Applications and Theory for Spectral Clustering

Pranav Rajbhandari (prajbhan) and Thor Truelson (ttruelso)

November 2020

1 Abstract

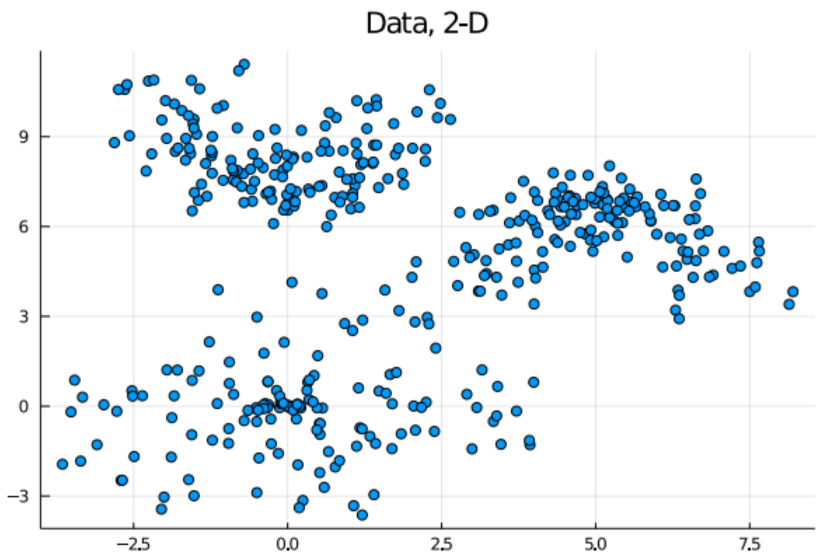
A common problem in data science is finding ways to group or "cluster" data into subsets by some metric of similarity.

An excellent method to cluster data is spectral clustering. To sort data in $\mathbb{R}^n$ by spectral clustering, we first create some similarity graph, then compute its Laplacian matrix. We then calculate the eigenvalues/vectors of the Laplacian. Interesting properties come from the eigenvalues of this graph, most importantly, each eigenvector with a small eigenvalue can represent a cluster, with a strong positive value for data points in its cluster, and a negative otherwise. Because of this, the number of these "small" eigenvalues can approximate the number of clusters we have. We can then sort a matrix of these eigenvectors into clusters with k-means or some other method, which will give us a clustering of the original data.

The following will requires some knowledge of Linear Algebra and the programming language Julia. In any case the authors hope that this is an interesting project.

2 Understanding the Problem

In data science there is data that must be understood as ”clusters” that are similar to each other. An example to motivate this is shown below



(generated from python)

Intuitively it seems rather easy to cluster these points into a downward curve, an upwards curve, and a blob at the bottom. However these intuitions must be more formal and translatable into computer algorithms. As such we set upon the problem trying to find a way to cluster data points in n dimensional space.

3 Preparing a proximity graph

Suppose we have m data points in a set X with each data point $\in \mathbb{R}^n$ and we want to sort them into k disjoint clusters.

In this step, we will construct, $G(V, E)$, with vertices $V = X$ and with E, the edges, as some measure of similarity between two vertices (self edges are not considered). These are some common ways to construct the graph

1. ϵ neighborhood graph and matrix
   We will construct the adjacency matrix by the following where, $\epsilon \in \mathbb{R}^+$
   $$e_{ij} \in E \iff ||x_i - x_j|| < \epsilon$$
   Observe that we create a circle or hyper-sphere of radius ϵ around every point x_i, and define an edge between x_i and all $x_j \in X \setminus \{x_i\}$ within the region. This relation will be symmetric since
   $$||x_i - x_j|| < \epsilon \iff ||x_j - x_i|| < \epsilon$$
2. K nearest neighbors graph.
   with $K \leq m$ then we want to make an edge from $x_i \in X$ to its K nearest neighbors (wrt. euclidean distance). For each x_i, we want find the set $S \subseteq X \setminus \{x_i\}$ with $|S| = K$ to minimize the expression.
   $$\sum_{x \in S} ||x_i - x||$$
   Observe this relation is not necessarily symmetric. Therefore, if e_{ij} exists, we must also define e_{ji}.

These are all valid ways of generating a set of edges given a graph in order to spectral cluster. Regardless of which option to choose, we now have $G(V, E)$.

4 The Laplacian

Using our constructed $G(V, E)$, we will compute an adjacency matrix A as in class i.e.

$$A_{ij} = \left\{ \begin{array}{cc} 1 & e_{ij} \in E \\ 0 & \text{else} \end{array} \right\}$$

The adjacency matrix represents the connections of the graph, with a 1 where the nodes are connected (for the data set, this represents data close together)

With this matrix we will then create another matrix D of the same dimensions of A i.e

$$D_{ij} = \left\{ \begin{array}{cc} A_{i*} \cdot \mathbf{1} & i = j \\ 0 & i \neq j \end{array} \right\}$$

Each row of this diagonal matrix is the sum of the corresponding row on the adjacency matrix. This matrix can be thought of as the total number of connections each node has.

We will now make a new matrix L such that

$$L = D - A$$

This is known as the (unnormalized) graph laplacian of A

4.1 Justification for the Laplacian

Let a cluster C be defined as a subset of V:

$$C \subseteq V \wedge |C| \geq 0$$

and that if two C exist i.e C_i and C_j that

$$C_i \cap C_j = \emptyset \iff i \neq j$$

Assume there are a positive number of clusters in the data

Assume that $|V| > 1$ and that we have multiple clusters, since otherwise would result in zero or 1 clusters and these cases are trivial.

We will make an vector s such that

$$s_i = \left\{ \begin{array}{cc} 1 & v_i \in C_1 \\ -1 & v_i \notin C_1 \end{array} \right\}$$

Observe that $\frac{1}{2}(1 - s_i * s_j) = \left\{ \begin{array}{cc} 1 & s_i \neq s_j \\ 0 & s_i = s_j \end{array} \right\}$

This value is effectively an indicator for if the vertices are in the same group.

Let the cut $R \in \mathbb{N}^+$ be defined as the number of edges between one cluster to another

$$R = \frac{1}{2} \sum_{i,j} A_{ij} \mid \ v_i, v_j$$

Where v_i, v_j are in different clusters. We divide by two as we are double counting in the method. This can be represented concisely as:

$$R = \frac{1}{2} \sum_{i,j} \frac{1}{2}(1 - s_i s_j) A_{ij} = \frac{1}{4} \sum_{ij} (1 - s_i s_j) A_{ij}$$

Using the indicator variable

We then define k as the vector where $k_i = \sum_j A_{ij}$ i.e the sum of the ith row of A

$$\begin{aligned} \Sigma_{i,j} A_{i,j} &= \Sigma_i k_i \\ &= \Sigma_i s_i^2 k_i \\ &= \Sigma_{i,j} s_i * s_j k_i I_{ij} \end{aligned}$$

We then know as

$$R = \frac{1}{4}\Sigma_{i,j} A_{ij} - s_i s_j A_{ij}$$

We can then express

$$R = \frac{1}{4}\Sigma_{i,j} s_i s_j (k_i I_{ij} - A_{ij})$$

And then notice that for matrix L'

$$L'_{ij} = k_i I_{ij} - A_{ij}$$

That this is true iff

$$L' = diag(A) - A$$

Which is iff

$$L = L'$$

Therefore if we have s then observe that

$$R = \frac{1}{4} s^T L s$$

4.2 Applying R and L

We know have a laplacian L. And we will choose an s such that R is minimized. and we can write s as a combination of of eiegenvectors of L. Which are found as L is symmetric by the spectral theorem. And these vectors span the m dimensional space.

Suppose $a_i \in R$ and that e_i is an eigenvector of L

$$s = \Sigma_i a_i e_i$$

We then observe that

$$s^T s = \Sigma_i a_i^2 = n$$

Which will be a constraint. We then proceed

$$\begin{aligned} 4R &= s^T L s \\ \iff &= \Sigma_i a_i v_i L \Sigma_j a_j v_j \\ \iff &= \Sigma_i a_i v_i \Sigma_j \lambda_j a_j v_j \\ \iff &= \Sigma_{ij} a_i a_j \lambda_j I_{ij} \\ \iff &= \Sigma_i a_i^2 \lambda_i \end{aligned}$$

As such we want to minimize this R and we will go with the assumption that

$$\lambda_1 \leq \lambda_2 \leq ...\lambda_m$$

. And we can observe that as we did this through a real symmetric matrix that there is a lower bound on these values.

$$0 \leq \lambda_1$$

So as we are trying to minimize the expression $\Sigma_i a_i^2 \lambda_i$ it make sense that we would choose λ_1 as this is the smallest λ. But observe that as we defined

$$L = D - A$$

Then the vector $s = (1, ..., 1)$

$$(D - A)e_1 = De_1 - Ae_1 = \vec{0}$$

This is because we defined the diagonal as the sum of the row of A. As such $De_1 = Ae_1$ as such we know that now matter what $\lambda_1 = 0$ and that $e_1 = \frac{\vec{1}}{\sqrt{n}}$ when normalized.

Thus we find no matter what we can cut the graph minimally if we put all data points into the same cluster. This is because there are no edges going from the one cluster to any other cluster. As such we have our solution to clustering...

Wait a sec, we already said this was a trivial solution as yes we can group all the points together but that would make this problem trivial. As such we will keep searching.

So now we will examine the eigenvalue that is the next smallest i.e. λ_2. The corresponding eigenvector is known as the "fiedler" vector. This vector will sort a graph into 2 clusters based on the idea of S where if an entry is positive then then a data point is in a group and if negative it is in another.

We can keep going in this way, taking k small eigenvalues and using their associated vectors to cluster the data into k clusters. But how will we know how many groups is best?

The majority of this is from *Foundation of Data Science* by Kannan [1].

4.3 Side note: conceptualizing Eigenvalues of Laplacians

The eigenvalues of the Laplacian matrix have many interesting properties, including:

- the algebraic multiplicity of the 0 eigenvalue represents the number of disconnected clusters in the graph.
- the second smallest eigenvalue (the Fiedler value) represents the minimum number of edges severed to break the graph into two clusters (note that if the graph is already not connected, this will be 0)
- the smallest positive eigenvalue (called the spectral gap) is correlated with the density of the graph, with a higher spectral gap meaning a denser gap

These three can all be conceptualized by the following (for graphs with many elements):

- Assume our graph is connected

  Recap: A Laplacian L contains the number of connections vertex i has at the diagonal entry $L_{i,i}$, and each connection from vertex i to j is recorded with a -1 at $L_{i,j}$ and $L_{j,i}$

  It should be clear from this definition that $\mathbf{1}$ is an eigenvector with eigenvalue 0, since the sum of each row is 0.

- Next, assume our graph is composed of two connected graphs (also in the adjacency matrix, order the graphs such that all the vertices of the first cluster are before the second cluster)

  Our laplacian would then be of the form

  $$\begin{pmatrix} L_1 & \mathbf{0} \\ \mathbf{0} & L_2 \end{pmatrix}$$

  with L_1 and L_2 being valid laplacians describing each connected region.

  Note: you can convince yourself of this by the logic that having a non-zero value in the zero region would imply the two disconnected regions have a connection between them

  It is clear from this that our graph could have two valid eigenvectors with eigenvalue 0, of the form $\mathbf{1}$ and $(x_1, \ldots, x_k, -y_1, \ldots, -y_{m-k})$ (also any linear combination of these)

  A way to view this is the eigenvalue describing the cluster, with positive values indicating membership

- this generalizes to graphs with n disconnected clusters, with the resulting laplacian (ordered based on the cluster)

  $$\begin{pmatrix} L_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & L_n \end{pmatrix}$$

and n resulting eigenvectors of eigenvalue 0, including a $\mathbf{1}$

each of these eigenvectors has only positives in the places where its respective cluster in

therefore, inspecting the elements of a single eigenvector will give a clear indicator of which data points should be in that cluster

for example, with 3 clusters with 2, 3, and 1 points, a matrix of the three 0 eigenvectors may look like

$$\begin{pmatrix} 1 & 1 & -1 \\ 1 & 1 & -1 \\ 1 & -1 & 1 \\ 1 & -1 & 1 \\ 1 & -1 & 1 \\ 1 & -1 & -1 \end{pmatrix}$$

running k-means using each row as a $\mathbb{R}^3$ input will return us the result of the three clusters we started with since each 3 dimensional point is on the corner of a cube corresponding to its original cluster

- The final step comes in considering what happens in clusters that aren't completely disconnected, but are very close

  Consider a matrix with two almost disconnected clusters

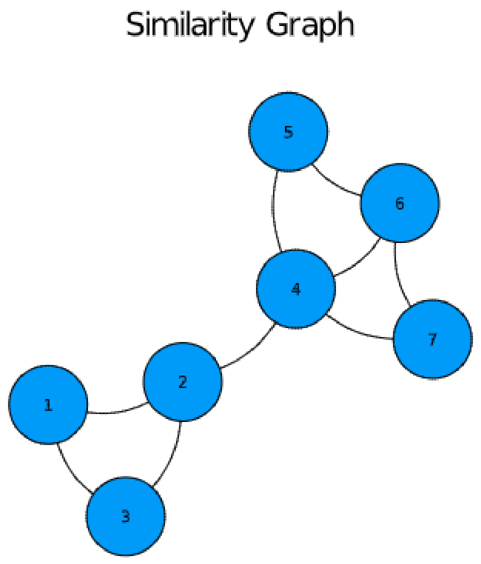
  Similar to before, the laplacian will look like

$$\begin{pmatrix} L_1 & K \\ K^T & L_2 \end{pmatrix}$$

  with K being a matrix with mostly 0s and a few -1s. Each -1 represents a connection between the two clusters

  Consider trying to fit the eigenvectors from the last example onto this matrix, specifically $(x_1, \ldots, x_k, -y_1, \ldots, -y_{m-k})$. It is clear that it *almost* works, except for the few times where K is non zero. To fix this, we must adjust the eigenvector to account for the rows, and raise the eigenvalue to accommodate

  This is less intuitive, so let's dive into proof by example



With a Laplacian

$$\begin{pmatrix} 2 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & -1 & 0 & 0 & 0 \\ -1 & -1 & 2 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 4 & -1 & -1 & -1 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 3 & -1 \\ 0 & 0 & 0 & -1 & 0 & -1 & 2 \end{pmatrix}$$

The matrix is intuitively divided into 2 clusters, the first 3 and the second 4 vertices. They are almost disconnected, apart from a connection between the second and the fourth points.

The matrix has a smallest positive eigenvalue .398 with eigenvector

$\begin{pmatrix} 1.384 & .833 & 1.384 & -.602 & -1 & -1 & -1 \end{pmatrix}$

We can see the underlying inferences in this eigenvector, as it shows that the first three (positive) points being in a cluster. In addition, we can also see that the second and the fourth points are slightly closer to 0, which implies that they are slightly connected to the other cluster.

As an added bonus, using k-means on this vector and the vector **1** would accurately place the points in the right clusters.

- This generalizes to larger data sets with more clusters, as using k-means with the smallest eigenvalues will result in sorting the data accurately.

This is neat! The human is almost completely obsolete. Now the only outside input required is how many groups to sort them into.

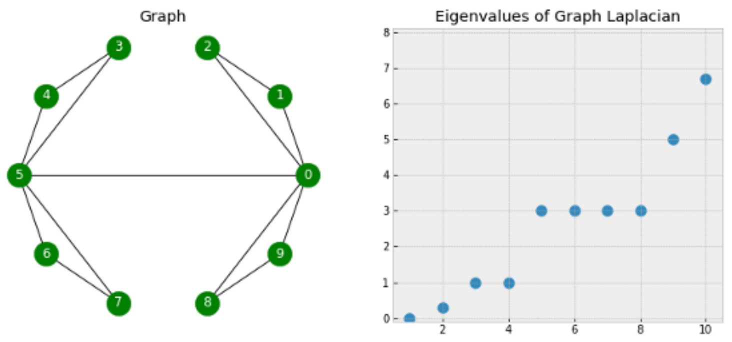
4.4 Number of groups

The number of groups that we are sorting the data into is an interesting question, especially with a connected graph where the distinctions are not very clear.

Because of the relation described above between small eigenvalues and groups, the number of groups should be well described by the number of ”small” eigenvalues. The question then becomes how to define ”small”.

This problem does not have a clear answer. However, what we can do is consider the gap between consecutive eigenvalues. When we notice a large jump (i.e. the next largest eigenvalue is considerably larger than all previous), we will consider the previous ones as ”small” eigenvalues.

For example:



Fleshman, W., 2019. Spectral Clustering Foundation and Application [3]

The number of groups we can split the data into can be seen in the distribution of the eigenvalues.

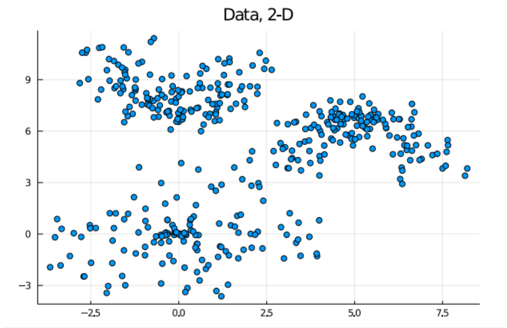
- The first e-value is 0, which means we can split the group into at least 1 disconnected cluster (big surprise)
- The next e-value is around .3, which implies it is slightly harder to split the graph into 2 clusters. We can see that in the graph by noticing that we must sever the middle connection to split the groups
- The next 2 e-values are higher, which implies more work must be done to split the data into 3 and 4 clusters. The clusters also become harder to visualize, though we can work out that the clusters may be splitting the left and right half into their top and bottom parts
- The later e-values are significantly higher, which implies it is much harder to split the graph into 5 or more clusters

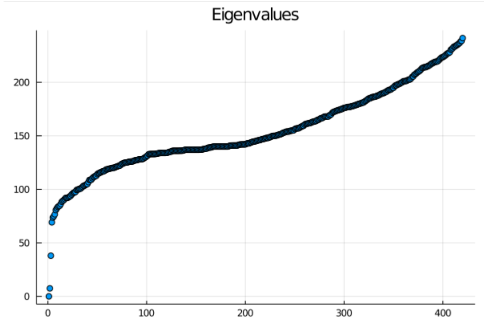
Notice that each time the e-value increased, we found a logical way to split the data, with increasing difficulty the higher the values were. First there was the trivial case of all the data in one cluseter, then there were two left/right clusters, then there were 4 clusters. This generalizes to all graphs, and in this way, the number of ”small” eigenvalues directly correlate with the number of clusters we can sensibly split up our data.

5 Application of Findings

From this, it is clear we must separate the graph into at least the multiplicity of the eigenvalue 0. This will cluster the graph into its disconnected clusters. The eigenvectors associated with these λs_i will be indicators of which data points are in these components.

To segment the graph further, we will cluster the graph into the number of ”small” eigenvalues





Consider the above data set. As we can see from either similarity graph, the data is amalgamated into about 2 or 3 clusters. These clusters all have relatively few edges between them. We can see as a result of this that the multiplicity of the eigenvalues close to zero is about 2 or 3. As such we can take the first 3 eigenvectors and use their values for each place i to determine which cluster vertex i is likely to belong to.

Of course we can increase the number of clusters even more by going onto the larger eigenvalues but this may result in unnecessary clusters.

6 Pseudocode and Algorithim

Note that parameters should include a similarity function, and either a tolerance for eigenvalues close to 0 or a number of groups to sort the data into.

By default we will use a epsilon neighborhood similarity function, but other functions can be used.

Algorithm 1: Algorithm for separating a dataset of vertices V into clusters based on a similarity function f

```
1  function Spectral Clustering ($V, f$);
2  Input: a set of vertices in n-dimensional space, a similarity function f,
     and either a definition of "close to 0" for eigenvalues or number of
     groups to sort into
3  Output: A set of cluster C' which contain all the vectors in clusters
     based on the similarity function
4  Create a graph G(V,E) where V = V and E is if two edges are similar.
5  Create a symmetric adjacency matrix based off the graph where each
     vertex doesn't self relate. Create a matrix D such that the diagonal
     entries of D are equal to the sum of its row in A.
6  Create the Laplacian matrix L = D - A
7  Find the eigenvalues and eigenvectors of L
8  Choose all the eigenvalues close to 0, and put them in a matrix $K$ with
     length $k$
9  perform k-means with parameter $k$ on matrix $K$ to get the clusters of
     each point.
10 return a list containing cluster membership of each vertex, in order
```

7 Examples

7.1 Naive dataset

As an illustrative example we will show the algorithim in a much more simple and smaller dataset.

As input, we will use fairly intuitive 2 dimensional data. We then take the number of these datapoints and then create an adjacency matrix.

From inspection of a plot of the data, it is fairly obvious there are two clusters, one near the bottom left and one near the top right. We run the epsilon neighborhood method on the data to get the adjacency matrix. We then are able to create a graph of the matrix as shown below and see that the our guess of two clusters is reasonable. We then compute the Laplacian.

We then find the eigenvalues for the Laplacian and observe how many eigenvalues are less than the tolerance range (.2). We found 2 valid eigenvectors corresponding to eigenvalues in this range. We can then use a simple algorithm such as k-means to sort a matrix of these two vectors

We can now re-plot the data and the graph and observe the groups we have created

In [1]:
```julia
#using Pkg;
#Pkg.add("RDatasets");Pkg.add("LinearAlgebra");Pkg.add("Statistics");Pkg.add("Clusterin
using RDatasets, LinearAlgebra, Statistics, Clustering, Plots, GraphRecipes;
```

In [2]:
```julia
#USER INPUT: data

a = [1 1 ; 2 2 ; 1 2 ; 2 1 ;  10 9 ; 9 10 ; 8 8 ; 7 6 ; 7 9 ; 10 10]
#a=convert(Matrix, dataset("datasets", "iris")[:,1:4])
num_vecs = size(a)[1];
```

In [3]:
```julia
#mean value, avg of all points
middle=sum(a[i,:] for i in 1:num_vecs)/num_vecs


#average distance from the mean
avg_epsilon=sum(norm(a[i,:]-middle) for i in 1:num_vecs)/num_vecs

#USER INPUT
epsilon=avg_epsilon/2
#change manually if you want?

#2d approximation for the data
Disp = [a[i,j] - mean(a[:,j]) for i = 1:size(a)[1], j=1:size(a)[2]]

U,σ,V = svd(Disp, full = true)


#our adjacency matrix, using epsilon sphere
A = zeros((num_vecs, num_vecs))
for i in 1:num_vecs
    for j in i+1 : num_vecs
        x = a[i,:]
        y = a[j,:]
        if(norm(x-y) < epsilon)
            A[i,j] = 1
            A[j,i] = 1
        end
    end
end

#Our laplacian (skipping the diagonal matrix)
L = -1*deepcopy(A)
diag_vals = sum(A, dims=2)
for i in 1 : size(A)[1]
    L[i,i] = 1*diag_vals[i]
end


#eigenstuff of our laplacian
F = eigen(L)

#USER INPUT
#METHOD 1: guesstimation of "small" eigenvalue
guesstimation=.2
```

```julia
counter = 0
for i in F.values
    if(i < guesstimation )
        counter = counter+1
    end
end
counter

#METHOD 2: just put in the number of clusters
#counter=2

#using k-means to cluster data
#may need to re-run, k-means uses a random starting point which may be metastable
result=kmeans(transpose(F.vectors[:,1:counter]),counter);
```

In [4]:
```
#Laplacian
L
```

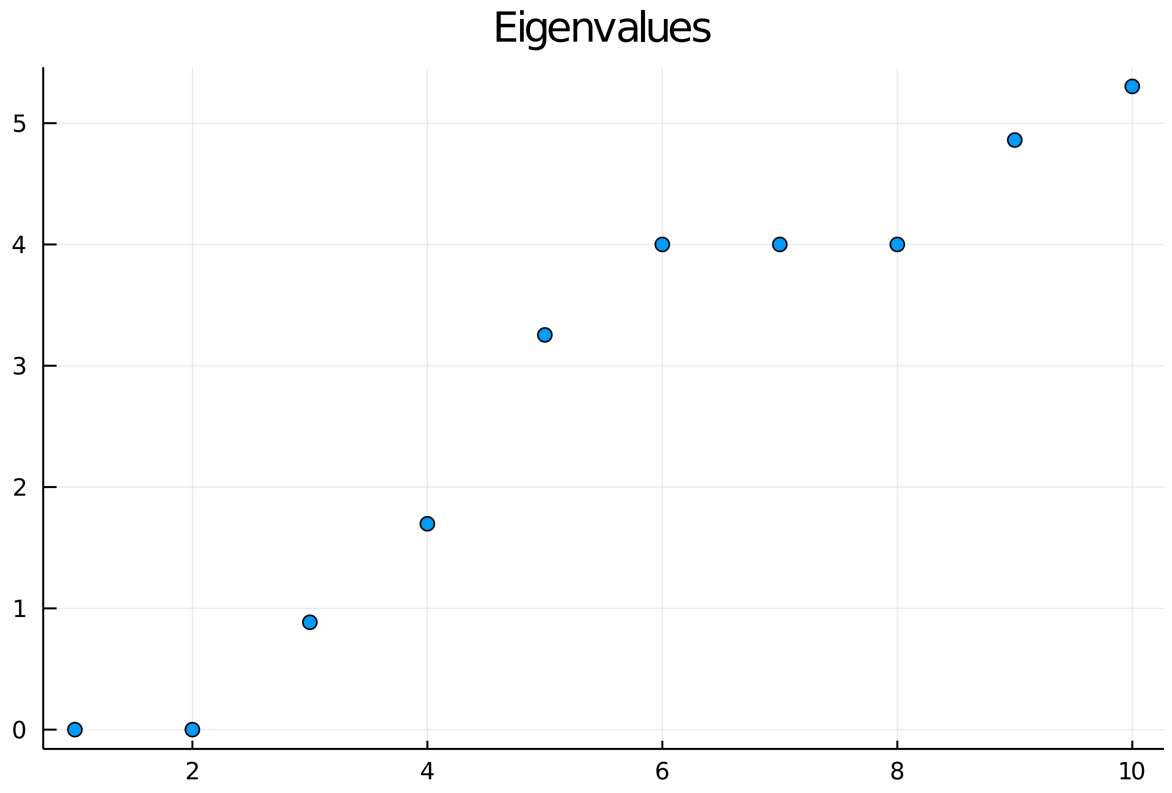
Out[4]:
```
10×10 Array{Float64,2}:
  3.0  -1.0  -1.0  -1.0  -0.0  -0.0  -0.0  -0.0  -0.0  -0.0
 -1.0   3.0  -1.0  -1.0  -0.0  -0.0  -0.0  -0.0  -0.0  -0.0
 -1.0  -1.0   3.0  -1.0  -0.0  -0.0  -0.0  -0.0  -0.0  -0.0
 -1.0  -1.0  -1.0   3.0  -0.0  -0.0  -0.0  -0.0  -0.0  -0.0
 -0.0  -0.0  -0.0  -0.0   3.0  -1.0  -1.0  -0.0  -0.0  -1.0
 -0.0  -0.0  -0.0  -0.0  -1.0   4.0  -1.0  -0.0  -1.0  -1.0
 -0.0  -0.0  -0.0  -0.0  -1.0  -1.0   4.0  -1.0  -1.0  -0.0
 -0.0  -0.0  -0.0  -0.0  -0.0  -0.0  -1.0   1.0  -0.0  -0.0
 -0.0  -0.0  -0.0  -0.0  -0.0  -1.0  -1.0  -0.0   2.0  -0.0
 -0.0  -0.0  -0.0  -0.0  -1.0  -1.0  -0.0  -0.0  -0.0   2.0
```

In [6]:
```
#k-means result
result.assignments
```

Out[6]:
```
10-element Array{Int64,1}:
 1
 1
 1
 1
 2
 2
 2
 2
 2
 2
```

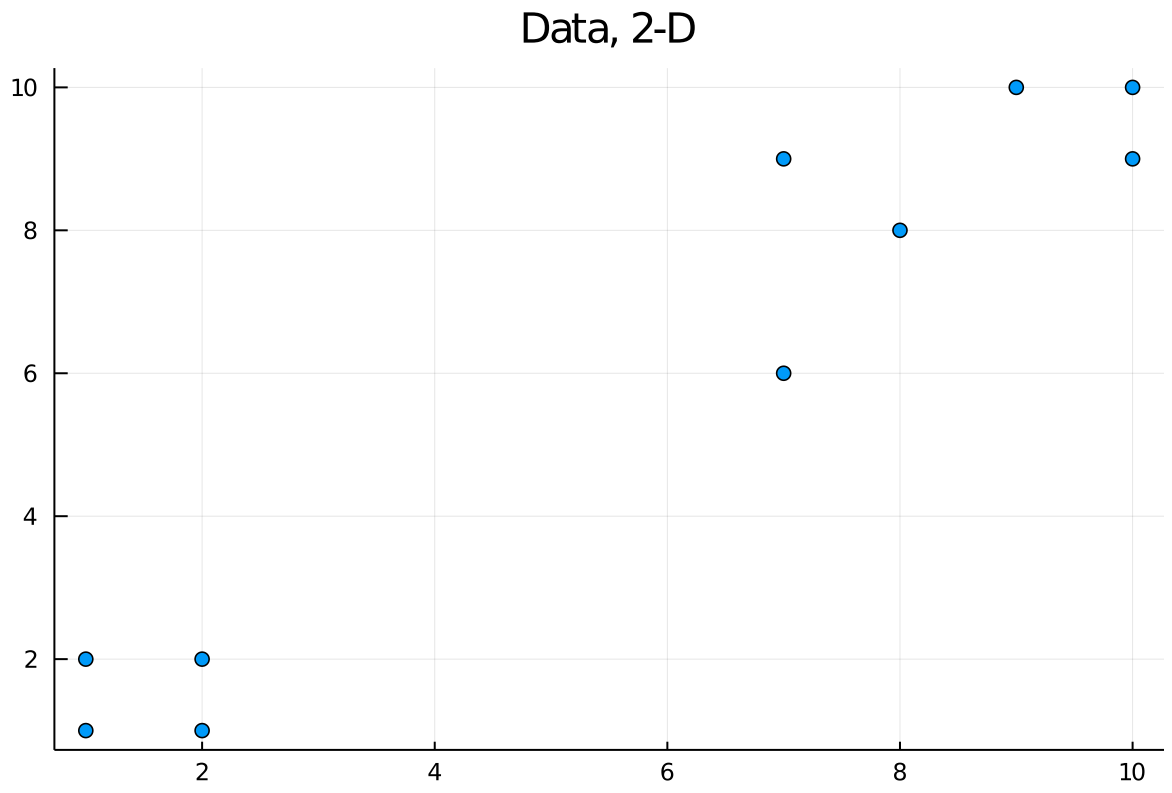
In [7]:
```
#plot eigenvalues
scatter([i for i in 1:size(F.values)[1]],F.values,labels=false,title="Eigenvalues")
```

Out[7]:



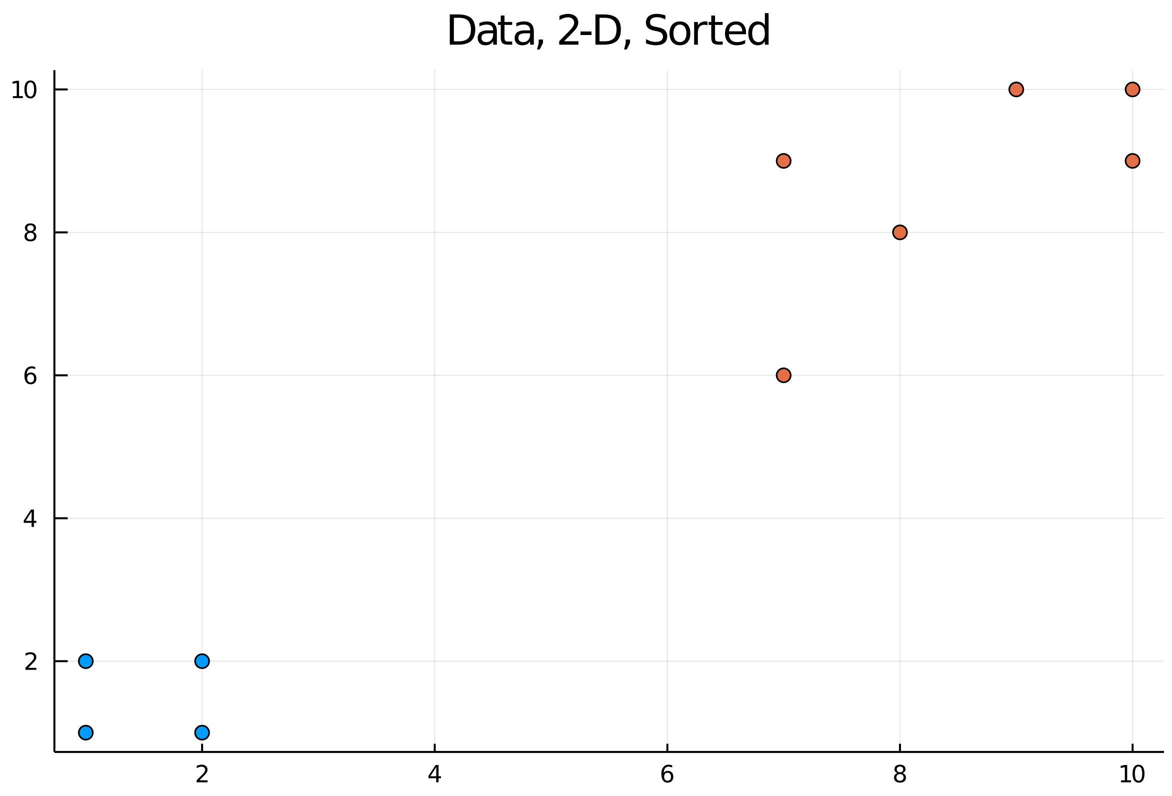
In [8]:
```
#displaying data with a 2d reduction
scatter(a[:,1], a[:,2],labels=false,title="Data, 2-D")
    #Disp*V[:,1], Disp*V[:,2],labels=false,title="Data, 2-D approximation")
```

Out[8]:



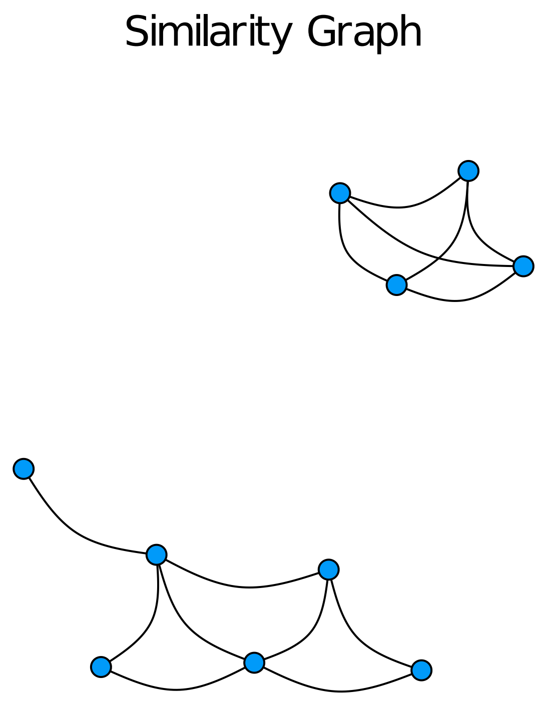
In [9]:
```
#Plot final assignments
colors = reshape(result.assignments, 1, num_vecs)
scatter(a[:,1], a[:,2] ,c = result.assignments, labels=false,title="Data, 2-D, Sorted")
    #Disp*V[:,1], Disp*V[:,2] ,c = result.assignments, labels=false,title="Data, 2-D ap
```

Out[9]:



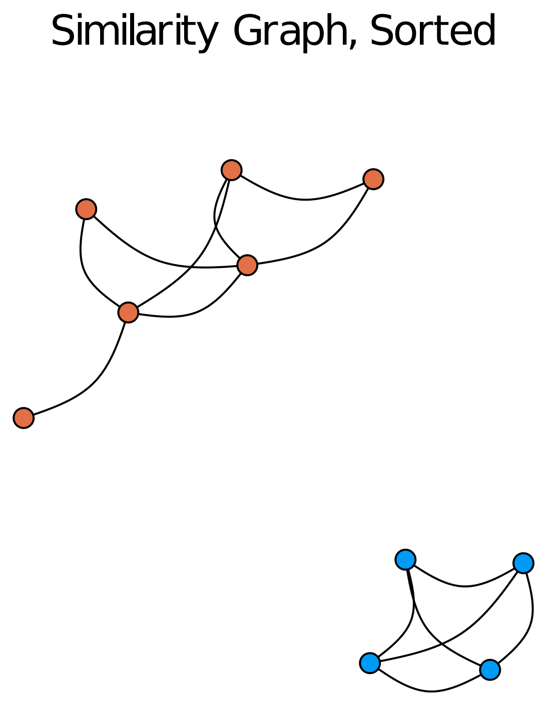
In [10]:
```
#graph the adjancy matrix
graphplot(A, markersize = .1,
    node_shape = :circle, title="Similarity Graph")
```

Out[10]:



In [11]:
```
#plot graph with assignments
graphplot(A, markersize = .1,markercolor=result.assignments,
    node_shape = :circle, title="Similarity Graph, Sorted")
```

Out[11]:



7.2 More complex data

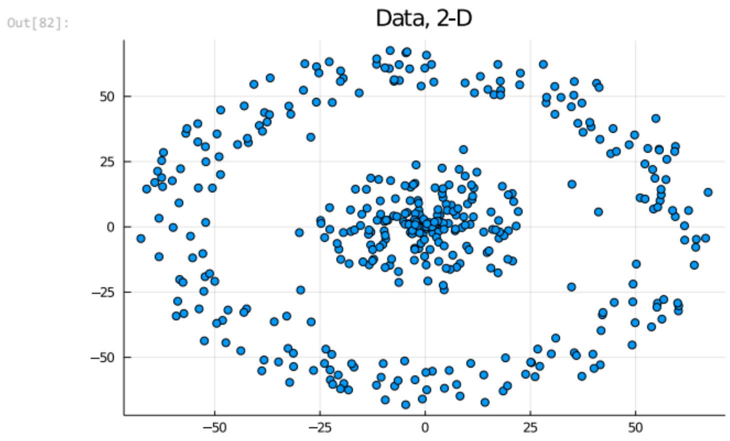
Using the same method, we can move on to more complex data. The data in this section was generated using python. To test the limits of the algorithm, there will be 420 random points following this distribution.

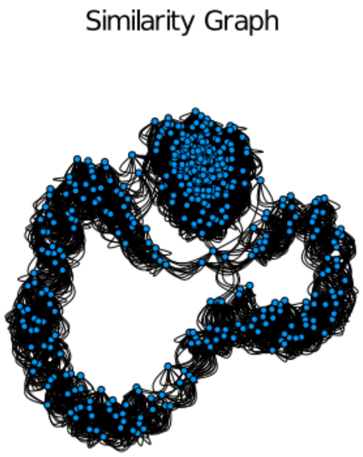
Note: The code will not be included to save space (save the virtual trees)

Note: the connected graphs for most large data sets will not be included, because I enjoy a room-temperature computer.

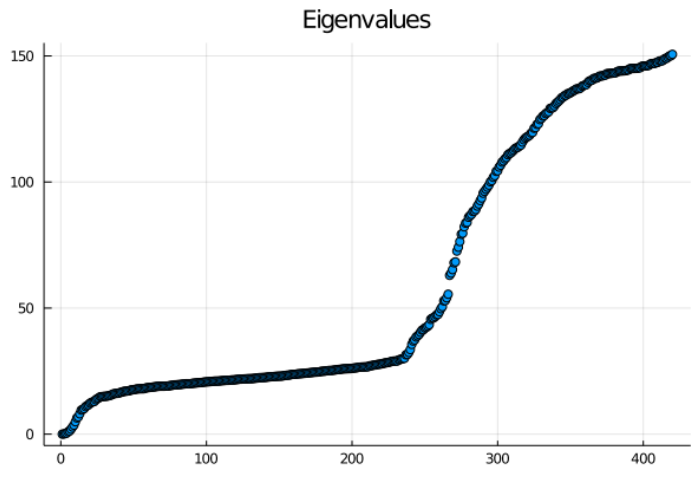
7.2.1 annulus 2d

This distribution was created by adding a center circle with a disc around it, with some noise.



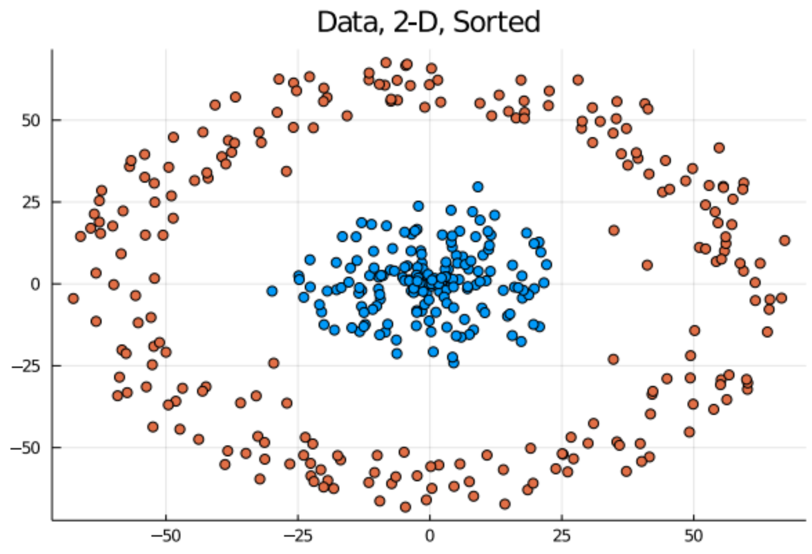


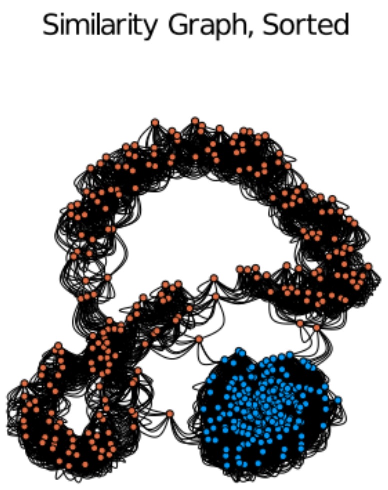
Note: We have now graduated into the big boy distributions, as there is a significant amount of noise and interconnectivity in the data



The eigenvalues were hard to distinguish, so instead of a tolerance, we input the number of clusters we wanted, 2.

Out[83]:





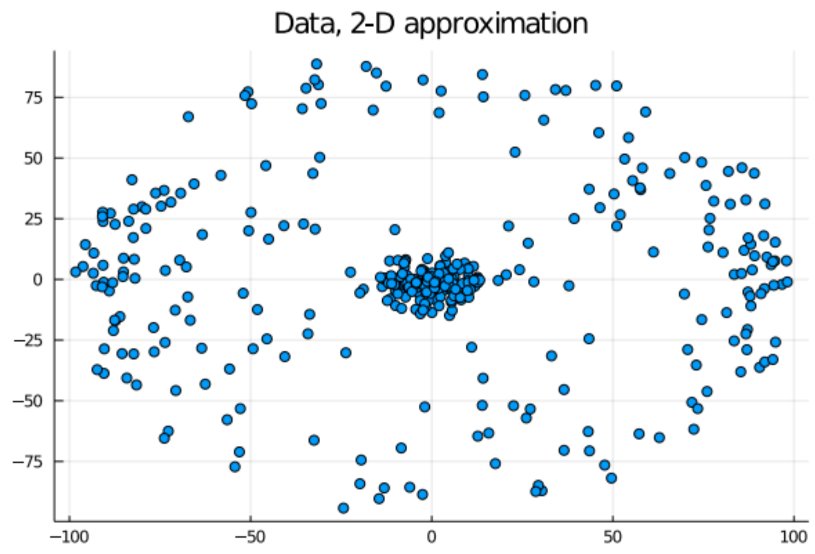
Note: In the cold month of December, generating this graph kept me and my laptop nice and toasty. However, this will be the last graph in this section that I produce.

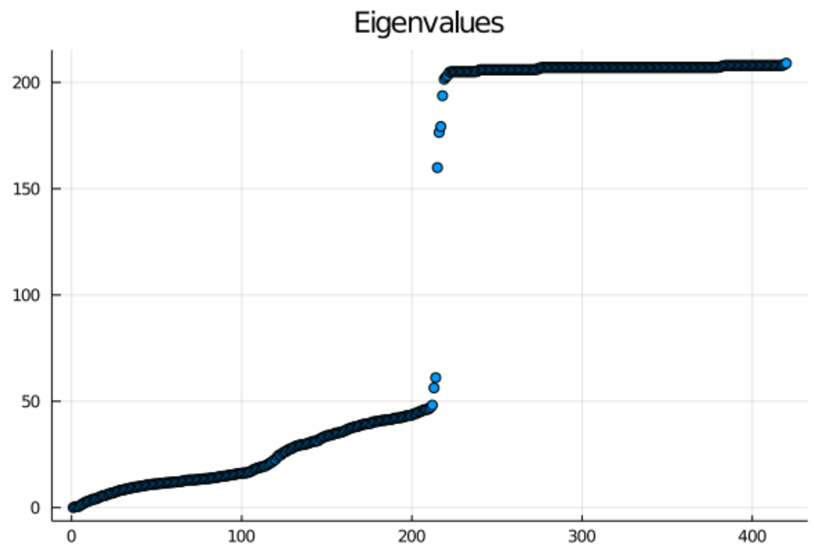
Conclusion:

From spectral decomposition we see that a lot of the variance in the distribution can be explained by two base distributions, which is exactly correct.

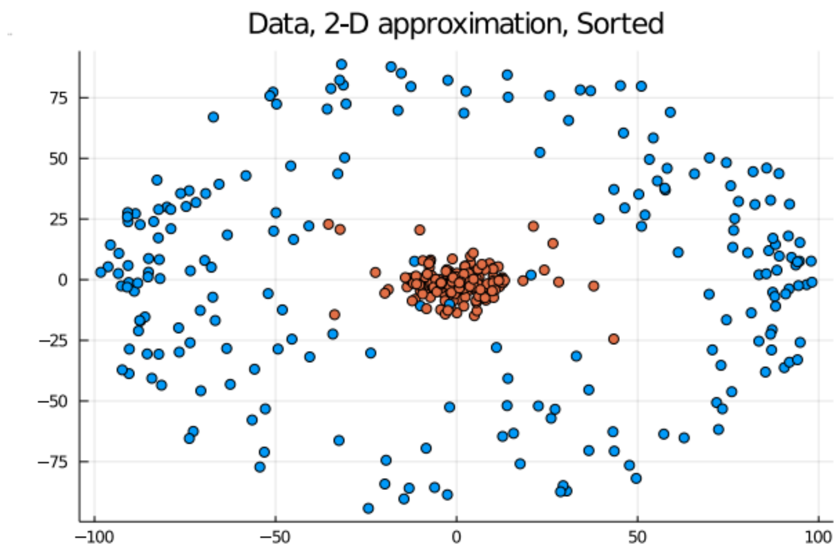
7.2.2 annulus 3d

This distribution was created by adding a center sphere with a shell around it, with noise.





again, the eigenvalues were hard to distinguish, so we forced the program to give us 2 clusters

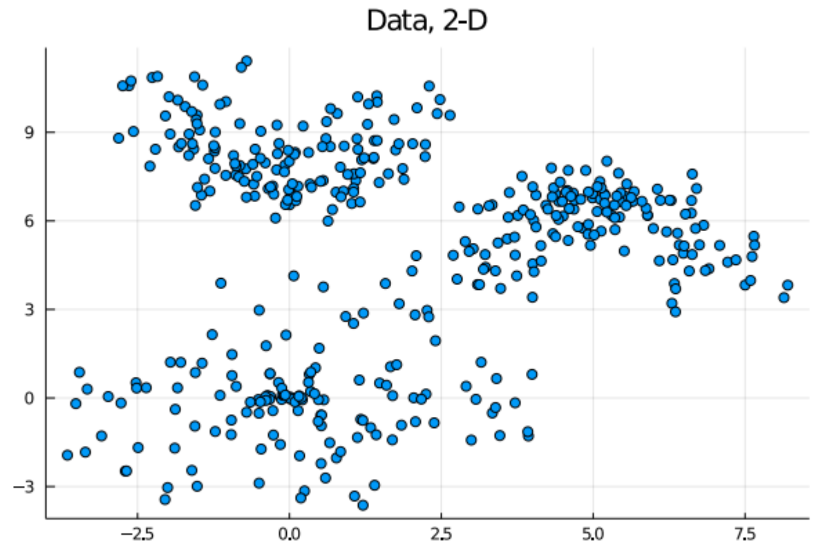


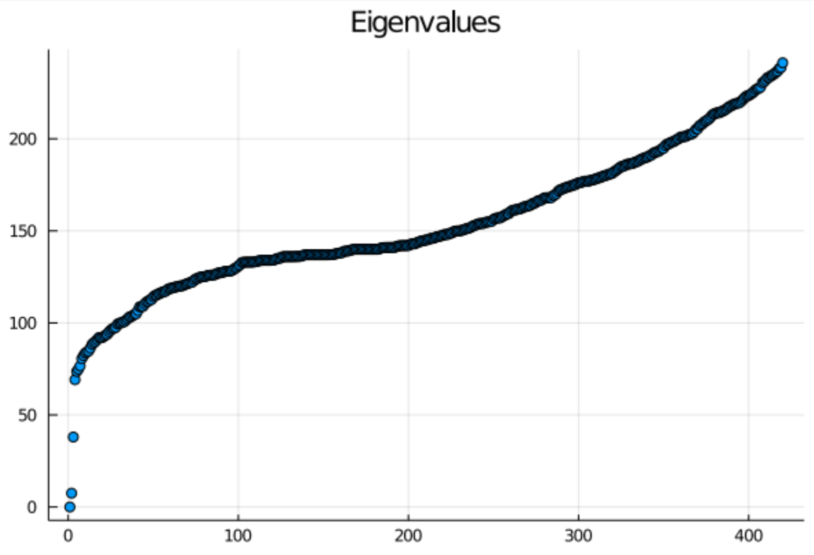
Conclusion:

From spectral decomposition we see that a lot of the variance in the distribution can be explained by two base distributions, which is exactly correct.

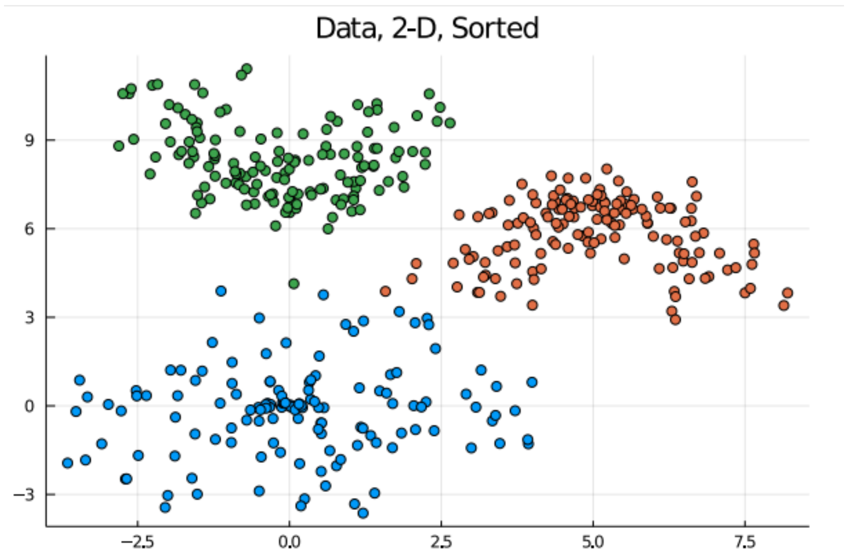
7.2.3 blob thing

This distribution is created from a parabola above the origin, a downward parabola to the right, and a blob near the origin, all with lots of noise.





We could see that two eigenvalues were small and very close to zero, and another one was significantly smaller than the rest.



Conclusion:

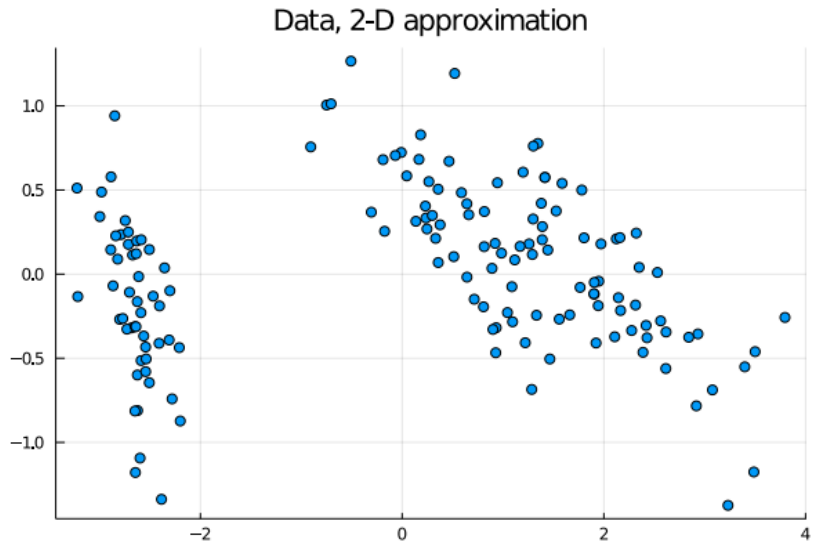
From spectral decomposition we see that a lot of the variance in the distribution can be explained by three base distributions, with two of them very close together, which is exactly correct.

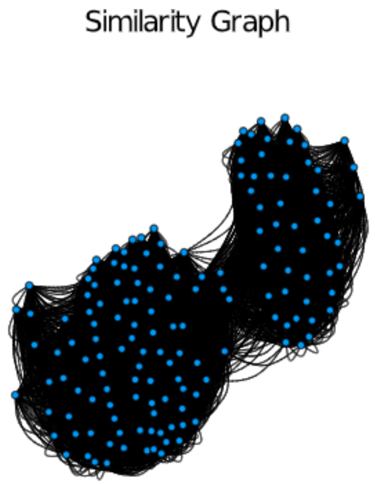
7.3 Real World Examples

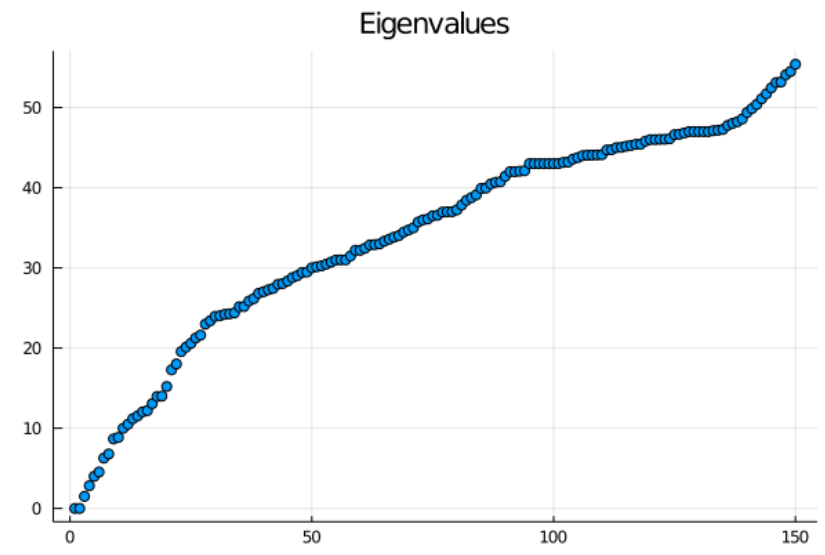
For the real world example, we used the same methods on elements of RDataset.
We tested it on the Iris distribution and the Quakes distrubution. [6] [7]

7.3.1 Iris

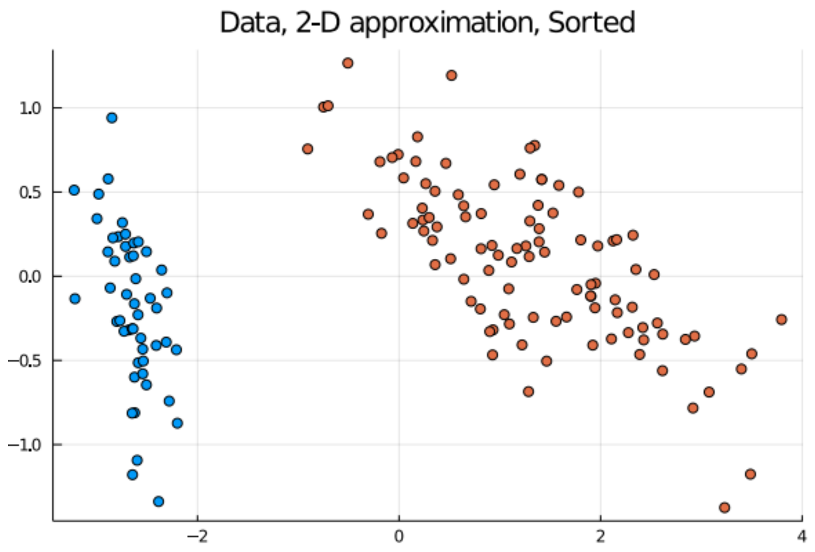
This distribution describes different flower measurements from three species [6]

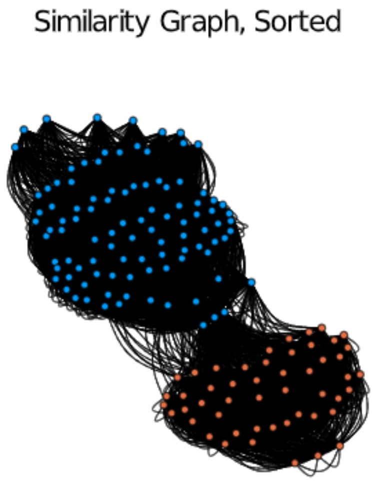






We could distinguish two near zero eigenvalues in the graph



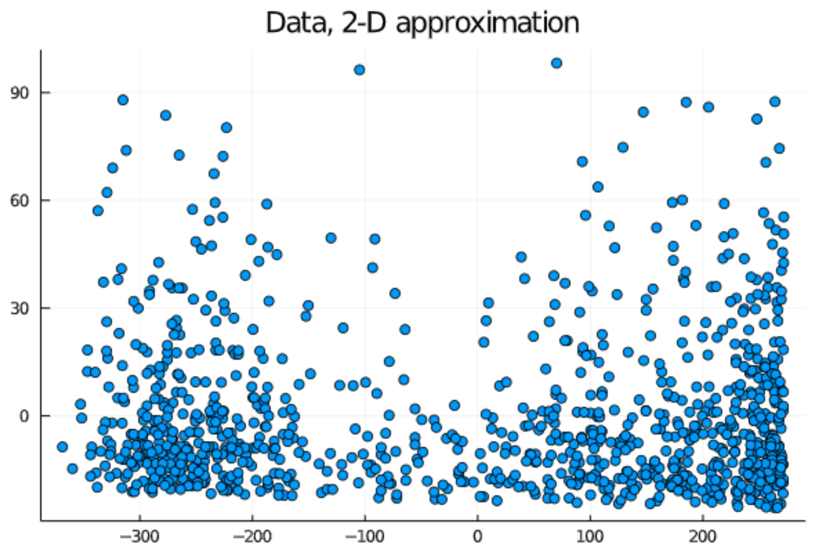


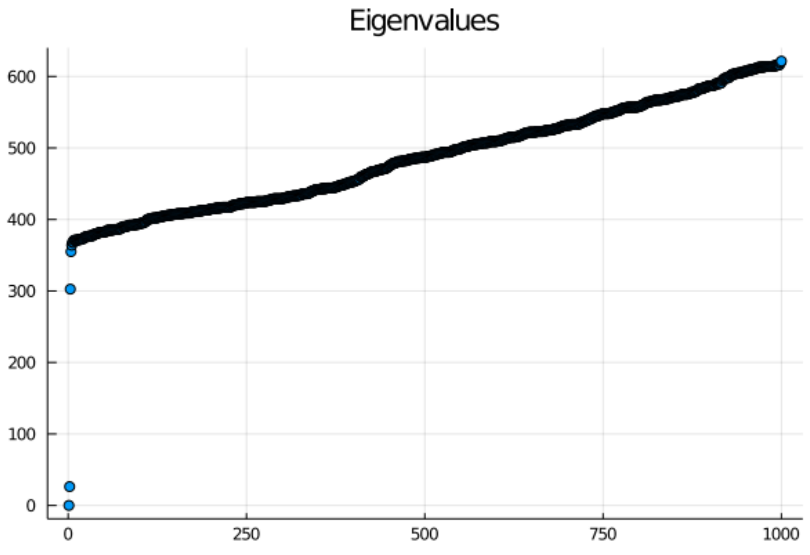
Conclusion:

From spectral decomposition we see that a lot of the variance in the distribution can be explained by two base distributions. While the actual data set has three types of flowers, the algorithm had trouble distinguishing two of them because they were very close together.

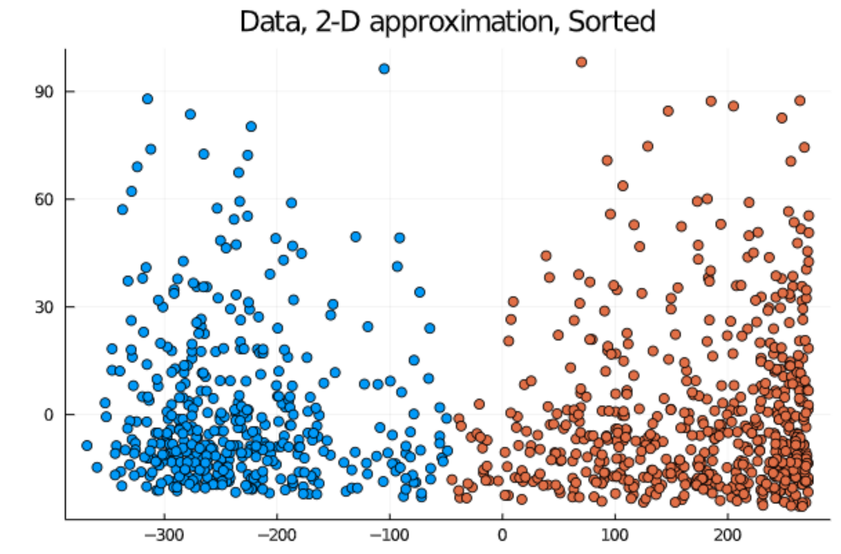
7.3.2 Quakes

This distribution is a record of 1000 earthquakes [7]. Measurements include Latitude, Longitude, depth (kilometers I assume), and magnitude (on the Richter scale probably). When inspecting the data I could find no obvious patterns, since there were way too much data to comprehend. I assumed there would be no clear patterns, which made the result even more surprising (foreshadowing).



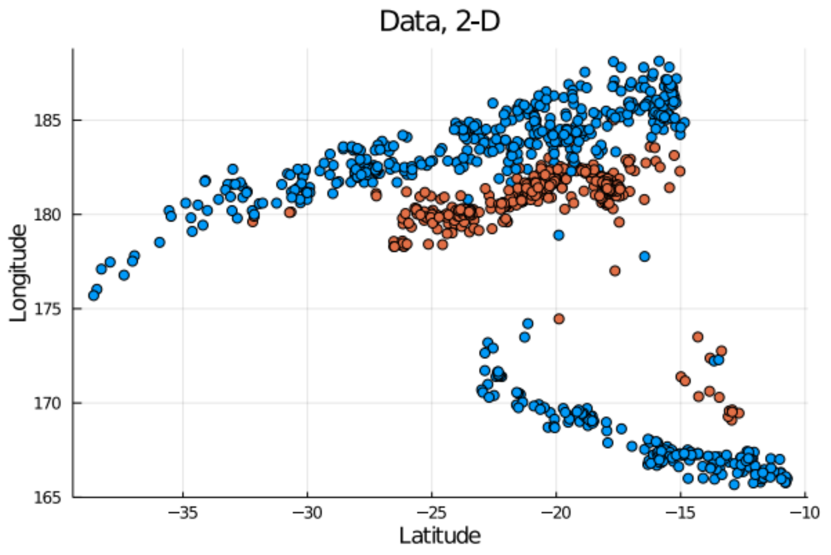


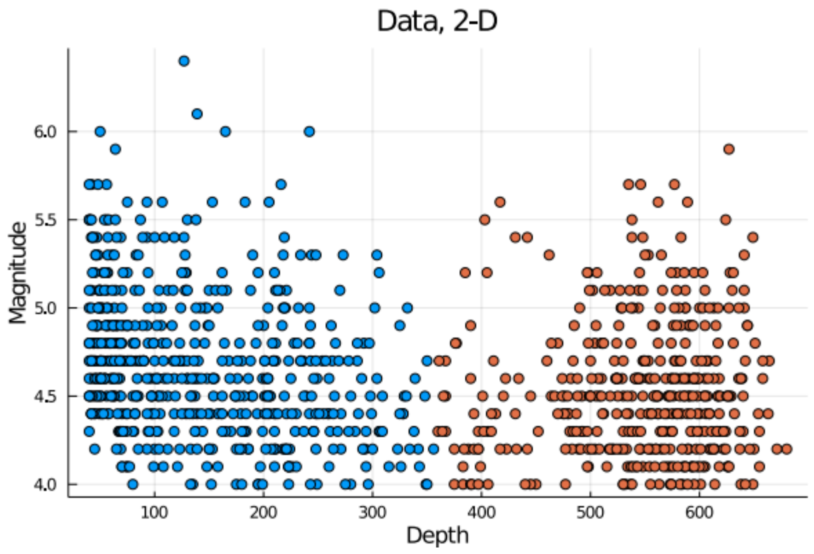
There are two very small eigenvalues close to zero with another one significantly smaller than the rest



It was very interesting that the data was being clumped strongly into 2 clusters. Spectral decomposition of the data implied that there were two different types of earthquakes

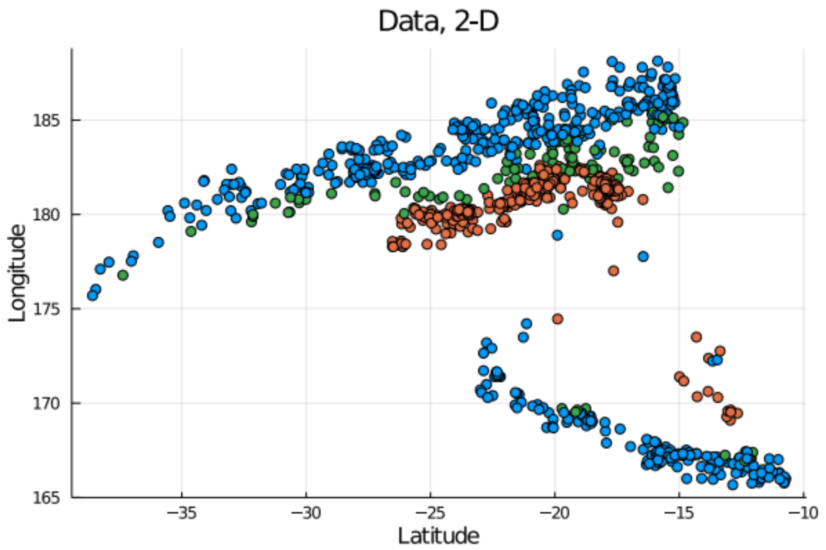
To further explore this, we looked at the locations and the depth/magnitude of these earthquakes.

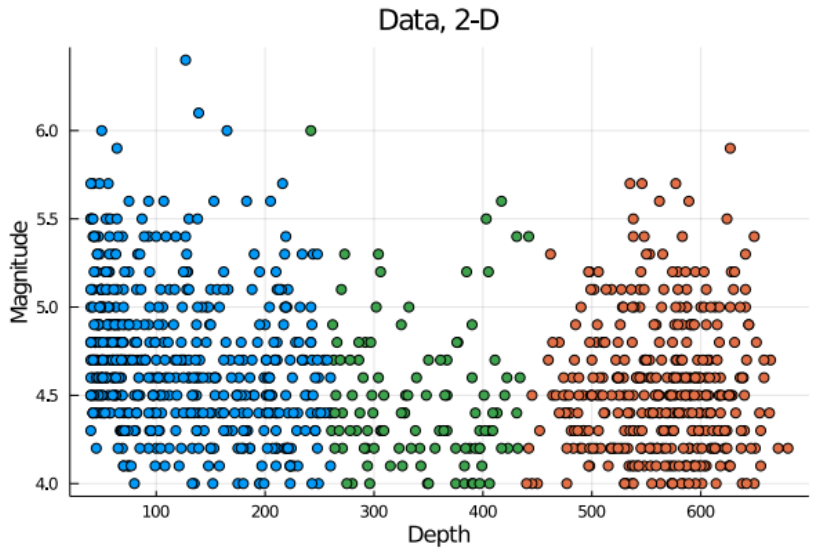




The spectral decomposition of this data implies that there a lot of the variance in earthquake data can be described by two types of earthquake, deep and shallow.

We then clustered the data into three parts to see what was up with the third smaller eigenvalue.





Conclusion:

From spectral decomposition we see that a lot of the variance in the distribution can be explained by three base distributions. It appears that there are three classes of earthquakes, one that is very deep, one that is shallow, and some in the middle. Interestingly, only the shallow ones and the deep ones seem to have higher magnitude. In addition, the locations of the types of earthquakes are very clustered together. From this analysis you could draw conclusions such as if you lived at around -20 Latitude and 100 Longitude, you would be way more likely to be struck by a deep earthquake than any other type.

From a quick google search, I found many science webpages, including a government page, saying that earthquakes are commonly broken down into three types, shallow, intermediate, and deep [4]. According to Alicia Chang, ”Shallow quakes are more damaging” and ”deep quakes are more widely felt” [5]. This would explain why the data groups seemed to peak in magnitude when the earthquakes were deep or shallow.

I don’t care enough about earthquakes to research this more, but the really cool part is that using spectral decomposition on the data gave an accurate description of this scientific classification without any knowledge of the underlying causes.

8 Bibliography

[1] Blum, A., Hopcroft, J., Kannan, R. (2020). Foundations of Data Science. Cambridge: Cambridge University Press. doi:10.1017/9781108755528

[2] Auollay, A., 2020. Spectral Clustering For Beginners. [online] Medium. Available at: https://towardsdatascience.com/spectral-clustering-for-beginners-d08b7d25b4d8 [Accessed 7 December 2020].

[3] Fleshman, W., 2019. Spectral Clustering Foundation and Application. [online] Medium. Available at: https://towardsdatascience.com/spectral-clustering-aba2640c0d5b [Accessed 9 December 2020]

[4] US Geological Survey, 1989. How shallow, deep earthquakes differ. [online] Medium. Available at: https://www.usgs.gov/natural-hazards/earthquake-hazards/science/determining-depth-earthquake?qt-science_center_objects=0qt-science_center_objects [Accessed 9 December 2020]

[5] Chang, A., 2016. How shallow, deep earthquakes differ. [online] Medium. Available at: https://phys.org/news/2016-08-difference-shallow-deep-earthquakes.html [Accessed 9 December 2020]

[6] JuliaStats, 2020. Iris data set. [online] Medium. Available at: https://github.com/JuliaStats/RDatasets.jl/blob/master/data/datasets/iris.rda [Accessed 9 December 2020]

[7] JuliaStats, 2020. Quakes data set. [online] Medium. Available at: https://github.com/JuliaStats/RDatasets.jl/blob/master/data/datasets/quakes.csv.gz [Accessed 9 December 2020]