopped due to time constraints.

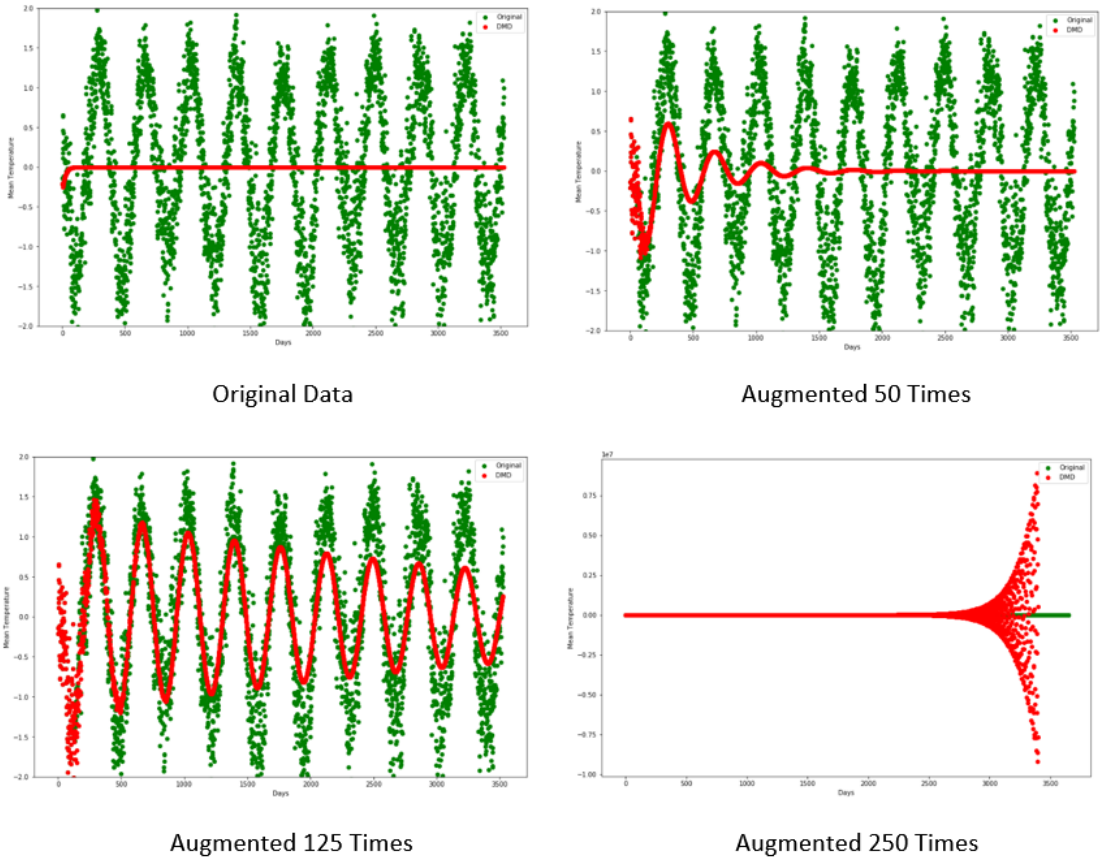

FIGURE 9. Original Data vs Augmented DMD Reconstructed Data

Another approach to create a more appropriate dataset is using less snapshots by splitting up the longer matrix. Breaking the dataset into many smaller ones, inputting them into the DMD algorithm, and combining all the data produced the following reconstruction. This reproduction led to some decently accurate results but because of the overall small size of snapshots and attributes, produced many extreme outliers that were inaccurate. This is displayed below.

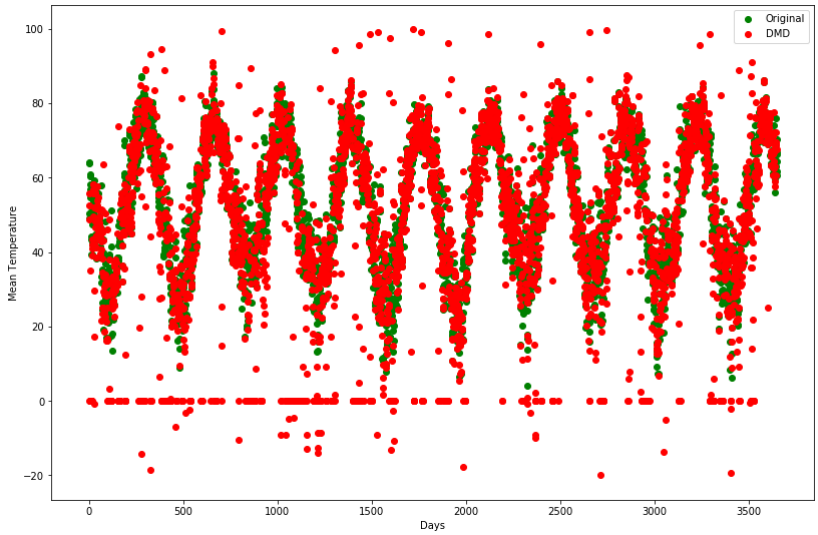

FIGURE 10. Original Data vs Fixed DMD Reconstructed Data

We then decided to test the prediction capabilities of our DMD algorithm. To do this, we ran DMD with a smaller dataset of 15 days and had it predict results for the following 10 days, allowing us to compare the results to the known data in the original dataset. We then plotted the mean temperature, mean visibility, wind speed, and fog/rain/snow forecast of the DMD results against the original data. The results are shown below, where predictions are not dead on accurate but do show hints of the actual dynamics of the original weather recordings.

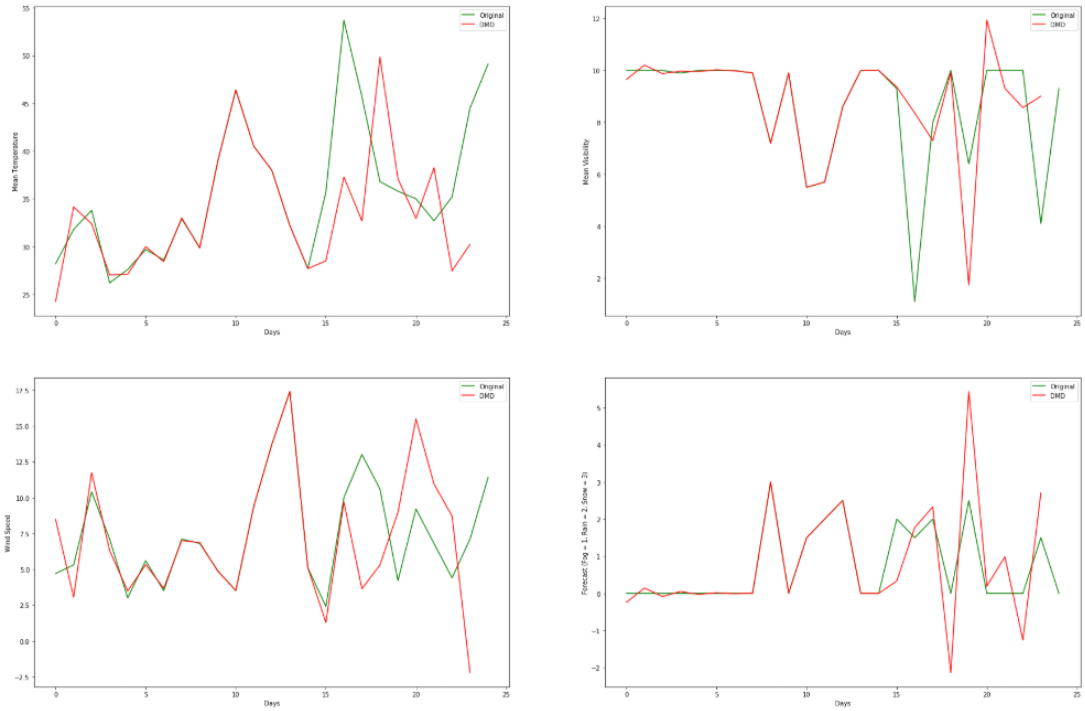

FIGURE 11. Original Weather Data vs DMD Predicted Data

With our numerical weather data reconstruction and predictions now providing some useful and expected results, we then wanted to test our algorithm on the water vapor images weather dataset to see how effective DMD would be at reconstructing an image. We created a Python script that could crop our images to focus on just the general New York area and transformed our images into column vectors, so that our dataset would consist of each column representing a different picture, and each row representing a single pixel in the images. We also had to use sparse svd in our DMD function as using full numpy svd led to memory errors due to the large size of our matrix. The reconstructions were surprisingly good, showing cloud patterns very close to the original images, even though the color seemed to be a little bit weaker on the reconstructed data.

After reconstruction, we tried to test the predictive capabilities of Dynamic Mode Decomposition. To do this, we ran through DMD with the first 30 columns and predicted the following 5 half hour intervals. The results are shown below, where the immediate prediction is a very good representation, but it seems that as you go further, the images almost go stagnant, leading to worse results the further you predict. We may be able to improve predictions with more snapshots, and this is a consideration for future work.

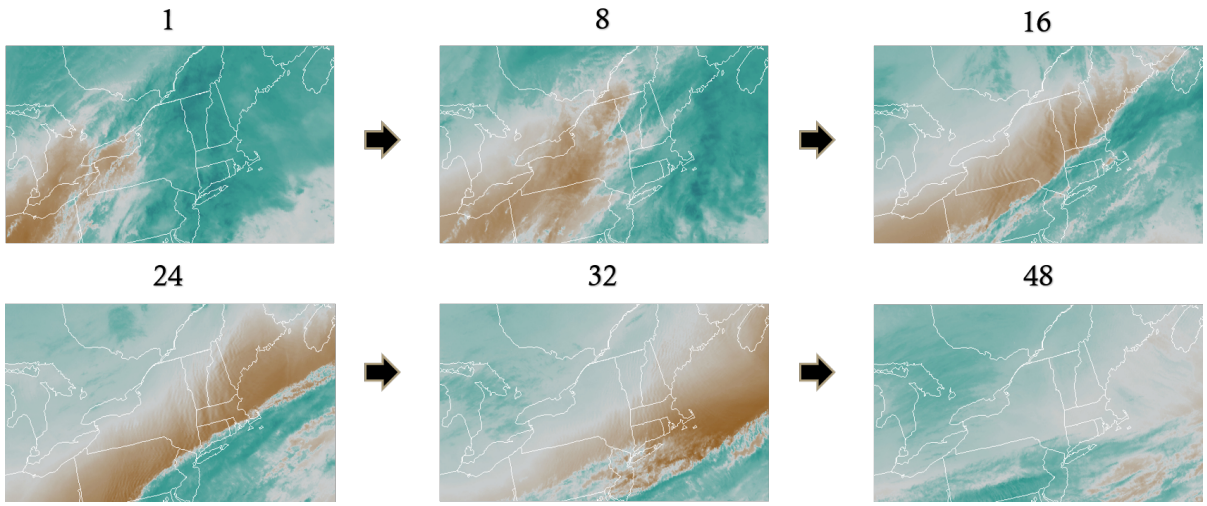

FIGURE 12. Original Water Vapor Image Data

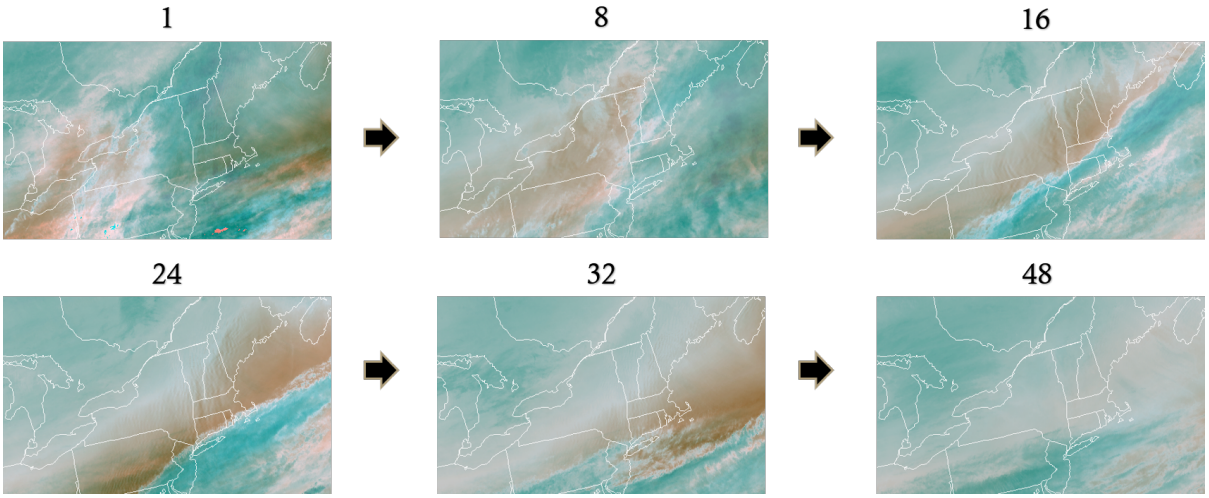

FIGURE 13. DMD Reconstructed Image Data

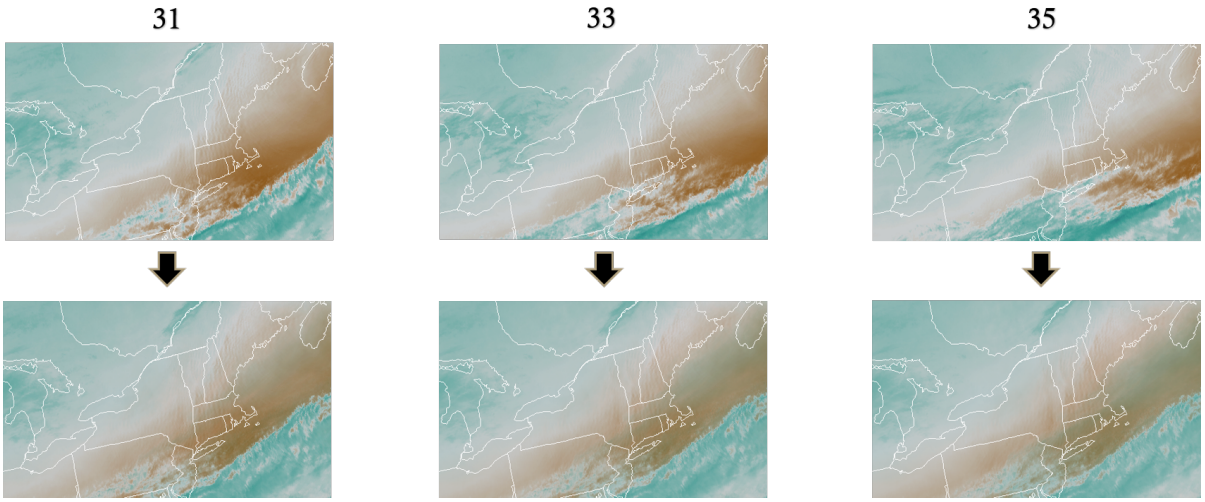

FIGURE 14. DMD Predicted Image Data

Discussion.

While conducting our research, we came across a number of problems concerning reconstruction and prediction using DMD on our weather dataset. At first, we were only able to somewhat accurately reconstruct the data using the A_r matrix since our dataset was full rank. However, this defeats the purpose of using dynamic mode decomposition, as its goal is to reduce the dimensionality of the dataset and reconstruct the data without having to compute the full A matrix. After reading past research papers on DMD [9][10], we discovered that the main issue we were facing was caused by the properties of our dataset. We learned that Dynamic Mode Decomposition will fail immediately if the data matrix is full rank and has no suitable low-dimensional structure. DMD finds its success when the dataset can be approximated by a low-rank matrix, where it can use the low dimensional structure to project future states of the system.

Another key to finding success in Dynamic Mode Decomposition is that the number of rows, or attributes per snapshot (weather recordings for a specific day in our case), should be larger than the number of columns, or number of data snapshots taken. This again relates to DMD wanting to make use of low-rank structure so that the number of DMD modes is less than the number of original attritbutes of the snapshots, therefore reducing the dimensionality of the system. This reduction of snapshot attributes can only occur if the number of snapshots is smaller in size than the attributes themselves, which was not the case for our original dataset, which only contained 17 rows of attributes compared to over 3000 columns of recorded snapshots. In order to account for these issues, we decided to split up our dataset 15 columns at time, and then appending the reconstructed and predicted data back together to form our full reconstruction, which yielded the much better results shown earlier.

When looking into why our DMD reconstruction was failing, we also came across a modified version of the technique, called Higher Order Dynamic Mode Decomposition (HODMD), which focuses on applying DMD to low spatial dimensional dynamical systems where the number of snapshot attributes is very small compared to the number of snapshots, such as our weather dataset. Explained well in [11], HODMD uses the higher order Koopman assumption to modify and add spatial dimensions to the snapshots of a system, where the number of extra dimensions, d, is tunable, as long as it is less than the number of snapshots. There is actually a python library named PyDMD [12] that contains a HODMD function, where one can specify the data matrix and the desired d, and it will reconstruct the data accordingly. We decided to explore this route, so we created a python script that would run through HODMD for every possible d and calculate the mean squared error of the reconstructed data compared to the original dataset. We then plotted the reconstructed data using the most accurate d, and we were able to get some surprisingly good results using the original normalized dataset, shown in the figure below.

In conclusion, this experiment was not a straight-forward or smooth process. As mentioned, the first attempt to directly plug the dataset into the DMD algorithm didn't work as expected because it was not the ideal size for the method to work. Because of this, we discovered various ways to adjust the dataset prior to using the algorithm. The first technique, augmentation, appeared to improve the replication results but increased computation time and resources. Testing augmentation was stopped due to this, but further testing will be considered because of the promising results this returned. We believe there may be an optimal augmentation number that reproduces the data, this would be an ideal method to code; a program that cycles through augmentation and returns the optimal number. The second dataset adjustment considered was splitting up the dataset into many smaller matrices and applying the DMD algorithm

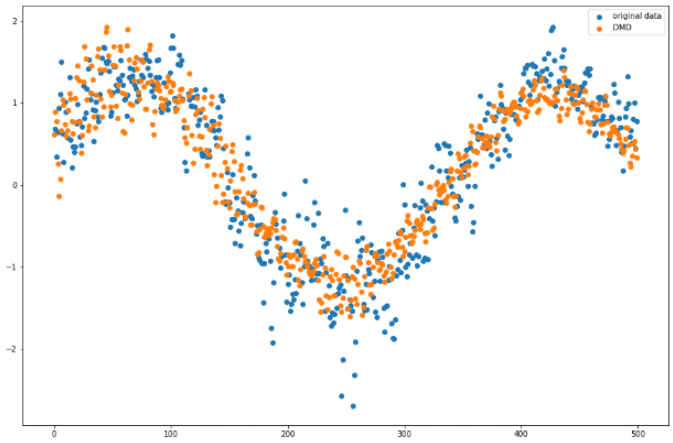

FIGURE 15. Original Data vs. HODMD Reconstructed Data

to each individual dataset. The replicated data was mostly accurate but did include some extreme outliers with the reconstructed datasets combined. Looking at the datasets individually also provided some promising forecasting results, but they were not always accurate due to the limited amount of data the algorithm considers with this adjustment. This method would in theory work if enough recording attributes were obtained; a possible option would be combining various weather station recordings into one dataset, allowing for a larger matrix.

While both tests provided proof that the idea of using DMD on weather forecasting would work with an appropriate dataset, our main focus shifted to the water vapor images dataset due to the fact that it had many more rows than columns and was therefore ideal for DMD. The algorithm worked well with compressing the images which is useful when dealing with a large quantity of high-quality images such as the ones used for weather radar. This alone provides a practical application for DMD, but in addition to compression, our results hinted to DMD being able to predict future states of the water vapor clouds analyzed in the project. Very immediate predictions showed good accuracy, and with more snapshots, prediction capabilities would most likely increase. Along with further testing on the results based on the number of snapshots used, other paths of future work include using different satellite imagery on the NOAA website that we can compress and predict other forecast variables such a cloud coverage, wind speeds, and temperature gradients.

Division of Labor.

Throughout this project, tasks were split up between the two authors, Michael Campiglia and Anton Paljusevic. Michael handled the LaTex preparaton, main DMD python function, dataset splitting technique, image results, Higher Order DMD, and methods. Anton handled the data collection, image cropping algorithm, augmentation technique, dataset description, and implementation of mathematical tools. Both authors contributed to the powerpoint, introduction, results, discussion, and references.

REFERENCES

[1] Neethu Mohan, K.P. Soman, S. Sachin Kumar, A data-driven strategy for short-term electric load forecasting using dynamic mode decomposition model, Applied Energy, Volume 232, 2018, Pages 229-244, ISSN 0306-2619, https://doi.org/10.1016/j.apenergy.2018.09.190. (http://www.sciencedirect.com/science/article/pii/S0306261918315009)

[2] Hua, Jia-Chen Roy, Sukesh McCauley, Joseph Gunaratne, Gemunu. (2015). Using dynamic mode decomposition to extract cyclic behavior in the stock market. Physica A: Statistical Mechanics and its Applications. 448. 172â180. 10.1016/j.physa.2015.12.059. Santosh Tirunagari, Samaneh Kouchaki, Norman Poh, Miroslaw Bober, David Windridge. Dynamic Mode Decomposition for Univariate Time Series: Analysing Trends and Forecasting. 2017. â¨hal-01463744âŠ

[3] Bi, Chongke, et al. "Dynamic mode decomposition based video shot detection." IEEE Access 6 (2018): 21397-21407.

[4] C.W. Rowley, I Mezic, S. Bagheri, P. Schlatter, and D.S. Henningson, "Spectral analysis of nonlinear flows." Journal of Fluid Mechanics 641 (2009): 85-113

[5] Hirsh, Seth M., et al. "Centering Data Improves the Dynamic Mode Decomposition." arXiv preprint arXiv:1906.05973 (2019).

[6] Santosh Tirunagari, Samaneh Kouchaki, Norman Poh, Miroslaw Bober, David Windridge. Dynamic Mode Decomposition for Univariate Time Series: Analysing Trends and Forecasting. 2017. â¨hal-01463744âŠ

[7] Kutz, J. Nathan et al. âOn Dynamic Mode Decomposition: Theory and Applications.â Journal of Computational Dynamics 1.2 (2014): 391â421. Crossref. Web.

[8] https://www.britannica.com/science/weather-forecasting/Progress-during-the-early-20th-century

[9] Mann, Jordan, and J. Nathan Kutz. "Dynamic mode decomposition for financial trading strategies." Quantitative Finance 16.11 (2016): 1643-1655.

[10] Schmid, Peter J. "Application of the dynamic mode decomposition to experimental data." Experiments in fluids 50.4 (2011): 1123-1130.

[11] Le Clainche, Soledad, and JosĂŠ M. Vega. "Higher order dynamic mode decomposition." SIAM Journal on Applied Dynamical Systems 16.2 (2017): 882-925.

[12] Demo et al., (2018). PyDMD: Python Dynamic Mode Decomposition. Journal of Open Source Software, 3(22), 530, https://doi.org/10.21105/joss.00530

MANHATTAN COLLEGE, COMPUTATIONAL METHODS FOR ANALYTICS