A Report on the Mapping and Analyzing of the Mariana Trench

Ezra Engelmann, Luke Morrissey, Nathan Wratney

April 2019

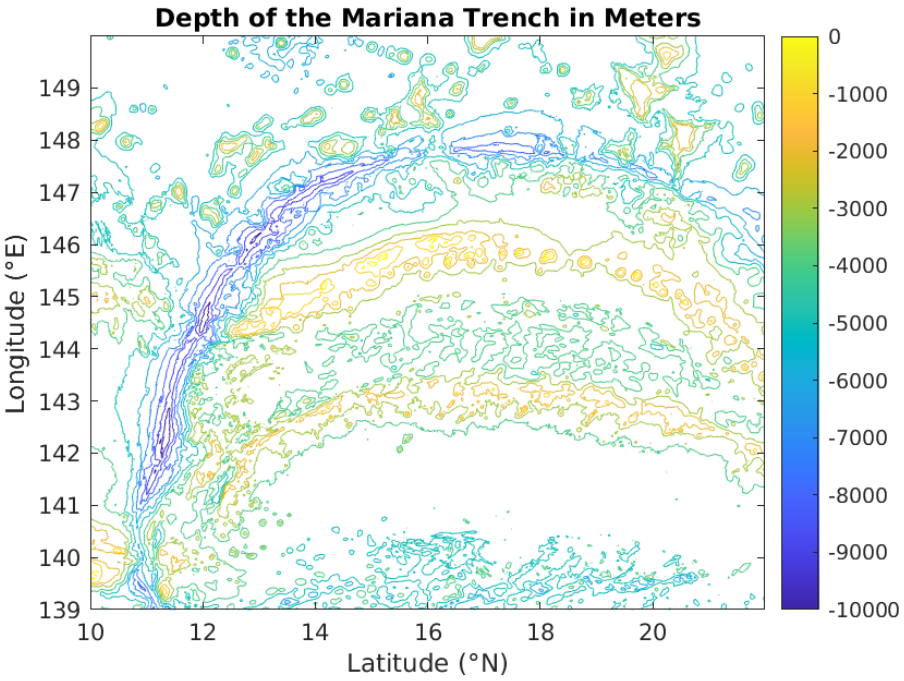
1 Introduction

We were contracted by the United States National Oceanic and Atmospheric Administration, henceforth referred to as NOAA, to simplify, interpret, and analyze bathymetric data collected on the Mariana Trench. Located between Japan and Papua New Guinea, the trench is a relatively unknown territory. The data fully maps the trench with high-resolution data points, and we have simplified this data using the Incomplete Standard Value Decomposition Process. Once completed, we constructed the trench with little to no errors, without processing the full extent of the file.

2 Investigating the Trench

2.1 Mapping the Trench

Having completed the Decomposition process, we constructed the graphic below.



A contour map of the Mariana Trench, as constructed from NOAA files obtained on 4/7/2020.

2.2 Analyzing the Depth

Having fully processed the data, we were able to determine the deepest point in the Mariana Trench, as well as its corresponding latitude and longitude.

The point is located at 11.333°N, 142.2°E at a depth of 10930 meters.

Defining the ocean floor to be at a depth of 6 kilometers, we analyze the data for all points deeper than the ocean floor, in order to calculate the average depth of the Mariana Trench.

We find that the average depth of the Mariana Trench is 7204.6 meters.

3 Computing Eigenvectors

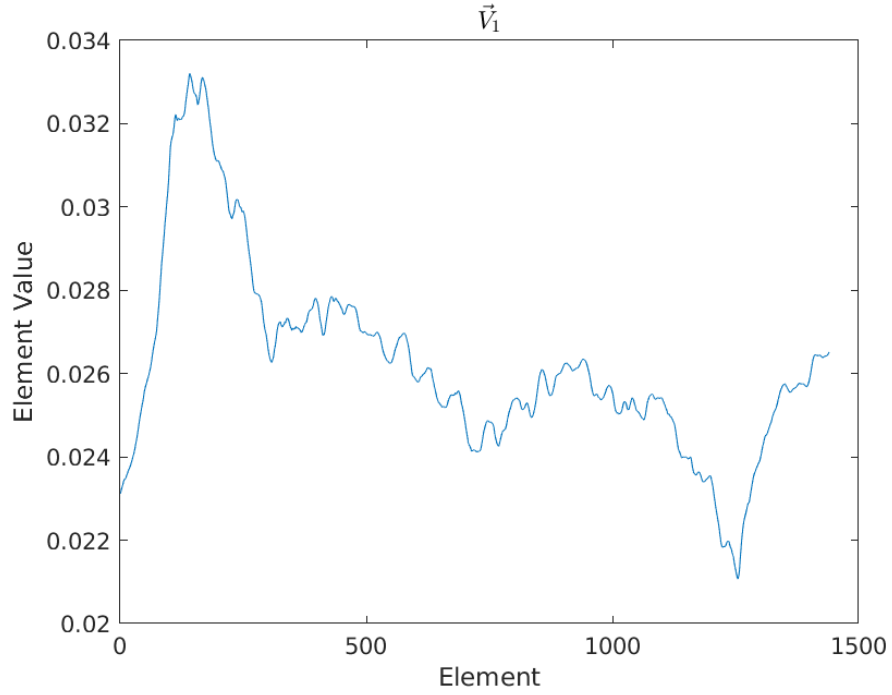
Representing the data above takes a lot of space which is very valuable. Let's explore how we can use some creative math to make things more efficient.

First, we code our algorithm to find the first eigenvector and eigenvalue of $A^T A$, letting A represent the depth matrix.

We start by using a random guess for the vector, with the correct size and magnitude 1.

Having created this matrix, we apply $A^T A$, then divide this new vector by its magnitude to ensure it is still 1. We repeat this process until the vector stops changing, resulting:

We call this vector $\vec{V_1}$, with eigenvalue of $3.8802 * 10^{13}$. We have provided a plot of the eigenvector as well for your convenience.



The eigenvector $\vec{V_1}$

We predict this works because $\boldsymbol{A^T A}$ is a symmetric matrix so its eigenvectors form a basis of $\mathbb{R}^n$ so

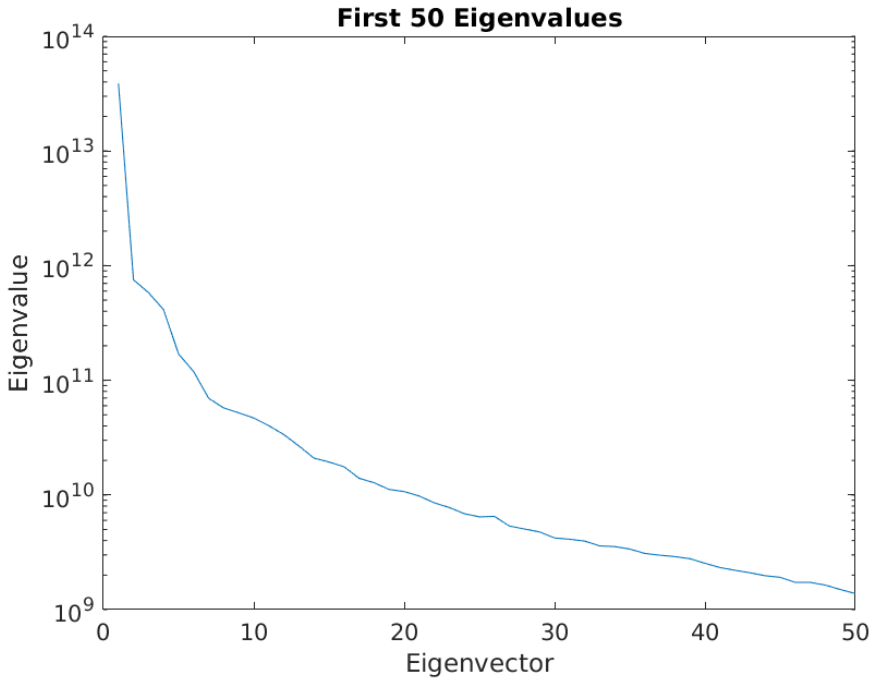
$$\vec{\boldsymbol{u}} = c_1 \boldsymbol{V}_1 + c_2 \boldsymbol{V}_2 + ... + c_n \boldsymbol{V}_n$$

If we apply $\boldsymbol{A^T A}$ to it many times we get:

$$\boldsymbol{A^T A}^k \vec{u} = \lambda_1^k c_1 \boldsymbol{V}_1 + \lambda_2^k c_2 \boldsymbol{V}_2 + ... + \lambda_n^k c_n \boldsymbol{V}_n$$

If $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq ... \geq \lambda_n$, then as k increases the λ_1 term will dominate and by normalizing it we will arrive at the first eigenvector.

Then, we used the Gram-Schmidt Orthogonalization process to compute the 50 largest eigenvalues, and their associated eigenvectors. We do this because we know the eigenvectors of a symmetric matrix like A^TA forms a basis and thus are orthogonal. We use this process in between multiplying by A^TA and dividing by the magnitude. Once completed, we stored the values in a 1440x50 matrix, and have provided a semilog plot of the eigenvalues below:



Semilog plot of the first 50 eigenvalues.

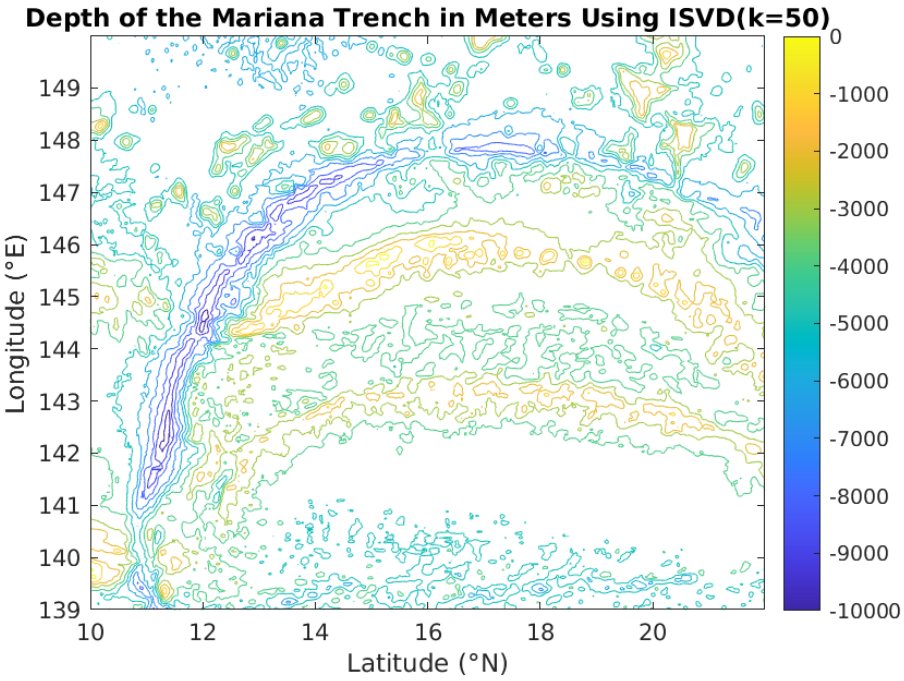
4 The Incomplete SVD Decomposition

Now we want to construct an incomplete SVD decomposition to reduce the amount of data we need to store without losing too much accuracy.Our first step is to take the square root of the eigenvalues along the diagonal of the 50x50 matrix we computed in the previous part. We call this matrix $\mathbf{\Sigma}$. Next, we defined a matrix $\mathbf{U}$, where each column of $\mathbf{U}$ is equal to $\mathbf{A}$ times the associated column of $\mathbf{V}$, divided by the associated element of $\mathbf{\Sigma}$.

$$\mathbf{A} \approx \boldsymbol{U\Sigma V}^T$$

The motivation behind making the incomplete SVD decomposition is we can save storage but keep the relevant features of $\mathbf{A}$. Specifically with k=50 we found that we only need to store 138,050 elements as opposed to 1,900,800 elements we need to store $\mathbf{A}$. This is because $\mathbf{A}$ stores 1320*1440 elements. Also $\mathbf{\Sigma}$ stores 50 elements, $\mathbf{U}$ stores 1320*50 elements and $\mathbf{V}$ stores 1440*50 elements. This means by using this decomposition we use less than 10% of the original amount of elements required to represent $\mathbf{A}$. Let's take a look at how much resolution we lost.

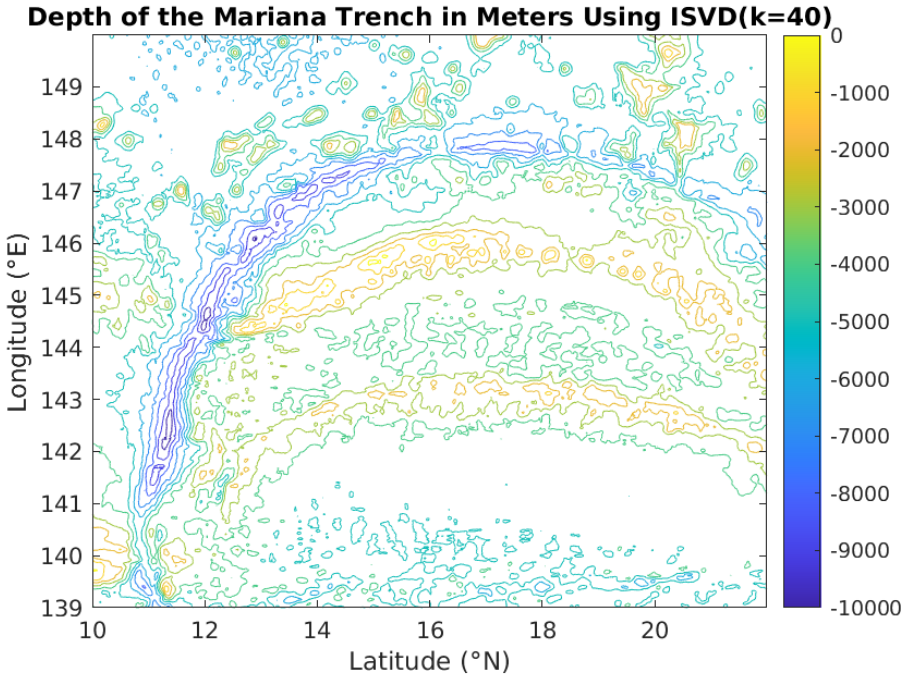
Below we have included the depiction of the Mariana Trench using the incomplete SVD decomposition.



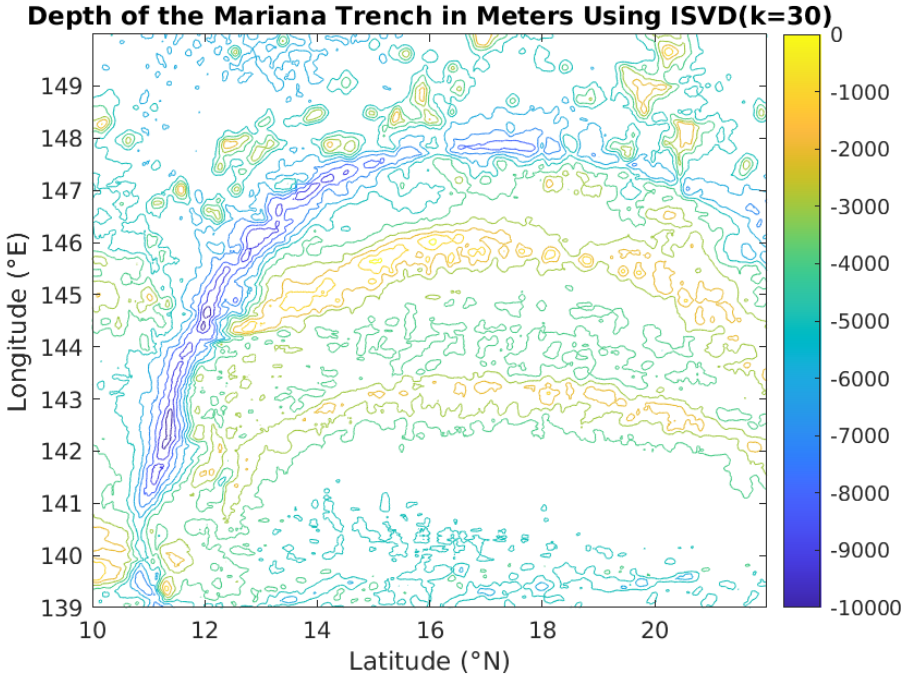
A contour map of the Mariana Trench, as constructed by the ISVD process on 4/7/2020.

When comparing this to the representation using the full data set it is hard to notice any differences and the ones you can notice are very minor. We were able to use a lot less elements and hardly lose and important features.

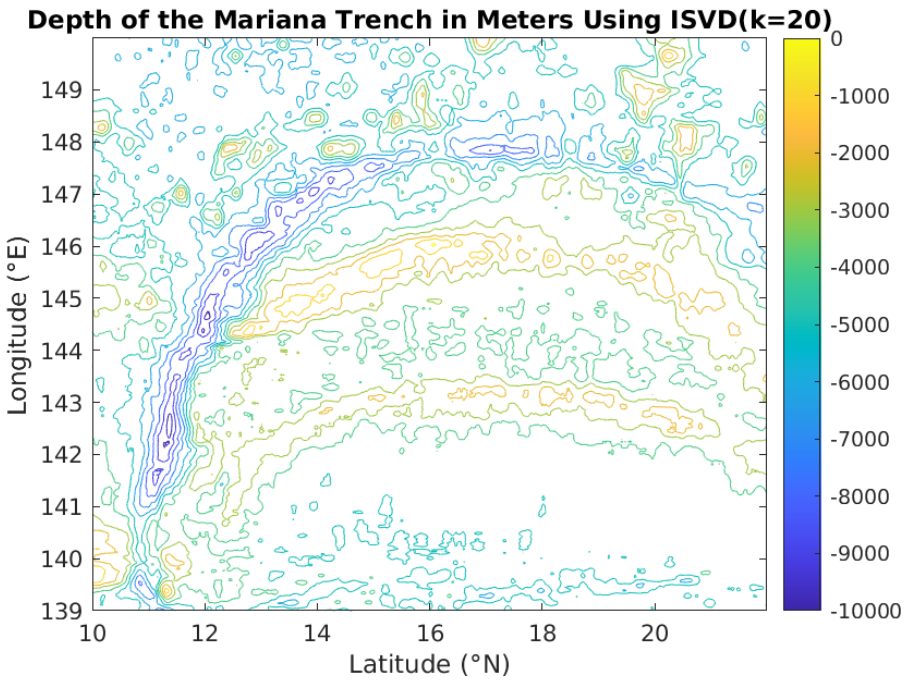
Let's now push it even farther by using k values of 40,30,20,10 and 100.



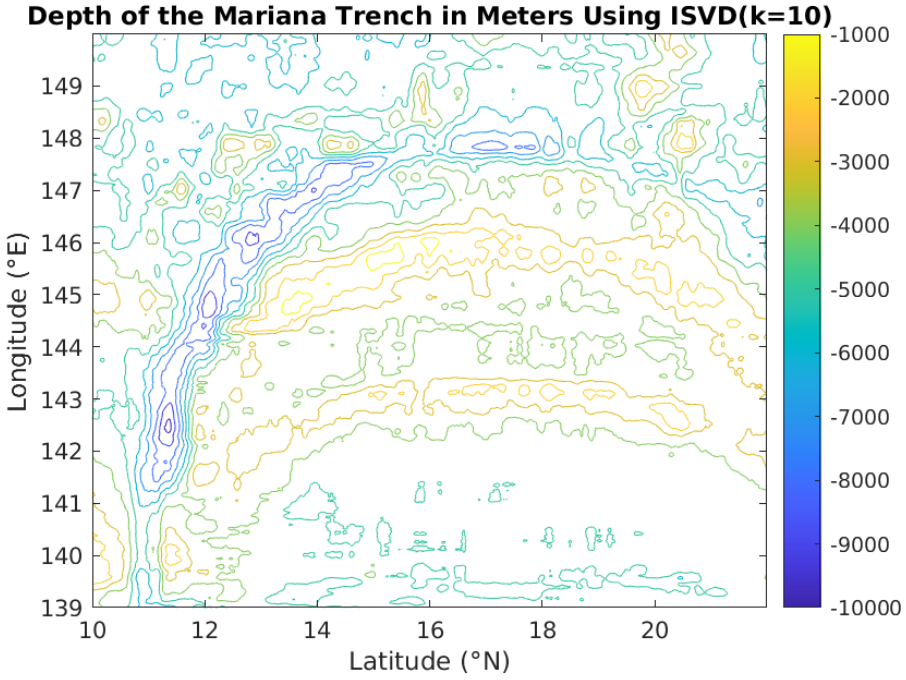
A contour map of the Mariana Trench, as constructed by the ISVD process with k=40 on 4/7/2020.



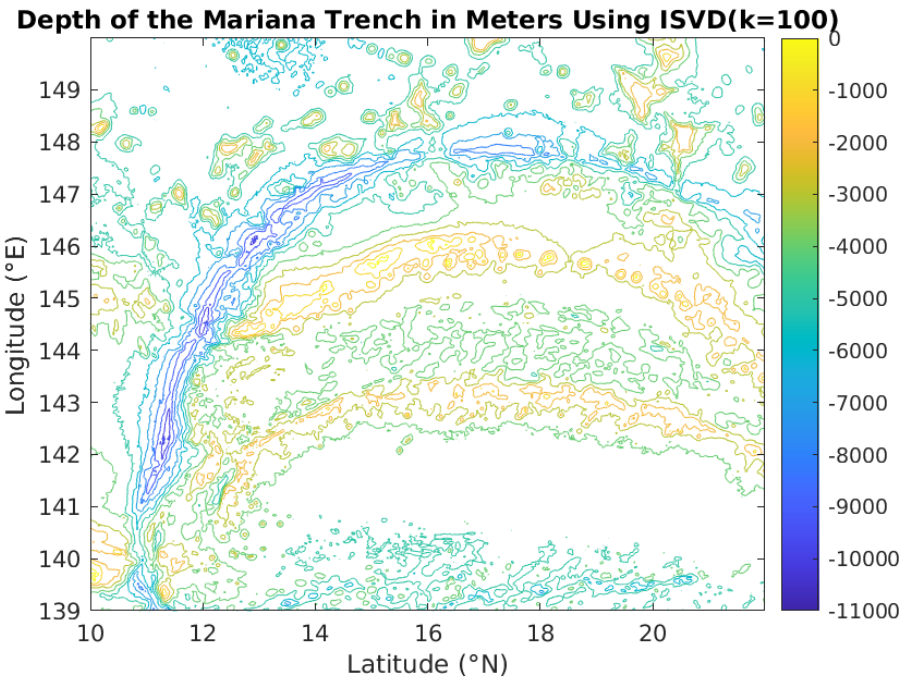
A contour map of the Mariana Trench, as constructed by the ISVD process with k=30 on 4/7/2020.



A contour map of the Mariana Trench, as constructed by the ISVD process with k=20 on 4/7/2020.



A contour map of the Mariana Trench, as constructed by the ISVD process with k=10 on 4/7/2020.



A contour map of the Mariana Trench, as constructed by the ISVD process with k=100 on 4/7/2020.

As you can see as the value of k decreases you lose more and more detail but the trench is still recognizable. The detail is lost but the main features prevail.

Overall using the incomplete SVD decomposition is an effective way to store elements of a matrix if you don't need the exact values. Even using only k=50 is very effective at recreating the data and reduces the amount of data you need to store significantly. By playing with the k value you can get the balance you desire of precision vs saved space. As demonstrated with the Mariana Trench it was very effective at reducing the size of the data needed to store to represent it.

5 MATLab Code

```
%Imports the Data
A = importdata('mariana_depth.csv');
lon = importdata('mariana_longitude.csv');
lat = importdata('mariana_latitude.csv');

%Makes new matrices that are the same size as A with lat and lon
latTot = zeros(1320,1440);
lonTot = zeros(1320,1440);

for i = 1:1320
    latTot(i,:) = lat';
end
for i = 1:1440
    lonTot(:,i) = lon;
end


%Makes first figure
f1 = figure(1);
contour(latTot,lonTot,A);
colorbar;
title('Depth of the Mariana Trench in Meters');
xlabel('Latitude ({\circ}N)');
ylabel('Longitude ({\circ}E)');
saveas(f1,'211.png');



%finds deepest and average depth
deep = findDeep(lon,lat,A)
avg = findAvg(A)


%finds first eigenvector
eig = findFirstEig(A);

%makes A'A and multiplies it by the eigenvector
AtA = A'*A;
AtE = AtA*eig;

%Finds eigenvalue by definition of eigen value
a = AtE(879,1)/eig(879,1)

b = AtE(538,1)/eig(538,1)


%plots first eigenvector
N = 1:1440;
N = N';
f2 = figure(2);
plot(N,eig);
title('$\vec{V}_1$','Interpreter','latex');
```

```
xlabel('Element');
ylabel('Element Value');
saveas(f2,'221.png');



%finds first 50 eigenvectors and stores them in matrix V
V = findHundredEig(AtA);
%finds eigenvalue of each eigenvector and stores it in vector eigValues
eigValues = findEigValues(V,AtA);

%Plots the first 50 eigen values on semilog plot
count = 1:50;
count = count';
f3 = figure(3);
semilogy(count,eigValues(1:50));
title('First 50 Eigenvalues');
xlabel('Eigenvector');
ylabel('Eigenvalue');
saveas(f3,'222.png');


%finds the total for A and the total for USV
totalA = 1320*1440
totalUSV = 1320*50 + 50 +1440*50

%computes USV'
USV50 = ISVD(50,V,eigValues,A);

%Plots USV'
f4 = figure(4);
contour(latTot,lonTot,USV50);
colorbar;
title('Depth of the Mariana Trench in Meters Using ISVD(k=50)');
xlabel('Latitude ({\circ}N)');
ylabel('Longitude ({\circ}E)');
saveas(f4,'233.png');


%same as above but uses k=40
USV40 = ISVD(40,V,eigValues,A);
f5 = figure(5);
contour(latTot,lonTot,USV40);
colorbar;
title('Depth of the Mariana Trench in Meters Using ISVD(k=40)');
xlabel('Latitude ({\circ}N)');
ylabel('Longitude ({\circ}E)');
saveas(f5,'ISVD40.png');

%same as above but k=30
USV30 = ISVD(30,V,eigValues,A);
f6 = figure(6);
contour(latTot,lonTot,USV30);
colorbar;
```

```
title('Depth of the Mariana Trench in Meters Using ISVD(k=30)');
xlabel('Latitude ({\circ}N)');
ylabel('Longitude ({\circ}E)');
saveas(f6,'ISVD30.png');

%same as above but k = 20
USV20 = ISVD(20,V,eigValues,A);
f7 = figure(7);
contour(latTot,lonTot,USV20);
colorbar;
title('Depth of the Mariana Trench in Meters Using ISVD(k=20)');
xlabel('Latitude ({\circ}N)');
ylabel('Longitude ({\circ}E)');
saveas(f7,'ISVD20.png');

%same as above but k = 10
USV10 = ISVD(10,V,eigValues,A);
f8 = figure(8);
contour(latTot,lonTot,USV10);
colorbar;
title('Depth of the Mariana Trench in Meters Using ISVD(k=10)');
xlabel('Latitude ({\circ}N)');
ylabel('Longitude ({\circ}E)');
saveas(f8,'ISVD10.png');

%same as above but k=100
USV100 = ISVD(100,V,eigValues,A);
f9 = figure(9);
contour(latTot,lonTot,USV100);
colorbar;
title('Depth of the Mariana Trench in Meters Using ISVD(k=100)');
xlabel('Latitude ({\circ}N)');
ylabel('Longitude ({\circ}E)');
saveas(f9,'ISVD100.png');

function [P] = findDeep(long,lat,A)
%goes through all points and finds min
o = 0;
a = 0;
min = 0;
for i = 1:1320
    for j = 1:1440
        if A(i,j) < min
            min = A(i,j);
            o = i;
            a = j;
        end
    end
end
P = [long(o),lat(a),min];

end

function [x] = findAvg(A)
```

```
%goes through all points and it they are less than 6000 finds the average
count = 0;
tot = 0;
for i = 1:1320
    for j = 1:1440
        if A(i,j) < -6000
            count = count + 1;
            tot = tot + A(i,j);
        end
    end
end

x = tot/count;
end

function [E] = findFirstEig(A)
%finds first eigenvector
tol = 1e-10;
uold = rand(1440,1);
uold = uold/norm(uold);
unew = A'*A*uold;
while norm(unew-uold) > tol
    uold = unew;
    unew = A'*A*unew/norm(A'*A*unew);
end
E = unew;
end

function [V] = findHundredEig(A)
%finds first 100 eigenvectors
V = zeros(1440,100);
for i = 1:100
    uold = rand(1440,1);
    uold = uold/norm(uold);
    unew = A*uold;
    unew = unew/norm(unew);
    while norm(uold-unew) > 1e-3
        uold = unew;
        unew = A*uold;
        for j = 1:(i-1)
            unew = unew - (unew'*V(:,j))*V(:,j);
        end
        unew = unew/norm(unew);
    end
    V(:,i) = unew;
end
end

function [E] = findEigValues(V,A)
%finds all the eigenvalues
E = zeros(100,1);
av = A*V;
for i = 1:100
    E(i) = av(100,i)/V(100,i);
```

```
end
end

function [O] = ISVD(x,V,eigValues,A)
%Does all the ISVD stuff
sigma = eye(x);
for i = 1:x
    sigma(i,i) = eigValues(i)^(1/2);
end
Vnew = V(:,1:x);
U = A*Vnew;
for i = 1:x
    U(:,i) = U(:,i)/sigma(i,i);
end
O = U*sigma*Vnew';

end
```