APPM 2360 Spring 2022

Project 2

Analysis of the Mariana Trench Utilizing Incomplete Singular Value Decomposition

Ferin Von Reich, Ben Scheck, Aaron Groudan

April 5, 2022

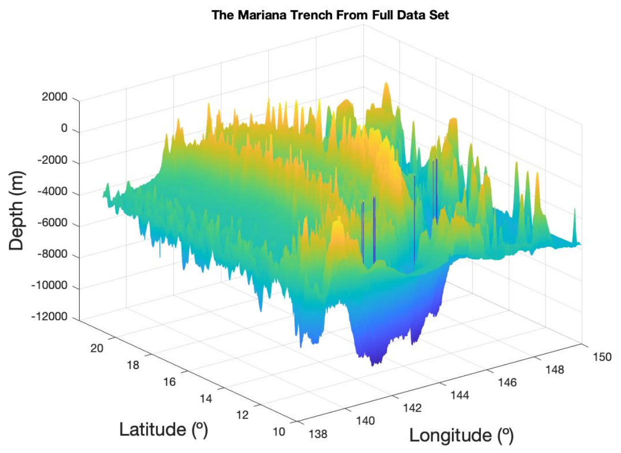
Contents

1 Introduction

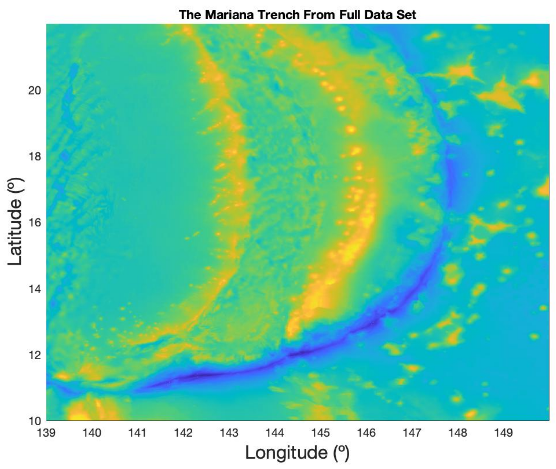
The Mariana Trench, lying in the Pacific Ocean between Japan and Papua New Guinea, is the deepest trench in the world. Because of its steep sides and vast depth, the trench provides a unique environment for scientific study. In this project, we will use bathymetric data provided by the United States National Oceanic and Atmospheric Administration (NOAA) to conduct an investigation of the trench. In addition, because this data is often difficult to work with, containing thousands of values, we will explore a means to reduce the size of the data while simultaneously maintaining the structure of the trench. This will allow for further, more computationally intensive investigation.

2 Initial Investigation

On initial investigation of the data, one finds that the Mariana trench is represented by three matrices. The matrices represent the depth (size 1320×1440), longitude (size 1320×1), and latitude (size 1440×1) of the data recorded. In the future we will refer to the depth matrix as $\mathbf{A}$. Plots of the data can be seen in Figure 1



(a) 3D Plot of $\mathbf{A}$



(b) 2D Plot of $\mathbf{A}$

Figure 1: Plots of Mariana Trench Depth, $\mathbf{A}$, in terms of latitude and longitude

Exploring this data, we find that the deepest point in the trench is 10.93 km below sea level at a latitude of 11.333° and a longitude of 142.2°. See Section 8.2 for the code used. In addition, nominally defining the ocean floor to have a depth of 6 kilometers, we can find the average depth of the trench by finding the mean of the values below 6 kilometers in $\mathbf{A}$. Thus, we find the average depth of the Mariana Trench to be 7.205 km. See Section 8.2 for the computation used.

3 Singular Value Decomposition

At this point, conducting further investigation would be impractical as the size of $\mathbf{A}$ is too large to quickly compute with. Thus, it is necessary to find a way to reduce the size of the data while preserving the structure of the trench. To simplify the data, we must first understand the concept of Singular Value Decomposition (SVD).

SVD is a factorization of a matrix that breaks up a matrix into three separate matrices such that $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices and $\mathbf{\Sigma}$ is a diagonal matrix. A pictorial representation of this can be seen in Figure 2

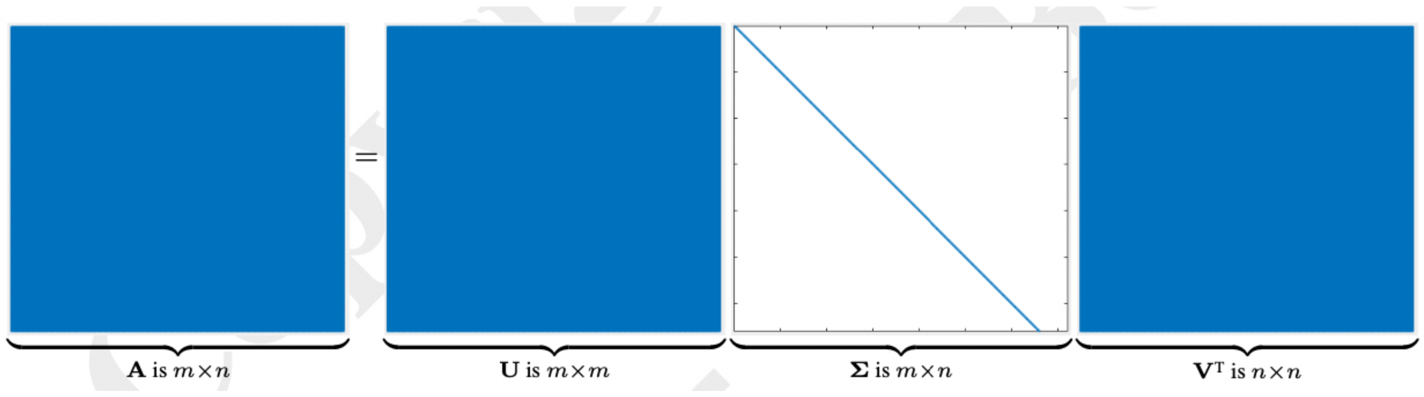

=
$\mathbf{A}$ is $m\times n$
$\mathbf{U}$ is $m\times m$
$\mathbf{\Sigma}$ is $m\times n$
$\mathbf{V}^{\mathrm{T}}$ is $n\times n$

Figure 2: SVD Representation of $\mathbf{A}$

As with standard factorization, each of the three matrices tell you information about the parent matrix $\mathbf{A}$. $\mathbf{\Sigma}$ contains the singular values which are the square roots of the eigenvalues of $\mathbf{A}^T\mathbf{A}$ and $\mathbf{V}$ represents the associated eigenvectors of those eigenvalues. $\mathbf{U}$ contains the eigenvectors of $\mathbf{A}\mathbf{A}^T$. Physically interpreted, the columns of $\mathbf{V}$ represent a basis of $\mathbb{R}^n$ and the columns of $\mathbf{U}$ represent a basis of $\mathbb{R}^m$. Written mathematically, this means:

$$\begin{aligned}\mathbf{A}\,\vec{\mathbf{x}} &= \mathbf{A}(c_1\vec{\mathbf{V}}_1 + c_2\vec{\mathbf{V}}_2 + \cdots + c_n\vec{\mathbf{V}}_n) \\ &= c_1\,\mathbf{\Sigma}_{11}\,\vec{\mathbf{U}}_1 + c_2\,\mathbf{\Sigma}_{22}\,\vec{\mathbf{U}}_2 + \cdots + c_n\,\mathbf{\Sigma}_{mm}\,\vec{\mathbf{U}}_m\end{aligned}$$

4 Incomplete Singular Value Decomposition

While SVD is useful for a deeper understanding of the parent matrix, it still does not reduce the number of values needed to represent the parent matrix. In fact, as one can see from Figure 2 the SVD matrices will actually require more values than the parent matrix. However, if we order the singular values and vectors such that $\mathbf{\Sigma}_{11} \geq \mathbf{\Sigma}_{22} \geq \cdots \geq \mathbf{\Sigma}_{mm}$, then for some $k < m$ we can make the approximation:

$$\begin{aligned}\mathbf{A}\,\vec{\mathbf{x}} &= c_1\,\mathbf{\Sigma}_{11}\,\vec{\mathbf{U}}_1 + c_2\,\mathbf{\Sigma}_{22}\,\vec{\mathbf{U}}_2 + \cdots + c_n\,\mathbf{\Sigma}_{mm}\,\vec{\mathbf{U}}_m \\ &\approx c_1\,\mathbf{\Sigma}_{11}\,\vec{\mathbf{U}}_1 + c_2\,\mathbf{\Sigma}_{22}\,\vec{\mathbf{U}}_2 + \cdots + c_n\,\mathbf{\Sigma}_{kk}\,\vec{\mathbf{U}}_k && \text{(where } k < m\text{)}\end{aligned}$$

This is to say that we are cutting off the action of matrix $\mathbf{A}$ a little early, but, because of the ordering, we are losing as little information as possible. This is called the Incomplete Singular Value Decomposition (ISVD), where we preserve essence of the completed SVD, but it assumes the large elements of $\mathbf{\Sigma}$ and the corresponding columns in $\mathbf{U}$ and $\mathbf{V}$ matter most, throwing away the columns that are less useful. This will thus allow us to reduce the values need to represent $\mathbf{A}$. This is shown pictorially in Figure 3.

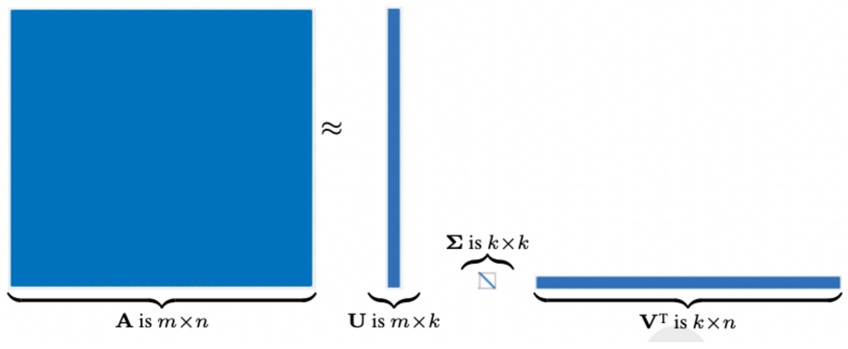

≈
$\mathbf{\Sigma}$ is $k \times k$
$\mathbf{A}$ is $m \times n$
$\mathbf{U}$ is $m \times k$
$\mathbf{V}^\mathrm{T}$ is $k \times n$

Figure 3: ISVD Representation of $\mathbf{A}$

5 Computing the First Eigenvalue and Eigenvector of ATA

Having described the mathematical foundation for ISVD, we will now explore its programmatic implementation. First, we need a method to find an eigenvalue from a matrix. To find the first eigenvector, and in turn the first eigenvalue of the depth matrix $\mathbf{A}$, we can use the following algorithm:

1. Guess a random unit vector with size corresponding to $\mathbf{A}$ (in the case of the Mariana Trench 1440).
2. Multiply $\mathbf{A}^T\,\mathbf{A}$ by the guess vector and divide the result by its magnitude to define the next unit vector.
3. Use the newly defined unit vector as the updated guess.
4. Repeat steps 3 and 4 until the vector stops changing, this is the first eigenvector $\vec{\mathbf{V}}_1$.

Defined more mathematically, this algorithm is represented as:

1. $\vec{\mathbf{u}}_1$ = a random unit vector with the correct number of components.
2. $\vec{\mathbf{u}}_{n+1} := \frac{\mathbf{A}^T\,\mathbf{A}\,\vec{\mathbf{u}}_n}{\|\mathbf{A}^T\,\mathbf{A}\,\vec{\mathbf{u}}_n\|}$
3. Repeat using $\vec{\mathbf{u}}_{n+1}$ as the guess until $\|\vec{\mathbf{u}}_{n+1} - \vec{\mathbf{u}}_n\| <$ acceptable precision

Note that $\|\vec{\mathbf{u}}\|$ is the magnitude of the vector $\vec{\mathbf{u}}$ that has N components. Thus, it is defined as:

$$\|\vec{\mathbf{u}}\| = \sqrt{\sum\ {}_{i=1}^{N} u_i^2}$$

Further exploration of why this algorithm works is necessary in a future project. However, a hypothesis as to why this algorithm works could be that we know that the vector $\vec{\mathbf{u}}_n$ can be expressed using the basis of the eigenvectors from $\mathbf{A}^T\,\mathbf{A}$, meaning that:

$$\begin{aligned}\mathbf{A}^T\,\mathbf{A}\,\vec{\mathbf{u}}_n &= \vec{\mathbf{u}}_n(\lambda_1\vec{\mathbf{V}}_\mathbf{1}\vec{\mathbf{V}}_\mathbf{1}^T + \lambda_2\vec{\mathbf{V}}_\mathbf{2}\vec{\mathbf{V}}_\mathbf{2}^T + \ldots + \lambda_n\vec{\mathbf{V}}_\mathbf{n}\vec{\mathbf{V}}_\mathbf{n}^T) \\ &= \vec{\mathbf{u}}_n\lambda_1(\vec{\mathbf{V}}_\mathbf{1}\vec{\mathbf{V}}_\mathbf{1}^T + \frac{\lambda_2}{\lambda_1}\vec{\mathbf{V}}_\mathbf{2}\vec{\mathbf{V}}_\mathbf{2}^T + \ldots + \frac{\lambda_n}{\lambda_1}\vec{\mathbf{V}}_\mathbf{n}\vec{\mathbf{V}}_\mathbf{n}^T) && \text{(factoring out } \lambda_1)\end{aligned}$$

Note that since we are computing the largest eigenvalue, we are assuming that $\lambda_1 >> \lambda_{2...n}$. Then, we normalize the vector so that it has magnitude 1, and perform the multiplication. We repeat this over, and over, lets say we repeat it g times. Then, we can say that:

$$\begin{aligned}
\mathbf{A}^T\,\mathbf{A}\,\vec{\mathbf{u}}_n &= \vec{\mathbf{u}}_n\lambda_1^g(\vec{\mathbf{V}}_\mathbf{1}\vec{\mathbf{V}}_\mathbf{1}^{\,T} + (\frac{\lambda_2}{\lambda_1})^g\vec{\mathbf{V}}_\mathbf{2}\vec{\mathbf{V}}_\mathbf{2}^{\,T} + ... + (\frac{\lambda_n}{\lambda_1})^g\vec{\mathbf{V}}_\mathbf{n}\vec{\mathbf{V}}_\mathbf{n}^{\,T}) \\
&= \vec{\mathbf{u}}_n\lambda_1^g\vec{\mathbf{V}}_\mathbf{1}\vec{\mathbf{V}}_\mathbf{1}^{\,T} && \text{(assuming that } \lambda_1 >> \text{all other eigenvalues)} \\
&= \lambda_1^g\vec{\mathbf{V}}_\mathbf{1}\vec{\mathbf{V}}_\mathbf{1}^{\,T} && (\vec{\mathbf{u}} \text{ is a unit vector, so it is negligible)} \\
&= \lambda_1^g
\end{aligned}$$

In other words, as we continue to perform the multiplication and normalization, the eigenvalue ratios go closer and closer to 0, since we are under the assumption that the first eigenvalue is the largest. Then, after enough multiplications, we can estimate those ratios to be zero, leaving a term that can be manipulated to provide the eigenvalue.

Having thus described the algorithm for finding the largest eigenvalue, we can now implement it on our data. Using this algorithm in Matlab (See Section 8.2), we find that for the depth matrix $\mathbf{A}$, the first eigenvector, denoted $\vec{\mathbf{V}}_1$, has an eigenvalue of $3.8802 * 10^{13}$. In addition, we can inspect the components by looking at the plot of $\vec{\mathbf{V}}_1$ as show in Figure 4.

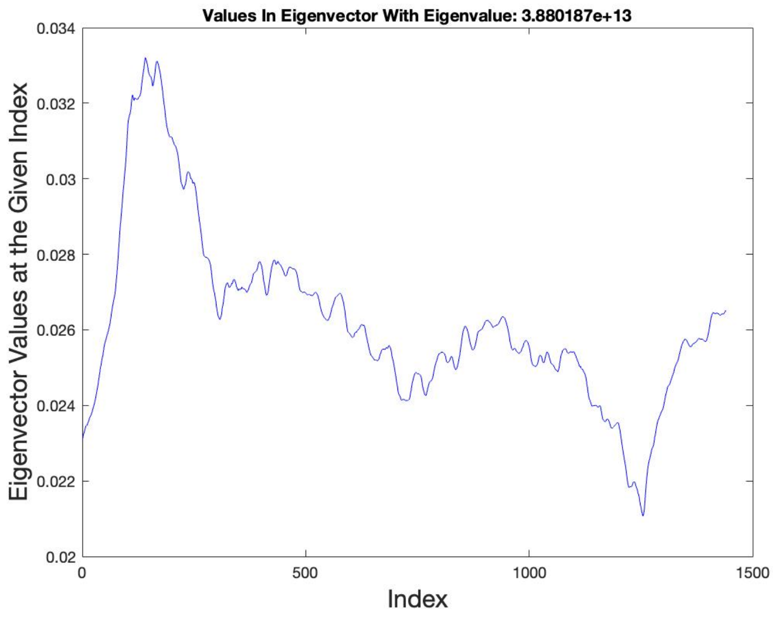


Figure 4: Plot of the Components of $\vec{\mathbf{V}}_1$

6 Computing the 50 Largest Eigenvalues of ATA

Having now found a way to compute an eigenvalue given a matrix, we need a way to calculate the i largest eigenvalues and and associated eigenvectors in order to perform ISVD. To do this, we can use the fact that every eigenvector in a symmetric matrix, for example $\mathbf{A}^T\,\mathbf{A}$, must be orthogonal to every other eigenvector. Thus, we can modify our algorithm for finding an eigenvalue from Section 5 to ensure that the vector that we are currently constructing is orthogonal to every eigenvector we have already constructed. In other words, if we are in the midst of computing eigenvector $\vec{\mathbf{V}}_i \approx \vec{\mathbf{u}}_n$ then we want to ensure that $\vec{\mathbf{u}}_{n+1}^T \vec{\mathbf{V}}_1 = \vec{\mathbf{u}}_{n+1}^T \vec{\mathbf{V}}_2 = \vec{\mathbf{u}}_{n+1}^T \vec{\mathbf{V}}_{i-1} = 0$.

To do this, we can utilize *Gram-Schmidt Orthogonalization.* Specifically, from a next guess vector $\vec{\mathbf{u}}^*_{n+1} = \mathbf{A}^T \mathbf{A}\, \vec{\mathbf{u}}_n$ having previously computed r eigenvectors $\vec{\mathbf{V}}_1 \ldots \vec{\mathbf{V}}_r$, we can "orthogonalize" the vector by computing:

$$\vec{\mathbf{u}}_{n+1} = \vec{\mathbf{u}}^*_{n+1} - \sum_{j=1}^{r-1} \left(\vec{\mathbf{u}}^{*T}_{n+1} \vec{\mathbf{V}}_{\mathbf{j}} \right) \vec{\mathbf{V}}_{\mathbf{j}}$$

Relying on these ideas, we can now alter our algorithm from Section 5 to compute the $i = 50$ largest eigenvalues and associated eigenvectors of $\mathbf{A}^T \mathbf{A}$. This creates the following algorithm to assign the eigenvectors to the matrix $\mathbf{V}$:

1. Initialize $\vec{\mathbf{V}}_1$ to $\vec{\mathbf{V}}_{50}$ as a matrix of zeroes (in this case of the Mariana Trench, this matrix will have size 1440×50))

2. for $i = 1$ to 50

   (a) $\vec{\mathbf{u}}_1$ = a random unit vector with the correct number of components.

   (b) $\vec{\mathbf{u}}^*_{n+1} := \mathbf{A}^T \mathbf{A}\, \vec{\mathbf{u}}_n$

   (c) $\vec{\mathbf{u}}_{n+1} := \vec{\mathbf{u}}^*_{n+1} - \sum_{j=1}^{i-1} \left(\vec{\mathbf{u}}^{*T}_{n+1} \vec{\mathbf{V}}_{\mathbf{j}} \right) \vec{\mathbf{V}}_{\mathbf{j}}$

   (d) $\vec{\mathbf{u}}_{n+1} := \frac{\vec{\mathbf{u}}_{n+1}}{\|\vec{\mathbf{u}}_{n+1}\|}$

   (e) Repeat b-d until $\|\vec{\mathbf{u}}_{n+1} - \vec{\mathbf{u}}_n\| <$ acceptable precision

   (f) Store the final $\vec{\mathbf{u}}_{n+1}$ as $\vec{\mathbf{V}}_i$

Converting this algorithm into Matlab, we are now able to compute the 50 largest eigenvalues, via their eigenvectors, of $\mathbf{A}^T \mathbf{A}$. Again, $\mathbf{A}$ represents the depth data for the Mariana Trench. See Section 8.2 for the code used for the calculation. Having calculated these eigenvalues, with the corresponding eigenvectors stored as $\mathbf{V}$, we can now create a semilog plot of them, as shown in Figure 5.

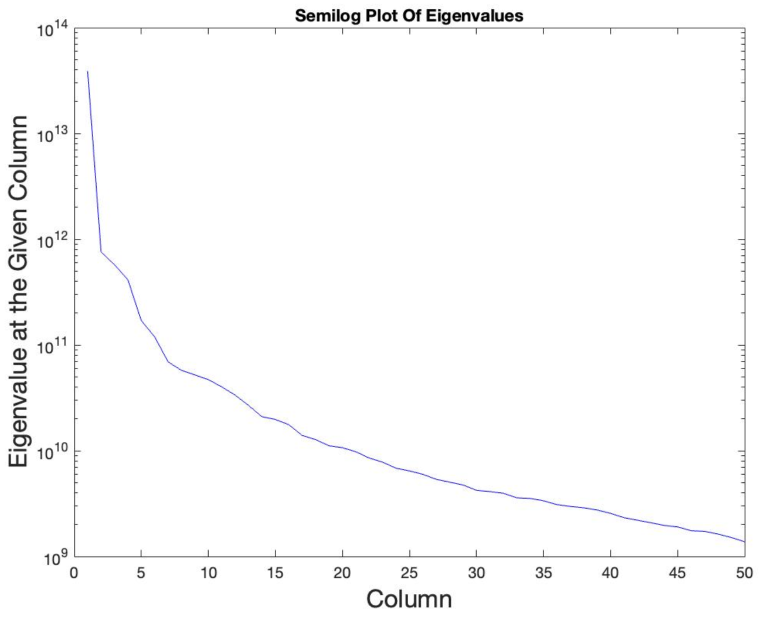


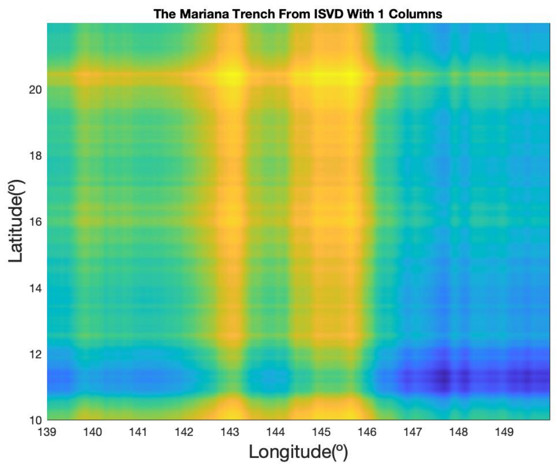
Figure 5: Semilog Plot of the 50 Largest Eigenvalues of $\mathbf{A}^T\,\mathbf{A}$
Note: Eigenvalues from Columns of $\mathbf{V}$

7 Conclusion

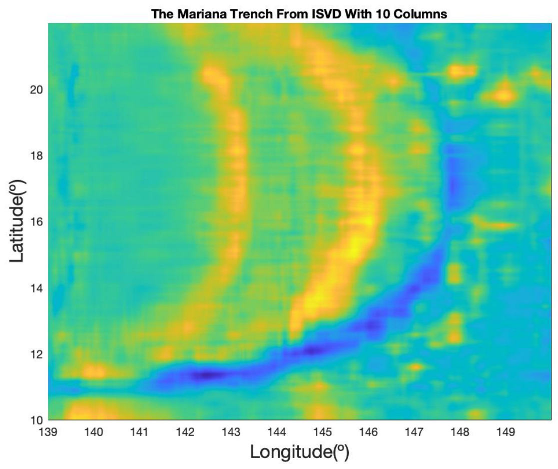
In this project, we analyzed a model of the Mariana Trench, with data from the NOAA. We first did a basic investigation wherein we graphed a representation of the depth data (matrix $\mathbf{A}$), found the deepest point in the trench, and found the average depth of the trench. Then, we described the process of Singular Value Decomposition, and how Incomplete Singular Value Decomposition could be used to reduce the memory needed to represent a data set. Thereafter, we described a method to find the largest eigenvalue in a matrix, and used that algorithm to find the largest eigenvalue in $\mathbf{A}^T\,\mathbf{A}$. Next, we modified that method to find that largest i eigenvalues of a matrix (where i is an arbitrary constant) and used the resulting algorithm to find the largest 50 eigenvalues of $\mathbf{A}^T\,\mathbf{A}$. Afterwards, we explained how to use these eigenvalues to construct the ISVD of a matrix, and used this method to generate numerous ISVD representations of the Mariana Trench, using different numbers of eigenvalues. Then, we quantified how much fewer values are needed to represent the depth of the Mariana Trench using ISVD. Penultimately, we demonstrated that using 50 eigenvalues, the ISVD representation of the Mariana Trench is very close to the real data, meaning further analysis can be conducted on the ISVD representation without significantly affecting the quality of results. Finally, we showed how the fidelity of the ISVD representation increases as the number of eigenvalues used increases, in turn causing the total number of values used to increase.

8 Appendix

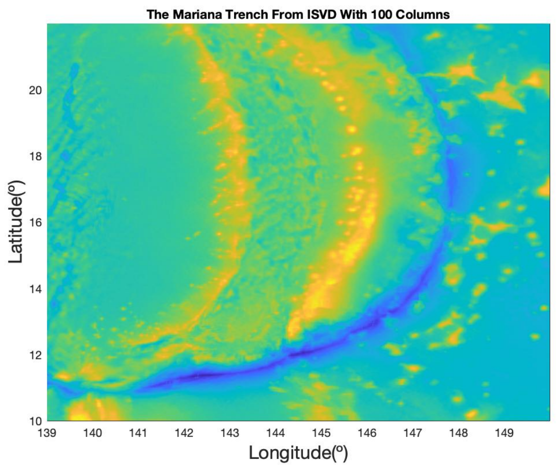
8.1 2D Plots of ISVD with Varying Number of Eigenvalue



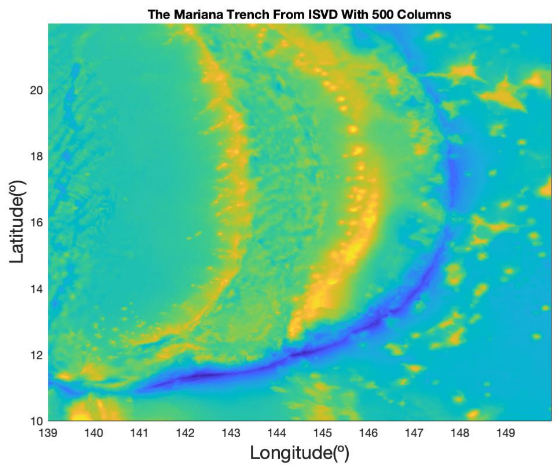
(a) Using 1 Eigenvalue



(b) Using 10 Eigenvalues



(c) Using 100 Eigenvalues



(d) Using 500 Eigenvalues

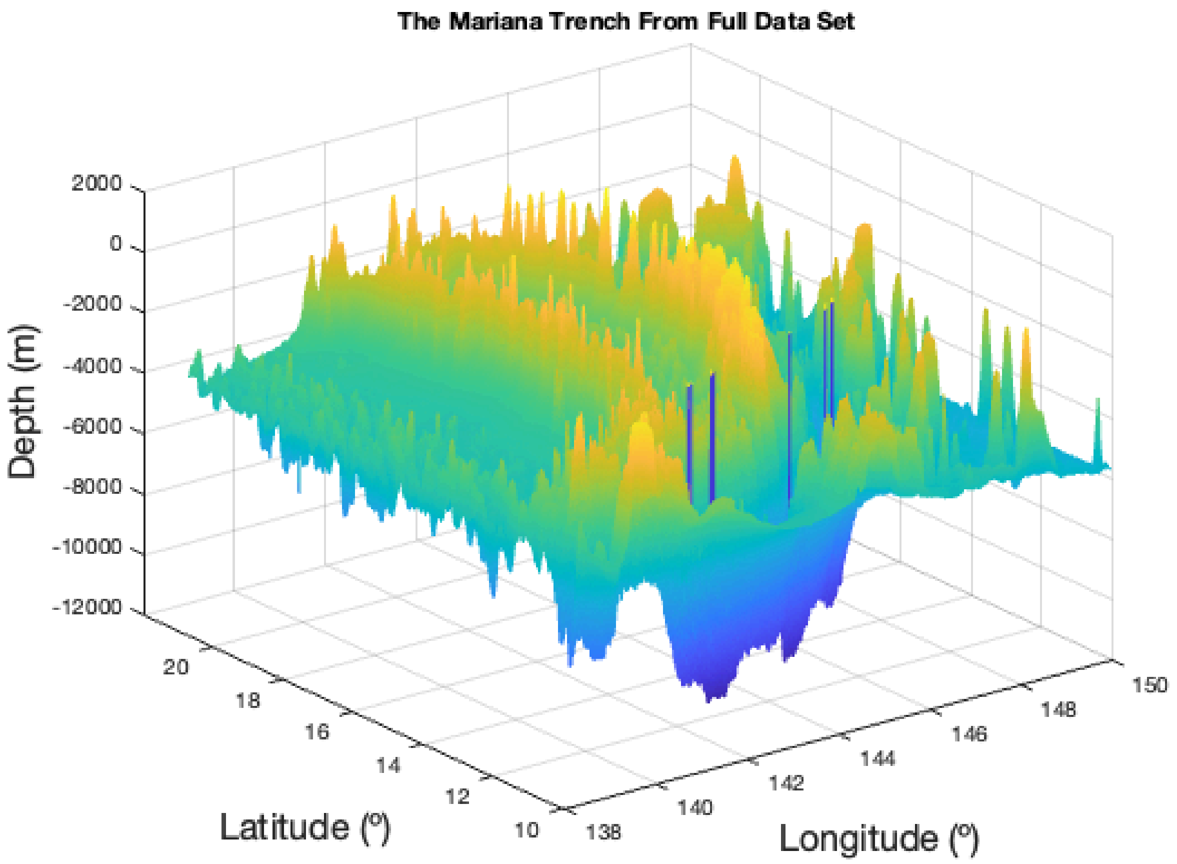
Figure 6: 2D Plots of ISVDs Using Different Numbers of Eigenvalues for Comparison

8.2 Code With Output for Reference

```
A = importdata('mariana_depth.csv');
lon = importdata('mariana_longitude.csv');
lat = importdata('mariana_latitude.csv');

%2.1.1
figure(4)
grid = meshgrid(lat,lon);
mesh(lon,lat,A',A'); %creates 3d rendering of mariana trench
%s = pcolor(lon,lat,A'); %this and the next line create the 2d color mapping
 of the mariana's trench
%set(s, 'EdgeColor', 'none');
xlabel('Longitude (º)','FontSize',16);
ylabel('Latitude (º)','FontSize',16);
zlabel('Depth (m)','FontSize',16);
title('The Mariana Trench From Full Data Set')

%%2.1.2 find the deepest point (lat, long, depth)

maxDepthArr1 = min(A);  %finds minimum in each column
maxDepth= min(maxDepthArr1); %-10930  finds minimum

latIndexOfMaxD = find(maxDepthArr1==maxDepth); %161
lonIndexOfMaxD = find( A(:,latIndexOfMaxD)==maxDepth);%385

latOfMaxDepth = lat(latIndexOfMaxD,1); %finds latitude of deepest point:
 11.333000
lonOfMaxDepth = lon(lonIndexOfMaxD,1); %finds longitude of deepest point:
 142.200000

fprintf('OUTPUT: \n',latOfMaxDepth,lonOfMaxDepth,maxDepth/1000);%Lat:
 11.333000 , Long: 142.200000 , Depth: -10930
fprintf('MAX DEPTH: Lat: %f , Long: %f , Depth : %f km
 \n',latOfMaxDepth,lonOfMaxDepth,maxDepth/1000);%Lat: 11.333000 , Long:
 142.200000 , Depth: -10930


%%2.1.3 what is the average of all points deeper than 6km

count=0; %used to count the number of points below 6km
currentTotal=0; %used to keep tract of the sum of the depths below 6km
numRows=1320;
numCol = 1440;%numRows adn numCol are used as upperbounds and are based on the
 dimensions of A

for i = 1:numRows
   for j = 1:numCol
      if(A(i,j)<-6000) %if the point is deeper than 6km, then update our
 variables that keep track
          count = count+1;
          currentTotal = currentTotal+A(i,j);
      end
   end
```

```
end
averageDepthUnder6km = currentTotal/count; %arithmetic mean
fprintf('Average Depth of the trench: %f (km)
 \n',averageDepthUnder6km/1000);%Average Depth of the trench: -7204.636665
```

```
OUTPUT:
MAX DEPTH: Lat: 11.333000 , Long: 142.200000 , Depth : -10.930000 km
Average Depth of the trench: -7.204637 (km)
```

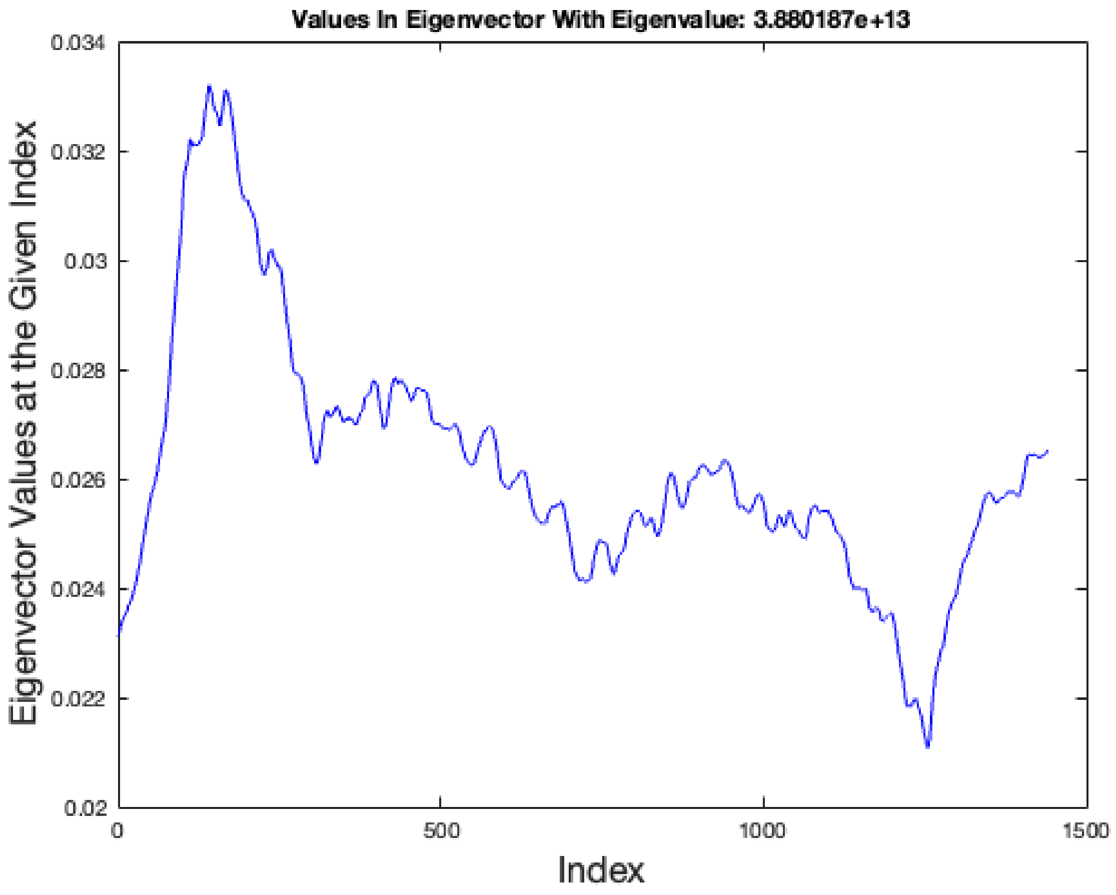


Published with MATLAB® R2021b

```
A = importdata('mariana_depth.csv');
lon = importdata('mariana_longitude.csv');
lat = importdata('mariana_latitude.csv');

%numRows = 1320;
%numCols = 1440;
%A = zeros(numRows,numCols);
%for i = 1:numRows
%   for j = 1:numCols
%      A(i,j)=randi(10,1);%first num in randi is random int generator
%   end
%end

fprintf('OUTPUT: \n');

%%2.2.1

A_t=transpose(A);
B= A_t*A;

[eVal1,v1] = findEigens(B);

figure(1);
plot(1:sizeColVect(B),v1,'blue');%Plots Values In Eigenvector
title(sprintf('Values In Eigenvector With Eigenvalue: %i', eVal1));
xlabel('Index','FontSize',16);
ylabel('Eigenvector Values at the Given Index','FontSize',16);


function [value, vector]=findEigens(matrix)%finds eigenvector and associated
 value by using the first method
    randUpBound = 10;

    u=randomUnitColVector1(sizeColVect(matrix),randUpBound);%generates a
 random vector to start with
    u1= matrix*u;
    u1=unitVect(u1);

    smallNum=1e-15;
    whileCounter =0;
    while(mag(u1-u)>smallNum)%does the same as above but does it until the
 unit vector becomes unchanging and therefore must be an eigenvector
        whileCounter=whileCounter+1;
        u=u1;
        u1= matrix*u;
        u1=unitVect(u1);
    end

    vector= u1;
    scaledV1 = matrix*vector;
    value = scaledV1(1,1)/vector(1,1);
    fprintf('whileCounter: %i \n',whileCounter);%
```

```
end

function sizeVect = sizeColVect(colVect)
    b=size(colVect);
    sizeVect=b(1);
end

function vect = randomUnitColVector1(size,randomUpperBound)
    vect = zeros(size,1);
    for k = 1:size
          vect(k,1)=randi(randomUpperBound,1);%first num in randi is random
 int generator
    end
    vect=unitVect(vect);
end

function vect = randomUnitColVector2(size)
    vect=randomUnitColVector1(size,10);
end

function unitVector = unitVect(array)
    unitVector = array/mag(array);
end

function magnitude = mag(array)
    magnitude = 0;
    for k = 1:sizeColVect(array)
      magnitude=magnitude+(array(k,1))^2;
    end
    magnitude= sqrt(magnitude);
end
```

OUTPUT:
whileCounter: 8



Published with MATLAB® R2021b

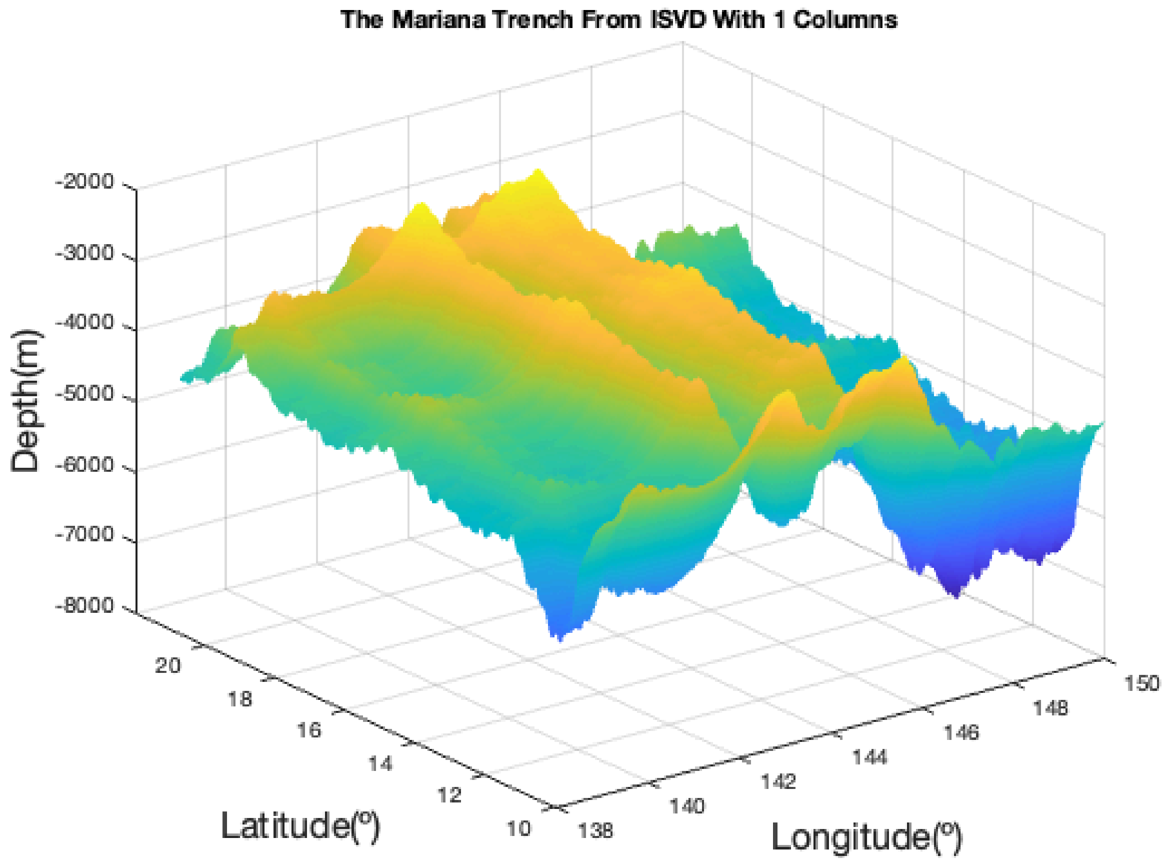
```
A = importdata('mariana_depth.csv');
lon = importdata('mariana_longitude.csv');
lat = importdata('mariana_latitude.csv');



%2.2.2
wantedEigenSize = 10;
fprintf('OUTPUT: \n');
numOfEignens =
 [1,10,50,100,500];%[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,22,24,26,28,30,32,
compressedRatings=zeros(1,size(numOfEignens,2));
accurateRatings=zeros(1,size(numOfEignens,2));
temp=zeros(1,size(numOfEignens,2));
for i = 1:size(numOfEignens,2)
    [accurateRating,compressedRating] = main(numOfEignens(i),A,lat,lon);
    fprintf('For ISVD With %i Columns: compression rating %f , accuracy rating
 %f (lower is better)\n',numOfEignens(i),compressedRating,accurateRating);
    compressedRatings(1,i)=compressedRating;
    accurateRatings(1,i)=accurateRating;
end

figure(100000);
plot(numOfEignens,compressedRatings,'red');
title(sprintf('Compression Rating for Different Number of Columns'));
xlabel('Number of Columns','FontSize',16);
ylabel('Compression Rating at the Given Number of Columns','FontSize',16);

figure(100001);
plot(numOfEignens,accurateRatings,'blue');
title(sprintf('Accuracy Rating for Different Number of Columns'));
xlabel('Number of Columns','FontSize',16);
ylabel('Accuracy Rating at the Given Number of Columns','FontSize',16);

function [accurateRating,compressedRating] = main(wantedEigenSize,A,lat,lon)
    A_t=transpose(A);
    B= A_t*A;

    [V,eigenValues]= betterEigen(B,wantedEigenSize);

    if(wantedEigenSize==50)
        figure(2);
        semilogy(1:wantedEigenSize,eigenValues,'blue');%plot of
        title(sprintf('Semilog Plot Of Eigenvalues'));
        xlabel('Column','FontSize',16);
        ylabel('Eigenvalue at the Given Column','FontSize',16);
    end
    %2.3.1
    sigma = zeros(wantedEigenSize,wantedEigenSize);
    U = zeros(size(A,1),wantedEigenSize);

    for i = 1:wantedEigenSize
```

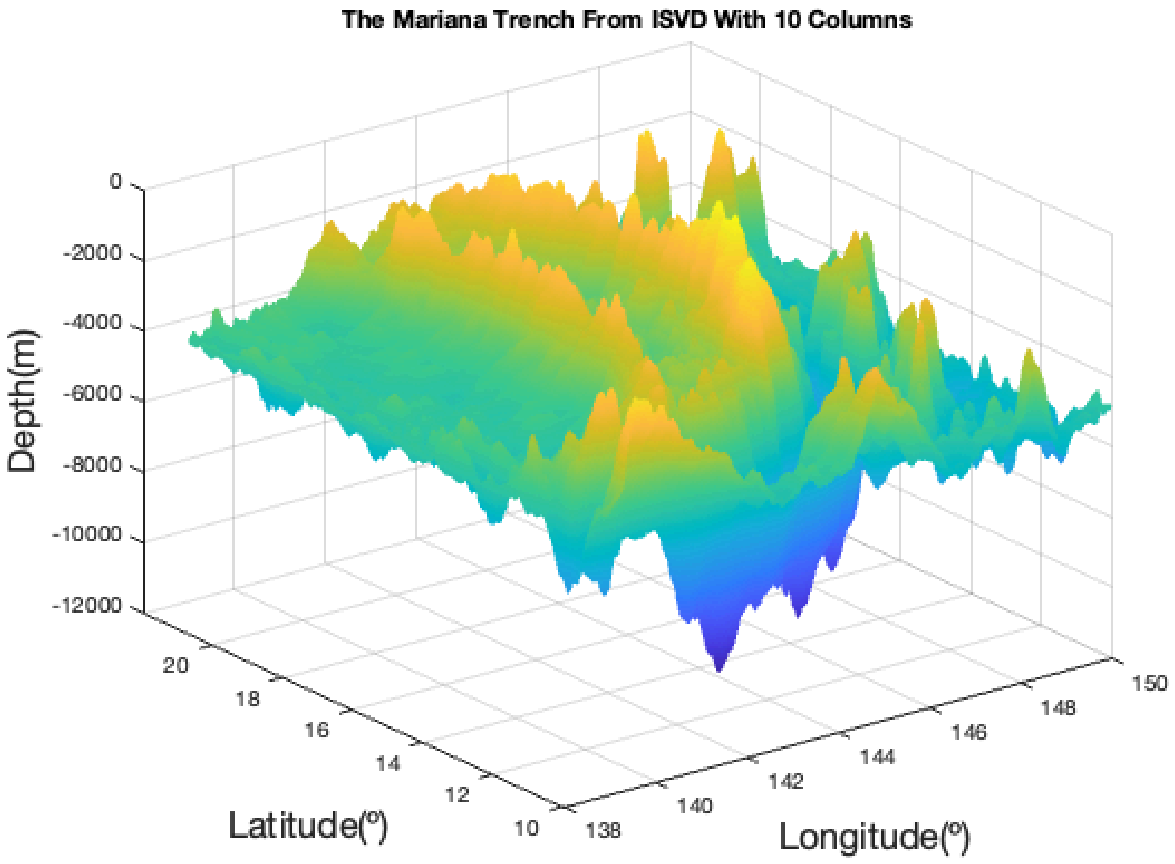
```
        sigma(i,i)=sqrt(eigenValues(1,i));
        U(:,i) = ( A*V(:,i) ) / sigma(i,i) ;
    end

    %2.3.3
    figure(wantedEigenSize)
    compressedA = U*sigma*transpose(V);
    grid = meshgrid(lat,lon);
    mesh(lon,lat,compressedA',compressedA');
    %s = pcolor(lon,lat,compressedA');
    %set(s, 'EdgeColor', 'none');
    xlabel('Longitude(°)','FontSize',16);
    ylabel('Latitude(°)','FontSize',16);
    zlabel('Depth(m)','FontSize',16);
    title(sprintf('The Mariana Trench From ISVD With %i
 Columns',wantedEigenSize));
    accurateRating = accuracyRating(A,compressedA);
    compressedRating = compressionRating(A, wantedEigenSize);

    %fprintf('compression rating : %f (lower is better), for an eigenspace of
 dimension: %i \n',accuracyRating(A,compressedA), wantedEigenSize);
    count=0; %used to count the number of points below 6km
    currentTotal=0; %used to keep tract of the sum of the depths below 6km
    for i = 1:size(compressedA,1)
       for j = 1:size(compressedA,2)
          if(compressedA(i,j)<-6000) %if the point is deeper than 6km, then
 update our variables that keep track
              count = count+1;
              currentTotal = currentTotal+compressedA(i,j);
          end
       end
    end
    averageDepthUnder6km = currentTotal/count; %arithmetic mean
    fprintf('Average Depth of the Trench For ISVD With %i Columns : %f (km)
 \n',wantedEigenSize,averageDepthUnder6km/1000);%Average Depth of the trench:
 -7204.636665

end

function total = compressionRating(A, numEigens)
    numInA = size(A,1)*size(A,2);
    total= (numEigens*(size(A,1)+numEigens+size(A,2)) )/numInA;
end

function total = accuracyRating(A,compressedA)
    total=0;
    for i = 1:size(A,1)
       for j = 1:size(A,2)
          total = total+ abs( A(i,j)-compressedA(i,j) );
       end
    end
    total=total/100000000;
end
```

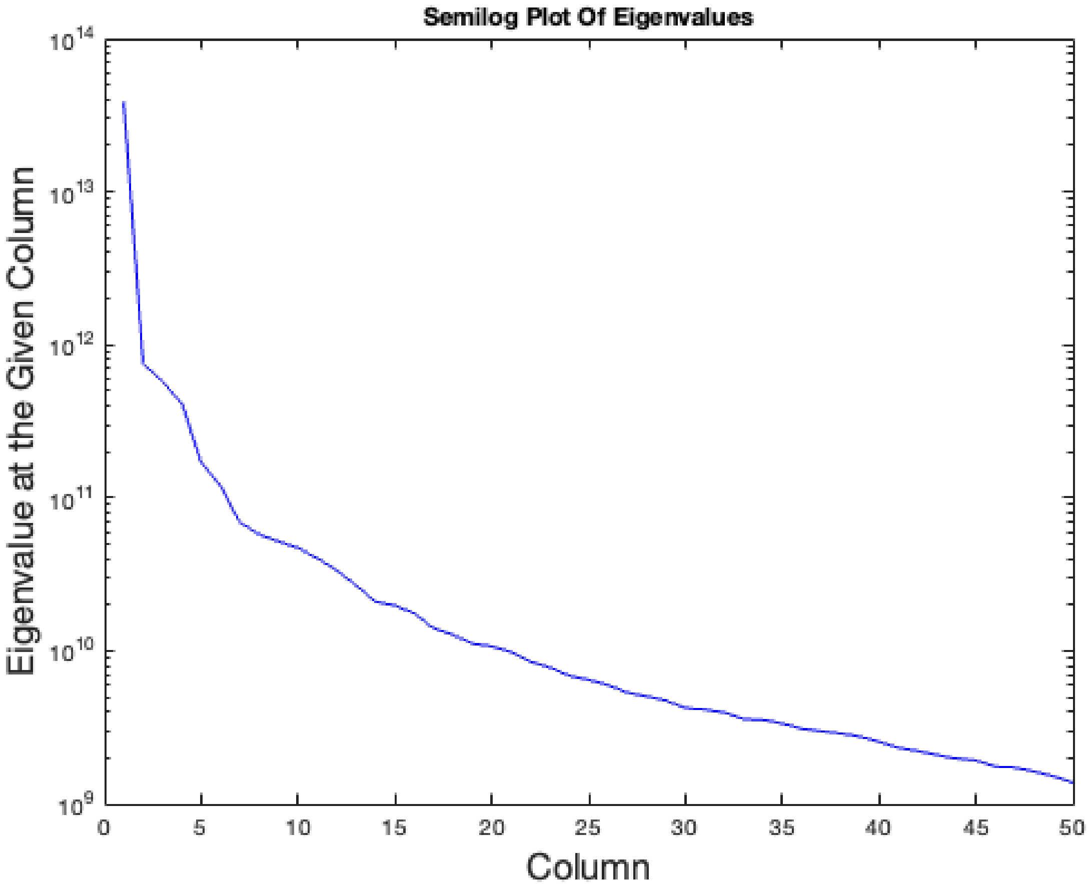
```
function [matrixOfVectors,matrixOfValues] = betterEigen(matrix,numOfEigens)
    %this method uses the second given algorithm to find the first
    %[numOfEigens] amt of eigenvectors adn their associated values.
    matrixOfVectors = zeros(sizeColVect(matrix),numOfEigens);
    matrixOfValues = zeros(1,numOfEigens); %row vectors
    for i = 1:numOfEigens
        u = randomUnitColVector2(sizeColVect(matrix)); %i


        u1star = matrix*u;%ii

        summationResult=0;
        for j = 1:(i-1)
            summationResult=summationResult
+(transpose(u1star)*matrixOfVectors(:,j))*matrixOfVectors(:,j);
        end

        u1=u1star-summationResult;%iii
        u1=unitVect(u1);%iv

        whileCount = 0;
        smallNumber = 1e-3;
        while(mag(u1-u)>smallNumber)
            whileCount=whileCount+1;
            u=u1;
            u1star = matrix*u;%ii

            summationResult=0;
            for j = 1:(i-1)
                summationResult=summationResult
+(transpose(u1star)*matrixOfVectors(:,j))*matrixOfVectors(:,j);
            end

            u1=u1star-summationResult;%iii
            u1=unitVect(u1);%iv
        end
        matrixOfVectors(:,i)=u1;

        %process of getting associated eigenvalue
        scaledV1 = matrix*u1;
        matrixOfValues(1,i) = scaledV1(1,1)/u1(1,1);


        %fprintf('whileCounter: %i \n',whileCount);%

    end
end

function sizeVect = sizeColVect(colVect)
    b=size(colVect);
    sizeVect=b(1);
end

function vect = randomUnitColVector1(size,randomUpperBound)
```

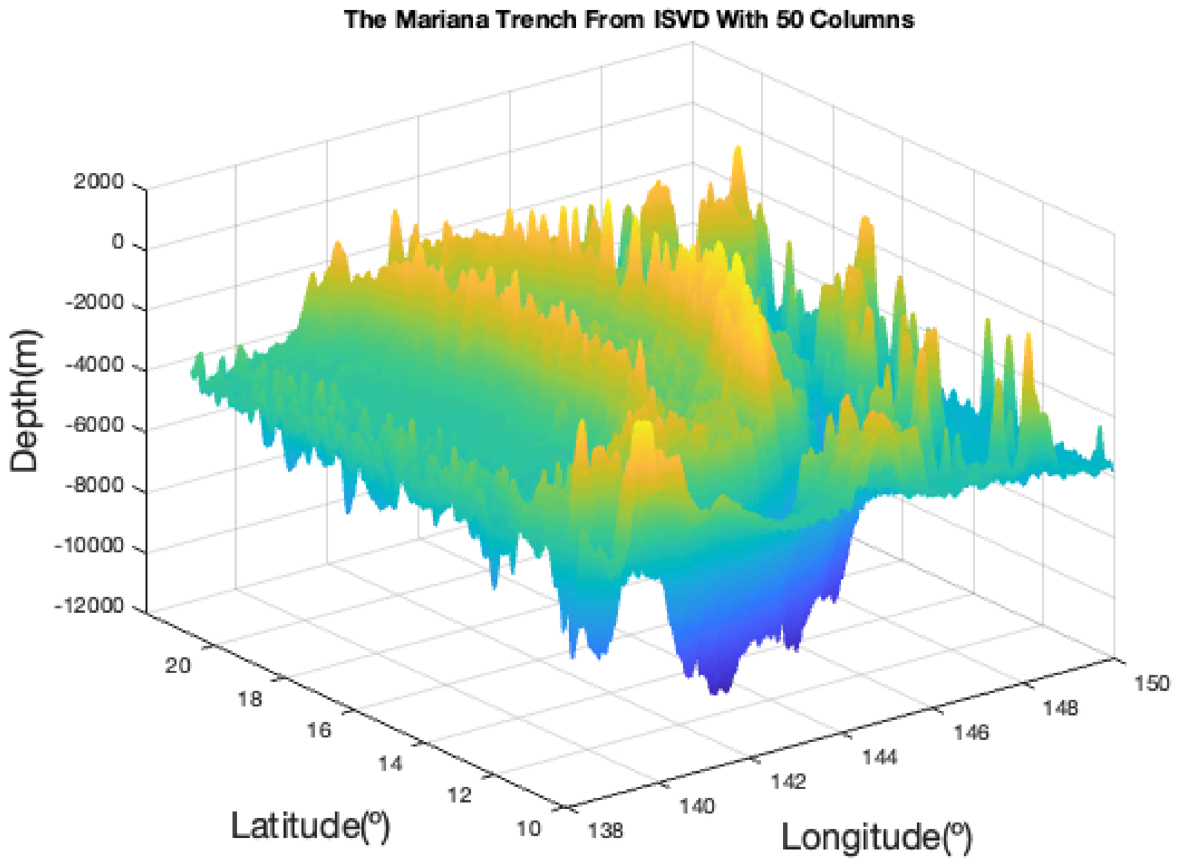
```
    vect = zeros(size,1);
    for k = 1:size
          vect(k,1)=randi(randomUpperBound,1);%first num in randi is random
 int generator
    end
    vect=unitVect(vect);
end

function vect = randomUnitColVector2(size)
    vect=randomUnitColVector1(size,10);
end

function unitVector = unitVect(array)
    unitVector = array/mag(array);
end

function magnitude = mag(array)
    magnitude = 0;
    for k = 1:sizeColVect(array)
      magnitude=magnitude+(array(k,1))^2;
    end
    magnitude= sqrt(magnitude);
end
```

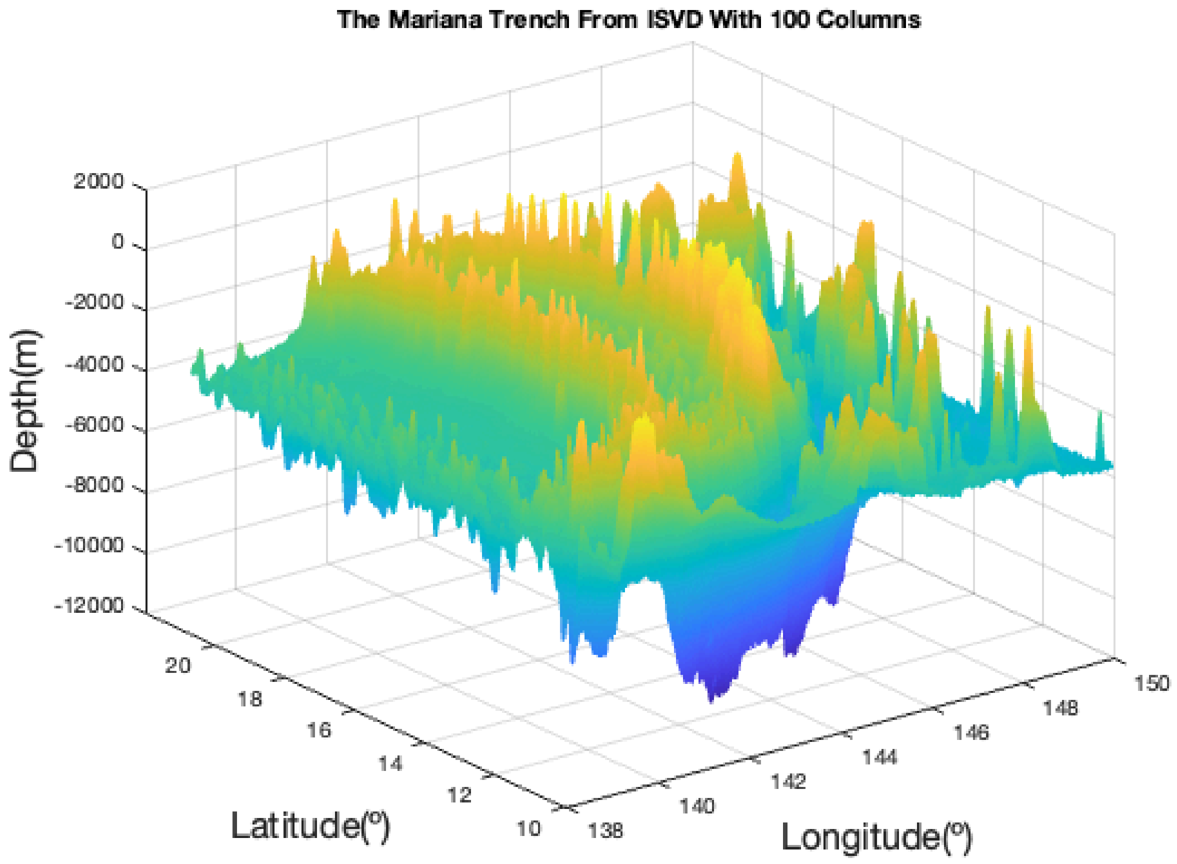
```
OUTPUT:
Average Depth of the Trench For ISVD With 1 Columns : -6.353018 (km)
For ISVD With 1 Columns: compression rating 0.001453 , accuracy rating
 15.437696 (lower is better)
Average Depth of the Trench For ISVD With 10 Columns : -7.048793 (km)
For ISVD With 10 Columns: compression rating 0.014573 , accuracy rating
 5.946841 (lower is better)
Average Depth of the Trench For ISVD With 50 Columns : -7.174009 (km)
For ISVD With 50 Columns: compression rating 0.073916 , accuracy rating
 1.869230 (lower is better)
Average Depth of the Trench For ISVD With 100 Columns : -7.196642 (km)
For ISVD With 100 Columns: compression rating 0.150463 , accuracy rating
 0.837695 (lower is better)
Average Depth of the Trench For ISVD With 500 Columns : -7.203606 (km)
For ISVD With 500 Columns: compression rating 0.857534 , accuracy rating
 0.134671 (lower is better)
```

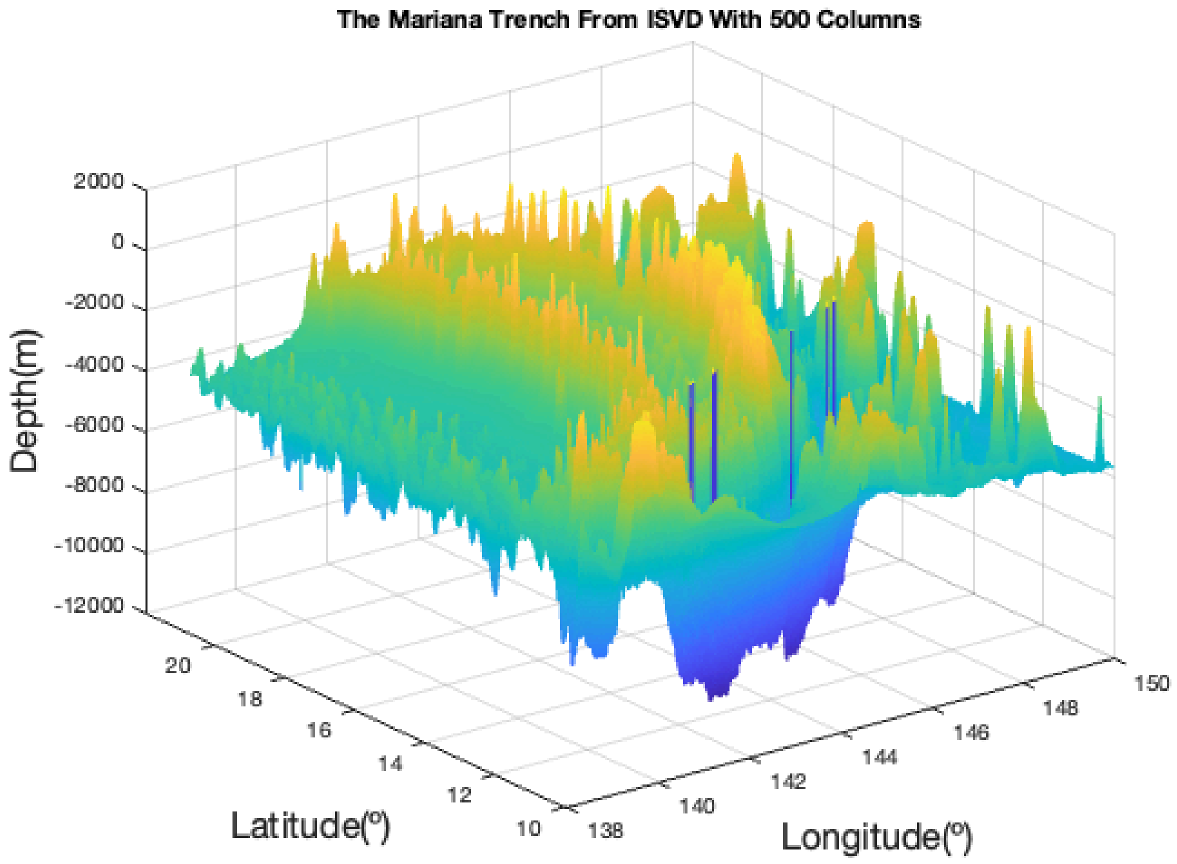

The Mariana Trench From ISVD With 1 Columns
Depth(m)
-2000
-3000
-4000
-5000
-6000
-7000
-8000
Latitude(º)
20
18
16
14
12
10
Longitude(º)
138
140
142
144
146
148
150

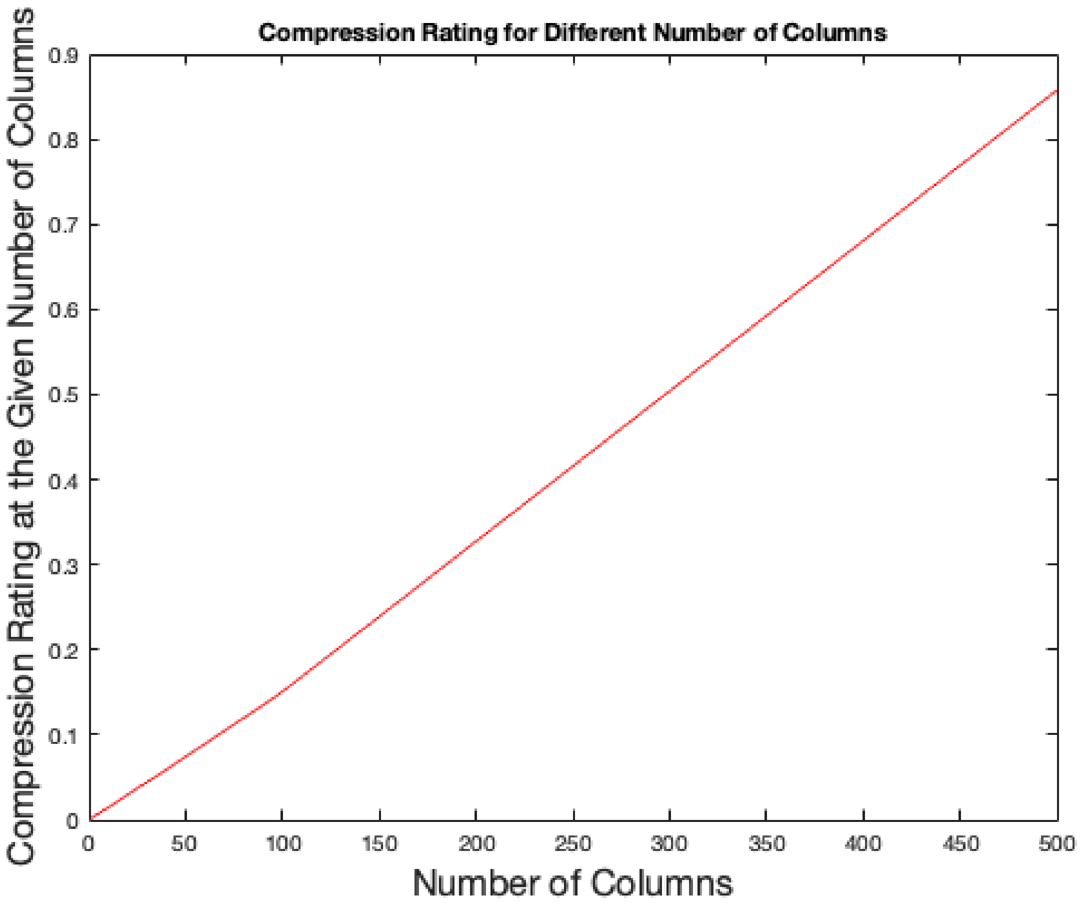

The Mariana Trench From ISVD With 10 Columns
Depth(m)
0
-2000
-4000
-6000
-8000
-10000
-12000
Latitude(º)
20
18
16
14
12
10
Longitude(º)
138
140
142
144
146
148
150


Semilog Plot Of Eigenvalues
10^{14}
10^{13}
10^{12}
10^{11}
10^{10}
10^{9}
Eigenvalue at the Given Column
0
5
10
15
20
25
30
35
40
45
50
Column

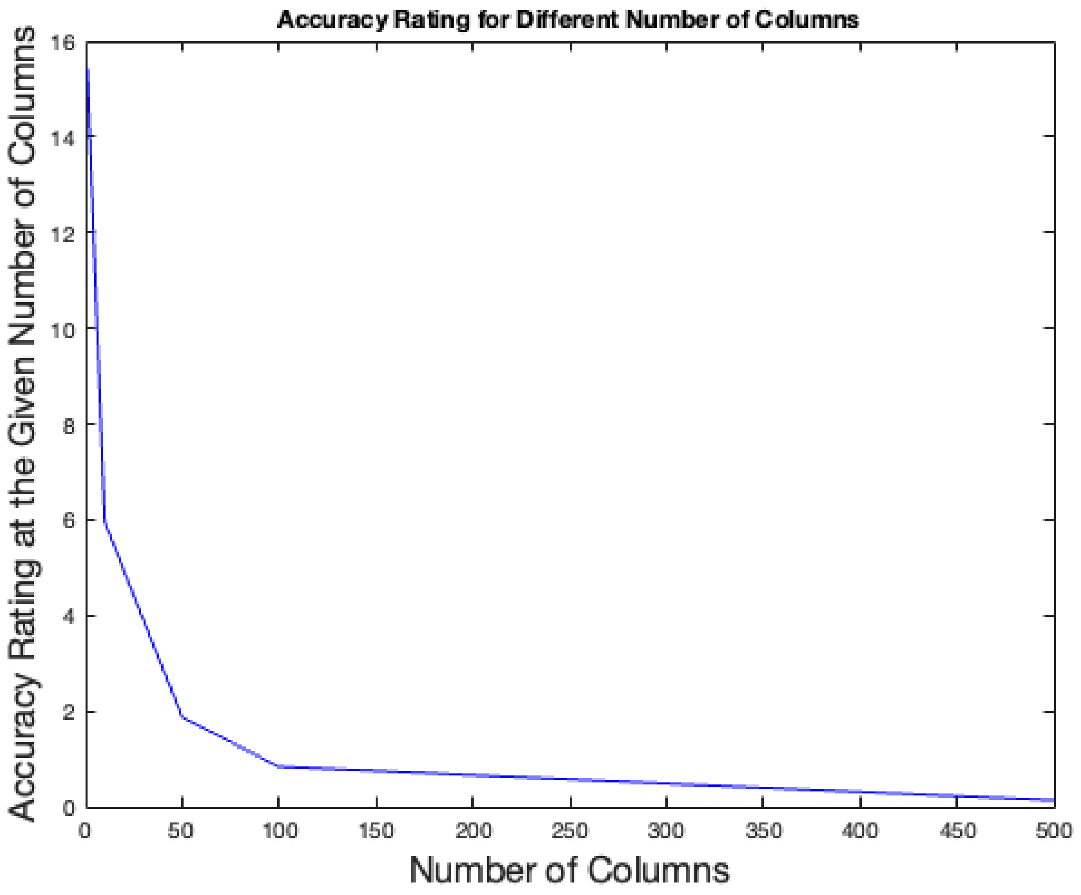

The Mariana Trench From ISVD With 50 Columns
2000
0
-2000
-4000
-6000
-8000
-10000
-12000
Depth(m)
20
18
16
14
12
10
Latitude(º)
138
140
142
144
146
148
150
Longitude(º)


The Mariana Trench From ISVD With 100 Columns
Depth(m)
2000
0
-2000
-4000
-6000
-8000
-10000
-12000
Latitude(º)
20
18
16
14
12
10
Longitude(º)
138
140
142
144
146
148
150


The Mariana Trench From ISVD With 500 Columns
Depth(m)
2000
0
-2000
-4000
-6000
-8000
-10000
-12000
Latitude(º)
20
18
16
14
12
10
Longitude(º)
138
140
142
144
146
148
150



Compression Rating for Different Number of Columns
Compression Rating at the Given Number of Columns
Number of Columns
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
50
100
150
200
250
300
350
400
450
500



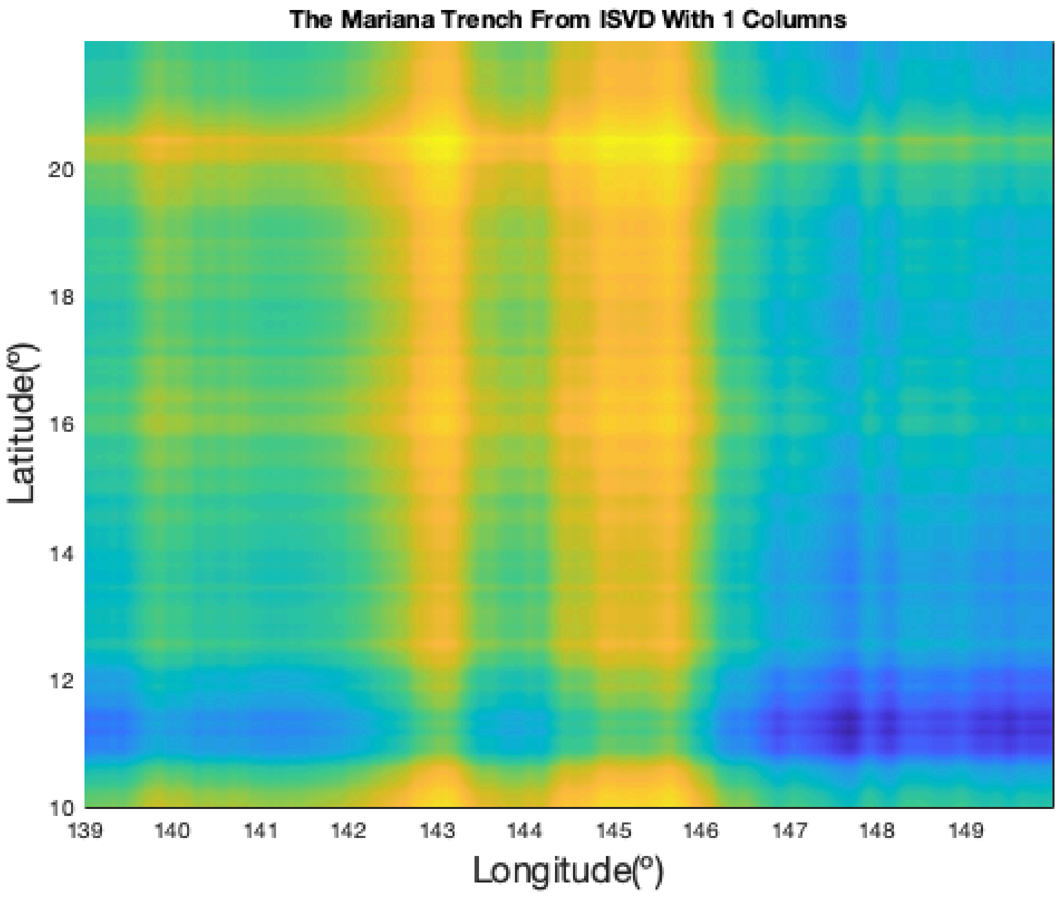
Published with MATLAB® R2021b

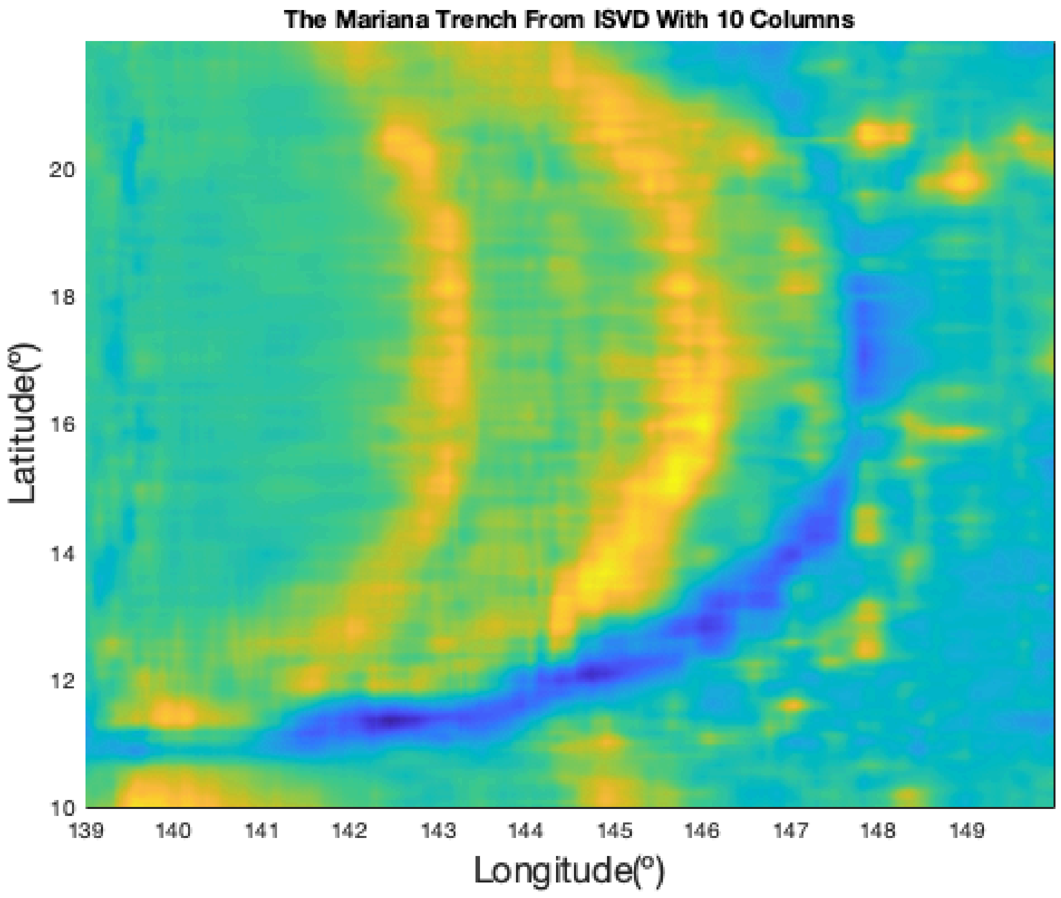
```
A = importdata('mariana_depth.csv');
lon = importdata('mariana_longitude.csv');
lat = importdata('mariana_latitude.csv');



%2.2.2
wantedEigenSize = 10;
fprintf('OUTPUT: \n');
numOfEignens =
 [1,10,50,100,500];%[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,22,24,26,28,30,32,
compressedRatings=zeros(1,size(numOfEignens,2));
accurateRatings=zeros(1,size(numOfEignens,2));
temp=zeros(1,size(numOfEignens,2));
for i = 1:size(numOfEignens,2)
    [accurateRating,compressedRating] = main(numOfEignens(i),A,lat,lon);
    fprintf('For ISVD With %i Columns: compression rating %f , accuracy rating
 %f (lower is better)\n',numOfEignens(i),compressedRating,accurateRating);
    compressedRatings(1,i)=compressedRating;
    accurateRatings(1,i)=accurateRating;
end

%figure(100000);
%plot(numOfEignens,compressedRatings,'red');
%title(sprintf('Compression Rating for Different Number of Columns'));
%xlabel('Number of Columns','FontSize',16);
%ylabel('Compression Rating at the Given Number of Columns','FontSize',16);

%figure(100001);
%plot(numOfEignens,accurateRatings,'blue');
%title(sprintf('Accuracy Rating for Different Number of Columns'));
%xlabel('Number of Columns','FontSize',16);
%ylabel('Accuracy Rating at the Given Number of Columns','FontSize',16);

function [accurateRating,compressedRating] = main(wantedEigenSize,A,lat,lon)
    A_t=transpose(A);
    B= A_t*A;

    [V,eigenValues]= betterEigen(B,wantedEigenSize);

    if(wantedEigenSize==50)
        figure(2);
        semilogy(1:wantedEigenSize,eigenValues,'blue');%plot of
        title(sprintf('Semilog Plot Of Eigenvalues From ISVD With %i
 Columns',wantedEigenSize));
        xlabel('Column','FontSize',16);
        ylabel('Eigenvalue at the Given Column','FontSize',16);
    end
    %2.3.1
    sigma = zeros(wantedEigenSize,wantedEigenSize);
    U = zeros(size(A,1),wantedEigenSize);
    for i = 1:wantedEigenSize
        sigma(i,i)=sqrt(eigenValues(1,i));
        U(:,i) = ( A*V(:,i) ) / sigma(i,i) ;
    end

    %2.3.3
    figure(wantedEigenSize)
    compressedA = U*sigma*transpose(V);
    %grid = meshgrid(lat,lon); %assist 3d plot
    %mesh(lon,lat,compressedA',compressedA'); %3d plot
    s = pcolor(lon,lat,compressedA');%2d plot
    set(s, 'EdgeColor', 'none');%assist 2d plot
    xlabel('Longitude(°)','FontSize',16);
    ylabel('Latitude(°)','FontSize',16);
    zlabel('Depth(m)','FontSize',16);
    title(sprintf('The Mariana Trench From ISVD With %i
 Columns',wantedEigenSize));
    accurateRating = accuracyRating(A,compressedA);
    compressedRating = compressionRating(A, wantedEigenSize);

    %fprintf('compression rating : %f (lower is better), for an eigenspace of
 dimension: %i \n',accuracyRating(A,compressedA), wantedEigenSize);
    count=0; %used to count the number of points below 6km
    currentTotal=0; %used to keep tract of the sum of the depths below 6km
    for i = 1:size(compressedA,1)
       for j = 1:size(compressedA,2)
          if(compressedA(i,j)<-6000) %if the point is deeper than 6km, then
 update our variables that keep track
              count = count+1;
              currentTotal = currentTotal+compressedA(i,j);
          end
       end
    end
    averageDepthUnder6km = currentTotal/count; %arithmetic mean
    fprintf('Average Depth of the Trench For ISVD With %i Columns : %f (km)
 \n',wantedEigenSize,averageDepthUnder6km/1000);%Average Depth of the trench:
 -7204.636665

end

function total = compressionRating(A, numEigens)
    numInA = size(A,1)*size(A,2);
    total= (numEigens*(size(A,1)+numEigens+size(A,2)) )/numInA;
end

function total = accuracyRating(A,compressedA)
    total=0;
    for i = 1:size(A,1)
       for j = 1:size(A,2)
          total = total+ abs( A(i,j)-compressedA(i,j) );
       end
    end
    total=total/100000000;
end
```

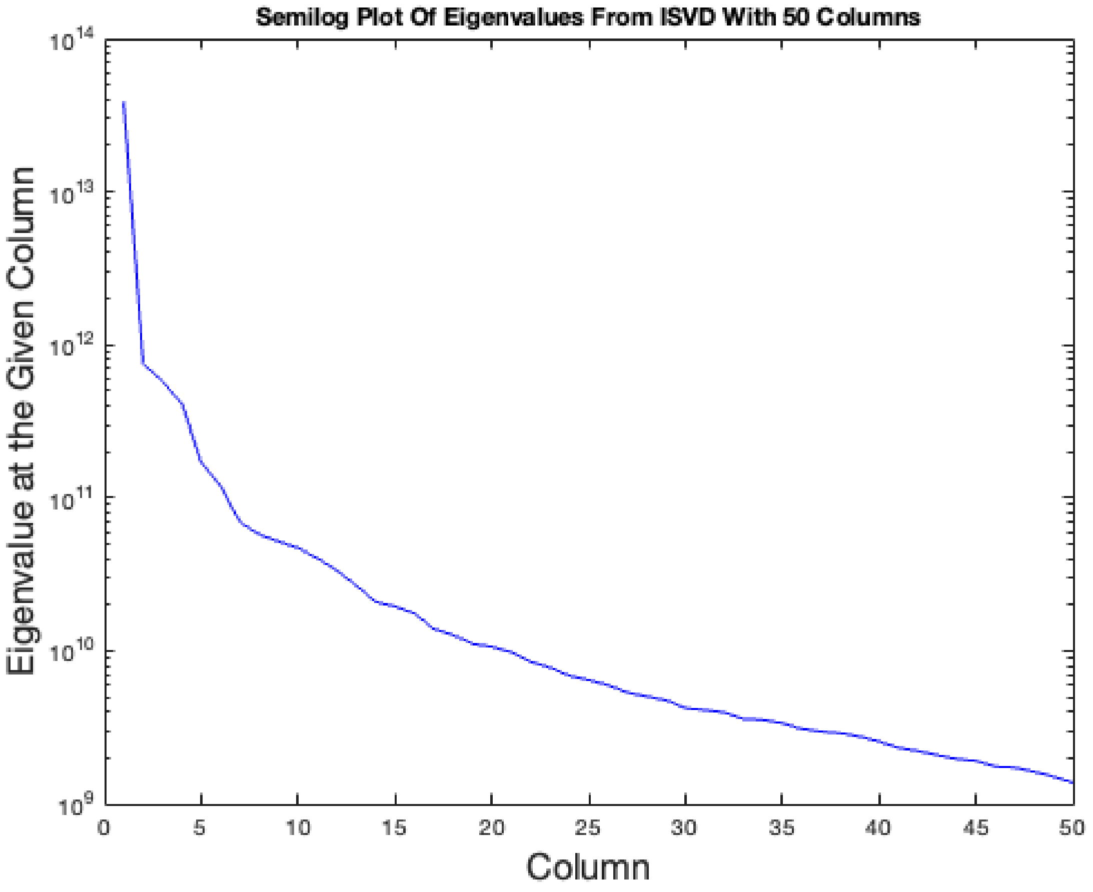
```
function [matrixOfVectors,matrixOfValues] = betterEigen(matrix,numOfEigens)
    %this method uses the second given algorithm to find the first
    %[numOfEigens] amt of eigenvectors adn their associated values.
    matrixOfVectors = zeros(sizeColVect(matrix),numOfEigens);
    matrixOfValues = zeros(1,numOfEigens); %row vectors
    for i = 1:numOfEigens
        u = randomUnitColVector2(sizeColVect(matrix)); %i


        u1star = matrix*u;%ii

        summationResult=0;
        for j = 1:(i-1)
            summationResult=summationResult
+(transpose(u1star)*matrixOfVectors(:,j))*matrixOfVectors(:,j);
        end

        u1=u1star-summationResult;%iii
        u1=unitVect(u1);%iv

        whileCount = 0;
        smallNumber = 1e-3;
        while(mag(u1-u)>smallNumber)
            whileCount=whileCount+1;
            u=u1;
            u1star = matrix*u;%ii

            summationResult=0;
            for j = 1:(i-1)
                summationResult=summationResult
+(transpose(u1star)*matrixOfVectors(:,j))*matrixOfVectors(:,j);
            end

            u1=u1star-summationResult;%iii
            u1=unitVect(u1);%iv
        end
        matrixOfVectors(:,i)=u1;

        %process of getting associated eigenvalue
        scaledV1 = matrix*u1;
        matrixOfValues(1,i) = scaledV1(1,1)/u1(1,1);


        %fprintf('whileCounter: %i \n',whileCount);%

    end
end

function sizeVect = sizeColVect(colVect)
    b=size(colVect);
    sizeVect=b(1);
end
```

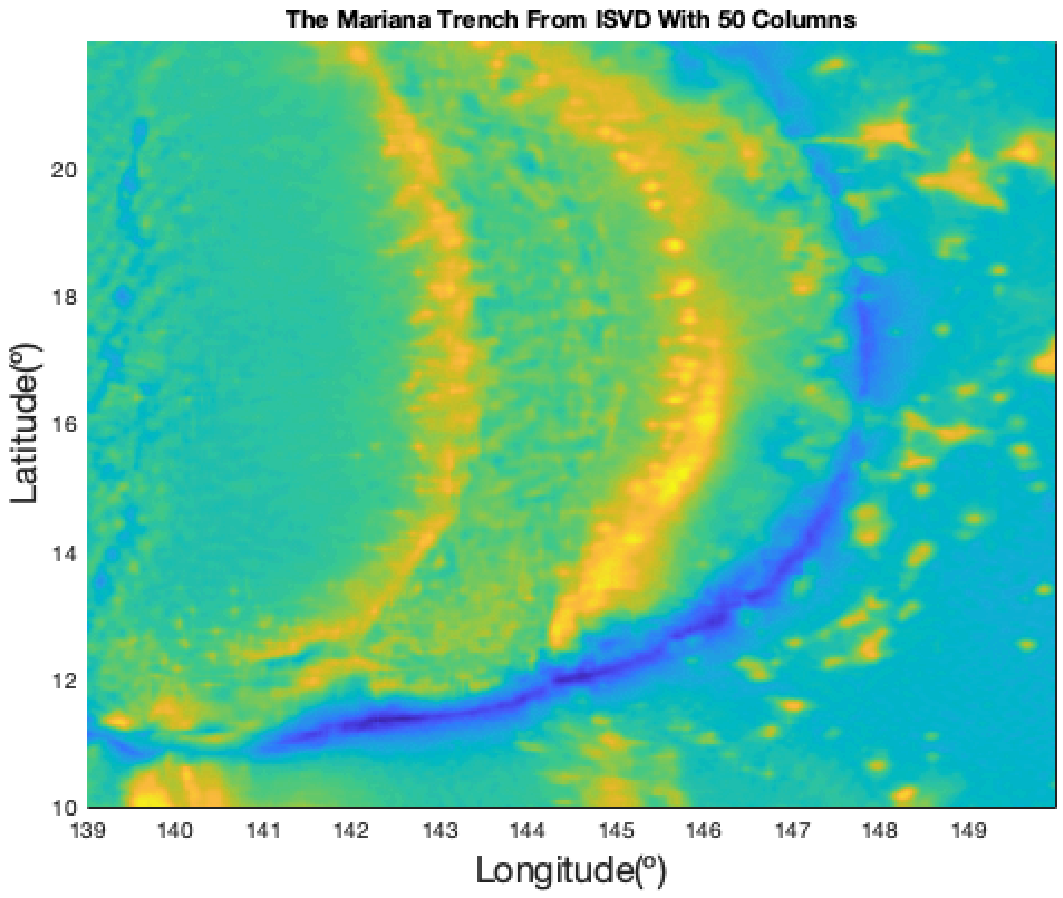
```
function vect = randomUnitColVector1(size,randomUpperBound)
    vect = zeros(size,1);
    for k = 1:size
          vect(k,1)=randi(randomUpperBound,1);%first num in randi is random
 int generator
    end
    vect=unitVect(vect);
end

function vect = randomUnitColVector2(size)
    vect=randomUnitColVector1(size,10);
end

function unitVector = unitVect(array)
    unitVector = array/mag(array);
end

function magnitude = mag(array)
    magnitude = 0;
    for k = 1:sizeColVect(array)
      magnitude=magnitude+(array(k,1))^2;
    end
    magnitude= sqrt(magnitude);
end
```

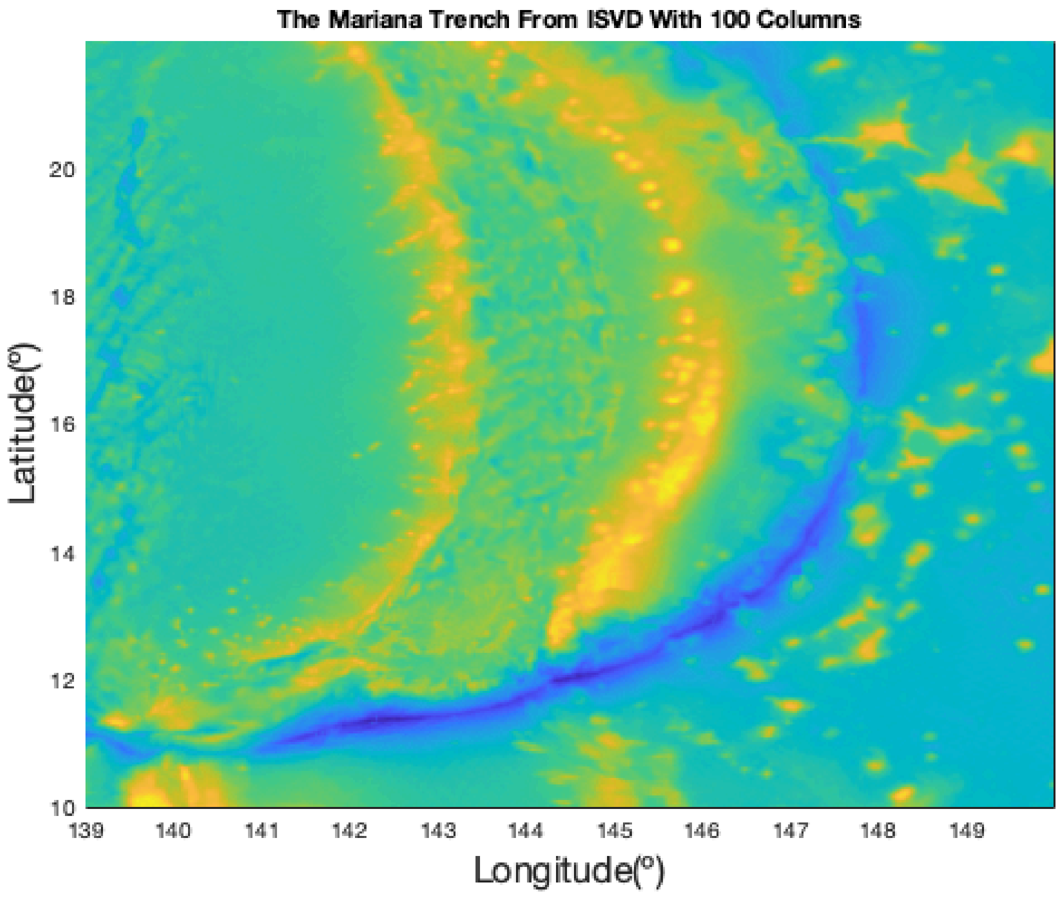
```
OUTPUT:
Average Depth of the Trench For ISVD With 1 Columns : -6.353018 (km)
For ISVD With 1 Columns: compression rating 0.001453 , accuracy rating
 15.437696 (lower is better)
Average Depth of the Trench For ISVD With 10 Columns : -7.049058 (km)
For ISVD With 10 Columns: compression rating 0.014573 , accuracy rating
 5.946921 (lower is better)
Average Depth of the Trench For ISVD With 50 Columns : -7.174004 (km)
For ISVD With 50 Columns: compression rating 0.073916 , accuracy rating
 1.869204 (lower is better)
Average Depth of the Trench For ISVD With 100 Columns : -7.196723 (km)
For ISVD With 100 Columns: compression rating 0.150463 , accuracy rating
 0.837677 (lower is better)
Average Depth of the Trench For ISVD With 500 Columns : -7.203576 (km)
For ISVD With 500 Columns: compression rating 0.857534 , accuracy rating
 0.134673 (lower is better)
```

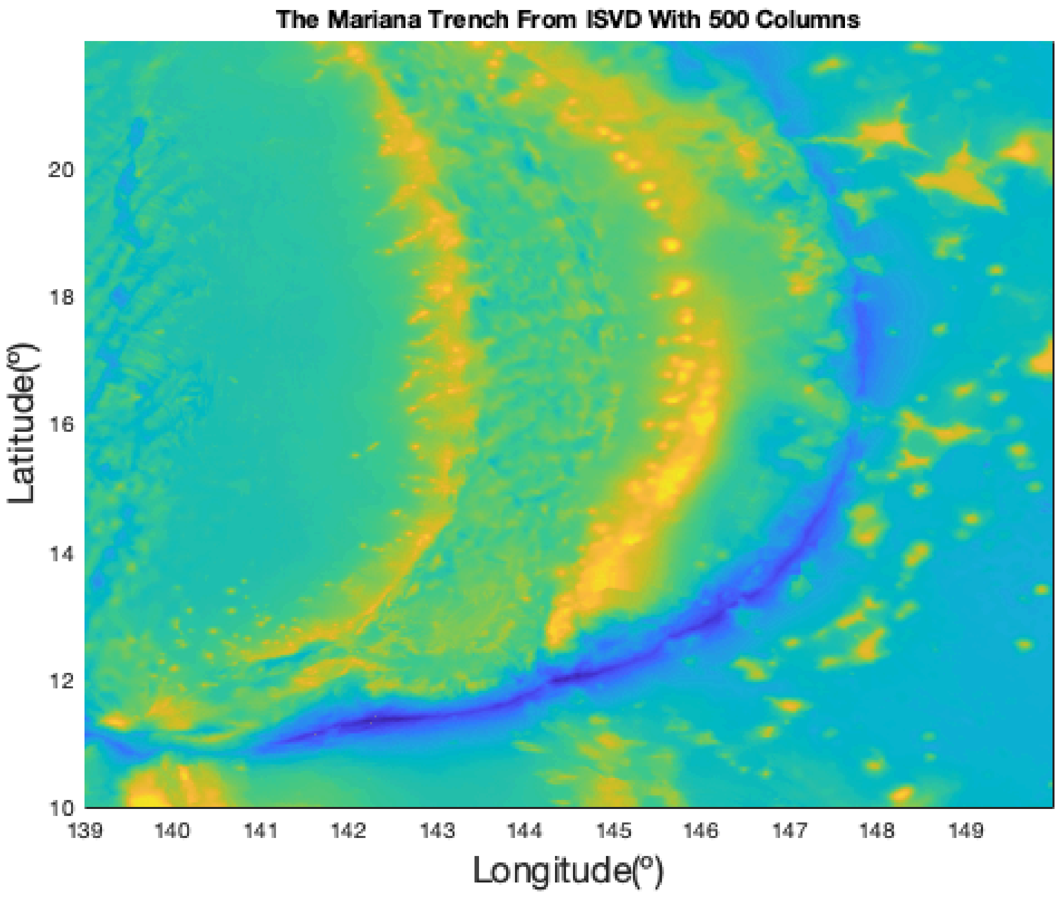

The Mariana Trench From ISVD With 1 Columns
Latitude(°)
20
18
16
14
12
10
139
140
141
142
143
144
145
146
147
148
149
Longitude(°)


The Mariana Trench From ISVD With 10 Columns
Latitude(°)
Longitude(°)
10
12
14
16
18
20
139
140
141
142
143
144
145
146
147
148
149


Semilog Plot Of Eigenvalues From ISVD With 50 Columns
Eigenvalue at the Given Column
Column
10^{14}
10^{13}
10^{12}
10^{11}
10^{10}
10^{9}
0
5
10
15
20
25
30
35
40
45
50


The Mariana Trench From ISVD With 50 Columns
Latitude(°)
20
18
16
14
12
10
139
140
141
142
143
144
145
146
147
148
149
Longitude(°)


The Mariana Trench From ISVD With 100 Columns
Latitude(º)
Longitude(º)
10
12
14
16
18
20
139
140
141
142
143
144
145
146
147
148
149



Published with MATLAB® R2021b