A Solution Manual and Notes for: Exploratory Data Analysis with MATLAB by Wendy L. Martinez and Angel R. Martinez.

John L. Weatherwax*

May 7, 2009

Introduction

Here you'll find various notes and derivations I made as I worked through this book. There is also quite a complete set of solutions to the various end of chapter problems. This book is quite nice in that it provides a variety of data sets one can use to explore the techniques discussed in the book. One of the nice things about this approach is that often with real data the most straight forward application of textbook procedures do not work. In this book, and with these notes, you can see many places where at first it might appear that a technique will work trivially but in fact they *do not* and can require some modifications or a different approach altogether to be made to work. This *learning by doing* is a very powerful approach and greatly facilitates the learning process.

I would appreciate constructive feedback (sent to the email below) on any errors that are found in these notes. I hope you enjoy this book as much as I have an that these notes might help the further development of your skills in exploratory data analysis.

*wax@alum.mit.edu

Chapter 1: Introduction to Exploratory Data Analysis

Additional Notes

Do to the frequent use of the various datasets this book provides we define "wrapper" functions to help facilitate loading and preprocessing many of the supplied datasets. For example we have the routine `load_iris.m` which is used to load in the iris dataset and possibly apply various preprocessing steps like standardization, sphering, etc depending on the input options passed in. This hopefully will make running the various experiments thought the book much easier in that only a single line of code is needed to input the data in any form desired. In addition, experiments involving apply various preprocessing steps can easily be performed by simply changing the values of the input arguments. Many of the problems in the later chapters are written using these auxiliary routines.

Exercise Solutions

Exercise 1.1 (exploratory data analysis)

Exploratory data analysis is the process of attempting to understand the given data set by performing transformations that hopefully make the data more transparent (in understanding) to the data analyst. Confirmatory data analysis is the process of posing a hypothesis that the data might satisfy and then confirming it with appropriate statistical tests. Because the hypothesis can be generated by exploring the data with EDA the two procedures often go hand in hand.

Exercise 1.2 (visualization of the `leukemia` dataset)

This problem is done in the MATLAB script `prob_1_2.m`. When that script is run it produces a gene expression map for 50 genes and 34 patients having two different type of Leukemia. We have normalized each gene so that across all patients, the average response is zero and the total standard deviation of all responses is one. This is a nice way to *compare* the gene expression profile produced by different patients. When the MATLAB script above is run we obtain the image shown in Figure 1 (left). We see that if we believe the observed gene expressions pattern found in the "training" patterns (the first 39 patients and plotted in Example 1.1) we would conclude that the patients indexed $39 - 50$ (the first 11 in this image) and those indexed $68 - 72$ (the last 5 patients in the above image) appear to have the ALL version of leukemia. In these patients the first 25 genes are highly expressed while the gene expression for last 25 genes are less so. In the "middle" set of patients, it appears that the opposite holds i.e. that the genes indexed $26 - 50$ highly expressed and the earlier numbered ones less so.

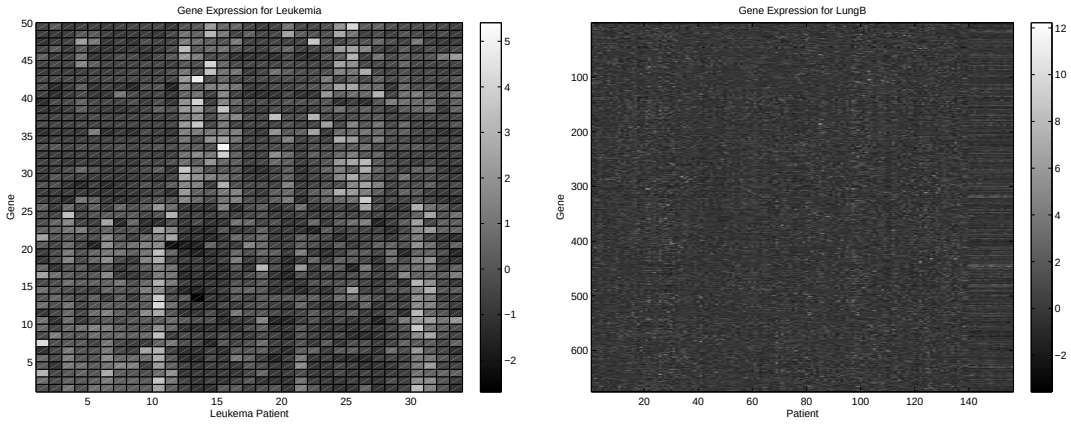

Figure 1: **Left:** The normalized (across patient) gene expression data for the `leukemia` dataset. Notice that the image is broken up into regions above and below about the 25th gene where the different types of Leukema have different gene responses. **Right:** The normalized (across patient) gene expression data for the `lungB` dataset. Note the "stripe" along the right hand side beginning at patient 140 or so of this image. These gene responses belong to normal patients.

Exercise 1.3 (visualization of the `lungB` dataset)

This problem is done in the MATLAB script `prob_1_3.m`. When the MATLAB script above is run we obtain the image shown in Figure 1 (right). In that image we notice that the *last* 17 patients or so have a gene response that is different than the first 139. These column numbers are obtained by looking closely at the various magnitudes in each column. This gives an indication that we maybe able to classify these two classes.

Exercise 1.5 (various standardizations)

The first standardization for one-dimensional data we consider is where we subtract the mean from the data and then divided by the standard deviation. In equation form

$$z = \frac{x - \bar{x}}{s} .$$

This is often called a z-transformation and can be computed directly in MATLAB with the function `zscore`. The two range transformations, given by

$$z = \frac{x}{\max(x) - \min(x)} \quad \text{or} \quad z = \frac{x - \min(x)}{\max(x) - \min(x)} .$$

are computed in the MATLAB function `range_trans.m`. This function takes an optional flag specifying which version of the above formula to use.

The transformation of sphering the data (also called whitening) which is applicable for multidimensional data is performed in the MATLAB function `Whitening_transform`, borrowed

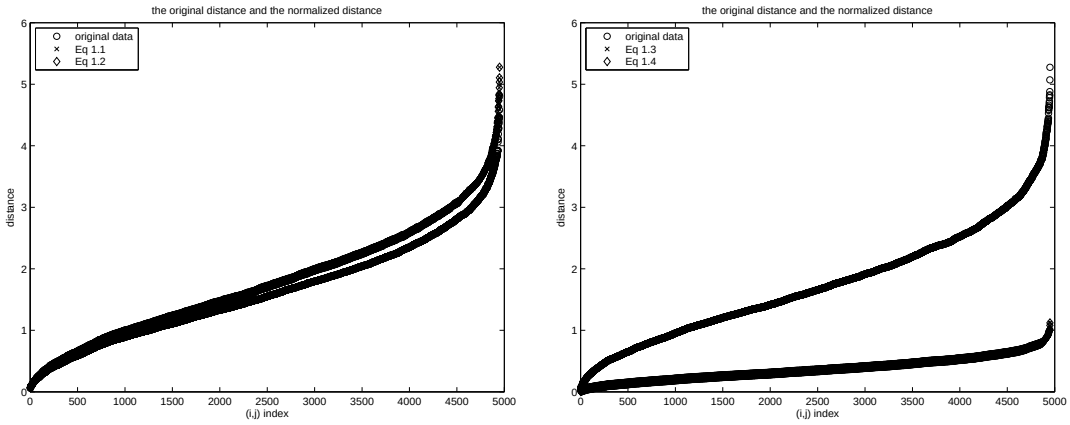

Figure 2: **Left:** The pairwise distances of the original data and the transformed data corresponding to Exercise 1.8. **Right:** The pairwise distances of the original data and the transformed data corresponding to Exercise 1.9.

from the classification toolbox provided in [3]. One caveat with this routine is that the input matrix is passed as a matrix of dimensions $[p, n]$, where p is the number of features and n is the number of samples. This is a transposed convention to the other routines used in this book. These versions are exercised, on random data, in the MATLAB script `prob_1_5.m`.

Exercise 1.7 (robust standardizations)

The suggested robust standardization is implemented in the MATLAB function `robust_norm.m`. One would run this routine in a similar way to the routines presented in Exercise 1.5 above.

Exercise 1.8 (standardizations and the Euclidean distance-part 1)

This problem is performed in the MATLAB script `prob_1_8.m`. When this script is run it produces a plot of the pairwise distances found in the original data and the pairwise distances found in the two scaled versions. The plot produces is reproduced in Figure 2 (left). We sort the pairwise distances so that the smallest distances are plotted first. We recall that for a standard normal the mean is zero so the two requested transformation are effectively the same (one subtracts the mean of our data while the other does not) . We see from the resulting figure that the distances found under the two transformations are indeed the same.

Exercise 1.9 (standardizations and the Euclidean distance-part 2)

This problem is performed in the MATLAB script `prob_1_9.m`. When this script is run we produce a plot in exactly the same way as in Exercise 1.8. We see that the pairwise distances

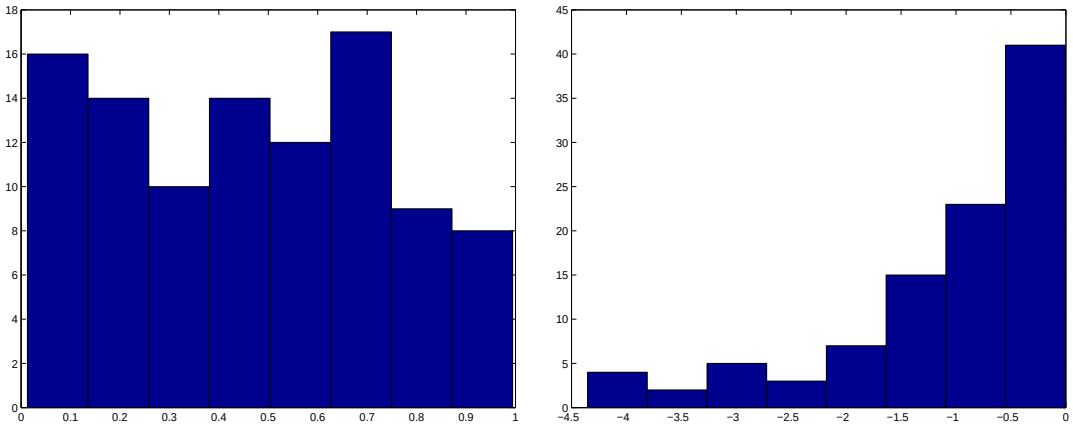

Figure 3: **Left:** A frequency histogram of $n = 100$ uniform random variables (drawn from $[0, 1]$) **Right:** A histogram of the transformed uniform variables $\log(x)$.

for each transformation are the same as expected. This plot can be seen in Figure 2 (right).

Exercise 1.10 (the log transform of uniform variables)

This problem is performed in the MATLAB script `prob_1_10.m`. When this script is run it produces a frequency histogram of the original data in Figure 3 (left). When the log transformation is applied to the data we obtain the frequency histogram shown in Figure 3 (right). We note that these two distribution appear to be very different.

Exercise 1.11 (transforming the `software` data)

This problem is performed in the MATLAB script `prob_1_11.m`. When this script is run it produces a plot of the "original" data (`log(prepsloc)` v.s. `log(defsloc)`), in Figure 4 (left). When the two suggested transformation are applied we obtain the plot shown in Figure 4 (right). The amazing thing is how compactly these two transformation seem to be able to describe the given data. The two transformations given, are to take for x the value of the variable `prepsloc` and then produce two derived variables T_1 and T_2 defined in terms of x by

$$T_1(x) = \log(1 + \sqrt{x}) \quad \text{and} \quad T_2(x) = \log(\sqrt{x})\,,$$

It would be quite remarkable to be able to somehow determine these transformation from the data directly.

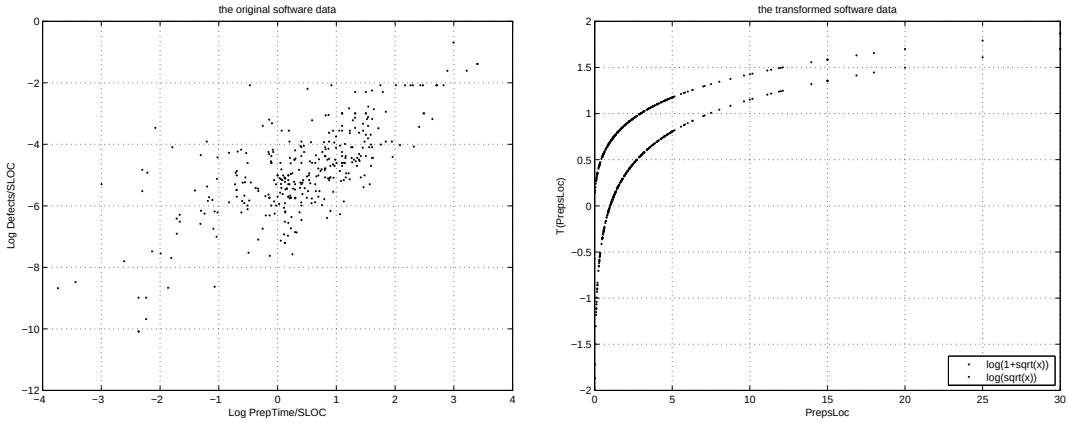

Figure 4: **Left:** A scatter plot of two variable from the `software` data set `log(prepsloc)` and `log(defsloc)`. **Right:** A scatter plot of the transformed variables as defined in the problem solution.

Chapter 2: Dimensionality Reduction - Linear Methods

Additional Notes and Comments

An SVD based idea for feature transformation

The SVD of a $n \times p$ matrix X is defined as the factorization of X such that

$$X = UDV^T .$$

Here V is of dimension $p \times p$, U is of dimension $n \times n$, and D is of dimension $n \times p$. If we desire to produce a matrix $\hat{X}$ that is of a smaller dimension than X i.e. say of dimension $n \times d$ with $d < p$ the book suggests a method of projecting each feature vector x to a smaller dimension via $U_d^T x$. Here U_d is the matrix created from U by considering only the first d columns of U.

An *alternative* method for reducing the dimension of X which is based simply on the dimensions of the matrices in the SVD decomposition is to restrict the matrix V by including only its first d *rows* in the product above. We then recompute an approximate X (say $\hat{X}$) of reduced dimension $n \times d$ as

$$\hat{X} = UD\hat{V}^T .$$

It is this matrix $\hat{X}$ that we attempt to perform classification and data visualization with since it will now be of the smaller dimension $n \times d$. In MATLAB notation this is performed with the following code

```
[U,D,V] = svd(X);
hatV = V(1:d,:);
hatX = U * D * (hatV.');
```

This method is implemented and tested in Exercise 2.16 (b), where it is found that it performs quite well in that there are well observed clusters using code like the above with no feature scaling or ordering on the `lungB` dataset. See the MATLAB code in the script `prob_2_16_lungB.m`.

Exercise Solutions

Exercise 2.1 (PCA on high variance random variables)

In this problem we apply PCA to the suggested data matrix. In this matrix the first feature `x1` has a significantly larger variance than the second two features `x2` and `x3`. The result of this is that when we use the largest eigenvalue associated with the PCA algorithm for

dimensionality reduction (in the covariance based case) we are effectively projecting *only* onto the first dimension i.e. the variance of the `x1` component dominates the projection. The correlation based PCA dimensionality reduction algorithm (because the correlation matrix has much more evenly distributed eigenvalues) specifies its first projection as effectively the average of all three components. This exercises shows that if one component has a much larger variance in comparison to the others the PCA dimensionally reduction algorithm based on the covariance matrix will be dominated by this component. In the example given here, if we compare the projected first component using the PCA algorithm with the unmodified component `x1` for the first five sample points we obtain the following

```
   11.9869     11.9868
    6.5294      6.5309
  -48.5296    -48.5303
   59.5491     59.5526
   14.9668     14.9662
```

From which we see that the two are effectively the same, empirically verifying the above conclusion.

This problem is implemented in the MATLAB script `prob_2_1.m`.

Exercise 2.2 (a scree plot of the `yeast` data)

Before even loading in the `yeast` data we can envision that in general the variances of the individual components in the `yeast` dataset will be much larger than that of the unit variances of the data generated using `randn`. As such, if we use a covariance based matrix for our PCA dimensionality reduction algorithm we will find that the projections are dominated in magnitude by the components with the largest variances. To compare the magnitude of the two sets of eigenvalues we will instead use the correlation based PCA dimensionality reduction algorithm on the `yeast` data. When we do this, and plot the resulting eigenvalues for both the `yeast` data and random data we obtain the scree plot as shown in Figure 5. In that plot we see the two curves cross at an index around 3, suggesting that this data could be projected into *three* dimensions without loss of information regarding the second order statistics of the data. This method of comparing the eigenvalues of the data correlation matrix with the eigenvalues from a dataset consisting of independent standard random variates seems to give equivalent results as the other methods discussed in the text for selecting the number of principal components to keep i.e. cumulative percentage of variance explained, the broken stick, and the size of variance method, all of which give approximately the same number of significant components.

This problem is performed in the MATLAB script `prob_2_2.m`.

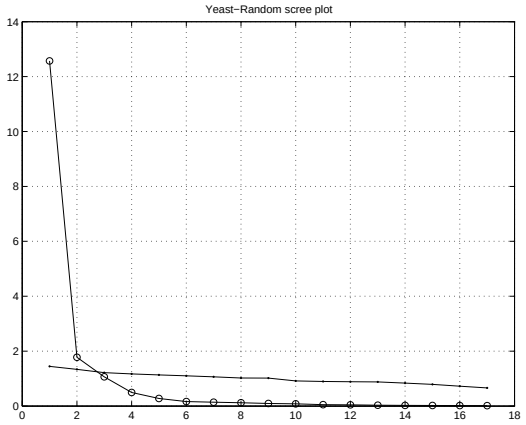

Figure 5: Scree plots of the `yeast` data and a corresponding scree plot for independent random data drawn from standard normals of the same dimension p as the `yeast` data. The crossover point at about three indicates that about only three dimensions need to be kept to explain fully the variance in this dataset.

Exercise 2.3 (experiments with PC scores)

This problem is worked in the MATLAB script `prob_2_3.m`.

Part (a): For a dataset consisting of p independent standard normal random variables we expect the total variance to be p. While considering the entire dataset (with no projection) the total sum of the PCA eigenvalues (under infinite data) should also equal this value. When we run the above script we indeed observe this behavior (as we increase the number of samples n the trace of the covariance matrix becomes closer and closer to $p = 3$). That the eigenvectors are orthogonal follows from the fact that that the covariance matrix is symmetric [11]. That the mean of the principal component scores is zero follows because the projections onto the eigenvectors of the covariance matrix can be thought of as a matrix multiplication. As such, the mean of the *projected* variables would be that projection matrix multiplied onto the mean of the unprojected variables. This later mean is zero so the projected mean would be zero also.

Part (b): Since the covariance matrix has and (i, j)th elements given by

$$E[(X_i - \bar{X}_i)(X_j - \bar{X}_j)]\,,$$

the component means can be thought of as being subtracted from the components and then the expectation taken. The upshot of this is that imposing a mean to our data has no effect on the resulting covariance. Thus the principal axes found in this part of the problem will be the *same* as found in the earlier part of the problem. A difference between this part and the results of Part (a) above does occur when we consider the *mean* of the projected variates. Since we have introduced a uniform shift of the original data, the mean of the projected PCA components will be shifted by the PCA eigenvector transformation multiplied by this mean. In the MATLAB code for this problem we compute both of these expressions and as n increases we see that the two values converge (as the should).

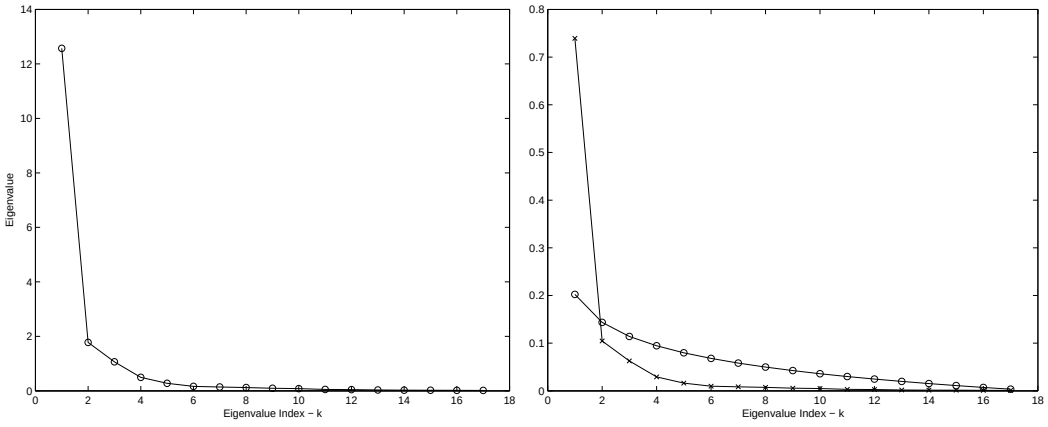

Figure 6: **Left:** A scree plot for the correlation matrix corresponding to the `yeast` data. The crossover point at about three indicates that about only three dimensions need to be kept to explain fully the variance in this dataset. **Right:** A scree plot of the eigenvalues of the correlation matrix corresponding to the `yeast` data and a plot of the function g_k the expected fractional length of the kth order statistics.

Part (c): When we demean and divide each column of our data by the sample standard deviation of that column and the compute the covariance matrix on the resulting data we are effectively doing the correlation calculation. Thus the difference between the correlation matrix and this matrix should be numerically zero. This can be observed by running the above code.

Part (d): For this part of the problem we project the original data onto the eigenvectors of the covariance matrix (i.e. compute the PC scores). We then find the covariance matrix of *this* new dataset. When the above MATLAB script is run we find that the resulting covariance is diagonal with the eigenvalues of the original dataset on the diagonal. These eigenvalues converge to the number 1 as the number of samples n increases. That the off diagonal components are zero means that the projected PC scores are uncorrelated.

Exercise 2.4 (PCA on the `yeast` correlation matrix)

In the MATLAB code `prob_2_4.m` we predict the number of "statistically significant" principal components based on the correlation matrix for the `yeast` data. We begin by plotting a scree plot which is shown in Figure 6 (left). From this plot it appears that four components appear to be a reasonable cutoff. Next we consider the percentage of variance explained. We do this by computing the cumulative sum of the sorted eigenvalues and then dividing these numbers by the total variance (sum of all the eigenvalues) found in this data. When we do this (and express the result as a percentage by multiplying by 100) we obtain the following 17 numbers:

```
73.9415   84.3993   90.6658   93.5881   95.2096   96.1828   97.0169
97.7369   98.3001   98.7833   99.0903   99.3583   99.5411   99.6848
```

```
99.8176   99.9182  100.0000
```

We pick the number of eigenvalues considered significant by selecting a threshold after which these number don't increase significantly. For this data such a threshold might be five or so. Next, to use the broken stick test we compare the given distribution of p eigenvalues with expected lengths one would find from randomly breaking a stick into p pieces. When we break a line segment into p random pieces and then sort them based on their length, the expected fractional length of the kth longest segment is given by the following expression

$$g_k = \frac{1}{p} \sum_{i=k}^{p} \frac{1}{i} .$$

For example $g_1 = \frac{1}{p} \sum_{i=1}^{p} \frac{1}{i}$ is the expected fractional length of the *longest* segment, while $g_p = \left(\frac{1}{p}\right)^2$ is the expected fractional length of the *shortest* segment. If we plot these two expressions on the same set of axis with the eigenvalues we obtain the result in Figure 6 (right). The index where the red curve (corresponding the eigenvalues of `yeast` correlation matrix) falls below the green curve (corresponding to the analytic "broken stick" values) would be an estimate of the number of components that are significant. Using this analysis we would predict one or two significant components. As a final method, we find the number of eigenvalues that are greater than the mean (or some fraction like 0.7 of the mean) of all the eigenvalues. According to this analysis we would take the top three eigenvalues. Thus all of these metrics provide a way of selecting the number of necessary components to retain in a projection. These results are very similar to the results obtained when considering the covariance matrix rather than the correlation matrix.

Exercise 2.5 (eigenvectors of $X^T X$ v.s. those of $X X^T$)

In the MATLAB script `prob_2_5.m` we generate a random data matrix X, construct the two matrices $X^T X$ and $X X^T$, and compute the eigenvalues of both. We generate a matrix X of size $[10, 5]$ which means that the matrix $X^T X$ is of dimension $[5, 5]$ while the matrix $X X^T$ is of dimension $[10, 10]$. The non-zero eigenvalues of each, however, are the *same*. When the above script is run it shows the two computed eigenvalues side-by-side for an empirical proof of this. An example output is shown below where the left column represents the eigenvalues of $X^T X$ while the right column represents the non-zero eigenvalues of $X X^T$.

```
13.7054   13.7054
 1.8604    1.8604
 0.8627    0.8627
 0.4844    0.4844
 0.1679    0.1679
```

When we compute the singular values of the matrix X using the SVD algorithm, from linear algebra we know that these singular values will be the square root of the eigenvalues of both

$X^T X$ or $X X^T$. We can use the MATLAB function `svd` to compute the SVD of the data matrix X and empirically verify this. The resulting singular values (squared) in this case are given by

```
13.7054    1.8604    0.8627    0.4844    0.1679
```

which we recognize as the same as the above.

When we compare the eigenvectors of $X^T X$ and $X X^T$ against the left and right singular vectors we recall that the left singular vectors of X are the columns of the matrix U and the right singular vectors are the columns of the matrix V in the SVD decomposition of the matrix X:

$$X = U D V^T .$$

When X is of dimension $[10, 5]$, U is of dimension $[10, 10]$, D is of dimension $[10, 5]$, and V is of dimension $[5, 5]$. When we run the above MATLAB code we extract the eigenvectors of $X^T X$ and $X X^T$. When we do this we see that, apart from sign, the eigenvectors of $X^T X$ are the *same* as the right singular vectors of X (i.e. the columns of the matrix V). In addition, the first 5 eigenvectors of $X X^T$ are the *same* as the first 5 singular vectors (again ignoring sign).

Exercise 2.6 (the eigendecomposition of the covariance matrix)

Given an dataset matrix X, we begin by computing the sample covariance matrix S. Storing the eigenvectors of S as columns of a matrix, A, and the eigenvalues on the diagonal of a diagonal matrix L we obtain the spectral decomposition of the matrix S as

$$S = A L A^T .$$

Given the eigenvectors of S as columns of a matrix A the eigenvalues can be determined from S via matrix multiplication as

$$L = A^T S A .$$

In the MATLAB script `prob_2_6.m` we empirically verify both of these equations.

Exercise 2.7 (the slopes of the scree plot)

When the MATLAB script `prob_2_7.m` is run it first plots a scree plot of the eigenvalues of the correlation matrix corresponding to the `yeast` dataset. It then plots the difference between consecutive eigenvalues. We see that the jump between the first eigenvalue and the second eigenvalue is quite significant while the difference between the other eigenvalues are less significant.

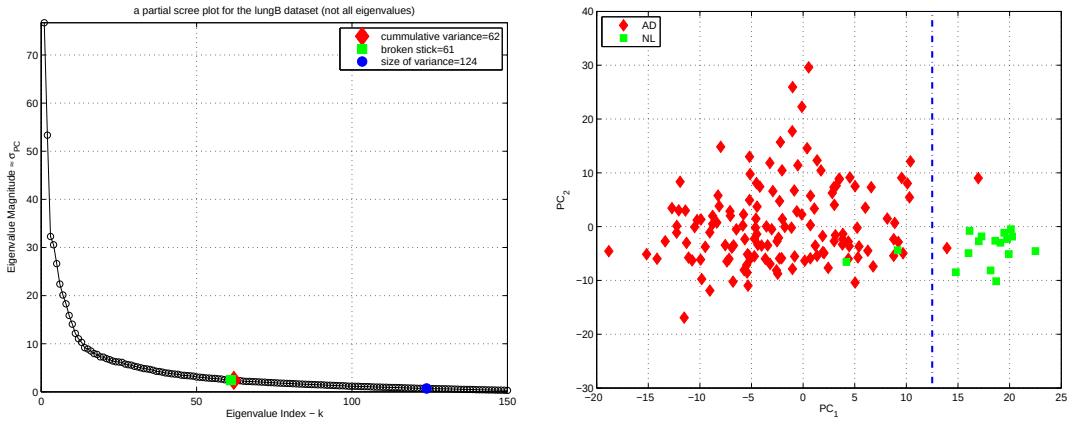

Figure 7: **Left:** A scree plot of a subset of the eigenvalues of the covariance matrix for the `lungB` dataset. **Right:** A plot of the `lungB` data projected onto the first two principal components (PC). Note the cluster separation between the AD and NL classes across the line $\mathrm{PC}_1 \approx 12.5$, in that if $\mathrm{PC}_1 < 12.5$ the sample comes from the class AD while if $\mathrm{PC}_1 > 12.5$ the sample comes from the class NL. The line $\mathrm{PC}_1 = 12.5$ is drawn as an dash-dotted overlay line.

Exercise 2.8 (PCA applied to various datasets)

In this problem we perform PCA dimensionality reduction on various datasets. To help determine the number of significant PCA components, each routine below calls MATLAB functions designed to implement the various methods discussed in the text and aimed at determining the number of significant eigenvectors. Specifically, we have implemented

- A scree plot in the MATLAB function `screePlot.m`
- The broken stick metric for selecting the eigenvalues that have variances larger than chance in the MATLAB function `brokenStick.m`
- The cumulative variance metric which selects a number of principal components such that the percentage of total variance is larger than the requested amount in the MATLAB function `cumVariance.m`
- To select the number of principal components that are larger than the mean of all eigenvalues (by some percentage) we have the MATLAB function `sizeOfVariance.m`

Since a-priori it is unknown if, the datasets from the book have variances of their individual components x_i on widely different scales, without any additional information as to the dimension of the individual components one might think that it would be prudent to use the *correlation* based PCA algorithm for dimensionally reduction. Using the correlation based PCA procedure would make sense in a situation where each component of a data sample was assumed to carry just as much weight (say in a classification task) as the others and it was desired to not have one coordinate dimension dominate the reduction procedure due

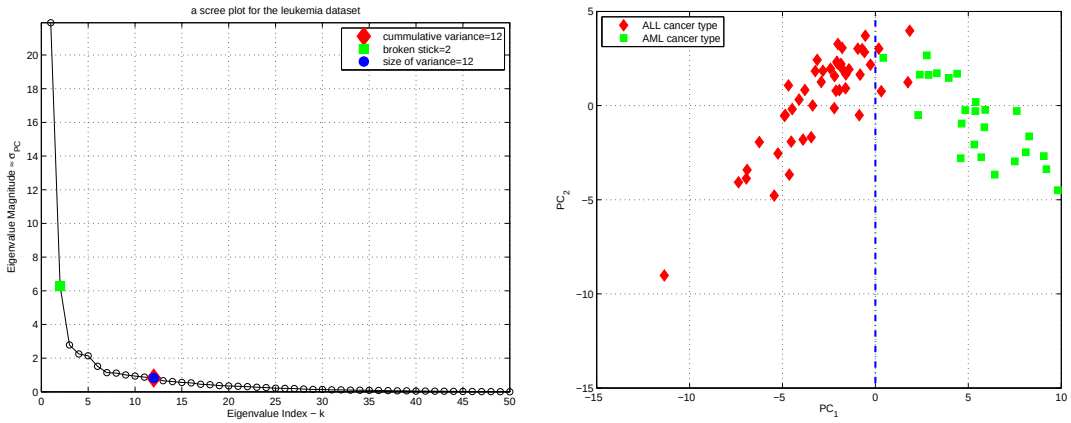

Figure 8: **Left:** A scree plot of the eigenvalues of the covariance matrix for the `leukemia` dataset. **Right:** A plot of the data projected onto the first two principal axis. Note the clear class separation across the line $\text{PC}_1 \approx 0$ in that if $\text{PC}_1 < 0$ the data sample if of cancer type ALL, while if $\text{PC}_1 > 0$ the data sample is of cancer type AML.

only to the *magnitude* of the variable. See Exercise 2.1 for an example where it is shown that if one component has a significantly larger variance than the others it will dominate the PCA dimensionally reduction procedure when used in a covariance matrix based PCA procedure. These considerations hold true usually in a more ideal world where the data analyst believes that there should be *equal* emphasis place on each feature variables and that none should be more heavily weighted than the others. This assumption is in fact most often *incorrect* in practice since often the data analysts uses whatever features are heuristically determined as discriminating and may not know before studying them which ones are the most important. In fact, if one is considering a *classification* problem often the individual features with larger variances carry *more* information than the ones with smaller variances since different classes hopefully give rise to wildly different features. In these cases it can be especially important to use the *covariance* based PCA dimensionality reduction procedure if one wishes to use it for class discrimination, visualization, or clustering. If the given features are chosen intelligently and features without information excluded by default (as is often the case with textbook datasets) this may not be such an important consideration. When this algorithm is used for dimensionality reduction on larger datasets that need dimensionality reduction *before* they can be processed by another algorithm, normalization of the variance can cause significant degradation of the secondary techniques. An example of this effect can be found in Exercise 2.16 where we attempt perform factor analysis on the `lungB` dataset. This dataset has 675 features before any reduction and normalizing the variance across features results in very poor clustering, while the unnormalized features can be clustered quite well. Thus rather than follow a rote scripted procedure for each of these datasets we will experiment with various options and select ones that seem to visually perform best (as far as cluster separation goes).

Part (a): For this part of the problem we will consider the `lungB` dataset. For this dataset we have $n = 156$ measurements of $p = 675$ gene responses. Because the number of features in this dataset is so much greater than the number of samples we expect that without significant feature selection we will never be able to design a robust classifier. Because the

gene response process is a "natural process", in that it occurs in Nature, we might expect that the gene response variable to be distributed as Gaussian random variables. Because of this as a preprocessing step we compute the zscore of each of the gene responses after which we compute the covariance matrix for the given set of samples.

In Figure 7 (left) we see the scree plot for the covariance matrix corresponding to this dataset. It appears that somewhere around 60 features are required to represent the variance in this dataset. When we compute the various metrics describe in the text we find

```
CumVar=        62; BrokenStick=        61; SizeVar=       124
```

When we compute the PCA projections from this covariance matrix we expect that if there are multiple classes that they would be best observed in the clustering of the data points along the most volatile projections, again since along these directions they would be the ones with the most feature variation. Since the PCA decomposition orders the most variable projection earlier than the later ones. In Figure 7 (right) we show the data projected into the first and second eigen-directions along with a color and symbol coding of the type of cancer each point corresponds to. We see that the two types of cancers are clearly delineated as different clusters in this projection.

This code is implemented in the MATLAB script `prob_2_8_a_lungB.m`.

For a second part of the problem we will consider the `leukemia` dataset. For this dataset we have $n = 72$ measurements of gene responses from individual patients each with different types of cancer. Each sample has $p = 50$ features, corresponding to different gene responses.

Following the same preprocessing steps as in the `lungB` dataset, in Figure 8 (left) we present a scree plot of the eigenvalues of the covariance matrix. It appears that somewhat around 10 features are required to adequately represent the variance in this dataset. We compute each of the metrics described in the book and find

```
CumVar=        12; BrokenStick=         2; SizeVar=        12
```

When we compute the PCA projection of this covariance matrix we expect that if there are multiple classes (or subclasses) that they should be best observed in a clustering of the data points in the most volatile projection, again since that would be the one with the most feature variation. In Figure 8 (right) we show the data projected into the first and second eigen-directions along with a color and symbol coding of the type of cancer. We see that the two types of cancers are clearly delineated across the line corresponding to $\text{PC}_1 \approx 0$. See the caption corresponding to that figure for more detail.

This code is implemented in the MATLAB script `prob_2_8_a_leukemia.m`.

Part (b): For this part of the problem we will consider the `oronsay` dataset. For this data it is noted that we have $n = 226$ measurements of $p = 12$ features/observations of the weight

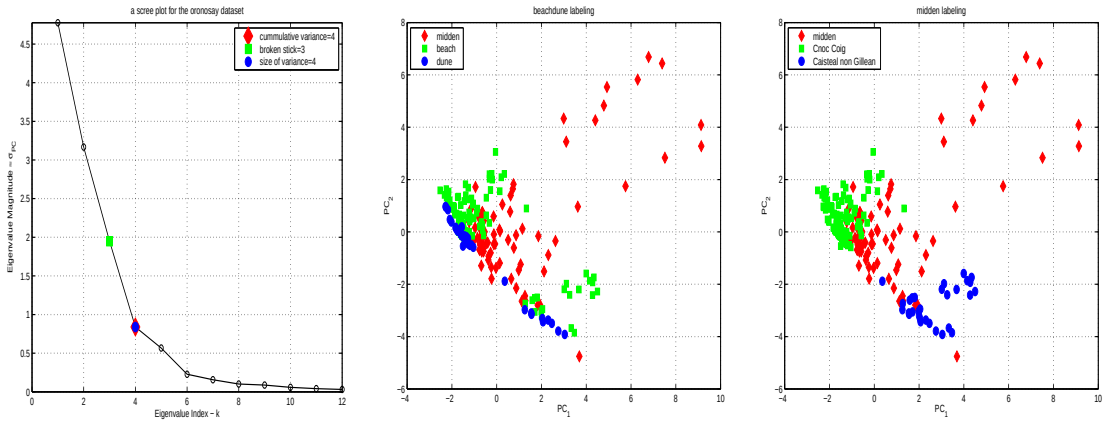

Figure 9: **Left:** A scree plot of the eigenvalues for the covariance matrix of the `oronsay` dataset. The projection of the `oronsay` data onto the first two principal components. **Center:** With labels representing whether the points are beach sand or dune sand. **Right:** With labels representing the location of the sand samples.

of sand of various sizes. Because this data comes from a physical process we preprocess this data by transforming each feature so that it has zero mean and unit standard deviation. In Figure 9 (left) we see the scree plot for the covariance matrix corresponding to this dataset. From this plot it appears that somewhere around 5 features are required to completely represent the variance in this dataset. When we compute the various metrics described in the book and we find

```
CumVar=          4; BrokenStick=          3; SizeVar=          4
```

In Figure 9 (center) and (right) we present the `oronsay` dataset projected on the first two principal directions. In Figure 9 (center) we classify the data according to the type of "place" the sand came from (midden, beach, or dune), while in Figure 9 (right) we classify the data according to the location where the sand was collected (midden, Cnoc Coig, or Caisteal non Gillean). Clustering of these classes somewhat visible from these plots.

This code is implemented in the MATLAB script `prob_2_8_b_ornosay.m`.

Part (c): For this part of the problem we will consider the `sparrow` dataset. For this data it is noted that we have $n = 49$ measurements of $p = 5$ features representing lengths of various body parts of sparrows after a storm. Since these are features that occur in Nature we will preprocess them in the same way as earlier (using a zscore).

In Figure 10 (left) we display a scree plot for the covariance matrix corresponding to this dataset. It appears that somewhere around only 2 features are required to represent the variance in this dataset. Computing the various other metrics describe above and we find

```
CumVar=          3; BrokenStick=          1; SizeVar=          1
```

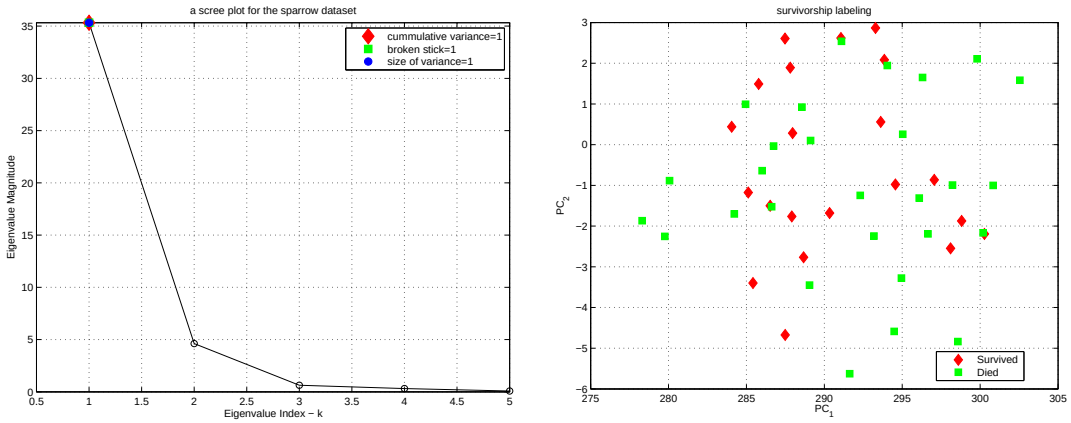

Figure 10: **Left:** A scree plot for the `sparrow` dataset. **Right:** The `sparrow` dataset projected onto its first two principal components. No obvious separation of these points into clusters exists.

In Figure 10 (right) we display a the `sparrow` data projected onto the first two principal components. In addition, we have classified the first 21 samples differently than the others since they correspond to the sparrows who survived the storm damage.

This code is implemented in the MATLAB script `prob_2_8_c_sparrow.m`.

Exercise 2.11 (the singular values of X v.s. the eigenvalues of XX^T and X^TX)

In this problem we use `rand` (or `randn`) to compute a bivariate data matrix X. We then directly compute X^TX, XX^T, and the eigenvalues of both using the MATLAB function `eig`. In addition, we compute the singular value decomposition of X using the MATLAB command `svd`. If we compare the resulting singular values with the square root of the non-zero eigenvalues of both X^TX or XX^T we see that they are equal. See also Exercise 2.5 for a discussion of the eigenvectors of X^TX and XX^T compared with the right and left singular values of the matrix X.

This problem is worked in the MATLAB script `prob_2_11.m`.

Exercise 2.12-13 (experiments with factor analysis using `factoran`)

We begin by applying the MATLAB function `factoran` to the given stock return data. A factor decomposition of the vector $\mathbf{x}$ means that the p component observed random variables $x_1, x_2, \cdots, x_p$ can be written in terms of a smaller number say $d < p$ unobserved *latent*

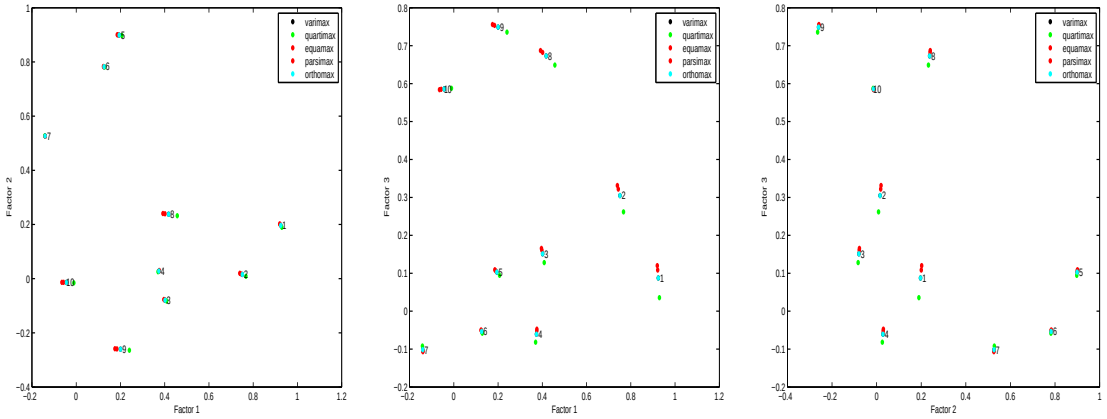

Figure 11: A sampling of the possible factor loadings for the `stockreturn` data for several types of rotations. **Left:** The return data projected into the space spanned by the first and second factors. **Center:** Projected into the space spanned by the first and third factors. **Right:** Projected into the space spanned by the second and third factors. Clustering of the points corresponding to the type of company is clearly evident from these plots.

variables (or common factors), $f_1, f_2, \cdots, f_d$ and p error terms $\epsilon_1, \epsilon_2, \cdots, \epsilon_p$ as

$$\begin{aligned} x_1 &= \lambda_{11} f_1 + \lambda_{12} f_2 + \cdots + \lambda_{1d} f_d + \epsilon_1 \\ x_2 &= \lambda_{21} f_1 + \lambda_{22} f_2 + \cdots + \lambda_{2d} f_d + \epsilon_2 \\ &\vdots \\ x_p &= \lambda_{p1} f_1 + \lambda_{p2} f_2 + \cdots + \lambda_{pd} f_d + \epsilon_p \,. \end{aligned}$$

The λ_{ij} are called factors loadings and along with the error terms ϵ_i are different for each sample, while the the common factors f_j are constant for *all* the variables considered. Since the computation of the factor loadings λ_{ij} is *not* uniquely defined from the above decomposition there are several different specifications available via options to the `factoran` function.

In this dataset, the first four columns of returns (with indexes $\{1, 2, 3, 4\}$) correspond to technology companies, the next three columns (with indices $\{5, 6, 7\}$) correspond to financial companies, and the final three columns (with indices $\{8, 9, 10\}$) correspond to retail companies. From this knowledge we might expect or hope for clustering of those points.

By default the `factoran` function generates its output according to a *varimax* rotation criterion. In this exercise we are asked to experiment with alternate rotations by specifying different options to the `factoran` function. Specifically, we choose to consider all of the available possible choices: no rotation, varimax, quartimax, equamax, parsimax, orthomax, and promax rotations. For each type of rotation, a three term factor decomposition is computed. We then plot projections of the factor loadings $\lambda_{\cdot,j}$ for various configurations of j (chosen from its possible range $j = 1, 2, 3$).

When we consider the scatter plots corresponding to the various type of projections we notice that the options of *no rotation* and *promax* are significantly different than the others. Because of this we plot these two factor decompositions on a different set of axis. To

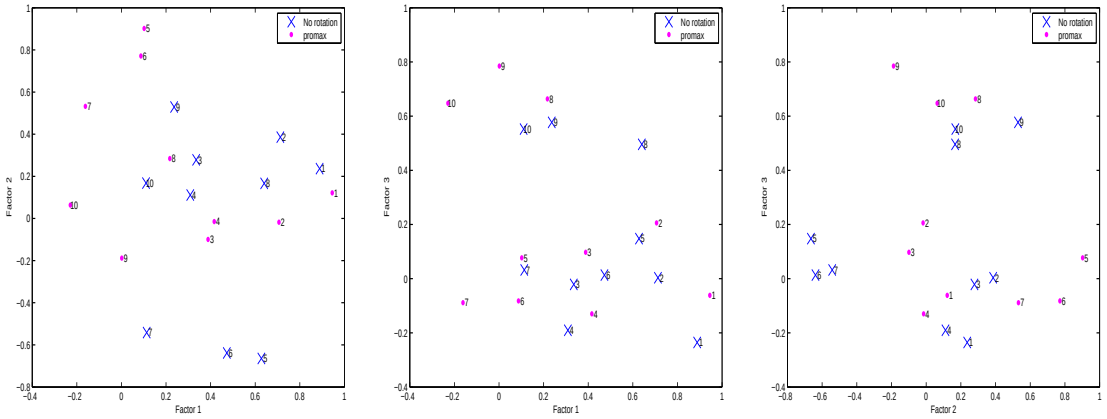

Figure 12: The possible factor loadings for the `stockreturns` data, under the no rotation and promax rotation. **Left:** We project our data into the space spanned by the first and second factor loadings. **Center:** The space spanned by the first and third factor loadings. **Right:** The space spanned by the second and third factor loadings.

begin with however, in Figure 11, we plot the projections corresponding to the: varimax, quartimaax, equamax, parsimax, and orthomax rotation options. We see that all of these projections produce results that are very similar to each other. There is definitely clustering of the data points. Depending on the projection space we see clustering of the points $5, 6, 7$ the points $\{1, 2\}$, and the points $\{8, 9, 10\}$. The points $\{3, 4\}$ are more ambiguous in that they could be assigned to the cluster containing $\{5, 6, 7\}$ in the Factor 1-Factor 3 projection or to the cluster containing the points $\{8, 9, 10\}$ in the Factor 1-Factor 3 projection.

In Figure 12, we plot the rotation options of no rotation and the promax rotation. We see that these projections produce results that are very similar to the earlier ones in that there is definitely clustering of the data points. In addition, the Factor 2-Factor 3 projections nicely group the companies by their type.

These two exercises are implemented in the MATLAB script `prob_2_12_N_13.m`.

Exercise 2.14 (projecting the stock data into two and four factors)

In this problem we perform factor analysis on the `stockreturns` dataset assuming two and four factors respectively. We first find the factors for the `stockreturns` data assuming two factors and the default varimax rotation. The results of this analysis are shown in Figure 13. We easily see clusters in the set of points $\{5, 6, 7\}$, $\{3, 4, 9, 10\}$, and $\{1, 2, 8\}$.

In the Figure 14 we show the six possible two-dimensional projections when we compute a factor model on the `stockreturns` dataset.

This problem is implemented in the MATLAB script `prob_2_14.m`.

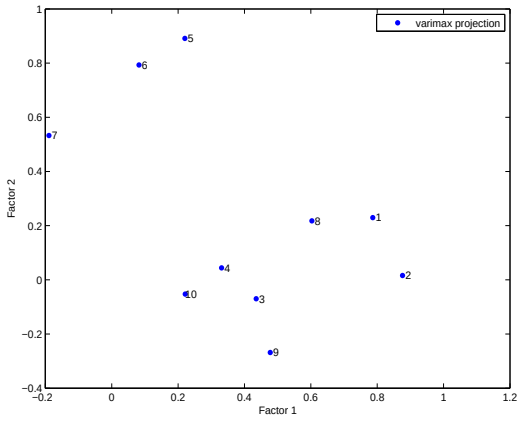

Figure 13: Computation of a two factor model for the `stockreturns` data. Clustering of points in this space is clearly visible.

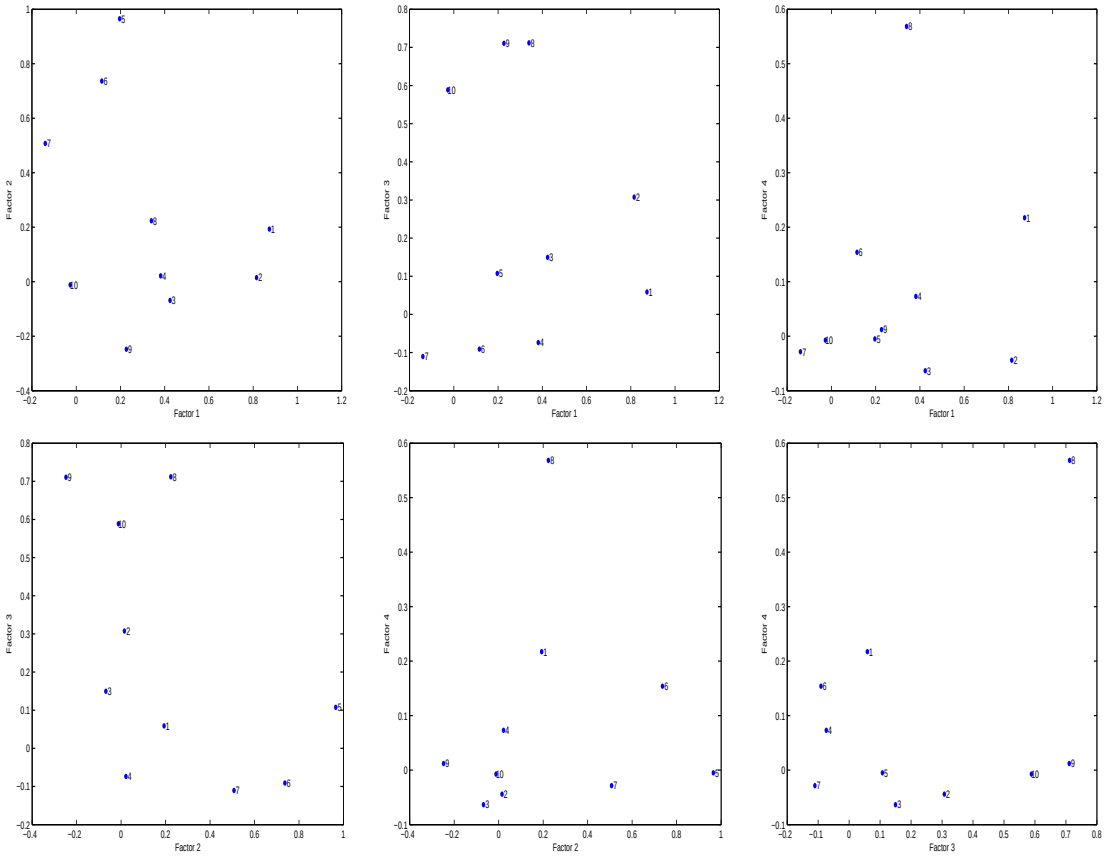

Figure 14: Plots of the projections from a four factor model for the `stockreturns` dataset.

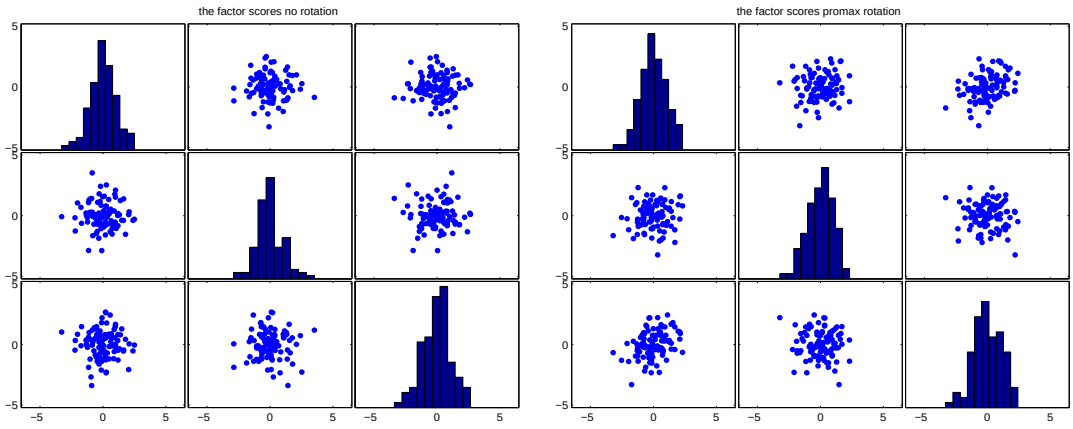

Figure 15: Computation of the factor scores produced by the `factoran` function under no rotation (left) and promax rotation (right) for the `stockreturns` dataset.

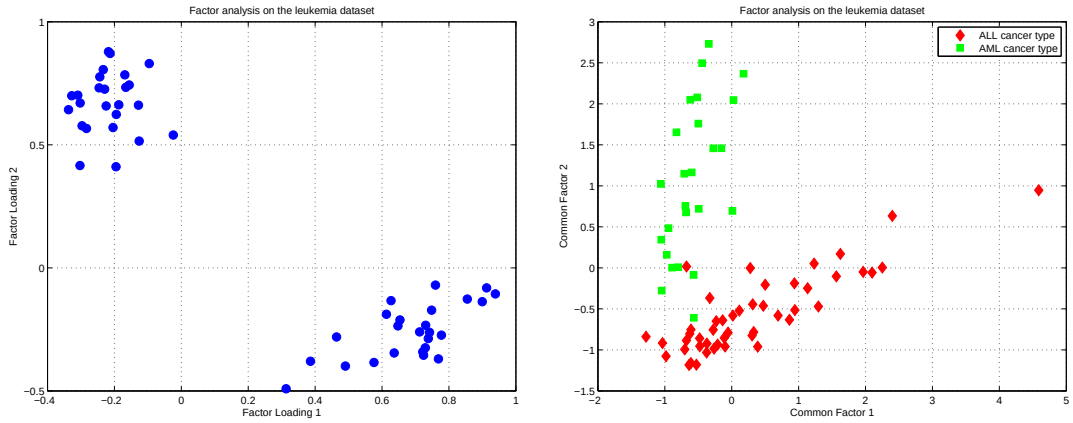

Figure 16: **Left:** Plots of the factor loadings produced by the `factoran` function under the default rotation (varimax) for the `leukemia` dataset and assuming a two factor model. **Right:** Plots of the common factors for the `leukemia` dataset. Note the strong clustering.

Exercise 2.15 (factor scores from the `factoran` function)

In Figure 15 we plot the factor *scores* produced by the `factoran` function on the `stockreturn` dataset under the assumption of a two factor model.

Exercise 2.16 (factor analysis on gene expression datasets)

Part (a): In this problem we perform factor analysis in the patient domain on the `leukemia` dataset by looking for *two* factors that could explain the given data. In general is to a good idea to start an EDA procedure by looking for only *two* different classes before trying to find more classes. If you cannot find at least two classes than there is little hope of finding more. When we use the MATLAB function `factoran` on the `leukemia` dataset we produce

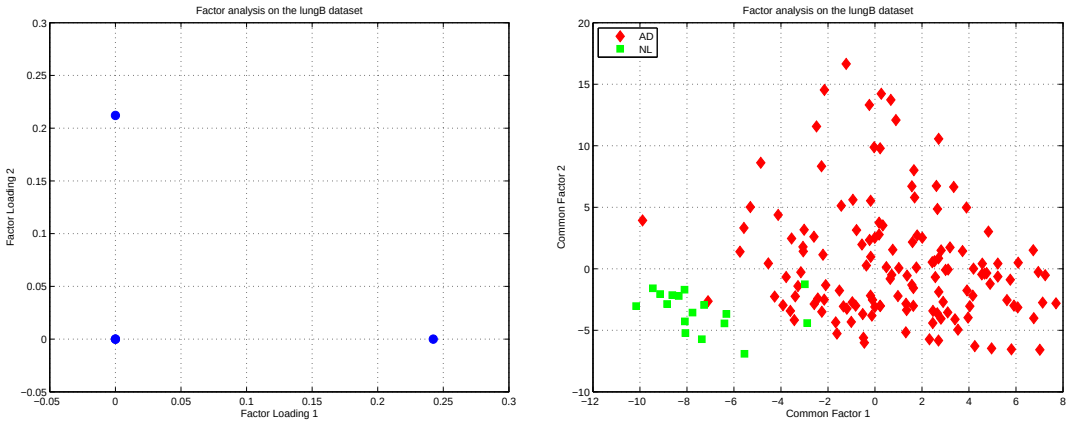

Figure 17: **Left:** Plots of the factor loadings produced by the `factoran` function under the default rotation (varimax) for the `lungB` dataset. **Right:** Plots of the common factors for the `lungB` dataset. Note the nice clustering, of the NL type of cancer represented by the square symbols in the lower left corner.

the figures seen in Figure 16. Notice the strong clustering in both the factor loading and the common factor domains.

This problem is implemented in the MATLAB script `prob_2_16_leuk.m`.

Part (b): For this part of the problem we attempt to apply clustering in the patient domain for the `lungB` dataset. Due to the large feature space 675 we begin this process by *sorting* the columns of the data in decreasing order by their variances. Since each feature corresponds to a gene response this variance ordering should help in the interpretation of the eigenvectors of the resulting covariance matrix in that the coefficients should roughly decay in magnitude with their index. Because the feature set is still quite large at 675 and results in a singular covariance matrix the `factoran` function crashes when passed the entire dataset we next perform a PCA dimensionality reduction procedure (down to a dimension of 20) on the resulting full 675 feature covariance matrix. For this dataset we found though experimentation that it was much better to *not* perform a z-score transformation on the resulting gene responses (i.e. the columns) before computing the covariance matrix. The results of the factor decomposition are displayed in Figure 17, where we can see nice clustering of the two cancer types. The NC type tightly clustered in the lower left corner.

This problem is implemented in the MATLAB script `prob_2_16_lungB.m`.

Exercise 2.17 (`idpettis` on independent random data)

For this problem we attempt to run the `idpettis` on two dimensional independent random data. We generate n random pairs with the `randn` function, call the `idpettis` function on this data and repeat this procedure for a given number of Monte-Carlo iterations. We generate sample sizes n of 10^l pairs for $l \in \{1, 2, 3, 4\}$. We expect to obtain an intrinsic

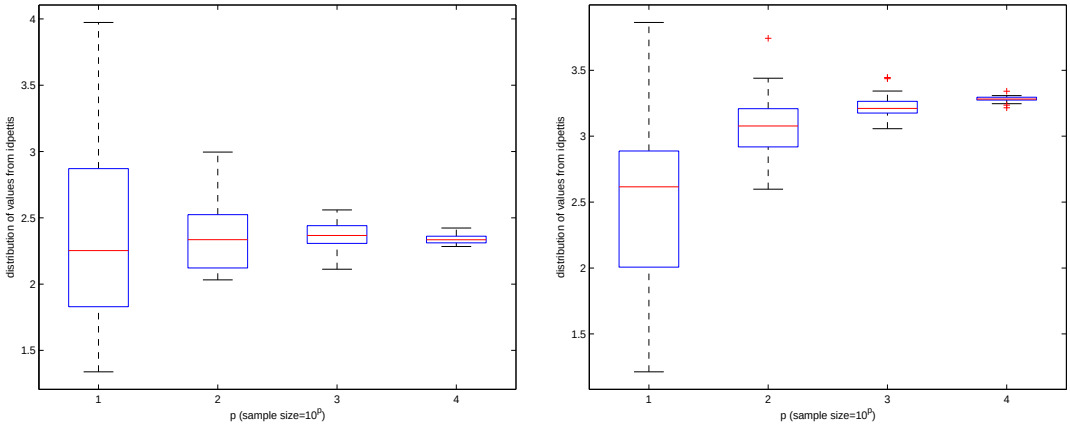

Figure 18: **Left:** A box plot representing the distribution of the intrinsic dimension produced by the `idpettis` MATLAB function when run on two feature data consisting of independent random variables. **Right:** A box plot representing the distribution of the intrinsic dimension estimates produced by the `idpettis` MATLAB when run on Fukunaga's Gaussian pulse.

dimensionality of 2 on random data like this since each component is independent. The results of this experiment are shown in Figure 18 (left), where we see that as the number of samples used to estimate the intrinsic dimensionality increases our estimate converges to the value 2.4.

This problem is implemented in the MATLAB code `prob_2_17.m`.

Exercise 2.18 (estimating the intrinsic dimensionality of a Gaussian pulse)

For this problem we generate random sample from the given waveform, use the MATLAB function `idpettis` to estimate their intrinsic dimensionality and repeat this experiment for several Monte-Carlo trials. When we do this we find the distribution of values from the `idpettis` function given in Figure 18 (right). There we see that under infinite samples size we would obtain a value approximately equal to 3.4, which is quite close to the expected value of 3.

This problem is worked in the MATLAB code `prob_2_18.m`.

Exercise 2.19 (the intrinsic dimensionality of the `yeast` dataset)

The `idpettis` function finds an intrinsic dimensionality of 13.75 for the `yeast` dataset. The PCA approach taken in Exercise 2.3 concluded that approximately three principal components were needed.

This problem is worked in the MATLAB code `prob_2_19.m`.

Exercise 2.20 (the intrinsic dimensionality of various datasets)

In this exercise we will compute the intrinsic dimensionality of each of these datasets and compare them to the dimension suggested by a PCA decomposition computed and discussed in Exercise 2.8.

Part (b): We compute the intrinsic dimensionality of the `ornosay` dataset to be 4.85, while PCA dimensionality reduction suggested a value of approximately $3 - 4$.

Part (c): We compute the intrinsic dimensionality of the `sparrow` dataset to be 4.91, while PCA dimensionality reduction suggested a value of approximately 1.

Part (d): We compute the intrinsic dimensionality of the `lungB` dataset to be 26.71, while PCA dimensionality reduction suggested a value of approximately 60.

We compute the intrinsic dimensionality of the `leukemia` dataset to be 12.20, while PCA dimensionality reduction suggested a value of approximately 12.

The estimates of the intrinsic dimensionality appears to be close in some cases and rather different in others. This problem is worked in the various MATLAB codes `prob_2_20_*.m` specific to each dataset.

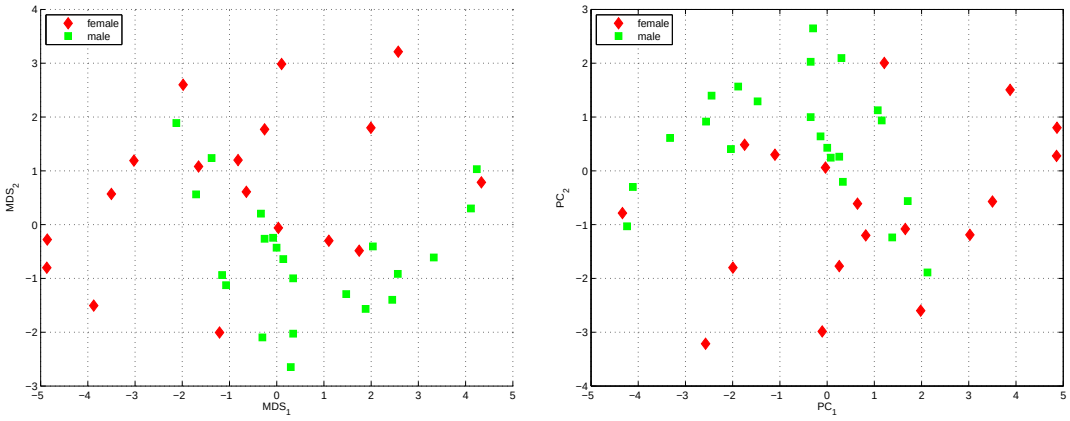

Figure 19: **Left:** A projection of the `skulls` dataset onto the first two axis produced the classical multidimensional scaling algorithm. **Right:** The projection onto the first two principal components of the `skulls` dataset. These two projections are the *same* expect for a rotation of 180 degrees counterclockwise of the MDS result. Note that a "diagonal" line (not shown) through the first figure would do a good job separating the data points into male and female classes.

Chapter 3: Dimensionality Reduction - Nonlinear Methods

Exercise Solutions

Exercise 3.1 (multidimensional scaling on the `skulls` dataset)

In this problem we perform classical multidimensional scaling on the `skulls` dataset with the MATLAB statistical toolbox command `cmdscale`. We then compare these results with the PCA dimensionality reduction procedure. In Figure 19 (left) we see the result of the multidimensional scaling while in Figure 19 (right) we see the results of the PCA dimensionality reduction. For this problem we used the MATLAB function `pdist` (with the argument "euclidean") to generate the matrix of dissimilarities to be used by the `cmdscale` command. Because of this, the multidimensional scaling and the PCA dimensionality reduction procedure result in the *same* projections [4].

This problem is implemented in the MATLAB script `prob_3_1.m`.

Exercise 3.2 (more multidimensional scaling on the `skulls` dataset)

In this exercise we consider the non-metric multidimensional scaling algorithm (implemented in the EDA toolbox function `nmmds.m`) and the SMACOF (scaling by majorizing a compli-

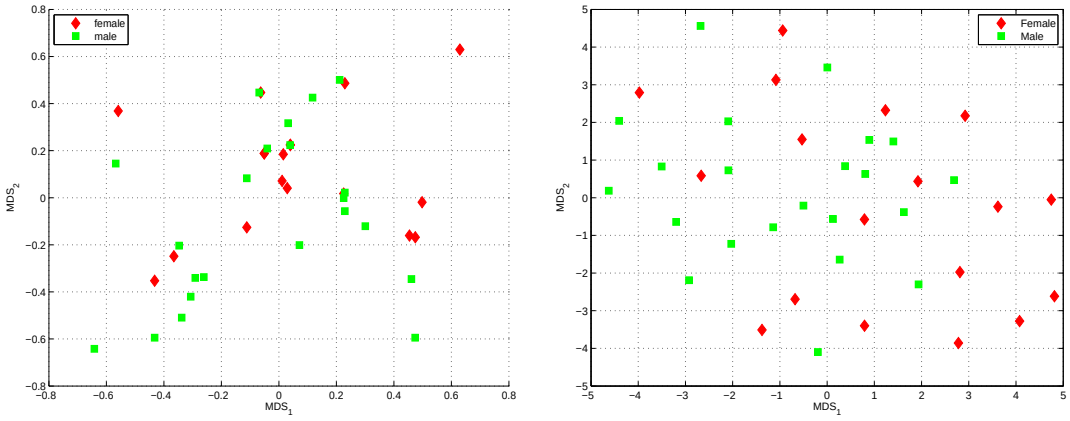

Figure 20: **Left:** A projection of the `skulls` dataset onto the first two axis produced from the non-metric multidimensional scaling algorithm `nmmds`. **Right:** The projection onto the first two axis produced from the SMACOF algorithm on the `skulls` dataset.

cated function) algorithm presented in Example 3.2 of the book on the `skulls` dataset. The results from these two algorithms along with male/female labels are shown in Figure 20 (left) and (right) respectively. The SMACOF results appear more random at least with respect to male/female labeling.

This problem is implemented in the MATLAB script `prob_3_2.m`.

Exercise 3.3 (multidimensional scaling into the third dimension)

For this problem we perform classical multidimensional scaling (MDS) on the `matchbpm` dataset and then view the resulting projection in three dimensions. When one performs this exercise the resulting three dimensional plot very clearly show clustering in the projection space. Using the MATLAB interactive "rotate" function on the figure window allows one to change the resulting view point and clearly observe the clusters in three dimensional space.

This problem is implemented in the MATLAB script `prob_3_3.m`, which when run allows the visualization suggested above to be easily performed.

Exercise 3.4 (SMACOF on the `oronsay` dataset)

For this exercise we apply the SMACOF algorithm on the `oronsay` dataset. The results from applying this algorithm is presented in Figure 21. These plots look qualitatively very similar to the ones presented when we did a PCA decomposition in Exercise 2.8 (b), in that clustering seems to be present in the labeling by location (i.e. midden labeling).

This problem is implemented in the MATLAB script `prob_3_4.m`.

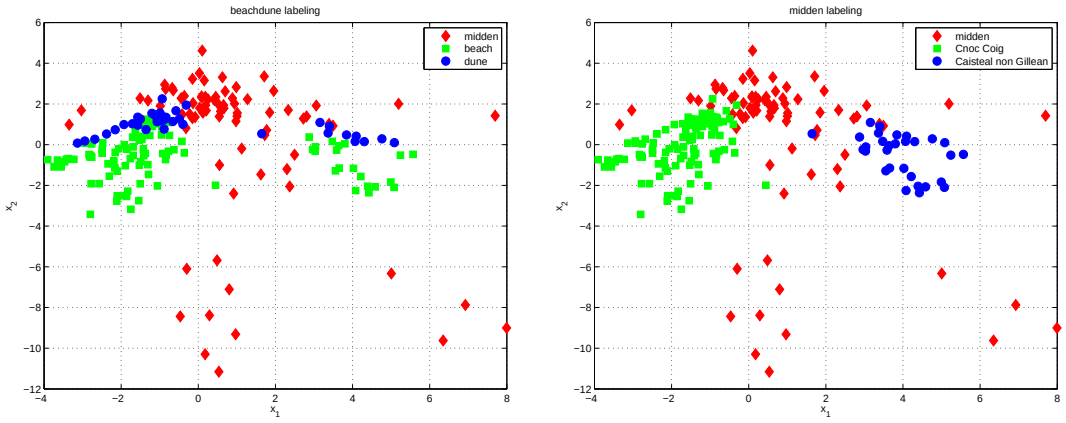

Figure 21: The projections for found by the SMACOF algorithm when applied to the `oronsay` dataset. **Left:** With labels representing the "beachdune" labeling. **Right:** With labels representing "midden" labeling. Notice that vertical lines (not shown) at approximately $x_1 \approx 0$ and $x_1 \approx 3$ in the midden labeling would do a good job of separating the sand samples according to where they were collected.

Exercise 3.7 (the command `som_show`)

The command `som_show(sM)` produces a U-matrix for all of the individual components on one graph. The command `som_show(sM,'comp',i)` produces a U-matrix for the i-th component.

These commands are exercised in the MATLAB script `prob_3_7.m`.

Exercise 3.8 (beachdune labeling of the `oronsay` dataset)

In this exercise we modify Example 3.6 from the text to use the beachdune labeling provided with the `oronsay` dataset. See the MATLAB script `prob_3_8.m` for an implementation of this.

Exercise 3.9 (plotting the GMT modes)

In this problem we plot the *modes* produced by the GTM algorithm when applied to the `oronsay` dataset. To aid comparison of this result with that when the means were considered, the modes are plotted on the *same* graph as the means. The result of this study is shown in Figure 22 (left). We see that many of the points are almost identical and that the two representation are quite similar.

This problem is implemented in the MATLAB script `prob_3_9.m`.

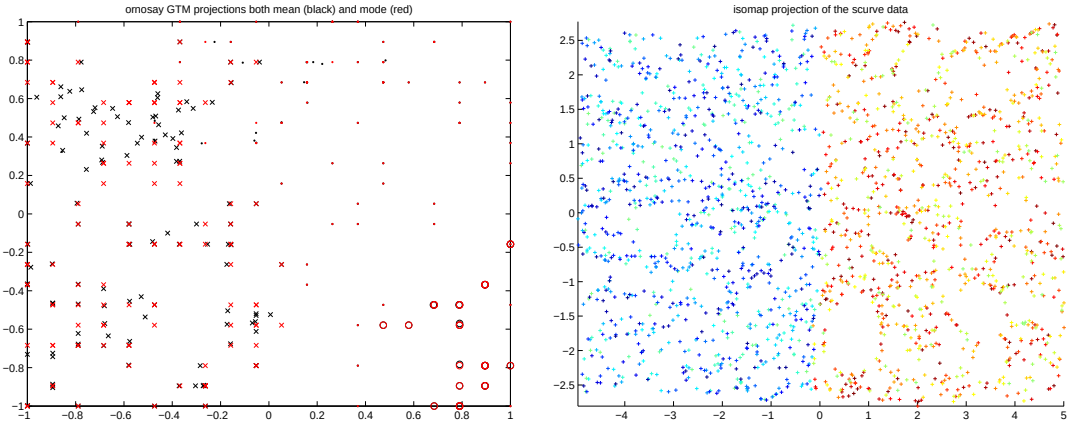

Figure 22: **Left:** The means represented by black crosses and the modes represented by red crosses found by the GTM projection when applied to the `oronsay` data. **Right:** The ISOMAP projection of the `scurve` dataset. Note the "hole" in the lower half of the image.

Exercise 3.10 (using the MATLAB command `mdscale`)

A similar exercise is qualitatively performed in Exercise 3.1. Please see the results there.

Exercise 3.11 (ISOMAP on the `scurve` dataset)

In this problem we apply the ISOMAP procedure to the `scurve` dataset. The resulting scatter plot from this procedure is shown in Figure 22 (right). Note the large hole in the lower half of the image (centered approximately on the coordinate $(0.5, -1.75)$ and the scattered holes throughout.

This problem is implemented in the MATLAB script `prob_3_11.m`.

Exercise 3.12 (LLE on the `swissroll` dataset)

In this problem we apply the Local Linear Embedding (LLE) algorithm to the `swissroll` dataset. The resulting scatter plot from this procedure is shown in Figure 23. It was noted that when running the `lle` command the choice of the desired output dimension `d` seemed significantly affect the resulting two-dimensional projections. Through experimentation we found that a value of $d = 3$ seemed to produce the most interesting result. Using this value the resulting two dimensional projection had a clear hole in it.

This problem is implemented in the MATLAB script `prob_3_12.m`.

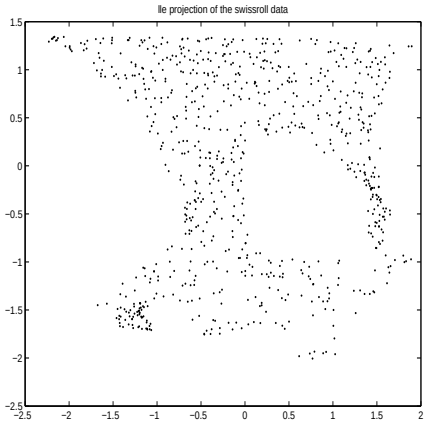

Figure 23: The Local Linear Embedding (LLE) projection of the `swissroll` dataset.

Exercise 3.13 (the ID of the `swissroll` and `scurve` datasets)

When we run the `idpettis` program to compute the intrinsic dimensionality (ID) of the `scurve` dataset we obtain a result of 2.0862. For the `swissroll` dataset we obtain an intrinsic dimensionality of 2.0849. From the construction of these two datasets we know that they are intrinsically two dimensional, which the `idpettis` algorithm is correctly recognizing.

This problem is implemented in the MATLAB script `prob_3_13.m`.

Exercise 3.14 (application of nonlinear dimensionality routines on various datasets)

Because classical multidimensional scaling using a Euclidean distance for a dissimilarity measure is *equivalent* to a PCA projection [4], in this exercise we have two choices to generate different projections from the ones already considered. The first is to use the EDA toolbox function `nmmds.m` to compute a non-metric multidimensional scaling. The second is to use the MATLAB function `mdscale` which can perform both metric and non-metric multidimensional scaling with non-dimensional scaling as the default. For simplicity and computational speed we will use the MATLAB function `mdscale`.

One thing about these runs must be noted. It was found that when running the Local Linear Embedding algorithm, `lle.m`, the maximum dimension of the desired embedding played a major factor in the qualitative projection obtained when we projected our data into two dimensions. For the results presented here, the final maximal dimension was selected that produced the most *interesting* two dimensional projection. This dimension was determined via trial-and-error.

Part (b): First, we apply the various non-linear dimensionality reduction techniques discussed in this chapter to the `yeast` dataset. The results from this set of experiments can be found in Figure 24. This problem is implemented in the MATLAB script `prob_3_14_b_yeast.m`.

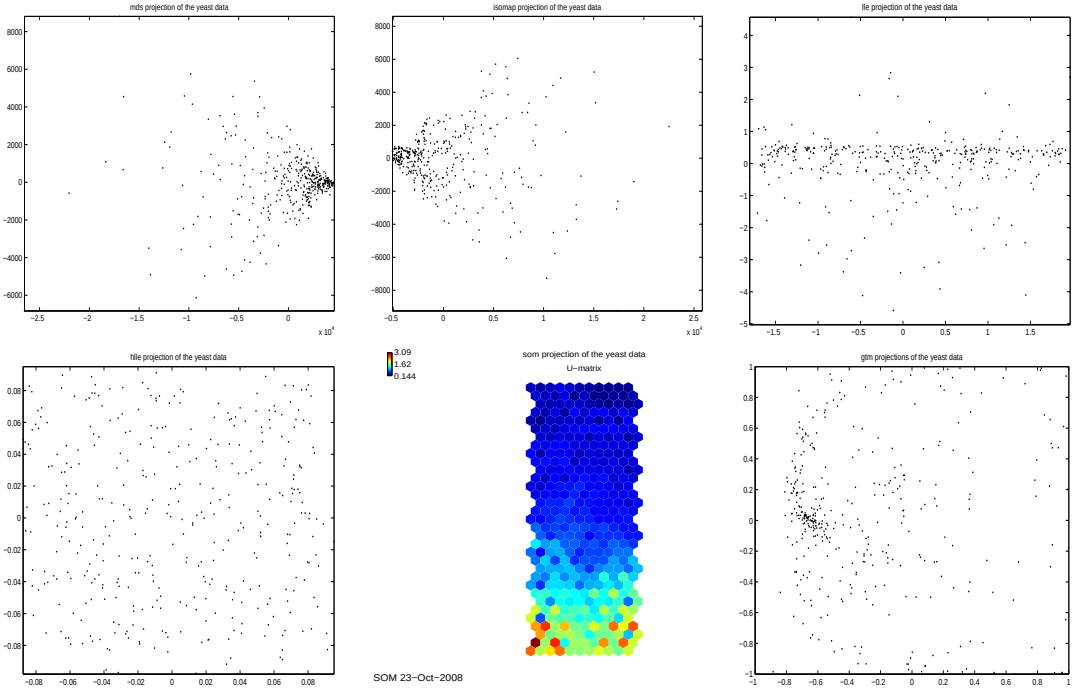

Figure 24: From left to right, top to bottom, we have the MDS, ISOMAP, LLE, HLLE, SOM, and GTM projections of the `yeast` dataset. The MDS and ISOMAP projections indicate the presence of outliers.

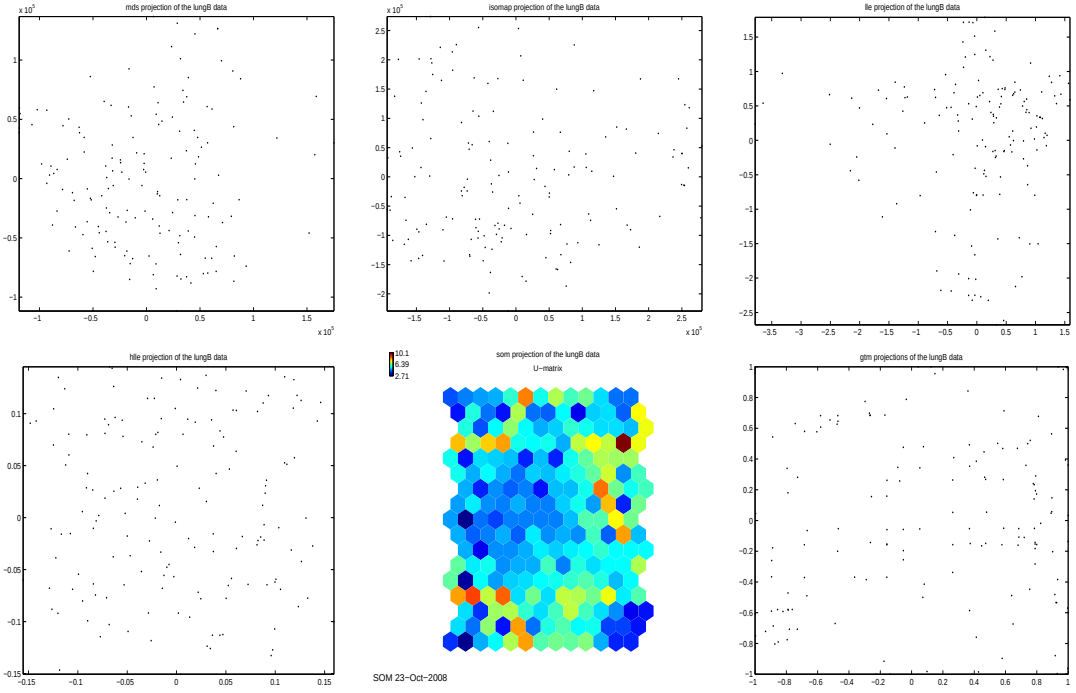

Figure 25: From left to right, top to bottom, we have the MDS, ISOMAP, LLE, HLLE, SOM, and GTM projections of the `lungB` dataset. The LLE and GTM projections indicate possible clustering.

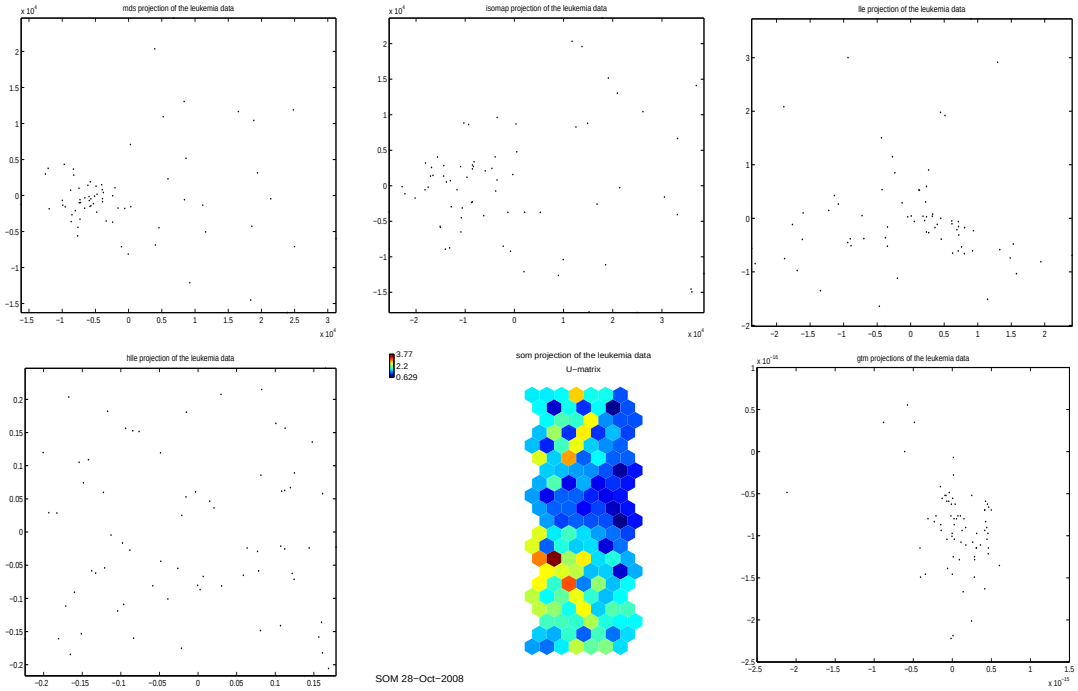

Figure 26: From left to right, top to bottom, we have the MDS, ISOMAP, LLE, HLLE, SOM, and GTM projections of the `leukemia` dataset. All projections indicate the presence of outliers and many projections (i.e. MDS and ISOMAP) indicated possible clustering.

Second, we apply the various non-linear dimensionality reduction techniques discussed in this chapter to the `lungB` dataset. The results from this experiment can be found in Figure 25 and is implemented in the MATLAB script `prob_3_14_b_lungB.m`.

Finally, we apply the various non-linear dimensionality reduction techniques discussed in this chapter to the `leukemia` dataset. The results from this experiment can be found in Figure 26 and is implemented in the MATLAB script `prob_3_14_b_leuk.m`.

Part (c): Here we apply the various non-linear dimensionality reduction techniques discussed in this chapter to the `iris` dataset. The results from this experiment can be found in Figure 27. This problem is implemented in the MATLAB script `prob_3_14_c_iris.m`.

Part (d): Here we apply the various non-linear dimensionality reduction techniques discussed in this chapter to the `pollen` dataset. The results from this experiment can be found in Figure 28 and is implemented in the MATLAB script `prob_3_14_d_pollen.m`.

Part (e): For this part of the problem we apply the various non-linear dimensionality reduction techniques discussed in this chapter to the various subdata sets from the entire `posse` data collection. As there are several objects present in the total dataset we will present the results from the various non-linear dimensionality reduction techniques one at a time.

We begin with the `croix` dataset. The results from this experiment can be found in Figure 29.

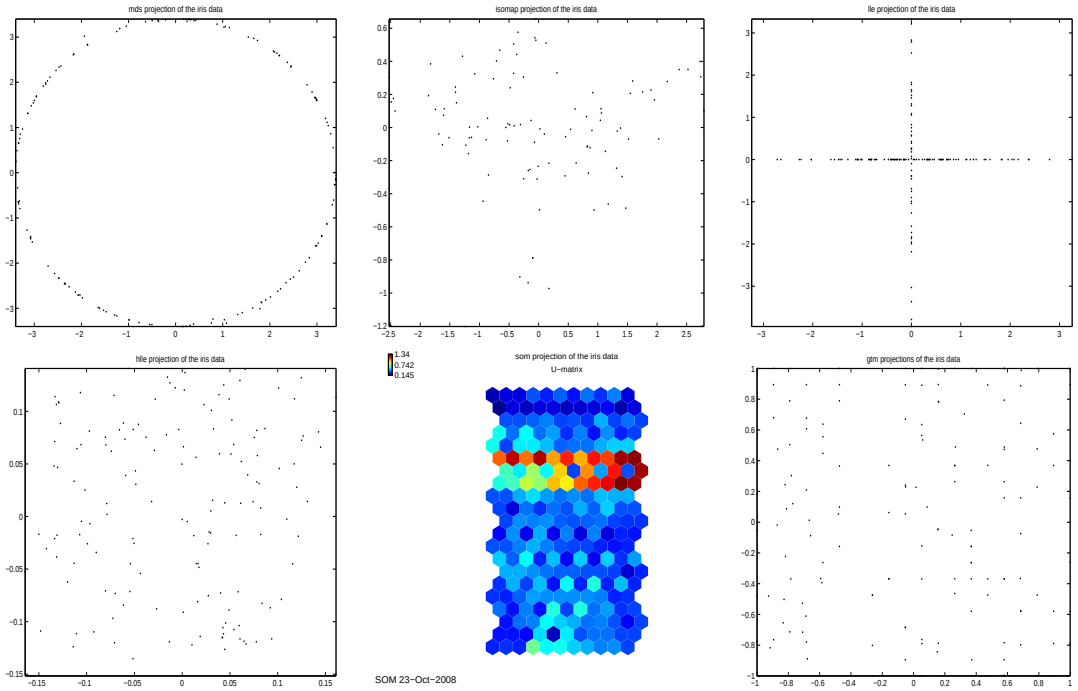

Figure 27: From left to right, top to bottom, we have the MDS, ISOMAP, LLE, HLLE, SOM, and GTM projections of the `iris` dataset. The projections found by the MDS and LLE algorithms are quite interesting. These results may indicate data clustering, depending on the sector of the circle (in the MDS case) or the quadrant (in the LLE case) where each of the flower type datum falls. Plotting class labels on these projections would be quite interesting and would verify or dispute these hypothesis.

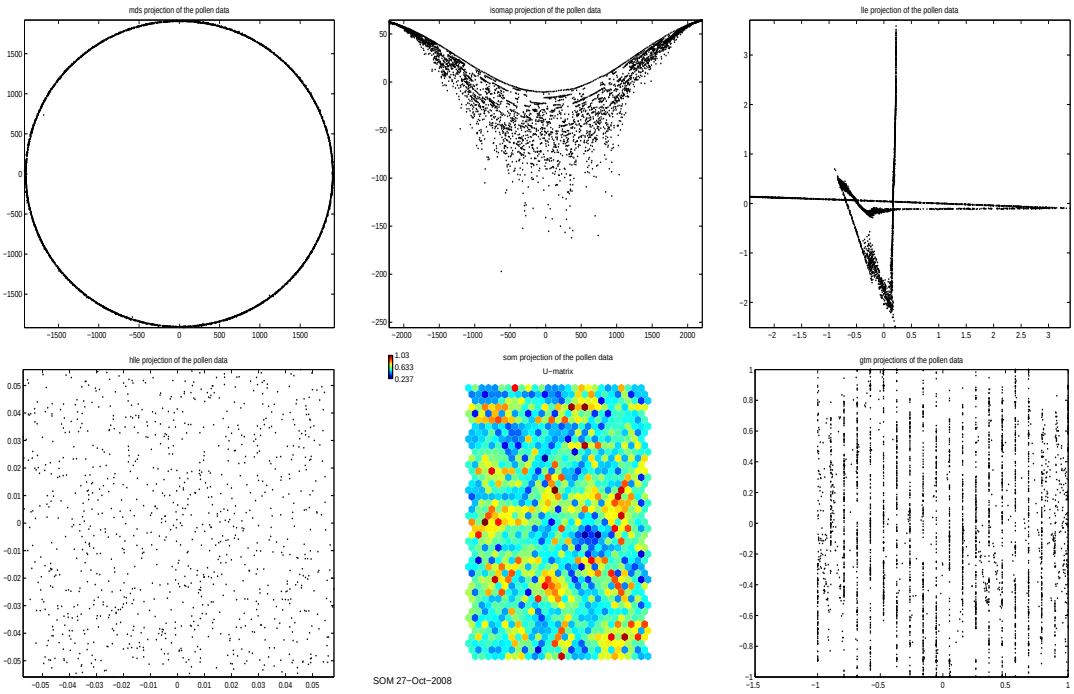

Figure 28: From left to right, top to bottom, we have the MDS, ISOMAP, LLE, HLLE, SOM, and GTM projections of the `pollen` dataset.

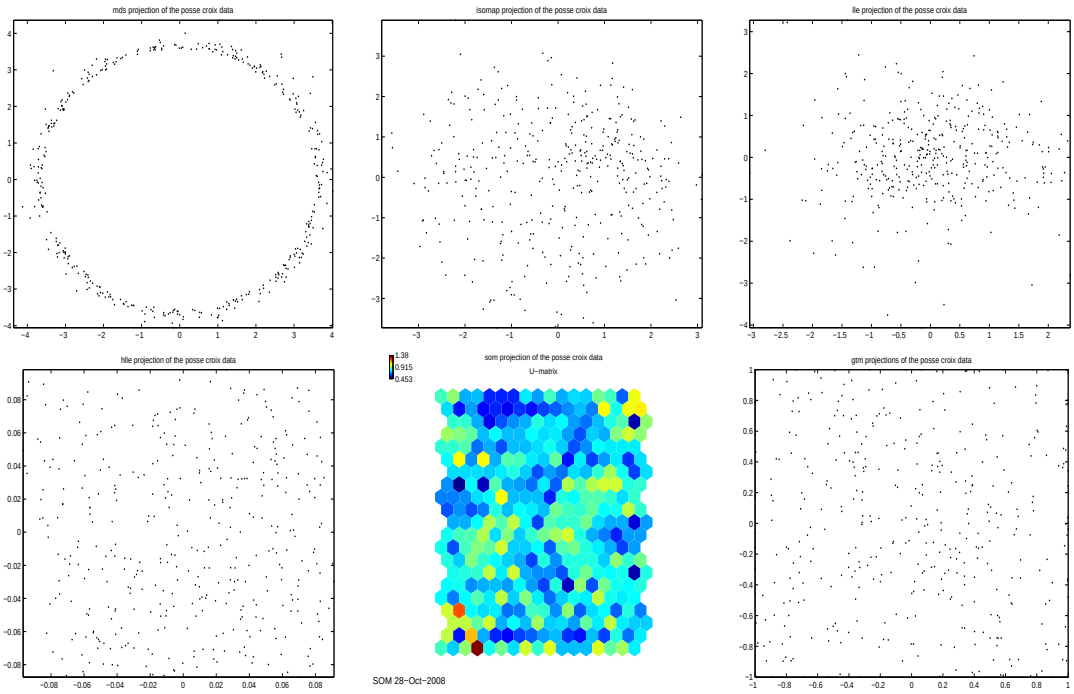

Figure 29: From left to right, top to bottom, we have the MDS, ISOMAP, LLE, HLLE, SOM, and GTM projections for the `croix` dataset from the `posse` data collection.

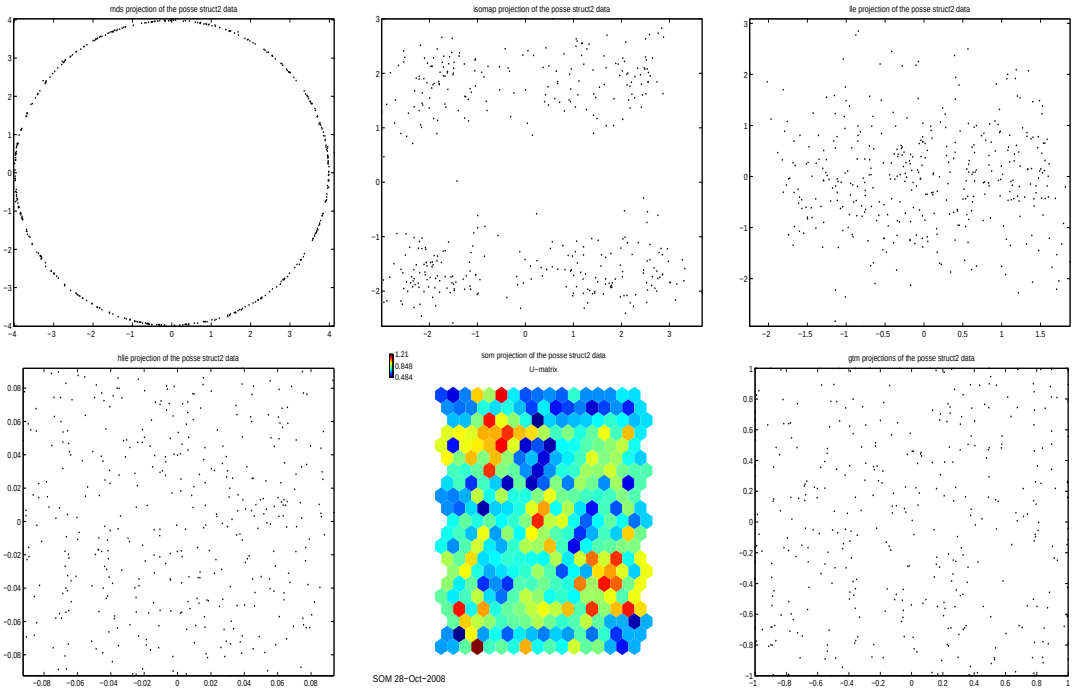

Figure 30: From left to right, top to bottom, we have the MDS, ISOMAP, LLE, HLLE, SOM, and GTM projections for the `struct2` dataset from the `posse` data collection.

Next we consider the `struct2` collection which represents an L-shaped structure. The results from this experiment can be found in Figure 30.

Next we consider the `boite` collection which represents an donut shaped structure. The results from this experiment can be found in Figure 31.

Next we consider the `groupe` collection which represents a set of four clusters. The results from this experiment can be found in Figure 32.

Next we consider the `curve` collection and the results from these experiments can be found in Figure 33.

Finally we consider the `spiral` collection which represents a set of four clusters. The results from this experiment can be found in Figure 34.

All of these plots can be generated by running the MATLAB script `prob_3_14_e_posse.m`.

Part (f): For this part of the problem we apply the various non-linear dimensionality reduction techniques discussed in this chapter to the `oronsay` dataset. The results from this experiment can be found in Figure 35 and are implemented in the MATLAB script `prob_3_14_f_oronsay.m`.

Part (g): For this part of the problem we apply the various non-linear dimensionality reduction techniques discussed in this chapter to the `skulls` dataset. The results from

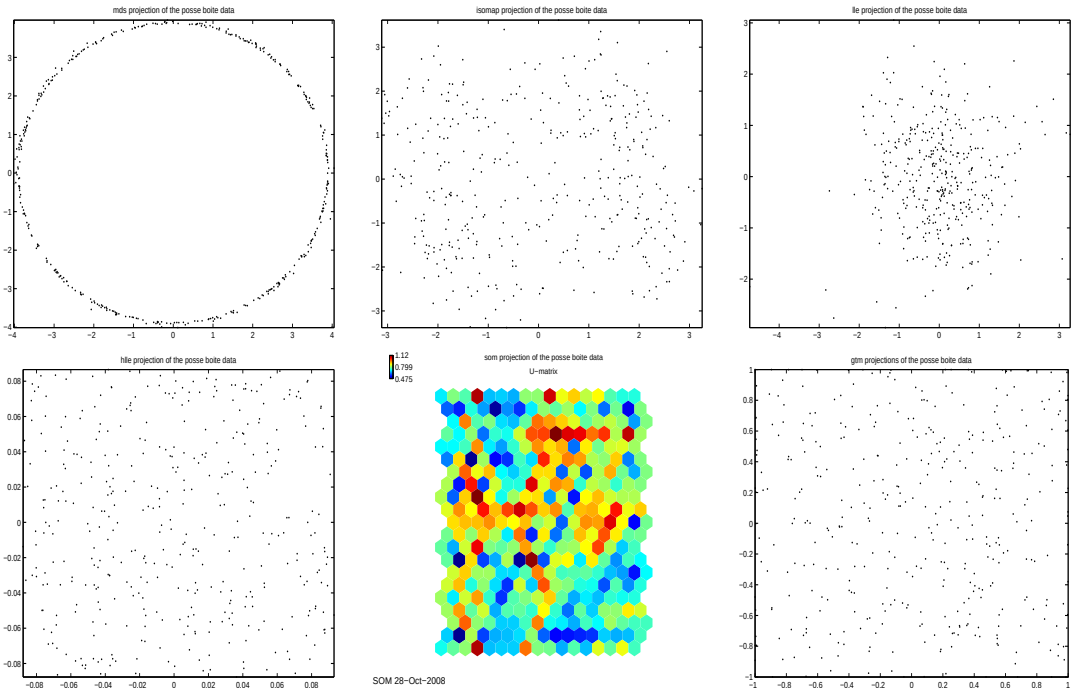

Figure 31: From left to right, top to bottom, we have the MDS, ISOMAP, LLE, HLLE, SOM, and GTM projections for the `boite` dataset from the `posse` data collection.

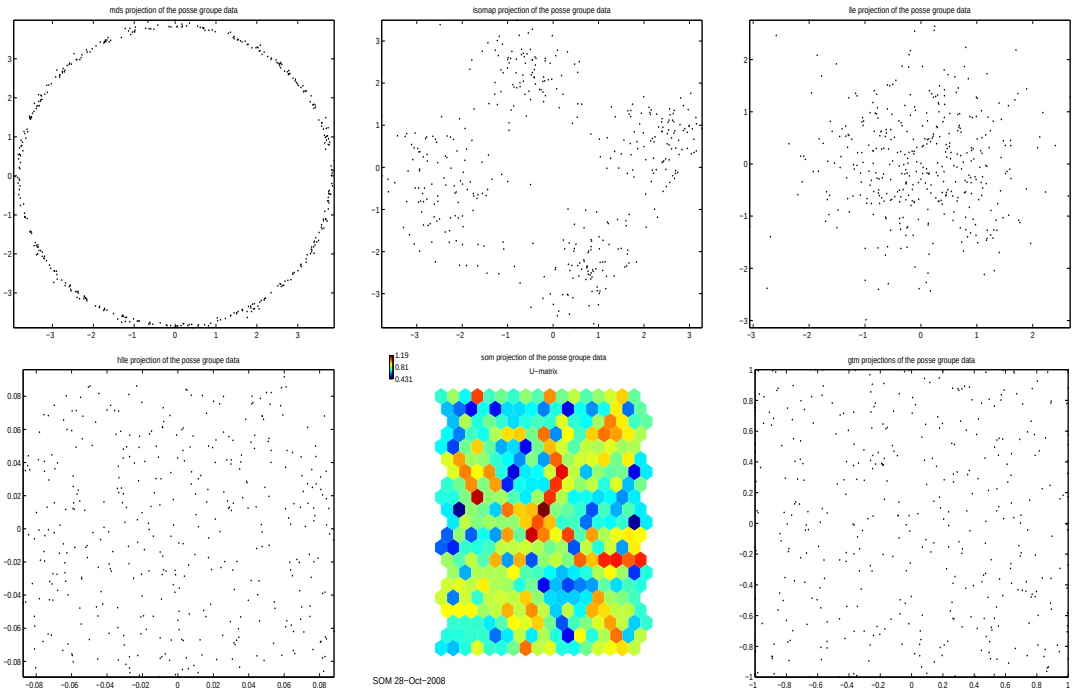

Figure 32: From left to right, top to bottom, we have the MDS, ISOMAP, LLE, HLLE, SOM, and GTM projections for the `groupe` dataset from the `posse` data collection.

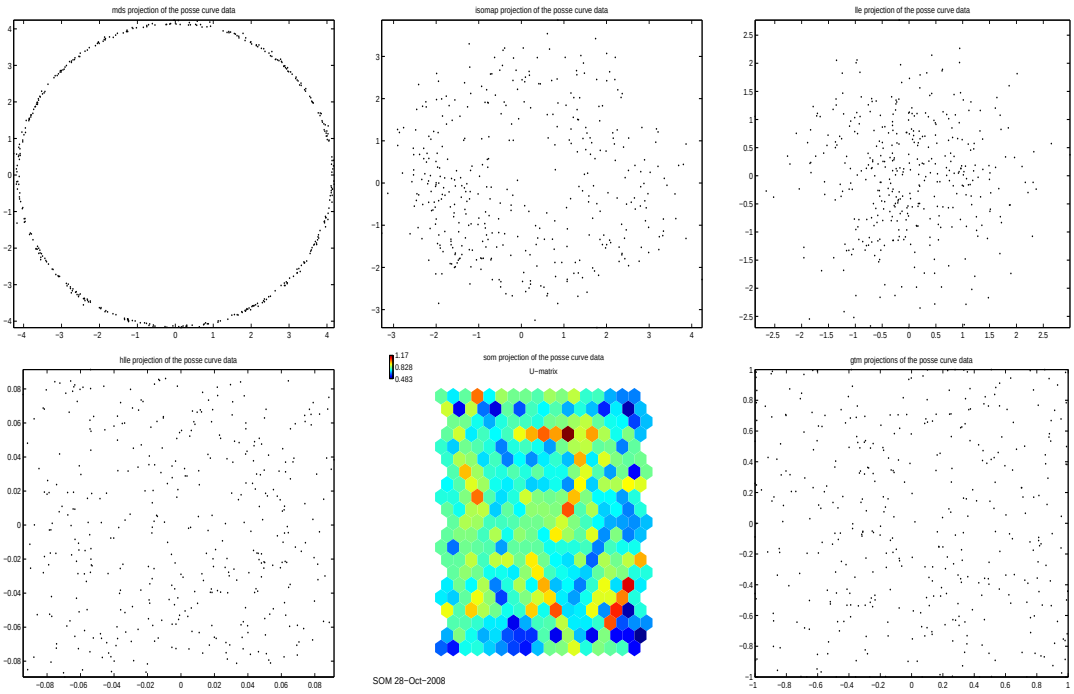

Figure 33: From left to right, top to bottom, we have the MDS, ISOMAP, LLE, HLLE, SOM, and GTM projections for the `curve` dataset from the `posse` data collection.

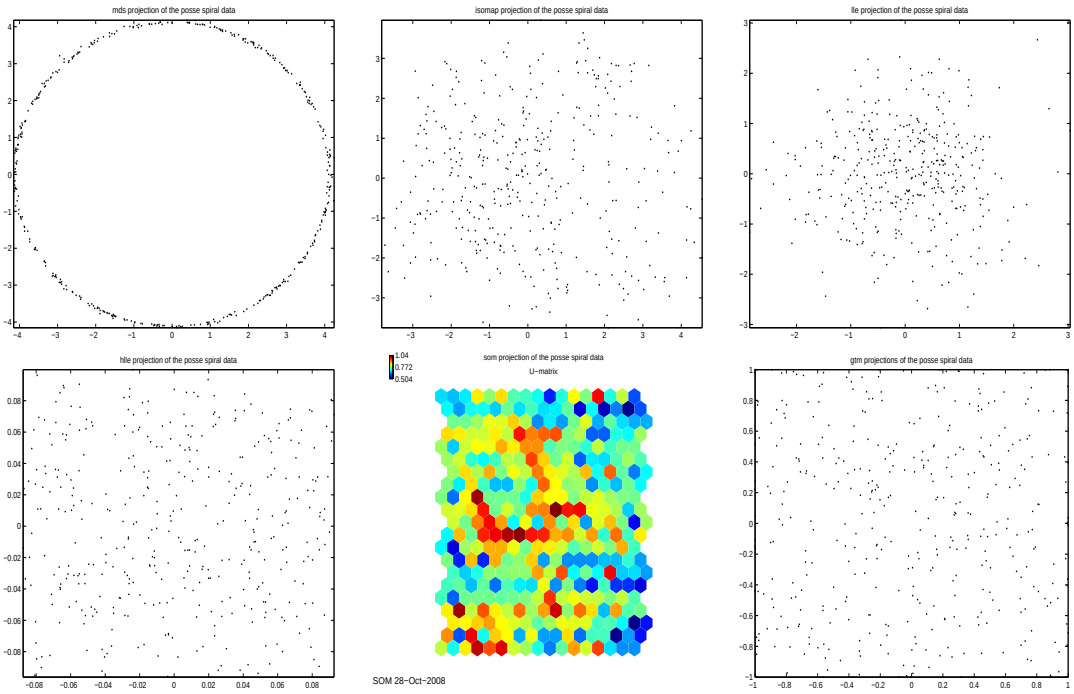

Figure 34: From left to right, top to bottom, we have the MDS, ISOMAP, LLE, HLLE, SOM, and GTM projections for the `spiral` dataset from the `posse` data collection.

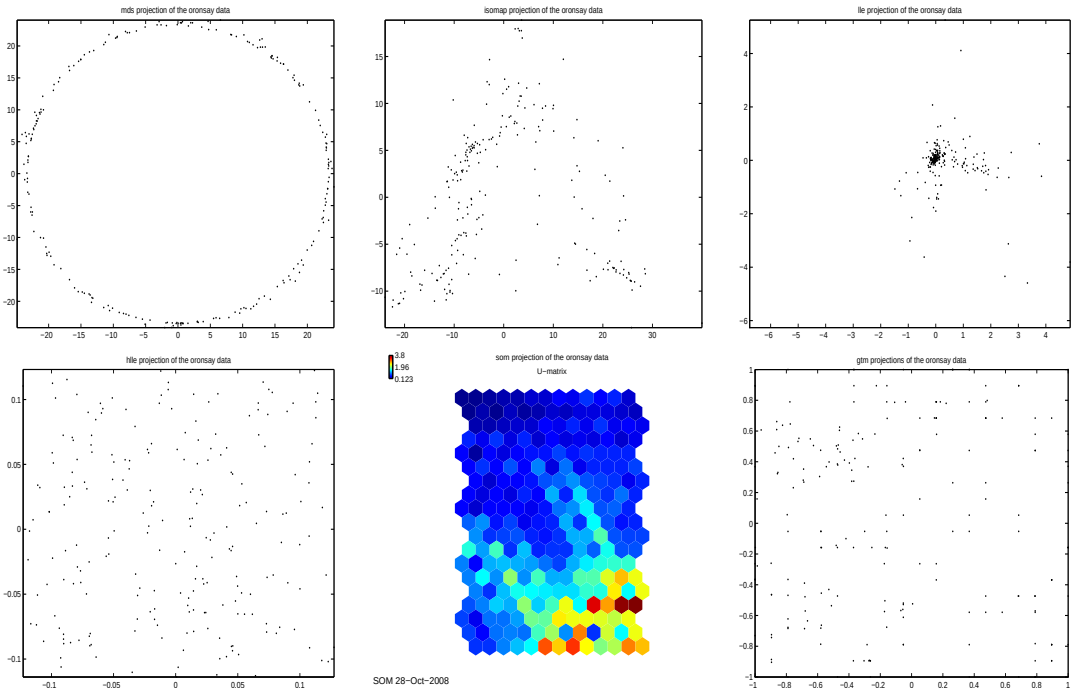

Figure 35: From left to right, top to bottom, we have the MDS, ISOMAP, LLE, HLLE, SOM, and GTM projections of the `oronsay` dataset.

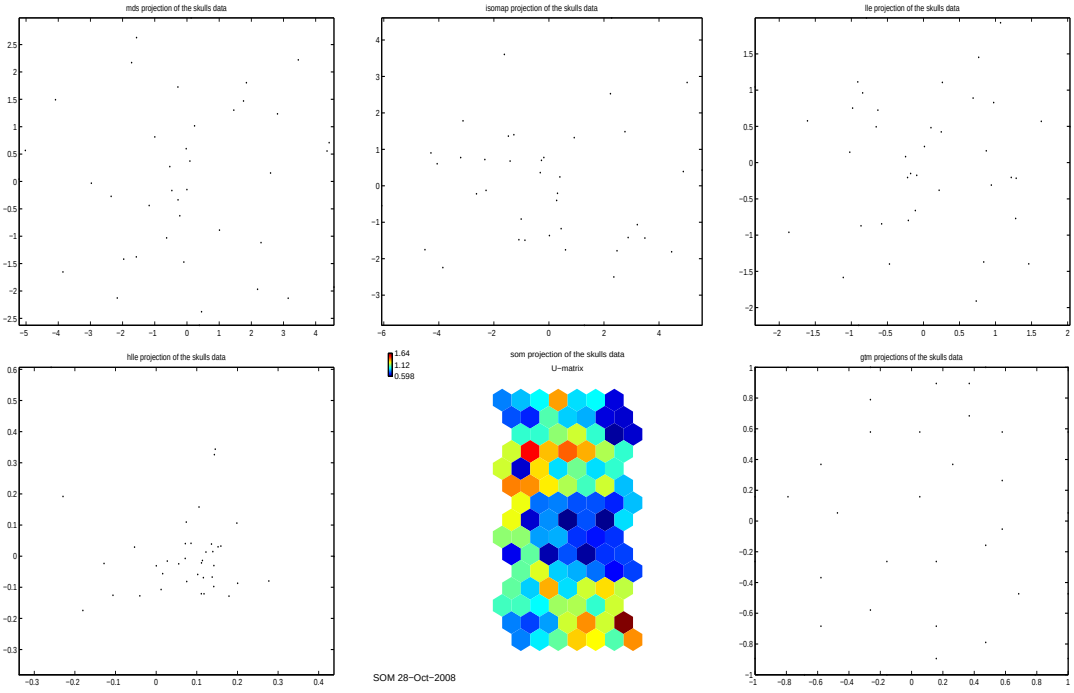

Figure 36: From left to right, top to bottom, we have the MDS, ISOMAP, LLE, HLLE, SOM, and GTM projections of the `skulls` dataset.

this experiment can be found in Figure 36 and are implemented in the MATLAB script `prob_3_14_g_skulls.m`.

Exercise 3.15 (different random configurations in the SMACOF algorithm)

For this problem we study how different configurations of initial conditions affect the SMACOF algorithm. In this problem the algorithm for SMACOF we specify an initial configuration of projected points with the MATLAB command

```
Z = unifrnd(-2,2,n,d);
```

In the MATLAB script `prob_3_15.m` we perform various initializations (by hand) of the projected points. One thing that is interesting is that the various initial configurations change the absolute final locations of the projected data but not the very prevalent clustering that is observable in the output.

Chapter 4: Data Tours

Extensions of the EDA Toolbox Functions

In many of the problems for this chapter we are asked to run various MATLAB data touring routines from the exploratory data analysis toolbox (EDT) on the data sets provided. With the versions of the MATLAB routines delivered, there is no provision provided to plot data points from different classes using different distinguishing markers. Because a great amount of information is usually contained in the class labeling and during these data tours it is very instructive to observe how the data from the different classes moves relative to each other on the website I provide modified toolbox function that provide this capability. Specifically, if one provides a sequence of class labels for data given to the data touring routines `pseudotour` and `intour` the corresponding data points will be represented differently by unique colors and symbols. See Exercises 4.4-6 where this capability is demonstrated when the corresponding MATLAB scripts are run. The modified versions of these two routines can be found on the accompanying website. These modifications are extremely helpful in visualizing clusters and experimenting with the given algorithms since any projection that results in clean classwise clustering is very easily observed in the resulting plots.

Exercise Solutions

Exercise 4.1 (a grand tour for the `yeast` data)

When we rerun the code from the suggested example, we see that the resulting projections of the `yeast` data look very much like an "arrowhead". Examples of two typical profiles that are generated from this grand torus tour when we run the MATLAB code `prob_4_1.m` are given in Figures 37 (left) and (right). The iterations of the grand torus tour produce an "arrowhead" that moves around pointing in different directions. From this data it appears that there is approximately one class clustered near the center, with additional data points located "behind" this core and distributed in a cone like shape. Note that this data's projection is very similar to the results obtained when one projected this data onto the first three eigenvectors, see Figure 2.3 in the book.

Exercise 4.2 (grand torus tours for various data sets)

In this problem we explore the projection results of applying the torus grand tour on several data sets.

Part (a): In the MATLAB script `prob_4_2_environmental` the `environmental` data set is subjected to a torus winding tour of Asimov. When that MATLAB code is run we see that as in the `yeast` data, the torus tour produces an arrowhead or wedge like set of data points

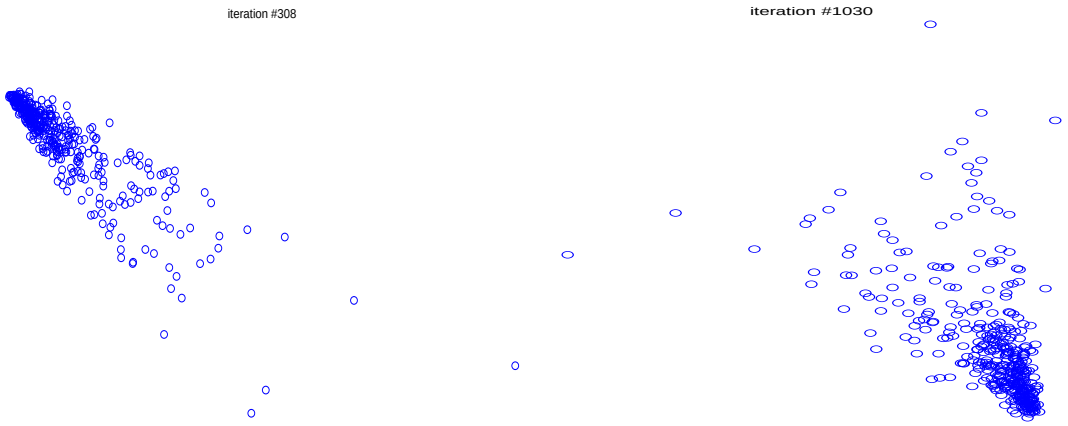

Figure 37: **Left:** One snapshot (iteration $k = 308$) of the grand tour of Asimov on the `yeast` data. **Right:** Another snapshot (iteration $k = 1030$) of the grand tour of Asimov on the `yeast` data. Note how both data sets have an arrowhead like appearance. Compare these plots with the eigen-projections in Figure 2.3 from the book.

that points in different directions as the iterations progress. Two examples, at different timesteps, are presented in Figure 38.

Part (b): In the MATLAB script `prob_4_2_oronsay` the `oronsay` data set is subjected to a torus winding tour of Asimov. When the above MATLAB script is run it appears that the this data lies in a planar like surface and rotating it produces various two dimensional projections of this plane. For example in Figure 39 (left) we are looking directly into the plane perpendicular surface of the data, while in Figure 39 (right) we are looking parallel to this plane and see the data projected onto a very linear surface.

Part (c): In the MATLAB script `prob_4_2_iris.m` the `iris` data set is subjected to a torus winding tour of Asimov. Since we know that the `iris` data set consists of three different species of iris, we would expect/hope that some degree of clustering would be observed. When we run the above script this hypothesis is found to be true in that for almost all projections we observe strong clustering. The strongest clustering is observed between the `setosa` species from the other two `versicolor` and `virginica` species. In some cases it appears that there might be three clusters indicating some separability of the `versicolor` and `virginica` classes. See Figure 40 (left) and (right) for two example projections from this data.

Part (d): In the MATLAB script `prob_4_2_posse.m` some of the individual data sets from the `posse` collection are subjected to a torus winding tour of Asimov. In most of these subcollections of data only the very *first* projection had any interesting structure. Once the grand tour began the initial structure observed in the first projection quickly degenerated and the set of points from that point onward looked much more random under. For this reason in this part of the exercise we only show the first projection for the of the data sets from the `posse` collection. Running the associated MATLAB code easily generates the remaining projections.

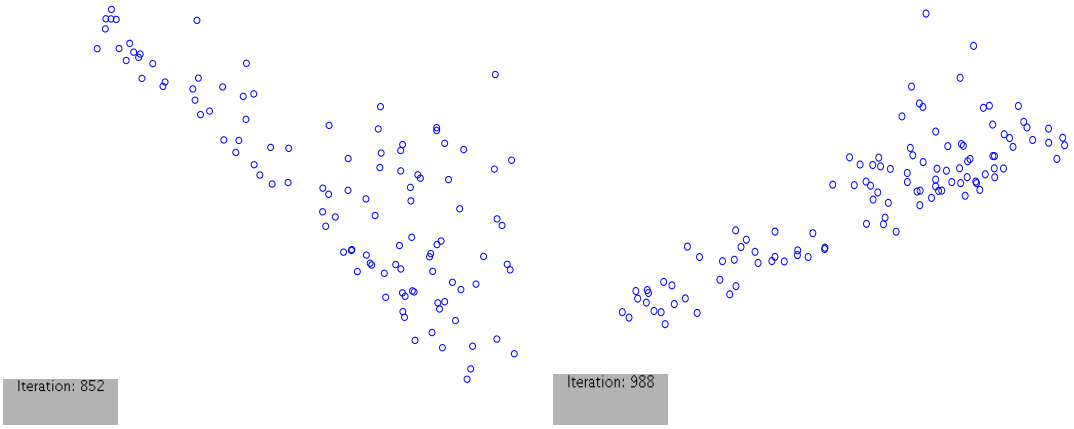

Figure 38: **Left:** One snapshot (iteration $k = 852$) of the grand tour of Asimov on the `environmental` data. **Right:** Another snapshot (iteration $k = 988$) of the grand tour of Asimov on the `environmental` data. Note the "wedge" like appearance of this data and the possible division of this wedge into two pieces by a possible gap in the "middle" of the wedge.

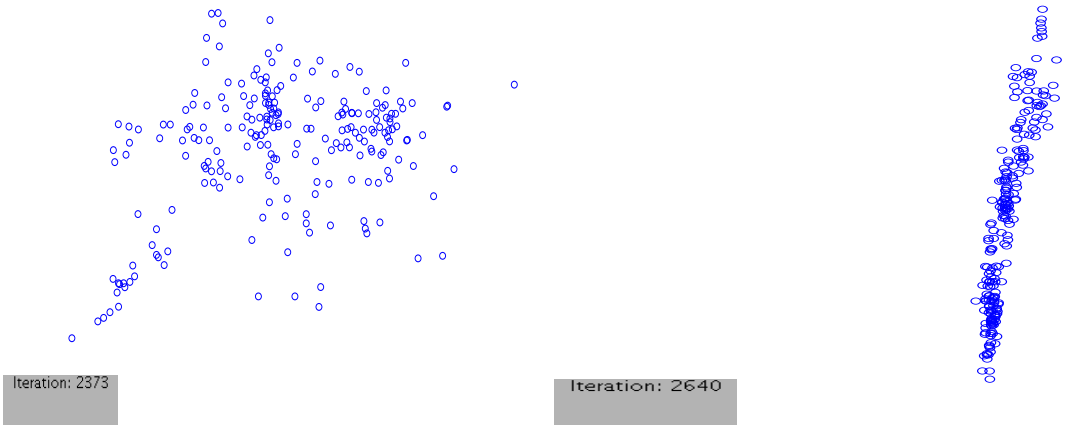

Figure 39: **Left:** Another snapshot (iteration $k = 2373$) of the grand tour of Asimov on the `ornosay` data. From this projection we observe possible clustering. **Right:** One snapshot (iteration $k = 2640$) of the grand tour of Asimov on the `ornosay` data.

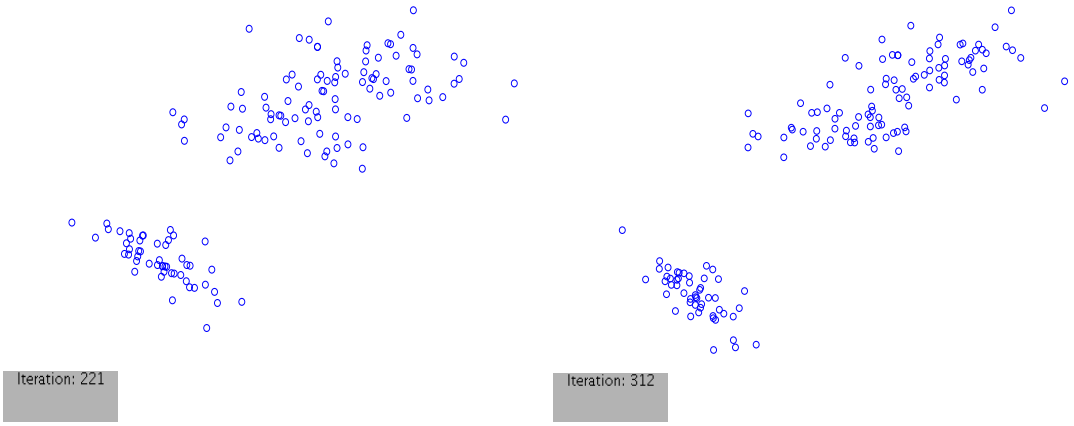

Figure 40: **Left:** One snapshot (iteration $k = 221$) of the grand tour of Asimov on the `iris` data. **Right:** Another snapshot (iteration $k = 312$) of the grand tour of Asimov on the `iris` data. These projections clearly show very strong separability of the data.

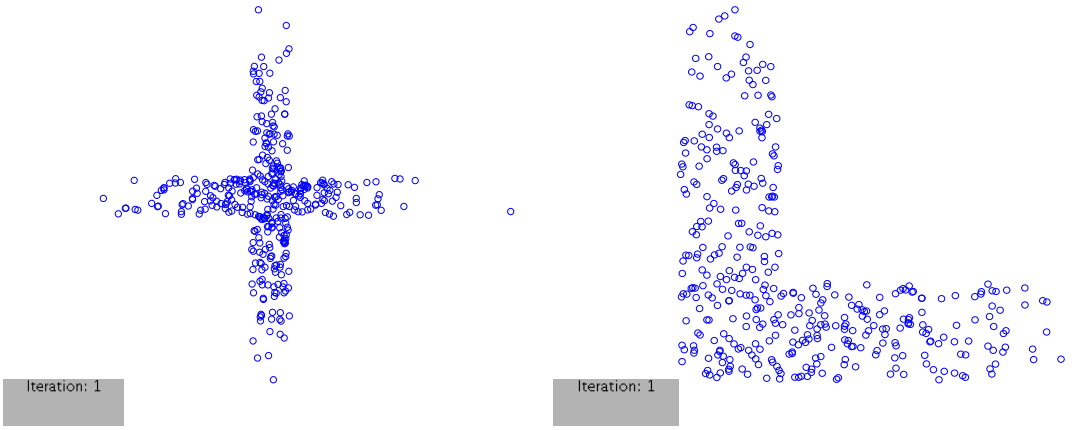

Figure 41: The first observed projections of the grand tour of Asimov on two data sets from the `posse` collection. **Left:** For the `croix` data set. **Right:** For the `struct2` data set.

In Figure 41 (left) we display the first projection from the `croix` data set. Notice the distinctive cross like appearance. In Figure 41 (right) we display the first projection of the `struct2` data which initially looks like an L-shape. In Figure 42 (left) we display the first projection of the `boite` data which initially looks somewhat like a donut or torus. In Figure 42 (right) we display the first projection of the `groupe` data which initially looks like a set of four clusters. In Figure 43 (left) we display the first projection of the `curve` data which initially looks like two distinct clusters of points separated by a region between them containing no data. In Figure 43 (right) we display the first projection of the `spiral` data which clearly shows a spiral-like pattern.

As stated before, when we run the MATLAB code `torustour` on the `posse` data sets, after the initial projection the remaining projections look like a spherical cluster of points with no observable structure. This emphasizes the conclusion that for some data sets there maybe only a *few* projections where the data looks significantly different than a random

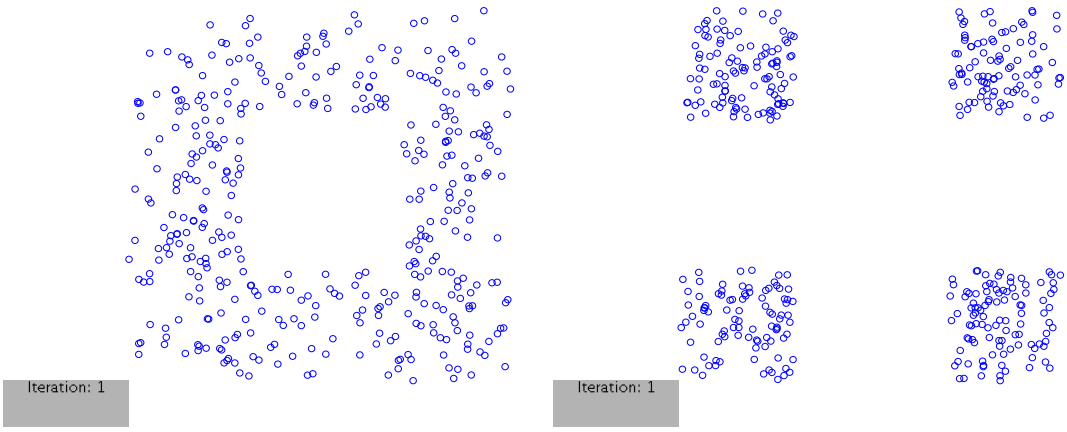

Figure 42: The first observed projections of the grand tour of Asimov on two data sets from the `posse` collection. **Left:** For the `boite` data set. **Right:** For the `groupe` data set.

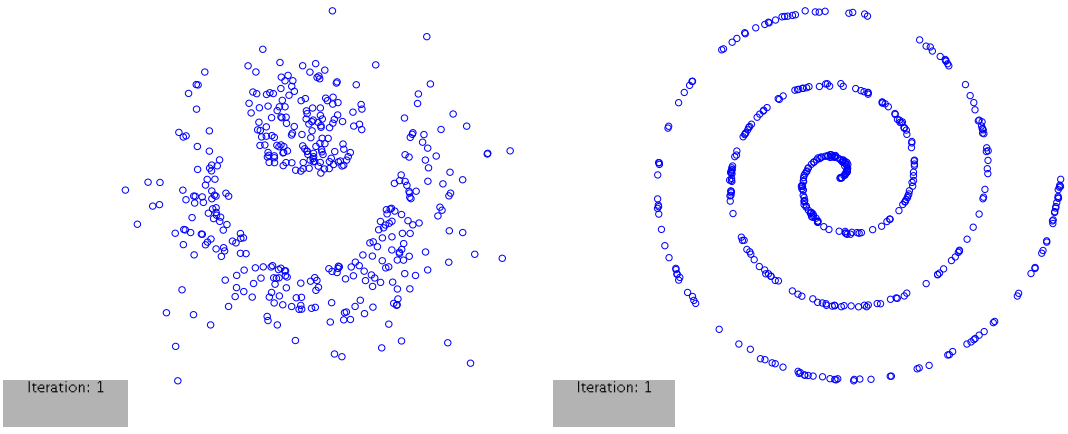

Figure 43: The first observed projections of the grand tour of Asimov on two data sets from the `posse` collection. **Left:** For the `curve` data set. **Right:** For the `spiral` data set.

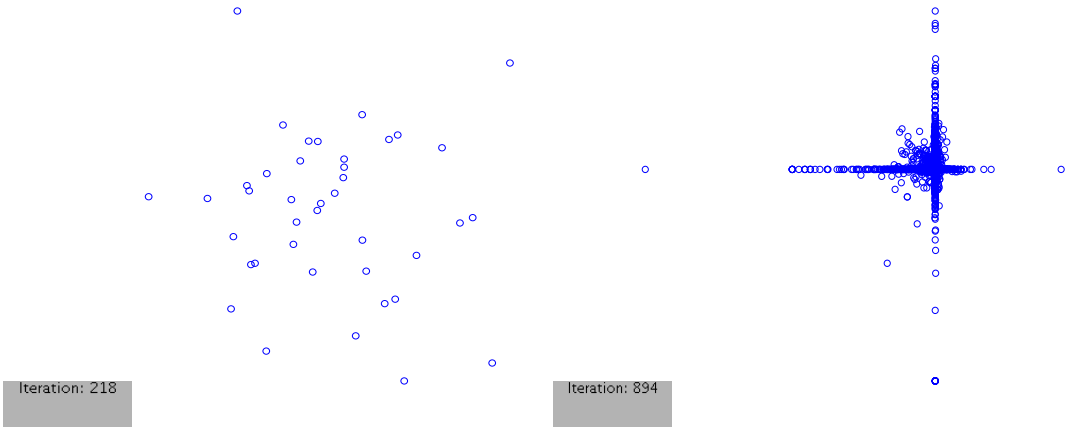

Figure 44: Two snapshots of the grand tour of Asimov. **Left:** The iteration $k = 218$ for the `skulls` data set. **Right:** The iteration $k = 894$ for the `spam` data set.

cloud of points. It would be interesting to see if the projection pursuit algorithm (PPEDA) could find these initial non-random projections in these data sets since it explicitly looks for projections that map the data into the most likely non-Gaussian configuration. See page 49 for a discussion of some computational experiments designed to answer this question.

Part (e): In the MATLAB script `prob_4_2_skulls.m` the data set `skulls` is subjected to the grand tour of Asimov. In this case due to the small number of samples in this data set (around 40) it is difficult to say very much. One thing that seems true is that the data seems to be somewhat spherically clustered as the data points stay in a sphere like cloud as the various rotations are performed. See Figure 44 (left) for a representative projection.

Part (f): In the MATLAB script `prob_4_2_spam.m` the data set `spam` is subjected to a grand tour of Asimov. When this script is run we see that the data seems to line up along "orthogonal axis" for most of the projections. An example projection is given in Figure 44 (right).

Part (g): In the MATLAB script `prob_4_2_pollen.m` the data set `pollen` is subjected to a torus grand tour of Asimov. When this script is run we see that the data quickly projects onto a straight line after which this line seems to rotate. Initially, the data seems "fuzzy" (Figure 45 (left)) or elliptical around a linear structure but this wider shape quickly collapses into a much tighter pencil like object (Figure 45 (right)) for much of the later parts of the tour.

Part (h): In the MATLAB script `prob_4_2_leukemia.m` the data set `leukemia` is subjected to a torus winding tour of Asimov. Since we know this data contains two type of cancers we might expect to find two clusters of points. In Figure 46 (left) and (right) we show two projections of the `leukemia` data set that are found during the tour. In both plots we see what appears to be two clusters of points that are "perpendicular" to each other. These two clusters might represent two different types of cancers.

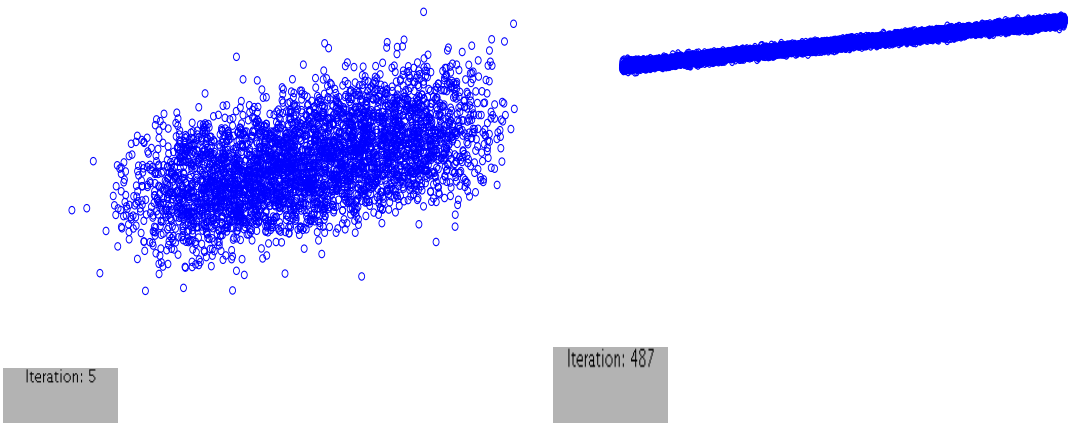

Figure 45: Two snapshots of the grand tour of Asimov on the `pollen` data set. **Left:** The iteration $k = 5$, where we see a "fuzzy" elliptical cloud of data. **Right:** The iteration $k = 487$, where we see the data collapse into a much tighter pencil like object.

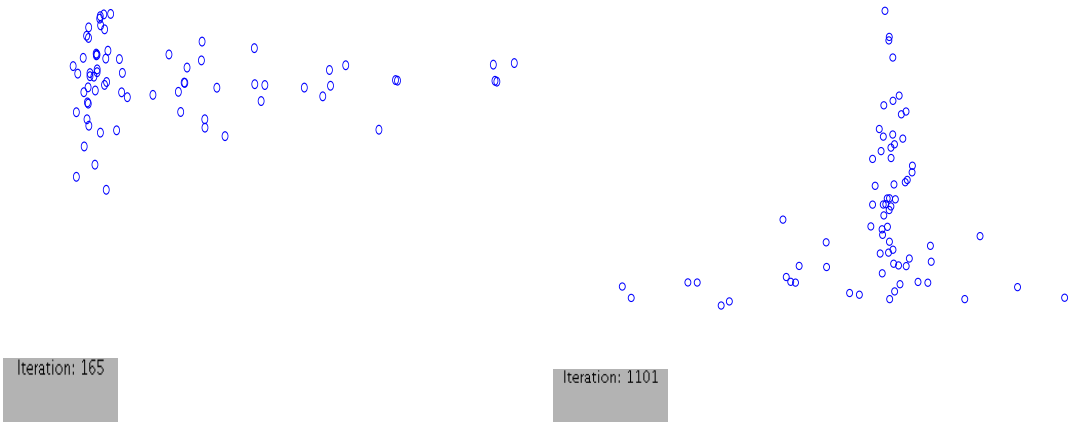

Figure 46: Two snapshots of the grand tour of Asimov on the `leukemia` data set. **Left:** The iteration $k = 165$. **Right:** The iteration $k = 1101$. Note in both projections there we see the presence of clustering.

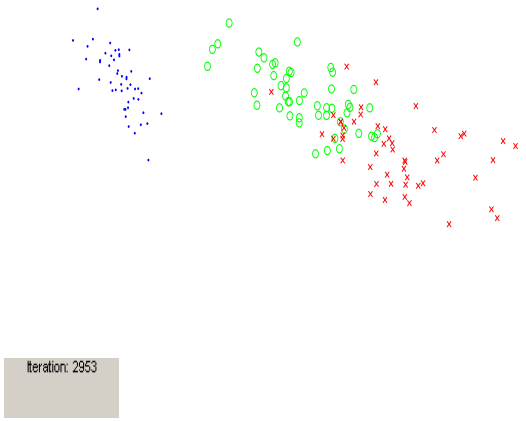

Figure 47: The iteration $k = 2952$ of the pseudo tour on the `iris` data set. We have plotted the three different types of iris with different symbols and colors. The expected clustering is clearly observable.

Exercise 4.3 (the pseudo grand tour)

In this problem we apply the pseudo grand tour to the `oronsay` data set. By running the MATLAB script `prob_4_3.m` one can observe the graphical output produced by the pseudo grand tour. For some of the tour an almost linear looking structure is present while for other parts clustering appears. The MATLAB script above has the feature that when run it will plot the classes associated with `midden` labeling which greatly facilities looking for clusters.

Exercise 4.4 (pseudo grand tours for various data sets)

In this problem we explore the results of the pseudo grand tour on several data sets.

Part (a): Here we apply the pseudo grand tour on the `environmental` data set. When one runs the MATLAB script `prob_4_4_environmental.m` an arrowhead looking structure appears and begins to rotate as the iterations proceed. This is very similar to the projections obtained by the grand tour (see Figure 38).

Part (b): For this subproblem we apply the pseudo grand tour to the `yeast` data set. When one runs the MATLAB script `prob_4_4_yeast.m` one sees what begins as a pointed cluster of points that spreads out into a funnel of data behind the tip. This result is very similar to the projections obtained by the grand tour (see Figure 37).

Part (c): Here we apply the pseudo grand tour to the `iris` data set. When one runs the MATLAB script `prob_4_4_iris.m` with colorized clustering (provided on the accompanying website as a modified EDA toolbox function `pseudotour` program) we easily observe the expected strong clustering of the data. See Figure 47 for an example projection from the pseudo grand tour. These projections are very similar to those obtained from the grand tour

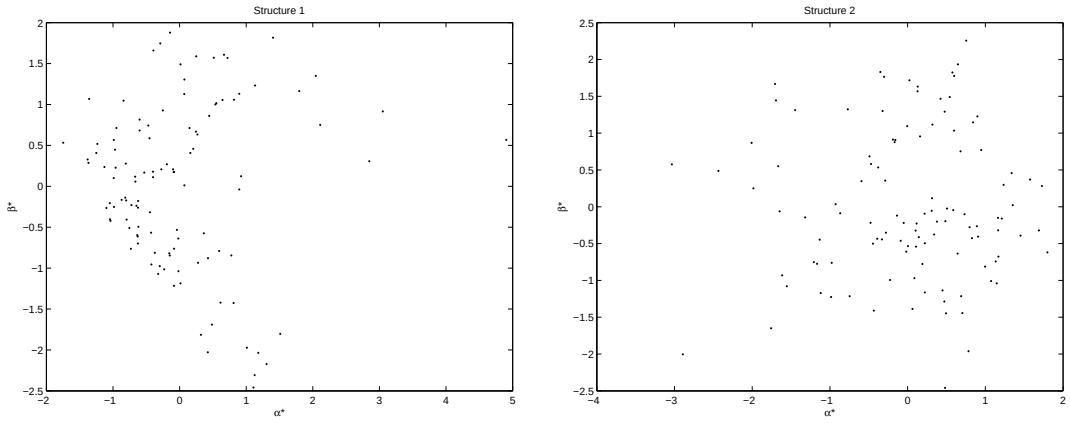

Figure 48: The first and second structures found when using the PPEDA algorithm on the `environmental` data set. Here we are using the chi-squared index for detecting projections that deviate from normality.

algorithm (see Figure 40).

Part (d): Here we apply the pseudo grand tour to the `posse` data set. See the MATLAB script `prob_4_4_posse.m` for an implementation of this procedure. For each of these data sets the pseudo grand tour quickly results in a spherical cloud of points lacking any distinguishing features.

Part (e): Here we apply the pseudo grand tour to the `skulls` data set. See the MATLAB script `prob_4_4_skulls.m` for an implementation of this procedure.

Part (f): Here we apply the pseudo grand tour to the `spam` data set. See the MATLAB script `prob_4_4_spam.m` for its implementation. When this script is run we observe an arrow head like object pointing in different directions for each of the projections.

Exercise 4.5 (the interpolation tour)

Part (a): Here we apply the interpolation tour on the `environmental` data set. This is implemented can be found in the MATLAB script `prob_4_5_environmental.m`

Part (b): Here we apply the interpolation tour on the `yeast` data set. This is implemented can be found in the MATLAB script `prob_4_5_yeast.m`

Exercise 4.6 (different target planes for the `oronsay` data)

This exercise is implemented in the MATLAB function `prob_4_6.m`. When this script is run we see the data pulling apart from a central location and some of it separates into distinct

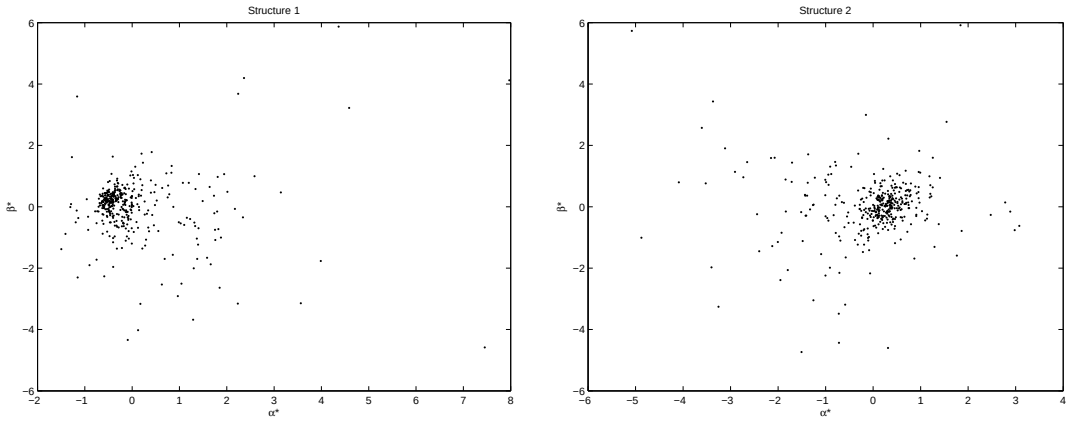

Figure 49: The first and second structures found when using the PPEDA algorithm on the `yeast` data set. Here we are using the chi-squared index for detecting projections that are not normal. Note that is looks like the first structure found is the result of outliers rather than actual non-Gaussian structure.

clusters.

Exercise 4.7 (projection pursuit applied to various data sets)

In this problem we explore the results of applying the projection pursuit algorithm of Friedman and Tukey [6] for exploratory data analysis (PPEDA) routine on several of the provided data sets.

Part (a): In the MATLAB script `prob_4_7_environmental.m` using the PPEDA code we extract the structure that is most non-Gaussian from the `environmental` data set. When this script is run, two of these projections are shown in Figure 48 (left) and (right). Running the `ppeda` code with the "moment" option gives very similar results.

Part (b): In the MATLAB script `prob_4_7_yeast.m` we extract the structure that is most non-Gaussian from the `yeast` data set. When this script is run, two of these projections are shown in Figure 49 (left) and (right). Running the `ppeda` code with the "moment" option gives very similar results.

Part (c): In the MATLAB script `prob_4_7_iris.m` we extract the structure that is most non-Gaussian from the `iris` data set. When this script is run, two of these projections are shown in Figure 50 (left) and (right). Since we know that this data is composed of multiple classes we would hope that the PPEDA algorithm would be able to find a projection in which the data points are strongly clustered. In the plot Figure 50 (left) we indeed see a projection with significant separation of the data. The second projection seen in Figure 50 (right) does not show this clustering. Subsequent structure removals produced by the PPEDA algorithm (not shown) do exhibit clustering.

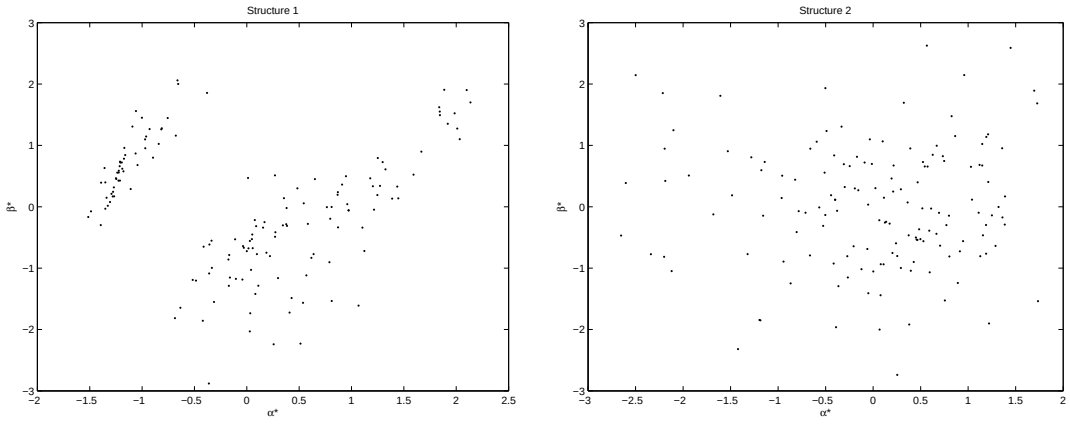

Figure 50: The first and second structures found when using the PPEDA algorithm on the `iris` data set. Here we use the chi-squared index for detecting projections that are not normal.

Part (d): In the MATLAB script `prob_4_7_posse.m` we extract the structure that is most non-Gaussian from the data sets provided from the `posse` data set. It is interesting that many of the higher projections obtained using PPEDA appear to be significantly more Gaussian looking that the initial projections of this data set might suggest. Especially since the PPEDA algorithm is developed specifically for the extraction of non-Gaussian structure. For example, in Figure 51 we show the first structure (the most non-Gaussian) found using the projection pursuit algorithm. These plots are to be compared with the two dimensional projections Figure 41 (left) and Figure 43 (right) both of which represent points that are significantly non-Gaussian. This result highlights the benefit of using several different techniques when beginning an initial study of a data set.

The PPEDA algorithm does find some interesting structure for some of the data sets in the `posse` collection. For example in Figure 52 we see the extracted projections from the `struct2` data. This data is designed to represent an L-shaped object (see Figure 41 (Right)). The `groupe` data set (Figure 53) also shows strong evidence of clustering.

Part (e): In the MATLAB script `prob_4_7_skulls.m` we extract the structure that is most non-Gaussian from the data sets in the `skulls` data set collection. See Figure 54 for examples of the first two structure removals.

Part (f): In the MATLAB script `prob_4_7_spam.m` we extract the structure that is most non-Gaussian from the `spam` data set. See Figure 55 for examples of the first two structure removals found in this data set.

Note that when the `spam` data set is processed "as is" the PPEDA algorithm takes a very long to compute the non-Gaussian projections. This is due (I believe) to the large disparity in magnitude between the feature vectors present. This might indicate that improved performance is possible if the feature vectors need to be normalized in someway.

Part (g): In the MATLAB script `prob_4_7_pollen.m` we extract the structure that is most

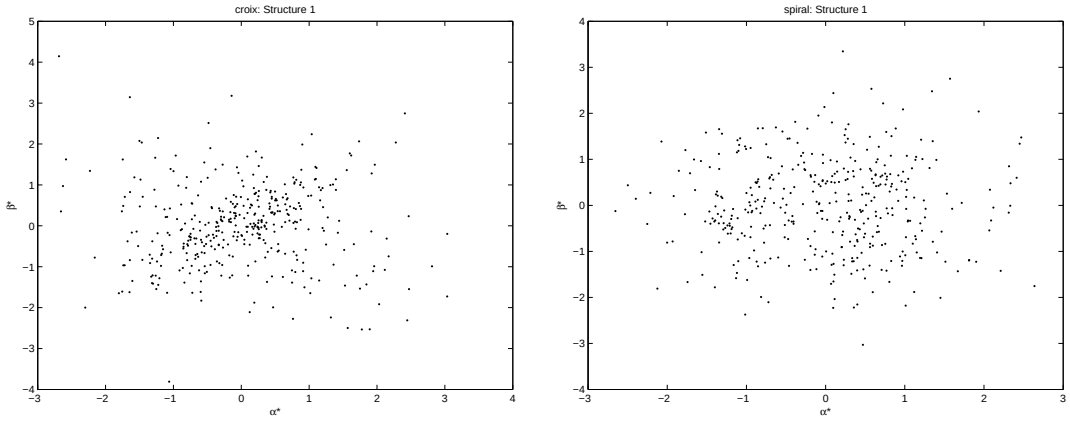

Figure 51: **Left:** The first structure removal when using the PPEDA algorithm on the `croix` data set. All additional structures found using the PPEDA algorithm are even more spherical/Gaussian looking. **Right:** The first structure removal when using the PPEDA algorithm on the `spiral` data set. All additional structures found using the PPEDA algorithm are more spherical/Gaussian looking than this one.

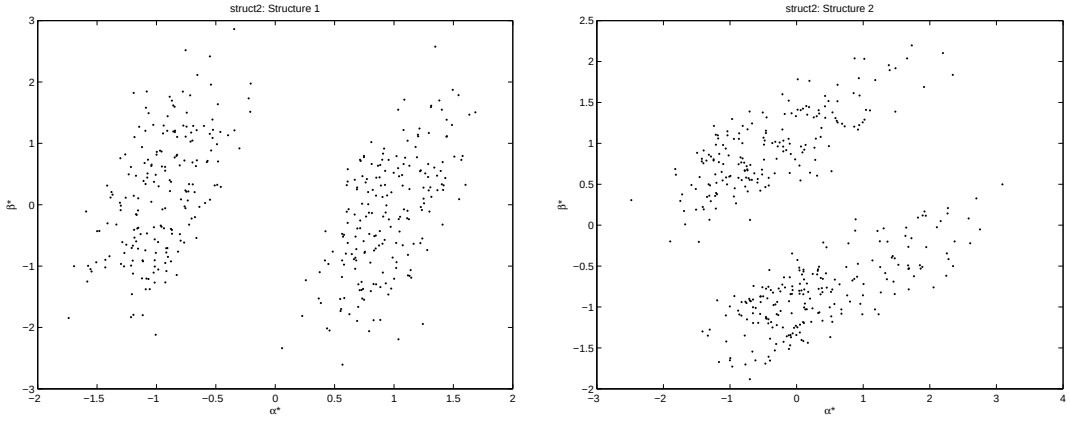

Figure 52: The first and second structure removal when using the PPEDA algorithm on the `struct2` data from the `posse` data set. Notice the nice cluster separation in both projections.

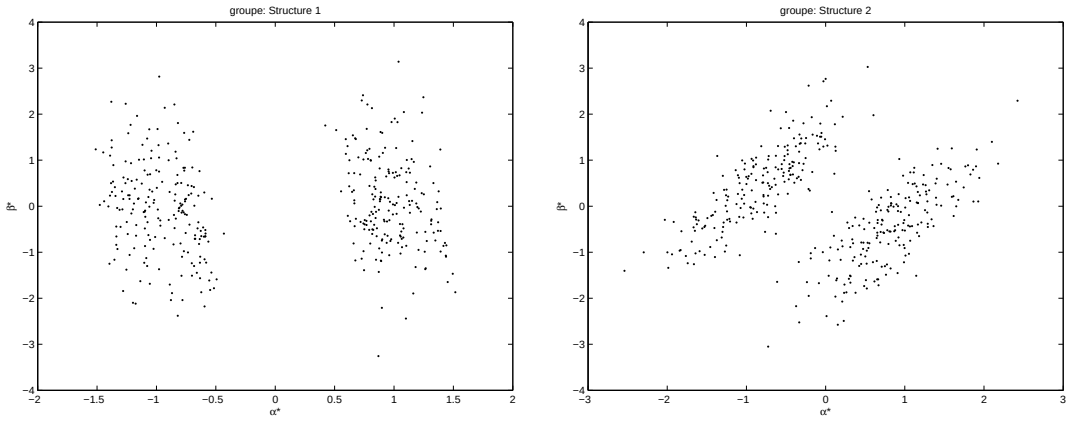

Figure 53: The first and second structure removal when using the PPEDA algorithm on the `groupe` data from the `posse` data set.

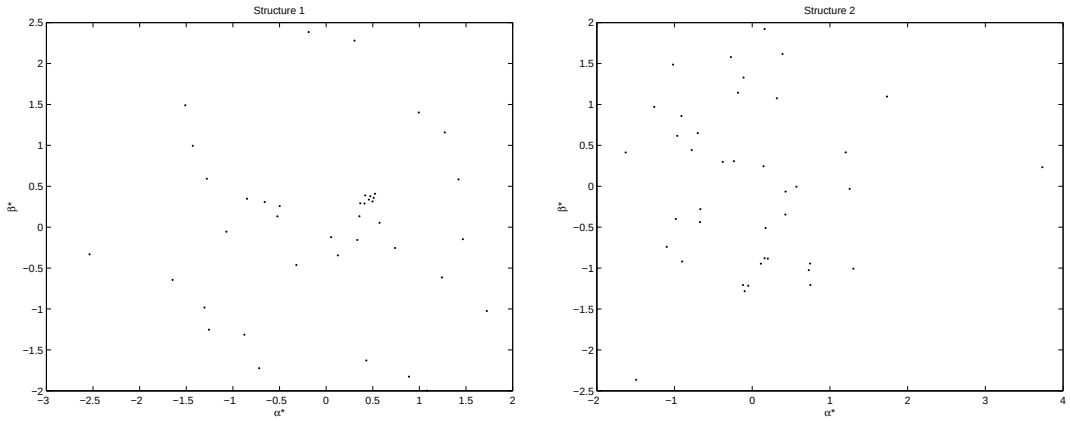

Figure 54: The first and second structure removal when using the PPEDA algorithm on the `skulls` data set.

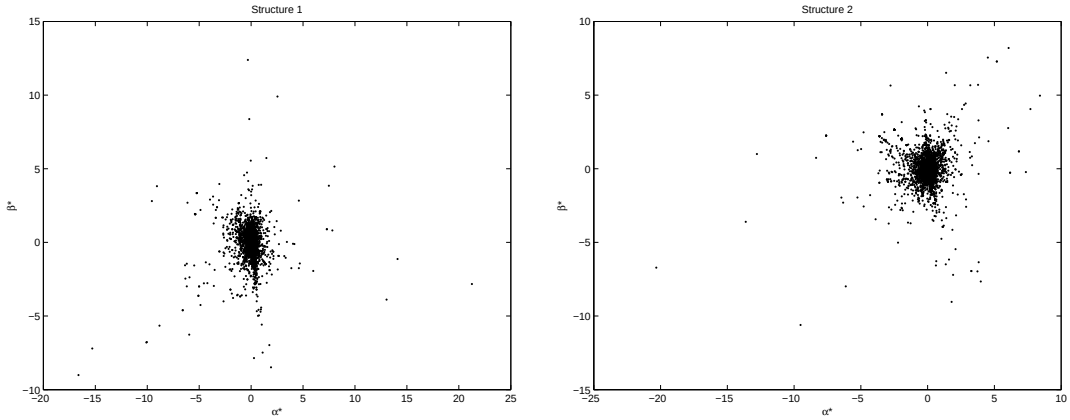

Figure 55: The first and second structure removal when using the PPEDA algorithm on the `spam` data set.

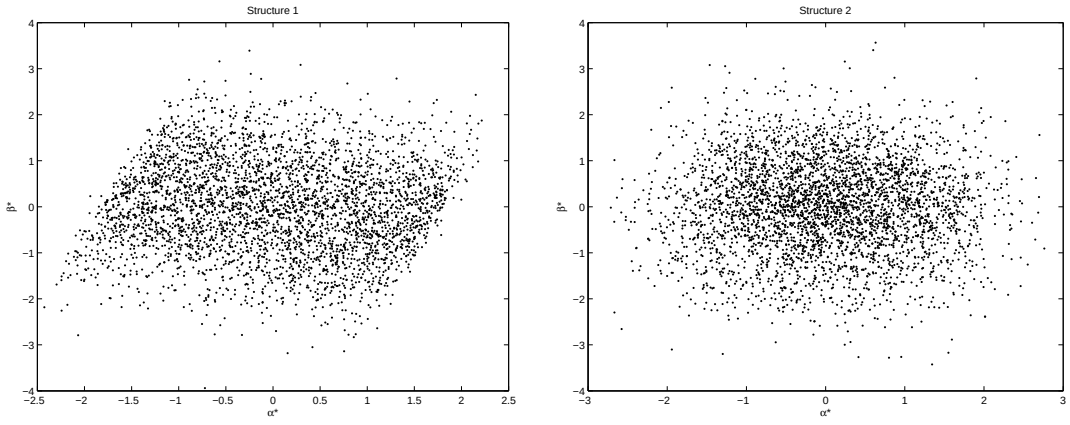

Figure 56: The first and second structure removal when using the PPEDA algorithm on the `pollen` data set.

non-Gaussian from the data sets in the `pollen` data set collection.

Part (h): In the MATLAB script `prob_4_7_leukemia.m` we extract the structure that is most non-Gaussian from the `leukemia` data set (not shown).

For a technique such as this one that attempts to extract the projections that are the most *non-Gaussian* it maybe beneficial to preprocess the data in an attempt to *make* it more Gaussian by sphering it or applying a MATLAB function like `zscore` to the feature vectors before an application of the routine `ppeda`. The motivation for this is to start the PPEDA algorithm "focused" on the more non-Gaussian parts of the data. It is unclear whether this preprocessing step would result in better clusters or degrade the algorithms performance by having it become overly concerned with any outliers.

Exercise 4.8 (looking for more structure in the `oronsay` data)

In the MATLAB script `prob_4_8.m` we look at removing more structure from the `oronsay` data set. When this script is run it produces five scatter plots of the two-dimensional projections associated with the first five results that maximize the various indices of performance (either the chi-squared or the moment index). When we run the code with the objective of maximizing the chi-squared index we find that many of the resulting scatter plots look similar. Two such projections are shown in Figure 57 (left) and (right). When we run the code aimed at maximizing the momentum index we obtain results very similar to the ones above.

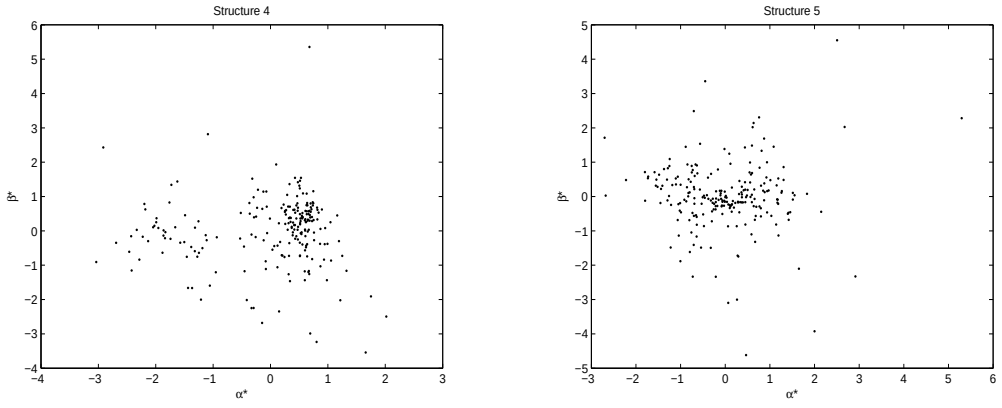

Figure 57: The fourth and fifth structure removal found when using the PPEDA algorithm on the `oronsay` data set. Here we use the chi-squared index as a measure of non-normality.

Chapter 5: Finding Clusters

Additional Notes

Example 5.7 from this chapter of the book is an example of computing the uniform gap statistic for various hypothetical numbers of clusters using the output of the hierarchical clustering algorithm `linkage`. With these statistics an estimates of the number of clusters present in the given data set can be obtained. For ease of use, we extracted this code into to the MATLAB function `gap_uniform.m`. See also (as requested in Exercise 5.9) the MATLAB function `gap_pca.m` where instead of uniform random draws over the range of the component variables we draw our random bootstrap samples according to the principal eigen-directions found using the data set. These two routines provide ways of estimating the optimal number of clusters found in ones data. These two routines in this chapter are designed to work with the agglomerative clustering routine `linkage`, while the actual gap statistic is more general than that and can be used with any clustering algorithm. Companion routines (with the same name) aimed at computing the gap statistics using *model* based agglomerative clustering (i.e. the routine `agmbclust`) can be found in Chapter 6.

In addition, for this chapter we developed the routine `mk_matching_matrix_N.m`, which when given two partitions say G_1 and G_2, will extract a "matching matrix" $\mathbf{N}$ from them. The elements n_{ij} of $\mathbf{N}$ represent a count of the number of observations in the ith partition G_1 that are also in the jth partition of G_2. This routine is used in computing the Fowlkes-Mallows index in Problem 5.20.

Exercise Solutions

Exercise 5.1 (clustering of the `iris` data set)

In this exercise we attempt to apply hierarchical clustering to the `iris` data set. See Figure 58 for the results of running the MATLAB `linkage` command with the Euclidean distance as a measure of dissimilarity and no data preprocessing.

In Figure 58 (left) we use the *single* linkage (nearest neighbor) criterion to determine the distances between clusters and in the resulting dendrogram we see strong evidence of the phenomena called **chaining**, where a given seedling cluster grows by incrementally adding additional data points. The result of this chaining is that a given cluster can end up with data points in the same clusters that when considered individually can be very different. This chaining is manifest in the "staircase" like attachment of the individual leaves in the dendrogram. In Figure 58 (center) we see the results of using the `linkage` command with the *centroid* option for determining the distance between clusters. In this case, we clearly see the phenomena called **inversions** where by the clusters created later in the clustering process can have a smaller inter-cluster distance than clusters created earlier. This is observed in the dendrogram by loops present in the dendrogram. Using the Mahalanobis distance rather

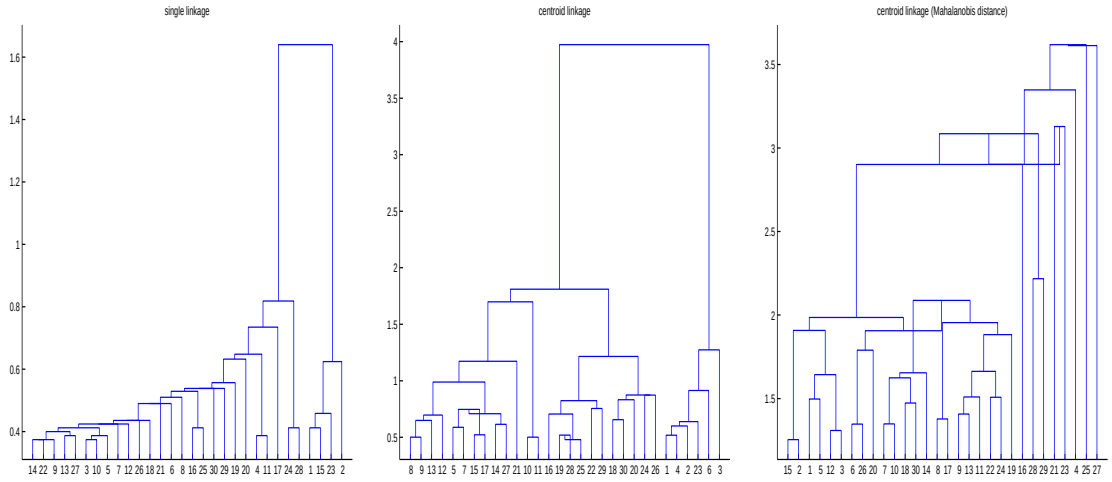

Figure 58: Some dendrograms for the `iris` data for various arguments to the MATLAB `linkage` command. **Left:** Using the option of single link clustering (the default). Note the phenomena of chaining. **Center:** Using the option for centroid clustering based on Euclidean distances. Note the inversions near the leaf clusters 7,15 and 19,28. **Right:** Using the option for centroid clustering based on Mahalanobis distances. Note the inversion near the leaf clusters 21 and 23. In all cases there are clearly two clusters, while in centroid based clustering (center plot) we see evidence for three clusters.

than the Euclidean distance does not improve things as we can see from Figure 58 (right).

Exercises 5.{2,3,5,10} (agglomerative clustering applied to various data sets)

For the data sets considered in these exercises, single link clustering almost always produces dendrograms that exhibit chaining. Because of this fact rather than present the single link results we will instead report on the results of applying agglomerate clustering (using the MATLAB command `linkage`) but with the *average* linkage option, which generally results in better clusters. If desired, the single link results can be obtained by running the individual MATLAB codes below.

Part (a): For this problem we apply agglomerative clustering to the `geyser` data set. See the MATLAB script `prob_5_2_geyser.m` for this implementation. Some of the results from running this script are presented in Figure 59. The gap-uniform procedure does a nice job at predicting two clusters which are then visually verified in the raw data as a viable clustering.

Part (b): For this problem we apply agglomerative clustering on the `singer` data set. This data set represents the height in inches of singers in the New York Choral Society. See the MATLAB script `prob_5_2_singer.m` for its implementation. Because height is a physical measurement we might expect it to be distributed normally among two classes corresponding to males and females. To study if this is true we apply the z-score transformation discussed in Chapter 1 to this data set before attempting agglomerative clustering. The results from running the above MATLAB script are shown in Figure 60. In Figure 60 (left) we see that

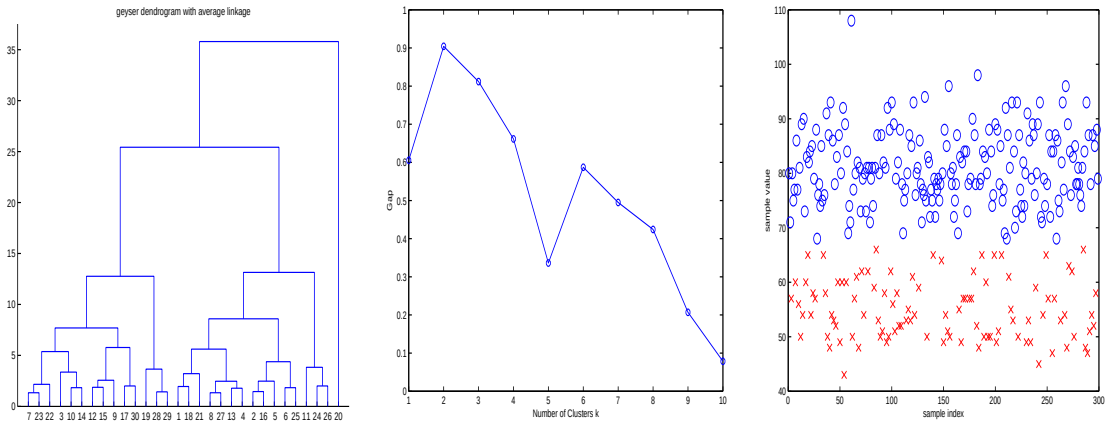

Figure 59: Some agglomerative clustering results for the `geyser` data set, using the MATLAB `linkage` command. **Left:** The dendrogram obtained using the option corresponding to average link clustering. **Center:** The resulting gap-uniform statistic plot, which provides evidence for two clusters. **Right:** The plot of the raw data color coded by its cluster membership (under the assumption of two clusters). Note the visual appearance of two "bands" of relatively well separated data.

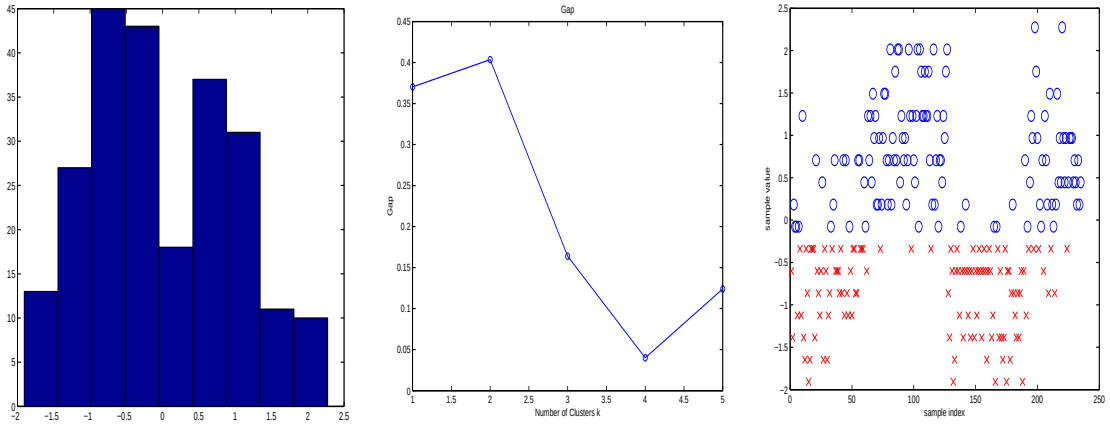

Figure 60: Agglomerative clustering applied to the `singer` data set. **Left:** A histogram of the raw `singer` data **Center:** The plot of the resulting uniform gap statistic obtained when using average link clustering. **Right:** The resulting clusters (color coded).

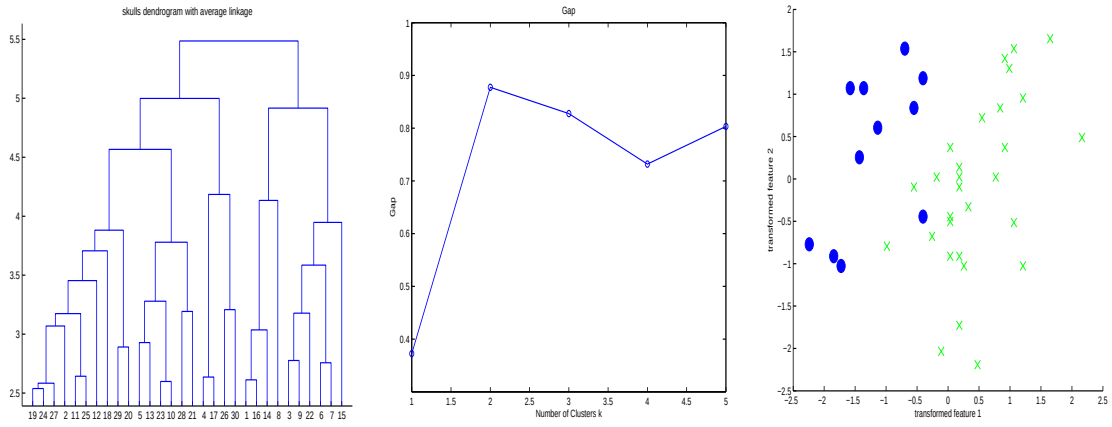

Figure 61: Agglomerative clustering on the `skulls` data set **Left:** The dendrogram produced using the `linkage` command with the option corresponding to average link clustering. Note two clusters maybe observed at the top most level of the tree. **Center:** The resulting uniform gap statistic plot. The idea that we have two clusters seems to be supported by this plot in that its peak is at $k = 2$. **Right:** The plot of the resulting agglomerative clusters under the assumption of two clusters.

the histogram of the data clearly shows two clusters which is confirmed in the gap statistic plot Figure 60 (middle). From this plot we see that the gap statistic obtains a maximum at the value of $k = 2$. A plot of the dendrogram obtained clusters (under the assumption of two clusters) is presented in Figure 60 (right).

Part (c): For this problem we apply agglomerative clustering on the `skulls` data set. See the MATLAB script `prob_5_2_skulls.m` for this implementation. Some of the results from running this script are shown in Figure 61. In performing this clustering we first transformed the data by subtracting from each point the global mean and dividing each point by the global standard deviation. There appears to be evidence for two clusters and in Figure 61 (right)

Part (d): For this problem we apply agglomerative clustering on the `spam` data set. See the MATLAB script `prob_5_2_spam.m` for its implementation. The problem with applying clustering directly to the `spam` data set is that the distribution of data under each class is very non-Gaussian. This can be seen by revisiting any of the two dimensional (or higher) projections we have considered on this data set earlier. The clustering of the data points seems to be stretched along orthogonal axis. In addition, the data set itself suffers in that a significant number of outliers exist. These outliers make it more difficult to perform cluster merging because they vastly skew the centroid locations used in many clustering methods. The effect from outliers can be controlled somewhat by using the "average" or "complete" linkage methods since they impose a more global view of the data when suggesting clusters.

To solve this problem, we began by attempting to apply agglomerative clustering *directly* to the `spam` data set. This is implemented exactly as the previous data sets we considered and is done in the MATLAB file `prob_5_2_spam_direct.m`. Some of the results obtained when this script is run are shown in Figure 62. We generally found several things of note when working with the `spam` data. The first is that it is necessary to use the argument "average" or

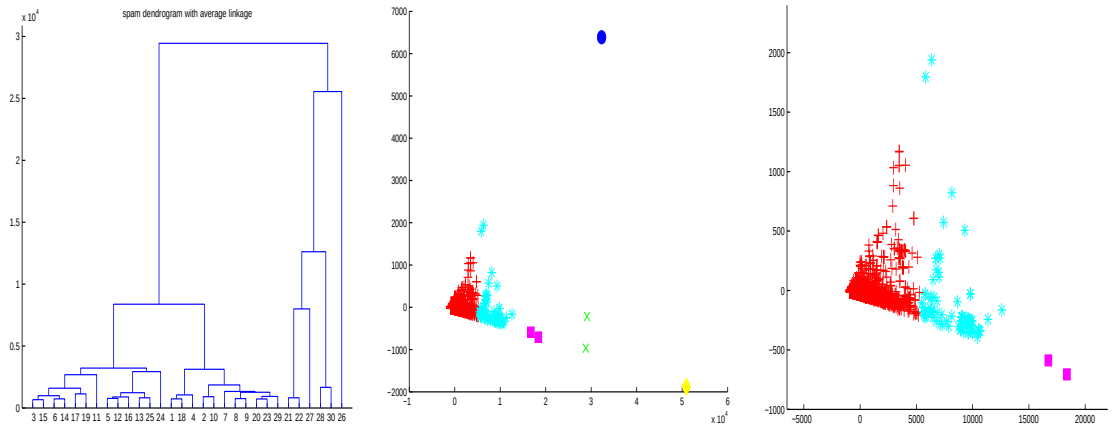

Figure 62: One of the more interesting examples from the book, agglomerative clustering of the `spam` data set. **Left:** The dendrogram of the `spam` data produced using the MATLAB `linkage` command with the option corresponding to average link clustering. Note the possible presence of outliers that appear in the dendrogram as a "column" of points with leaf number of 21, 22, 27, 30, and 26 on the right side of this plot. These outliers appear reasonably separated from the larger clusters shown as the two branches that split around a dissimilarity value of 0.9 and found the left region of the plot. **Center:** The labeled classes under the assumption of *six* classes. **Right:** A zoomed plot of the suggested clusters showing visually the possibility of two reasonably sized clusters, that might correspond to spam and non-spam email.

"complete" to the MATLAB `linkage` command for the reason suggested above. The second is that preprocessing the `spam` data seems to help the clustering algorithm find valid clusters. We found that first performing a z-score of the data and then either performing PCA based dimensionality reduction or SVD dimensionality reduction with column ordering seemed to provide nice looking dendrograms that did not display chaining. After this preprocessing step was performed we then applied the MATLAB `linkage` command to the resulting data.

An example of the dendrogram produced when running this code is given in Figure 62 (left). In that figure we see two larger clusters on the left side of the figure. In addition, we show the clusters that result under the assumption of six clusters in Figure 62 (middle). In that figure, we see the outliers clearly denoted as they become clusters containing only a few points. By plotting six clusters we have been able to remove the presence of the outliers and focus on clusters that probably more accurately represent an actual phenomena of interest. A plot around the central region containing the two largest clusters is shown in Figure 62 (right). It should be noted that when run on this data set the `gap` statistic continually predicted only one cluster. This might lead to the conclusion that the gap statistic is not very robust to outliers and should not be used to determine the number of clusters when the data has a significant number of them.

An alternative procedure that could be used to deal with outliers but that was not pursued further here, would be to iteratively perform clustering and removal of clusters that are below a given size threshold. If this algorithm was implemented on the `spam` data set one might find that in the first few iterations we remove the outliers and the later clusterings

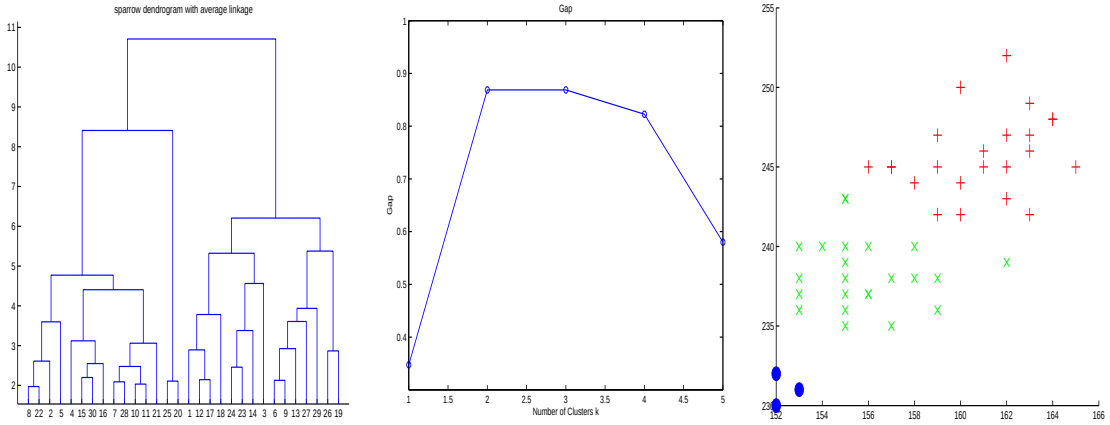

Figure 63: Agglomerative clustering of the `sparrow` data set. **Left:** The dendrogram produced using the `linkage` command with the option corresponding to average link clustering. **Center:** The resulting gap statistic plot which indicates the possibility of two or three clusters. **Right:** The plot of the resulting clusters under the assumption of *three* clusters. Note the three points in the lower left corner might be outliers.

provide separation between spam and non-spam.

Part (e): For this problem we applied agglomerative clustering to the `sparrow` data set. This is implemented in the MATLAB script `prob_5_2_sparrow.m`. In Figure 63 (left) we see two large clusters and two possible "outlier" clusters labeled in the dendrogram by the leave numbers 20 and 25. The gap statistic does a good job of predicting the number of clusters and returns the value of two or three (see the gap plot in Figure 63 (middle)). If we plot the resulting clusters under the assumption that there are three we obtain the plot seen in Figure 63 (right). In this later plot we see three points located in the lower left corner of this figure that maybe "outliers". Whether these points belong in their own cluster or should actually be included in the larger cluster of green crosses (which would happen if we had assumed only two clusters) is for the data analysts to decide.

Part (f): For this problem we apply agglomerative clustering on the `oronsay` data set. See the MATLAB script `prob_5_2_oronsay.m`, for the implementation of this. Some of the results from running this routine are shown in Figure 64. For this problem the dendrogram is shown in Figure 64 (left). Since MATLAB does not denote how many elements are in each dendrograms leaf node, in that figure we see the possibility for a variety of clusters. The uniform gap statistic plot for this data set is shown in Figure 64 (middle). This plot clearly suggests the possibility of three clusters. Finally, in Figure 64 (right) we present the clusters that are found when we assume *five* clusters. This is helpful in observing what the agglomerative clustering algorithm clustered into clusters with only a few data points. From this plot we see that since several clusters contain only a few points these points are potential candidates for outliers.

Part (g): For this problem we apply agglomerative clustering on some of the gene expression data sets. We begin with the `lungB` data set. See the MATLAB script `prob_5_2_lungB.m` for its implementation. Some of the results from running this routine are shown in Figure 65.

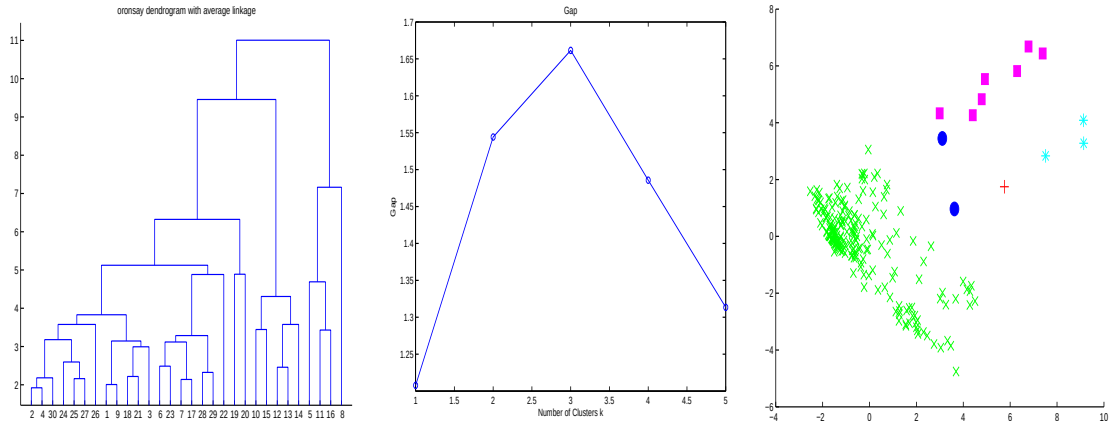

Figure 64: Agglomerative clustering on the `oronsay` data set. **Left:** The dendrogram produced using the `linkage` command using average link clustering. **Center:** The resulting gap statistic plot, which clearly suggests a hypothesis of three clusters. **Right:** The plot of the clusters that result under the assumption of *five* clusters.

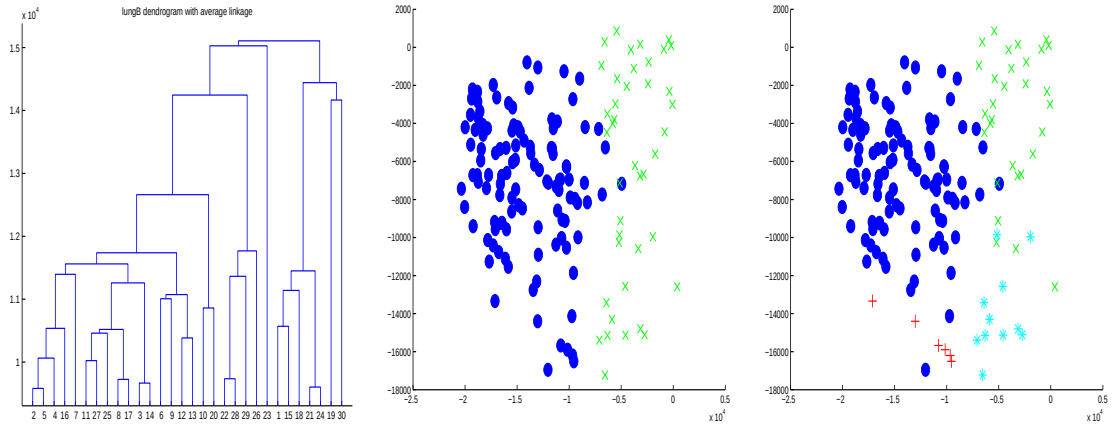

Figure 65: **Left:** The dendrogram of the `lungB` data produced using the `linkage` command, using the option corresponding to average link clustering. **Center:** The resulting clusters. **Right:** The plot of the clusters compared with the known class labels.

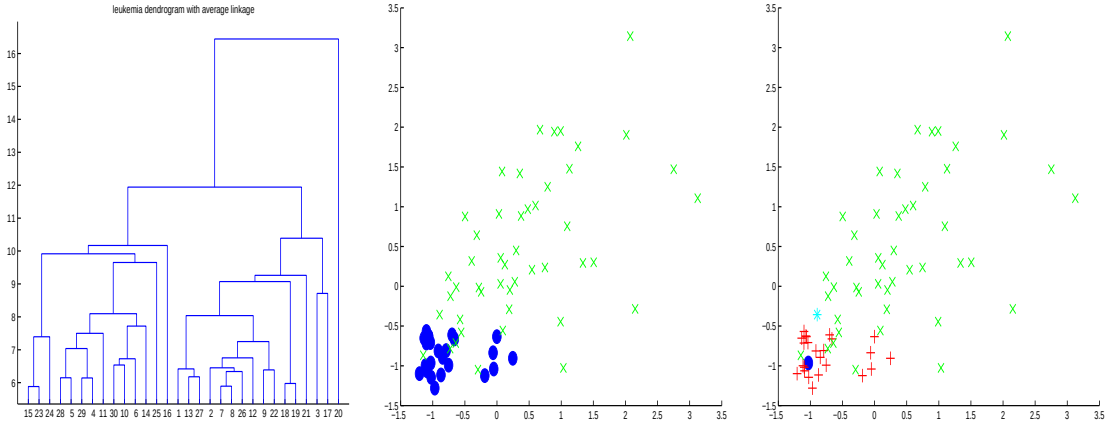

Figure 66: **Left:** The dendrogram of the `leukemia` data produced using the `linkage` command, using the option corresponding to average link clustering. **Center:** The resulting clusters. **Right:** The plot of the clusters compared with the known class labels.

The gap statistic (not shown) on this data set predicted two clusters. To compare how well the unsupervised clustering algorithm performed in comparison to the known labels provided by the `lungB` data set we also plot the predicted clusters against the known class labels in Figure 65 (right). Data points where the clustering and true classes agree are labeled all in one color. Points where the two algorithms disagree are labeled in a different color. We see that the agglomerative clustering algorithm seems to predict a good number of samples i.e. the number of miss-classified samples is relatively small.

Next we considered the `leukemia` data set. See the MATLAB script `prob_5_2_leukemia.m` for its implementation. Some of the results from running this routine are shown in Figure 66. The gap statistic (not shown) predicted two clusters. As in the earlier part of this problem, to compare how well the unsupervised clustering algorithm performed in comparison to the known class labels provided by the data set we also plot the predicted clusters against the known class labels in Figure 66 (right). We see that the clustering result seems to predict a good number of samples i.e. the number of miss-classified samples is rather small.

Exercise 5.7 (the gap statistic on uniform random data)

For this problem we generate two dimensional uniform random data and then attempt to apply hierarchical clustering to it. We then compute the uniform gap statistic on the resulting data set to compute an estimate the number of clusters. Since the data we generate is random we expect the resulting clustering to be based on the specific random sample drawn and to have no actual generalization ability. When we run the MATLAB script `prob_5_7.m` that implements this procedure we produce a plot of the uniform data, and then two plots corresponding to the gap statistic. See Figure 67 for the output obtained when we run the above script. In Figure 67 (middle) we are plotting the actual and expected values of $\log(W_k)$ where W_k is a measure of tightness of each of the clusters. To define this expression we first

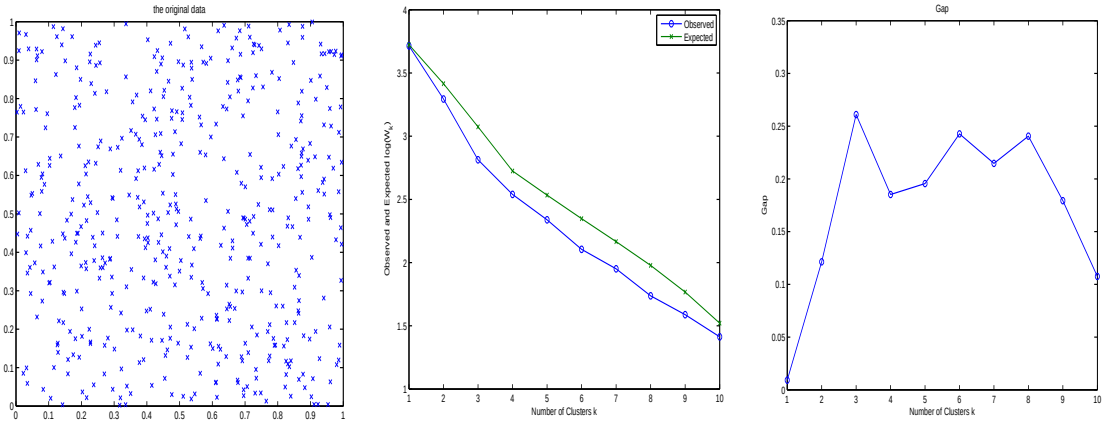

Figure 67: **Left:** The original uniform two-dimensional random data. **Center:** The observed v.s. expected value of $\log(W_k)$ as a function of k (the hypothetical number of clusters) where the expected value of W_k is taken under the assumption of uniform random data. Note how closely these two curves are to each other. The observed values of $\log(W_k)$ is certainly within the standard error the bootstrap generated values of $\log(W_k)$. **Right:** The standard error adjusted difference between the measured $\log(W_k)$ and bootstrapped $\log(W^*_{b,k})$ curves. The fact that the magnitude of this difference is small (of magnitude 0.2) indicates that the suggested number of clusters (here 2) is probably due to noise.

introduce the expression, D_r, the pairwise distance of all points in the cluster r as

$$D_r = \sum_{i,j \in C_r} d_{ij} \,. \tag{1}$$

Then for a clustering with k clusters we average this value over all of the k clusters. If n_r is the number of points in the rth cluster define W_k as

$$W_k = \frac{1}{2} \sum_{r=1}^{k} \frac{1}{n_r} D_r \,. \tag{2}$$

Using these values in Figure 67 (right) we plot the deviation of the observed W_k from bootstrap estimated values $W^*_{k,b}$. The distance between these two expressions is how much the data clustering deviates from what would be expected to be obtained randomly. That is for B bootstrap samples of "uniform clustering" we plot (as a function of k the number of clusters)

$$\text{gap}(k) = \frac{1}{B} \sum_{b=1}^{B} \log(W^*_{k,b}) - \log(W_k) \,. \tag{3}$$

Exercise 5.8 (the gap statistic on well separated clusters)

For this problem we generate some two dimensional data that are produced from well separated clusters and then apply hierarchical clustering to it. After the clustering we then use the uniform gap procedure to estimate the number of clusters by comparing the observed

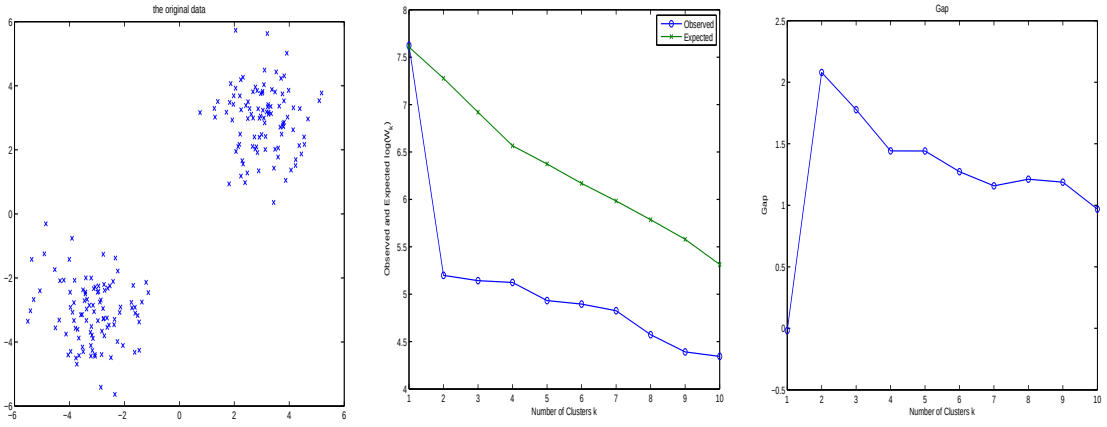

Figure 68: **Left:** The original data, generated as samples from spherical Gaussians with two different centers. **Center:** The observed v.s. expected value of $\log(W_k)$ as a function of k (the hypothetical number of clusters) where the expected number of clusters is taken under the assumption of uniform random data. Note how differently these two curves are from each other. **Right:** The standard error adjusted difference between the measured $\log(W_k)$ and bootstrapped $\log(W^*_{b,k})$ curves. Note the largest difference occurs at the value $k = 2$ and corresponds to the actual number of clusters present in the data set.

clustering with the null hypothesis that the data is drawn from a uniform random distribution. We would hope that the gap procedure could extract the correct number of clusters. When we run the MATLAB script `prob_5_8.m` we first produce a plot of the data considered, and then the same two plots as in Exercise 5.7 (see above) on the gap statistic. See Figure 68 for the output obtained when we run the above script.

Exercise 5.9 (the gap-PCA procedure)

Following the discussion in the text we implement the gap-PCA procedure in the MATLAB function `gap_pca.m`. Our implementation is based on Example 5.7 from the book, which is packaged in the MATLAB function `gap_uniform.m`. The only modification required for `gap_pca.m` is to compute a singular value decomposition (SVD) on the column centered data set and then generate random variables according to the size of the singular values obtained. When one runs the MATLAB script `prob_5_9.m` we apply the gap-PCA algorithm to uniform data (as in Exercise 5.7) and to two cluster data (as in Exercise 5.8). Results from these computational experiments are shown in Figure 69. Note that these results match well with what was found when we performed the same experiments using the gap-uniform statistic.

Exercise 5.11 (`mojenaplots`)

For this exercise we consider some of the data sets from Exercise 5.2 above and apply the upper tail rule and present `mojenaplot` of the clusters that result. For this exercise we

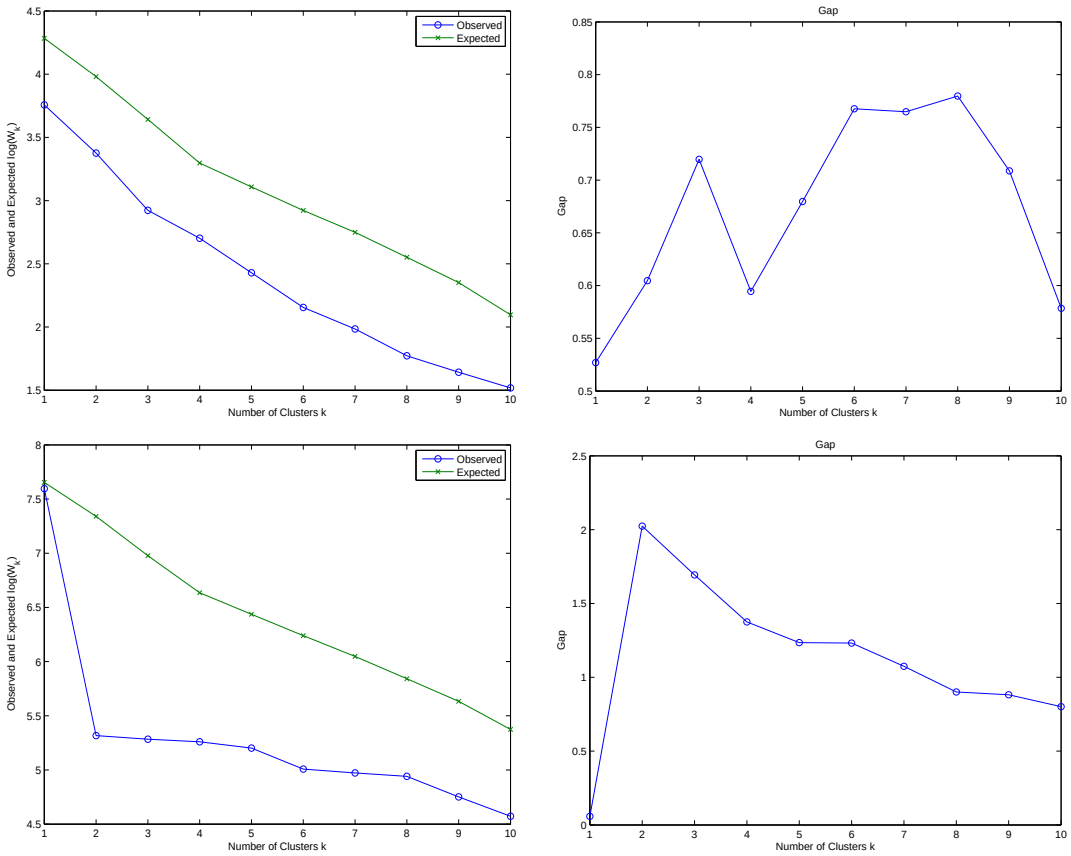

Figure 69: **Top Left:** The observed v.s. expected values of $\log(W_k)$ as a function of cluster number where the actual data is two-dimensional uniform random and data for the null hypothesis is drawn randomly from its principal components. See the discussion on the PCA-gap procedure in the book. The fact that two curves are so close to each other indicates that there is no strong statistical difference between the two data distributions. **Top Right:** The standard error adjusted difference between the $\log(W_b(k))$ curves on the observed data set and the average of the bootstrapped sample estimates $\log(W^*_{b,k})$. The fact that the magnitude of this difference is so small again indicates that any suggested number of clusters is probably due to noise. **Bottom Left:** The same plot as presented above (Top Left) but with data that explicitly was generated from two clusters. Note how different the two curves are in this case. **Bottom Right:** The same plot as presented above (Top Right) but with the two cluster data set. Note that the maximum of this curve is clearly at $k = 2$ the correct number of clusters.

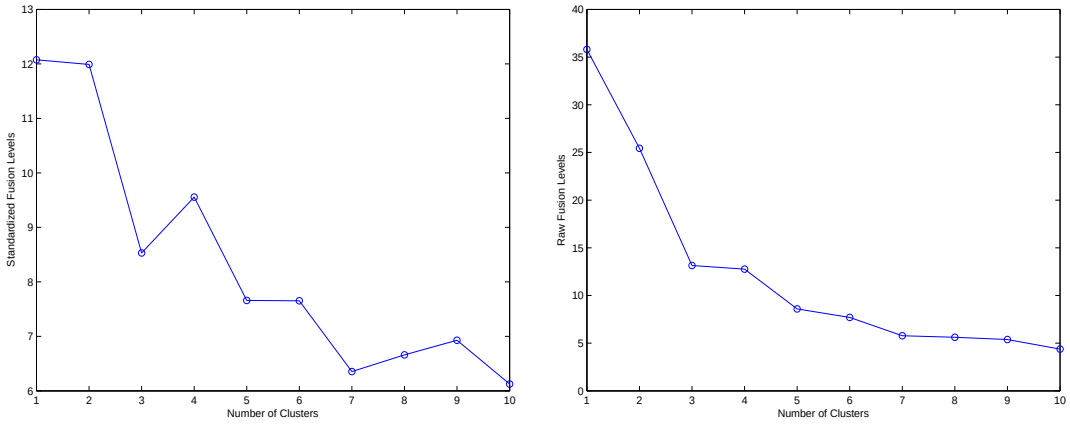

Figure 70: The upper tail rule on the `geyser` data set. **Left:** The standardized fusion levels. **Right:** The raw fusion levels. Both plots have "elbows" at the index $k = 3$ indicating that three clusters might be present.

closely follow example 5.5 from the book.

Part (a): See the MATLAB script `prob_5_11_geyser.m` for an implementation of this where we apply the graphical Mojena rule to the `geyser` data set. When we run that script we obtain the plots shown in Figure 70.

Part (b): See the MATLAB script `prob_5_11_spam.m` for an implementation of the application of the graphical Mojena rule to the `spam` data set. When we run that script we obtain the plots shown in Figure 71.

Exercise 5.12,13 (values of the Rand index)

The rand index would be zero if *none* of the proposed clusterings agree on their assessment of the numbers of clusters. For example, if we have a five object data set with two proposed clustering given by

$$c_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \text{and} \quad c_2 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix},$$

then our Rand index should be zero. In this proposed clustering, the c_1 labeling suggests that all data points should belong to the *same* cluster, while the c_2 suggests that all the data points as belonging to *different* clusters. These two views of our data are conflicting and as such result in a zero Rand index. As another example of the values produced by Rand

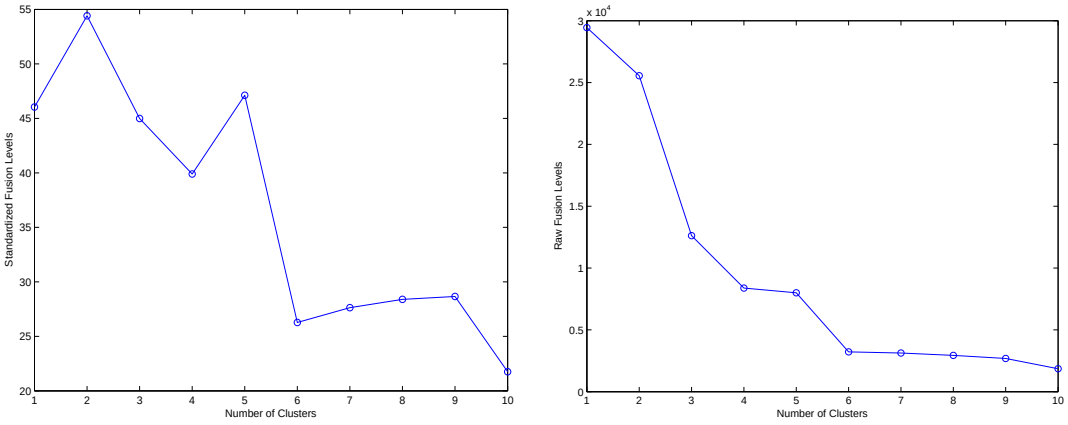

Figure 71: The upper tail rule on the `spam` data set. **Left:** The standardized fusion levels. **Right:** The raw fusion levels. Both plots seem to have "elbows" around the index $k = 3$ or $k = 4$ indicating that three or four clusters might be present in this data set. Given how noisy this data set is this is quite a good result.

indexes consider another proposed clusterings of our five points

$$c_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 2 \end{bmatrix} \quad \text{and} \quad c_2 = \begin{bmatrix} 2 \\ 2 \\ 2 \\ 1 \\ 1 \end{bmatrix},$$

In this case the two clusterings agree in that the first three points should be in one cluster while the last two points belong in their own cluster. In this case, the Rand index will be one indicating perfect agreement of the clusterings. Finally, we can compute the Rand index for two copies of the *same* clusterings. This will give the Rand index of one, again indicating perfect agreement of the clusters. See the MATLAB script `prob_5_12.m` where we numerically demonstrate these results by calling the `randind` and `adjrand` functions.

Exercise 5.14,15 (k-means v.s. agglomerative clustering of `oronsay`)

For this exercise we apply k-means and agglomerative clustering (which has already been done in Exercise 5.2) to the `oronsay` data set using the correct number of known classes of $k = 3$. We then plot the resulting clustering using a silhouette plot, and report the Rand and adjusted Rand index between the reported clusters and the true ones. See the MATLAB script `prob_5_14.m` for an implementation of this procedure. When this script is run it produces the two silhouette plots shown in Figure 72. In addition, the mean silhouette value for the agglomerative clustering is 0.82 while for k-means clustering it is 0.67. This provides some indication that agglomerative clustering may provide a better clustering. The Rand index for the agglomerative clustering takes the values of 0.44 or 0.03 depending on whether one uses the function `randind` or `adjrand` to calculate it. For k-means clustering the corresponding values are 0.68 and 0.39. Both the Rand index and the adjusted rand

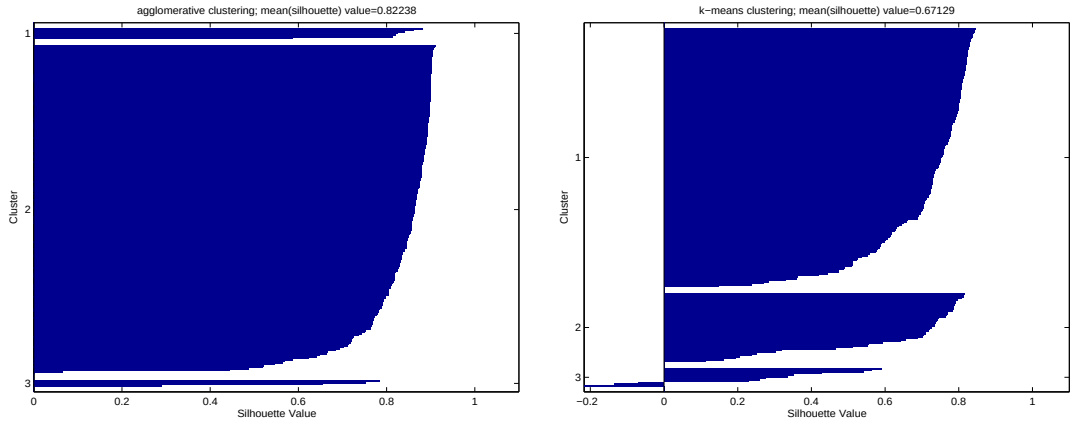

Figure 72: Silhouette plots of the `oronsay` data set assuming $k = 3$ clusters. **Left:** Using agglomerative clustering. **Right:** Using k-means clustering. Note that under both clusterings three clusters appears to be a good choice as the mean silhouette value is quite large.

index have values between zero and one, with the value of zero indicating clusterings that are relatively dissimilar and the value of one representing clusters that are similar. Both of these indicate that the k-means algorithm may have produced clusters more like the true clustering.

Next we repeat the above clustering experiments but for a range of values for the number of clusters to extract. See the MATLAB file `prob_5_14_various_k.m` where these experiments are performed. To asses the number of clusters present in this data set one method is to follow Kaufman and Rousseeuw [7] where we select the number of clusters that gives the *largest* average silhouette value. When we plot the average silhouette values for both clustering methods as a function of the number of clusters we obtain the results shown in Figure 73 (left).

Next we plot values of the Rand index [9] and adjusted Rand index of the resulting clusters against the known true cluster labels. Here we considered truth to be obtained from the "midden" class labeling. This plot is presented in Figure 73 (right). Ideally we would select the number of clusters that produces the *largest* value of the Rand index computed in this way. In most unsupervised learning tasks we would not actually know the true class label and so could not directly use this metric.

Exercise 5.17 (Example 5.7 with the gap-PC method)

In this exercise we run the gap-PC method on the `lungB` data set. When we run the `prob_5_17.m` code we produce the plots shown in Figure 74. From the plot in Figure 74 (right) we see strong evidence for *two* clusters in agreement with the known truth for the given data set. In Figure 74 (left) we conclude since the two curves are relatively separated that the results are significant.

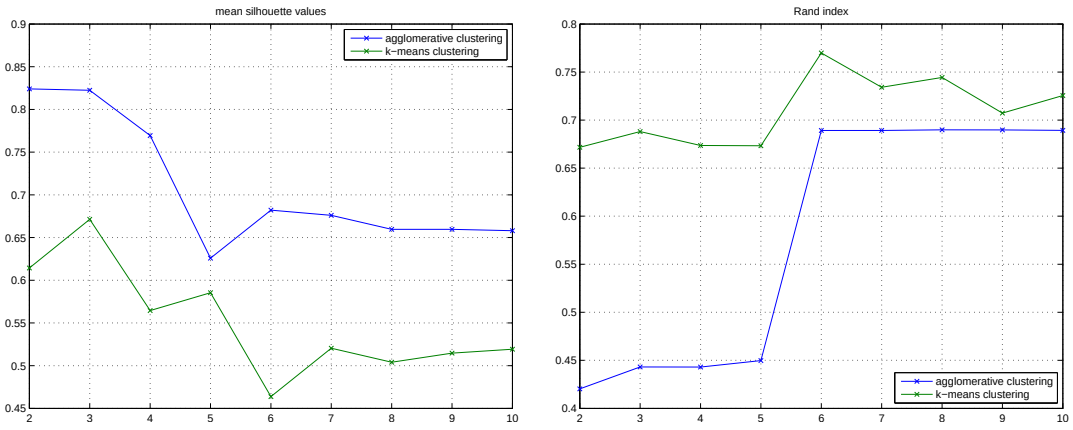

Figure 73: Cluster validity metrics for the `oronsay` data set for both agglomerative and k-means clustering. **Left:** Mean silhouette values as a function of number of assumed clusters. As larger values are better we would conclude the data contained *three* clusters (the starting x-axis value starts at $k = 2$). **Right:** The Rand index compared to the the known clustering labels for "midden" labeling. Since the k-means result is larger than the agglomerative clustering result for all k values we can conclude that the k-means procedure may yield clusters more aligned with truth.

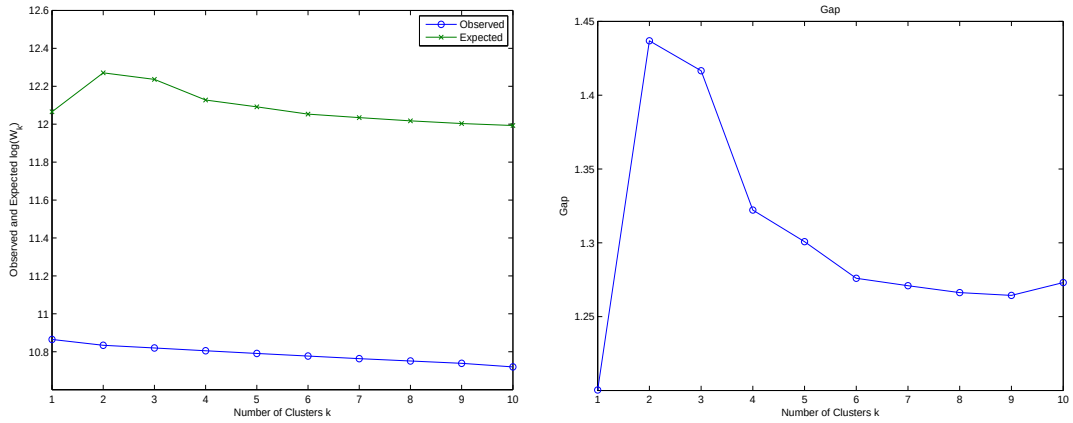

Figure 74: Results of applying the PCA-*gap* procedure to the `lungB` data set.

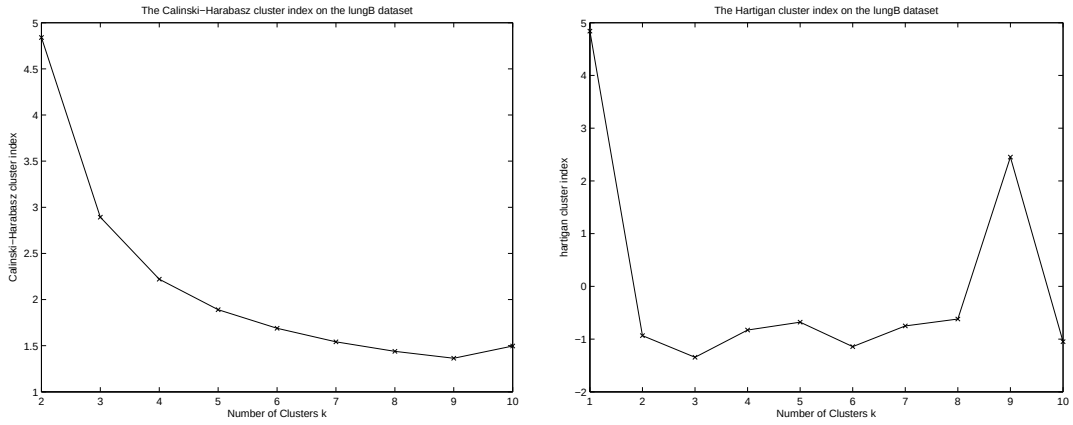

Figure 75: **Left:** Results of applying the cluster selection criterion of Calinski-Harabasz to the `lungB` data set. We find the largest value of ch_k at $k = 2$ suggesting *two* clusters in agreement with example 5.7. **Right:** Results of applying the cluster selection criterion of Hartigan to the `lungB` data set.

Exercise 5.18 (Calinski and Harabasz's cluster index)

The Calinski and Harabasz's cluster index, denoted ch_k, for a data partition with k clusters as describe in the text is implemented in the MATLAB function `cal_har_k_index.m`. To use this function in the MATLAB script `prob_5_18.m` we load the `lungB` data set and compute the value of ch_k as a function of k for clusters found by agglomerative clustering and the "average" linkage option. A plot of the results of this procedure is shown in Figure 75 (left). This metric gives a suggestion of $k = 2$ clusters in agreement with what we had concluded earlier.

Exercise 5.19 (Hartigan's cluster index)

For this problem, Hartigan's cluster index hart_k, is used to compare the relative merits of clustering a given data set with k or $k + 1$ clusters. The implementation of this index is described in the text accompanying this problem and is implemented in the MATLAB function `hartigan_k_index.m`. To exercise this function, in the MATLAB script, `prob_5_19.m`, we load the `lungB` data set and from it compute the values of hart_k as a function of k for $1 \leq k \leq 10$. The estimated number of clusters is the *smallest* value of k that results. A plot of the results when this procedure is run is shown in Figure 75 (right). We see that the Hartigan index concludes that this data set contains two or three clusters.

Exercise 5.20 (the Fowlkes-Mallows index)

In computing the Fowlkes-Mallows index we require the "matching matrix" $\mathbf{N}$ defined in the "additional notes" section for this chapter above. To facilitate this calculations we

implemented this procedure in the MATLAB function `mk_matching_matrix_N.m`. Using this, the MATLAB function `fowlkes_mallows_index.m` implements the Fowlkes-Mallows cluster index as described in this problem. We exercise both of these routines in the MATLAB script `prob_5_20.m` on the `oronsay` data set. When we run this command, we find a Fowlkes-Mallows cluster index of 0.70 when comparing k-means clustering against the true class labels under "midden" labeling. The corresponding index for agglomerative clustering is 0.62. In both cases, we clustered our data assuming three classes and used for the truth clustering labels the values specified from the "midden" labeling.

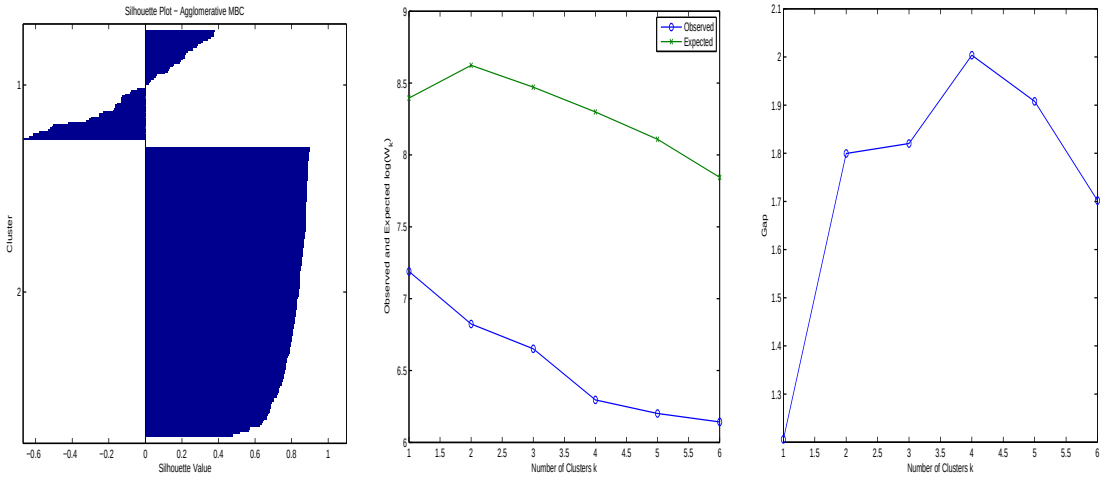

Figure 76: Some agglomerative model based clustering results for the `oronsay` data set. **Left:** The silhouette plot associated with agglomerative based clustering assuming *two* clusters. One cluster appears to be quite stable while the second cluster less so. **Center:** The observed v.s. expected value of $\log(W_k)$ as a function of k (the hypothetical number of clusters) where the expected value of W_k is taken under the assumption of uniform random data. **Right:** The standard error adjusted difference between the measured $\log(W_k)$ and bootstrapped $\log(W^*_{b,k})$ curves. See Equation 2 for the definition of W_k.

Chapter 6: Model-Based Clustering

Additional Notes

As discussed in Chapter 5 we can implement the gap statistic using any clustering scheme. As this chapter deals with model-based clustering we have implemented a function that computes the gap statistic using the `agmbclust` function to perform clustering on the null hypothesis (the set of random points). The function that we implemented (`gap_uniform.m`) has the same name as the corresponding one in Chapter 5 that used agglomerative clustering but is located in a different directory (`Chapter6`) to avoid namespace collision. This function is used in several of the exercises below.

Exercise Solutions

Exercise 6.1 (model-based agglomerative clustering on the `oronsay` data set)

See the MATLAB script `prob_6_1.m` for an implementation of this problem. From the gap-statistic we have observe two specifications of what the number of clusters might be. If we consider the gap-statistic results obtained from `gap_uniform.m`, we have two possible values for the number of clusters found in this data set. The first ($k = 2$) is the point where the

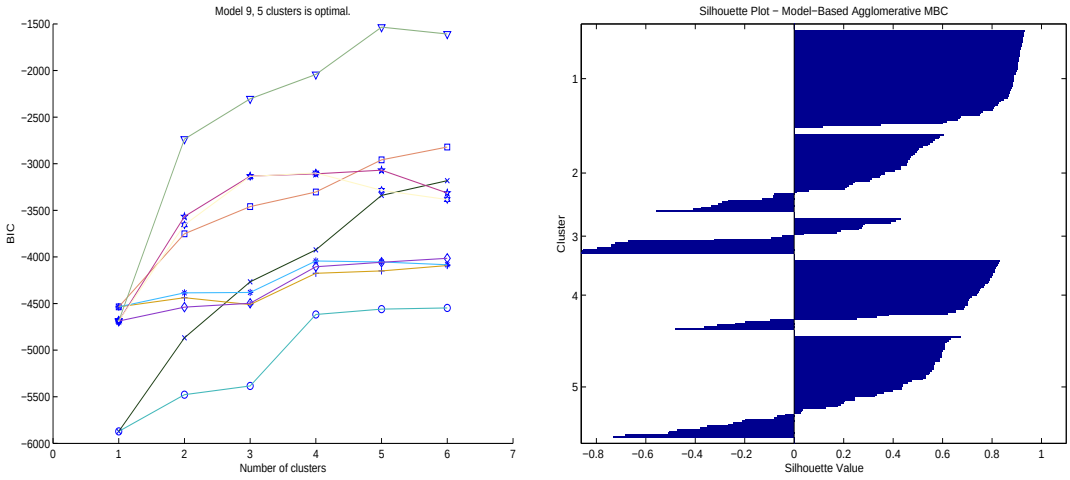

Figure 77: Some model-based clustering results for the `oronsay` data set. **Left:** The resulting Bayesian Information Criterion (BIC) plots that result when the `mbclust.m` function is run. This procedure estimates that the maximum BIC is produced under model number nine (the most general covariance structure) with *five* clusters. **Right:** The silhouette plot associated with agglomerative based clustering and assuming *five* clusters.

observed values of W_k *first* differs statistically (by one standard deviation) from the average of the bootstrap values $W^*_{k,b}$. The second ($k = 4$) is where the curve of W_k is maximally different from the curve of the average of the bootstrap samples $W^*_{k,b}$. In Figure 76 (right) the gap statistic obtains its maximum at *four* clusters. Considering each of these two clustering options the mean silhouette width in the two cluster case is given by 0.57, while that for four clusters is given by 0.52. If we follow Kaufman and Rousseeuw [7] by selecting the number of clusters that gives the *largest* average silhouette value, we would select two clusters for this data set. A silhouette plot of two clusters is shown in Figure 76 (left).

Exercise 6.2,11 (full model-based clustering on the `oronsay` data set)

For this problem we apply full model-based clustering on the `oronsay` data set using the MATLAB function `mbclust`. See the MATLAB script `prob_6_2.m` where we implement this procedure. When we run the code in that script we obtain the plots shown in Figure 77. The Bayesian Information Criterion (BIC) selects the ninth covariance model (the most general covariance specification) and five clusters. In Figure 77 (left) we plot the BIC as a function of cluster number for each of the nine models. One see that indeed the ninth model (the topmost curve) has the largest BIC and that it peaks at the value of $k = 5$. In Figure 77 (right) we present a silhouette plot of the clusters corresponding to $k = 5$. The average silhouette value in this case is given by 0.42. Note that this is smaller than the two values computed in Exercise 6.1.

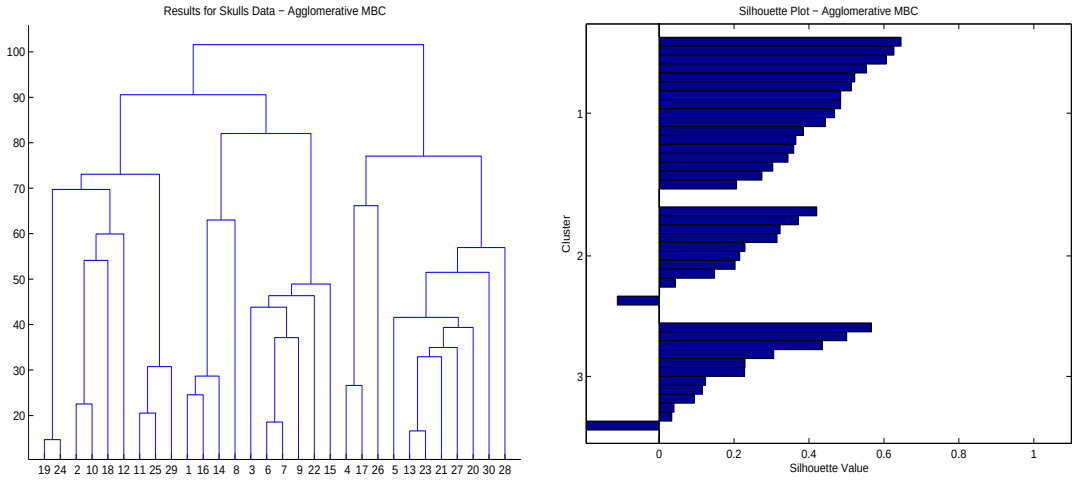

Figure 78: **Left:** Some agglomerative model-based clustering results for the `skulls` data set. **Right:** The silhouette plot associated with agglomerative based clustering and assuming three clusters.

Exercise 6.5 (cluster size and cluster volume)

In the Banfield and Raftery [1] and Celeux and Govaert [2] decomposition of the k-th covariance matrix in a finite mixture model given by

$$\Sigma_k = \lambda_k D_k A_k D_k^T \,, \tag{4}$$

the variable λ_k governs the volume of the k-th cluster and is a "global" scaling factor for the k-th cluster. The orientation and size of this cluster is determined by the matrix D_k, which specifies the principal directions and A_k which specifies (via the normalized eigenvalues) the magnitude for a single standard deviation move in each direction. The volume of the cluster is specified once these components are determined. Defining the size of the the cluster to be the number of observations falling into this cluster we have that for a finite mixture model given by

$$f(x; \hat{\pi}_k, \hat{\mu}_k, \hat{\Sigma}_k) = \sum_{k=1}^{c} \hat{\pi}_k \phi(x; \hat{\mu}_k, \hat{\Sigma}_k) \,, \tag{5}$$

this number for the k-th cluster must be proportional to $\hat{\pi}_k$, which is determined by the Expectation Maximization (EM) learning procedure.

Exercise 6.7 (agglomerative model-based clustering on various data sets)

For this problem we use the Exploratory Data Analysis (EDA) toolbox function `agmbclust.m` on the suggested data sets.

Part (a): See the MATLAB script `prob_6_7_skulls.m` for the application of agglomerative model-based clustering on the `skulls` data set. When that code is run the gap-statistic

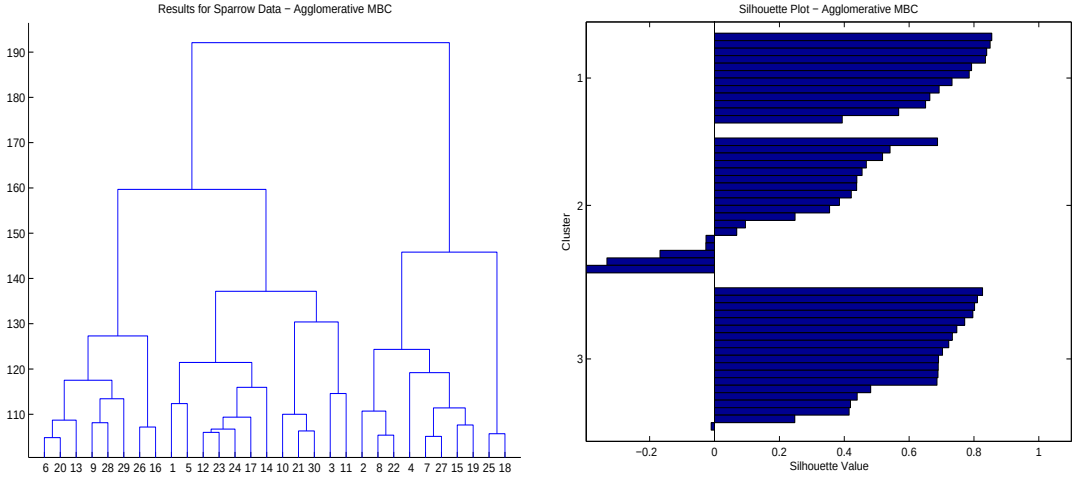

Figure 79: **Left:** Some agglomerative model-based clustering results for the `sparrow` data set. **Right:** The silhouette plot associated with agglomerative based clustering under the assumption of three clusters.

predicts three clusters. The dendrogram produced under this assumption is displayed in Figure 78 (left). Plotting a silhouette plot with three clusters gives the result in Figure 78 (right). Barring a few possible out-ling points three clusters appears to cluster the data quite well.

Part (b): See the MATLAB script `prob_6_7_sparrow.m` for the application of agglomerative model-based clustering on the `skulls` data set. When that code is run the gap-statistic predicts three clusters. The dendrogram produced is shown in Figure 79 (left) while the silhouette plot under the assumption of three clusters is given in Figure 79 (right). The average silhouette value for this clustering is 0.499.

Part (c): See Exercise 6.2 (above) where we apply agglomerative model-based clustering on the `oronsay` data set.

Part (e): See the MATLAB script `prob_6_7_lungB.m` for the application of agglomerative model-based clustering on the `lungB` data set. When that code is run the gap-statistic predicts two clusters. The dendrogram produced is shown in Figure 80 (left) while the silhouette plot under this assumption is given in Figure 80 (right). The average silhouette value for this clustering is 0.074. From the dendrogram it looks like something like the phenomena on "chaining" is occurring. The three points to the far left of the dendrogram should certainly be studied as possible outliers. Were these points found to be outliers they could be removed and clustering attempted again.

Exercise 6.9 (two or four clusters in the `iris` data set)

See the MATLAB code `prob_6_9.m` for an implementation of this problem. When this script is run it produces the plot show in Figure 81. There we see that two clusters is a very good clustering while four clusters is some what worse. The fact that two clusters provide a good

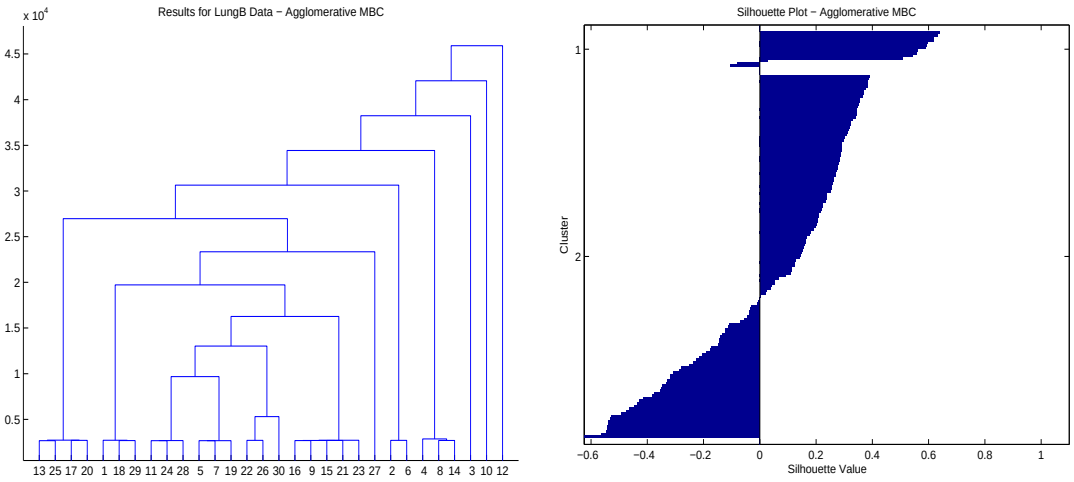

Figure 80: **Left:** Some agglomerative model-based clustering results for the `lungB` data set. **Right:** The silhouette plot associated with agglomerative based clustering and assuming two clusters. Note that this does not appear to be a particularly good clustering.

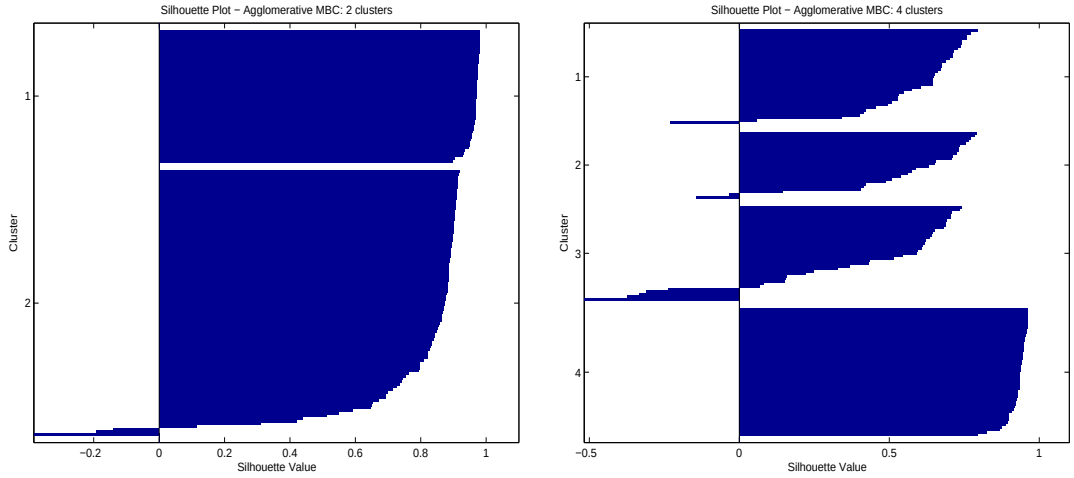

Figure 81: Some agglomerative model-based clustering results for the `iris` data set. **Left:** The silhouette plot under the assumption of *two* clusters. **Right:** The silhouette plot under the assumption of *four* clusters.

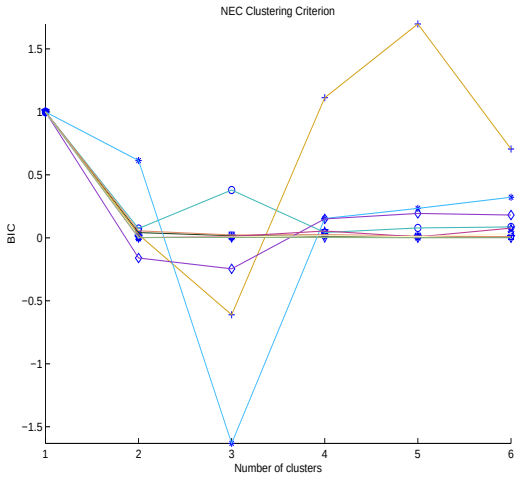

Figure 82: Plots of the NEC clustering criterion for the nine covariance models from Exercise 6.2. The minimum value of all these curves suggests this data is best modeled with *three* clusters and using the fourth covariance specification.

clustering follows from the known fact that that two (of the three present) species of iris are somewhat similar. The species *Iris versicolor* and *Iris virginica* are similar and more difficult to distinguish between than the species *Iris setosa*.

Exercise 6.10 (implementing the NEC)

For this problem we began by modifying the EDA toolbox function `mixclass.m` to return an estimate of the *posteriori* probability that x_i belongs to cluster k and to return the mixture density $\hat{f}$ evaluated at each of the data samples x_i. That is we now return

$$\hat{\tau}_{ik}(x_i) = \frac{\hat{\pi}_k \phi(x_i; \hat{\mu}_k, \hat{\Sigma}_k)}{\hat{f}(x_i; \hat{\pi}_k, \hat{\mu}_k, \hat{\Sigma}_k)} \quad \text{for} \quad k = 1, \cdots, c; \quad i = 1, \cdots, n\,,$$

and the value of $\hat{f}(x_i)$ given in Equation 5 calculated at each of the input samples x_i. Here $\phi(\cdot)$ is the normal density function. A MATLAB function to implement the NEC criterion is then implemented in `NEC.m`.

Exercise 6.11 (using the NEC)

See Exercise 6.11 for a discussion of the implementation of the NEC and see Exercise 6.2 above where we select the number of clusters that corresponds to its *minimum* value. The gap statistic (see exercise 6.1 above) suggests either two or four clusters while the NEC suggests *three* clusters when using the fourth covariance specification.

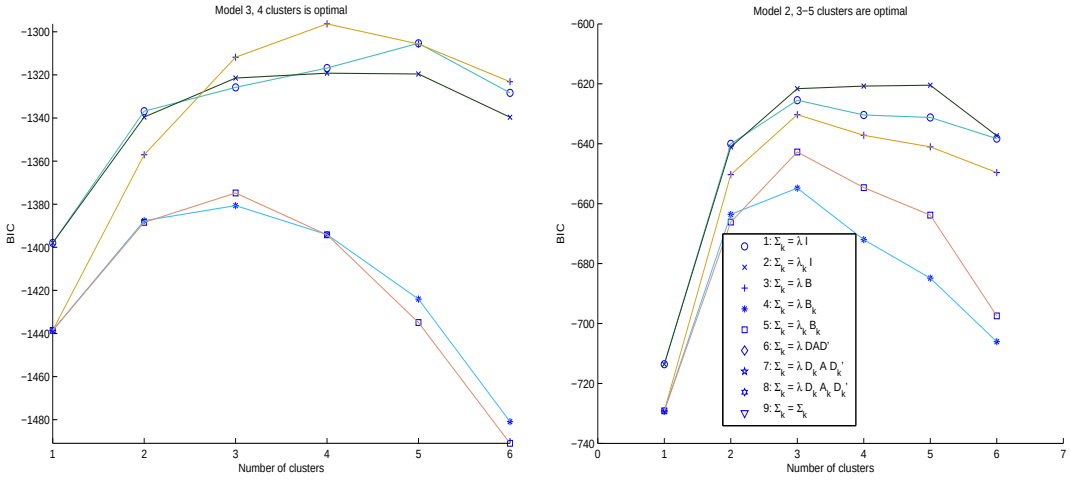

Figure 83: **Left:** The BIC plots for the `skulls` data set. **Right:** The BIC plots for the `sparrow` data set.

Exercise 6.15 (using the best model but not the best number of clusters)

For this exercise we apply model based agglomerative clustering as in example 6.7 to the `iris` data set where we found that the best model was model 9 (the most general specification with variable volume, shape, and orientation). Under this model the Bayesian Information Criterion (BIC) selected *two* clusters in contrast to the known number (three) of species of iris considered in Fishers [5] famous study. Example 6.8 from the book considered a silhouette plot of the *three* cluster result. In this exercise we plot the *two* cluster result. This problem is implemented in the MATLAB script `prob_6_15.m`. When this code is run it produces a silhouette plot (not shown) that shows very nice clustering and is much like the one in Figure 81 (left).

Exercise 6.17 (full model-based clustering on various data sets)

For this problem we use the Exploratory Data Analysis (EDA) toolbox function `mbclust.m` on the suggested data sets. As discussed in the text sometimes the EM algorithm may fail to converge in which cases the BIC array will have NaN's as some of its values. In these cases we will simply select the best model (the one with the *largest* BIC) from the ones where this statistic could be computed. Often when the experiments presented below the BIC statistic of the each model under the *first* cluster had values that were much larger than the others. This looked inconsistent given the remaining estimates and these points were deleted/removed before presenting the results below. More work is needed to make these procedures more robust.

Part (a): See the MATLAB script `prob_6_17_skulls.m` for the application of model-based agglomerative clustering for the `skulls` data set. The BIC plots for each model type and for a maximum of six clusters is shown in Figure 83 (left). This procedure suggests that the

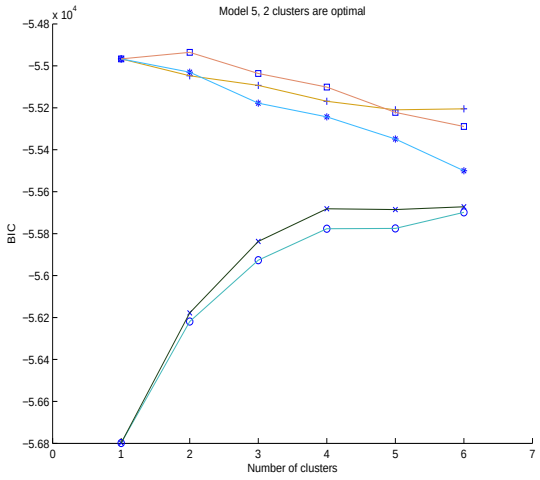

Figure 84: The BIC plots for the `lungB` data set.

third covariance model with four clusters seems to be optimal.

Part (b): See the MATLAB script `prob_6_17_sparrow.m` where we apply model-based agglomerative clustering on the `sparrow` data set. The result of running this code is presented in Figure 83 (right). In that plot we see that the *second* model with 3,4, or 5 clusters appears optimal. Each of these three possible clusters appear to have almost identical Bayesian Information Criterion.

Part (c): See Exercise 6.2 (above) where we apply model-based agglomerative clustering on the `oronsay` data set.

Part (e): See the MATLAB script `prob_6_17_lungB.m` where we apply model-based agglomerative clustering on the `lungB` data set. When this is run it produces a plot like that shown in Figure 84. This is an example where the various curves look quite different and might lead one to question the validity of these results. Taken as is, they imply that the fifth covariance model with two clusters is optimal for this data set. It is encouraging that we correctly predict the hypothesized true number of classes.

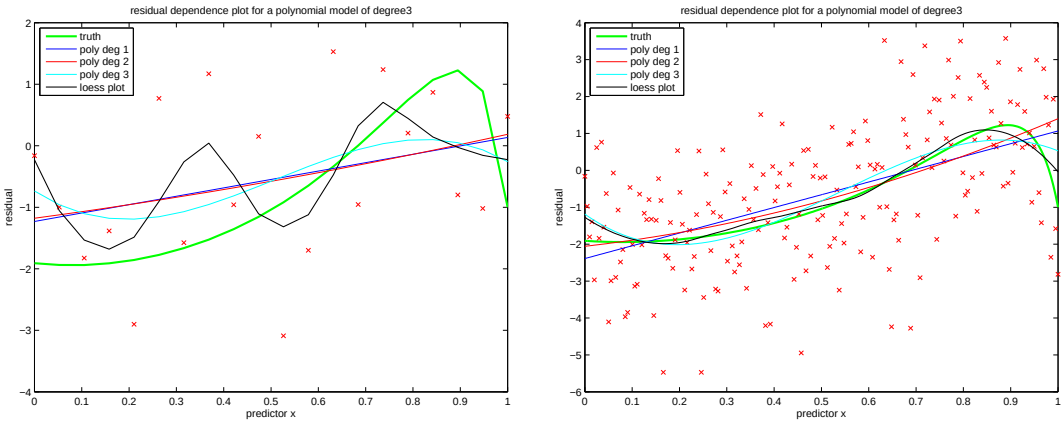

Figure 85: Examples of fitting three polynomial models and a loess smooth to noised samples of the function $y(x)$ given in Exercise 7.1. The unnoised function $y(x)$ is drawn in both plots in green. **Left:** Here we have taken $M = 20$ samples for $x_i = \frac{i}{M}$. Note that in this case the loess plot produces a model that seems to be fit more to noise that to signal. **Right:** Here we have taken $M = 200$. In both of these plots we can see the well known "edge effect" where a given models predictive performance degrades as we get closer to the edges of the independent variables domain. Also notice that in the large sample limit all models with sufficient capability (the cubic and loess smooths) are able to predict models that match the truth quite closely.

Chapter 7: Smoothing Scatterplots

Exercise Solutions

Exercise 7.1 (using the functions `polyval` and `polyfit`)

For this problem we choose to generate data for the independent variable, x, on the interval $[0, 1]$ at M evenly spaced points $x_i = \frac{i}{M}$ for $i = 0, 1, \cdots, M$. At each of the points we evaluate the compute the function y at these points where the function $y(x)$ is given by

$$y(x) = 4x^5 + 6x^2 - 1 + \frac{1}{x - 1.1}\,.$$

This function was specifically chosen to *not* be in the space of the polynomials over which we are considering as a possible model candidates (that is first, second, and third degree polynomials). We also choose to consider two cases for the value of M. The first value of M will correspond to the case where we only take a relatively few samples, say 20, from our x domain $[0, 1]$. This represents the small sample performance of these fits. In the second case for M we will take significantly more samples, say 200, and represents the large sample limit of the performance of our polynomial fits. To each of the computed $y(x_i)$ values we add a some Gaussian random noise of a constant variance. The results from fitting three polynomial models (using the Matlab command `polyfit`) are shown in Figure 85.

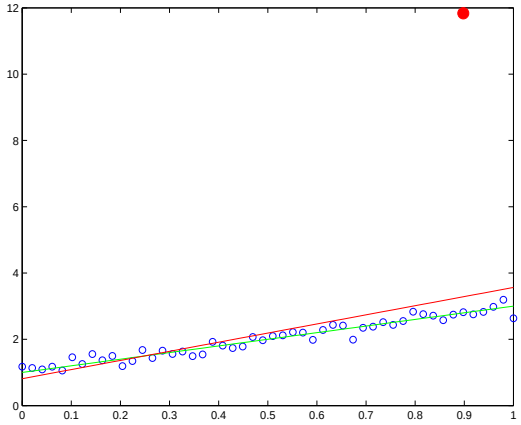

Figure 86: The results obtain when running the experiment presented in Exercise 7.2. Note the outlier in the upper right corner. The true model is plotted as a green line while the approximate model is plotted as a red line.

In addition to the three polynomial models we have also presented the model that results when using the exploratory data analysis toolbox function, `loess`, to generate a loess smooth for the given points. In the loess smooth model we have taken parameters $\alpha = 0.5$ and $\lambda = 2$. In that figure we see that the linear and quadratic polynomial models produce almost identical looking models that are both very nearly linear. The cubic and loess models do perhaps the best job at fitting the data. We have implemented this exercise in the the MATLAB script `prob_7_1.m`. Running it produces plots like those shown above.

Exercise 7.2 (the presence of outliers in `polyfit`)

For this problem we are told to perform the following computational experiment

- generate data from a linear model $y(x) = \beta_0 + \beta_1 x$.
- apply some small amount of noise to the samples $y_i = y(x_i)$ values.
- replace one value of y_i with an "extreme" value i.e. create an outlier in our data set.
- fit a linear model to this new data set containing the outlier.
- compare the exact model and the estimated model.

This exercise is implemented in the MATLAB script `prob_7_2.m`. When this script is run we obtain the results in Figure 86. Performing additional numerical experiments with this code one notices a few things. The presence of the outlier point obviously adversely affects our ability to estimate the true coefficients in the linear model. How much the outlier affects these estimate of β_i seems to depend on how large the outlier is relative to the default added noise. If the added noise is of the same order of magnitude as the outlier then the outlier

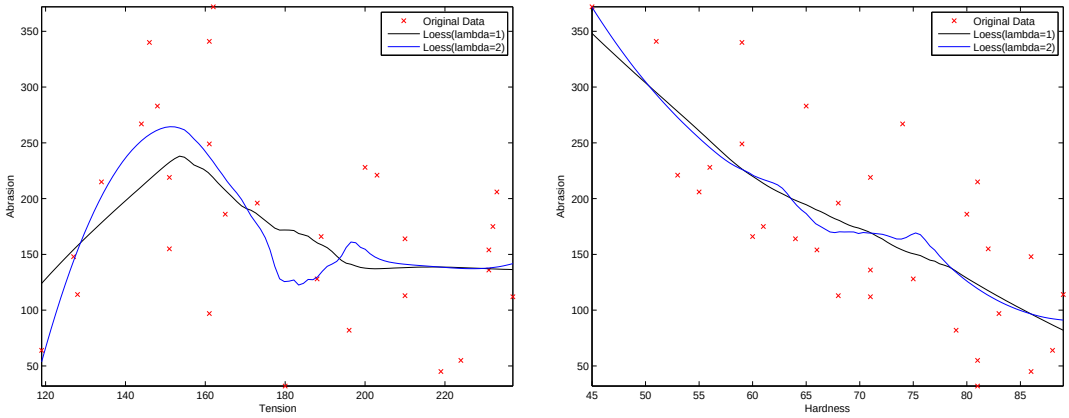

Figure 87: Loess smooths for $\lambda = 1, 2$ of the `abrasion` data set. **Left:** Loess smooths where the independent variable is tensile strength and the dependent variable is abrasion loss. **Right:** Loess smooths where the independent variable is hardness and again the dependent variable is again abrasion loss. Both values for λ seem to do quite a good job fitting this data.

does not have much affect. A second observation is that if there are a great number of data samples already then the addition of only one outlier does not have much affect regardless how large it is. In Figure 86 we selected a noise variance that is significantly smaller than the magnitude of the outlier. Because of this choice we can clearly see its influence in skewing the best fit line in that direction.

Exercise 7.3 (using the MATLAB basic fitting toolbox)

This exercise can be performed by running the MATLAB script `prob_7_3.m` and selecting choices the found under the "Tools" and then "Basic Fitting" menus.

Exercise 7.4 (loess curves for the `abrasion` data set)

For this exercise we use the `loess` function provided in the exploratory data analysis toolbox to fit loess smooths to the abrasion data set and is implemented in the MATLAB script `prob_7_4.m`. When this script is run it produces the results shown in Figure 87. We plot loess smooths for both $\lambda = 1$ and $\lambda = 2$. We notice that with this data set even thought the abrasion loss variable has a relatively large variance (it is rather noisy) *both* values of λ are able to capture the trend of the data quite well.

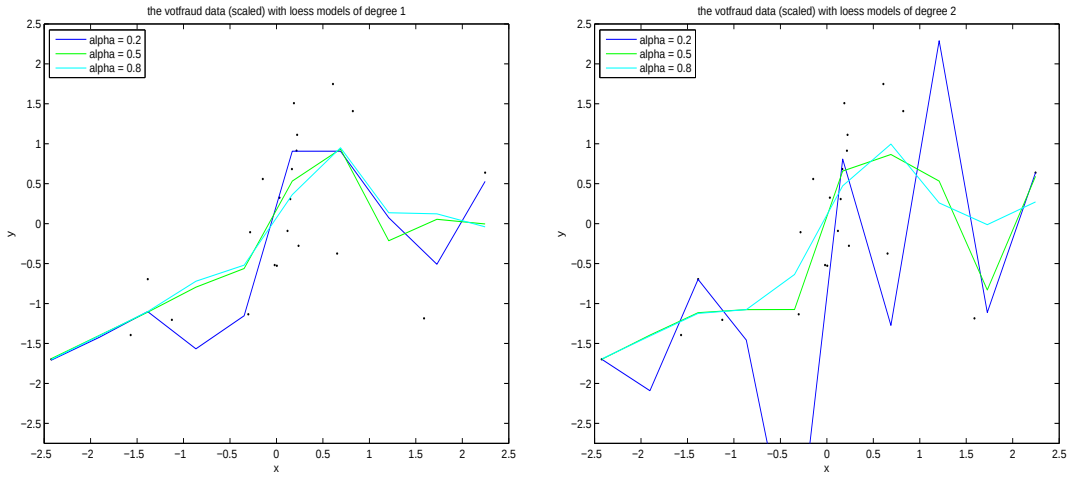

Figure 88: The results obtain when running the experiment presented in Exercise 7.6. **Left:** Loess smooths of the `votfraud` data set when $\lambda = 1$. **Right:** Loess smooths of the `votfraud` data set when $\lambda = 2$.

Exercise 7.6 (loess curves for the `votfraud` data set)

For this exercise we load the `votfraud` data set and produce loess plots for various values of α and for $\lambda \in \{1, 2\}$. This exercise is implemented in the MATLAB script `prob_7_6.m`. When this is run it produces the results shown in Figure 88 (left) and (right). We see that when $\alpha = 0.2$ the loess fit is too local in that it places too much weight on points close to the point we are evaluating the loess smooth at. This produces models that are very oscillatory in appearance. This large degree of oscillation is the result of fitting our model to the noise in the data and is not to be believed. The other values of α produce curves that are much more smooth and are more likely to be capturing a real effect in the data. Note that these plots are done on data that has been transformed by standardizing (z-scored) since using the raw `votfraud` data alone results in MATLAB issuing a significant number of warnings about the conditioning of the matrices involved in the `wfit` function. Transforming the data in this simple way removed these warnings and resulted in a more stable application of this algorithm. Given these results the value of $\alpha = 0.8$ looks to be a good choice.

Exercise 7.7 (looking at the residuals of a loess smooth)

As in exercise 7.6 we load the `votfraud` data set and apply the loess data smoothing procedure to the given data points. The resulting plots of the resulting residuals from this procedure are shown in Figure 89 (left) and (right). We see that in the case of $\lambda = 1$ we would conclude that the values of α of 0.5 and 0.8 are much better than the others at fitting the true trend to the data. This conclusion also holds when $\lambda = 2$. The general consideration to be made when looking at residual plots likes these is that one would like to see all of the residual values located in a fixed banded region around the line $y = 0$. The width of this band should be independent of the x variable and have relatively few outling points. In this

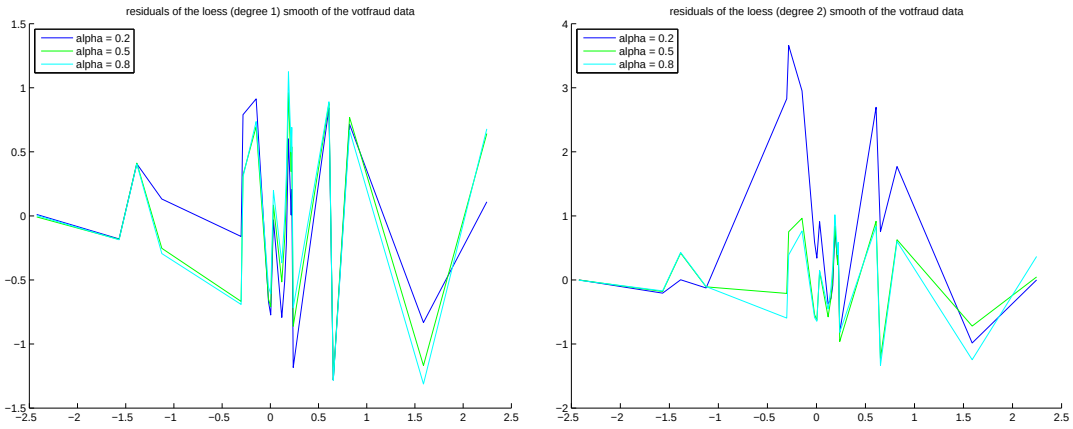

Figure 89: The residuals that result when running the experiment presented in Exercise 7.6 (various loess model fits). **Left:** Residuals of the loess smooth of the `votfraud` data set when $\lambda = 1$ (first order polynomials). **Right:** Residuals of the Loess smooths of the `votfraud` data set when $\lambda = 2$ (second order polynomials).

case then the error in the fit has approximately a constant variance under different values of the independent variable.

Exercise 7.8 (loess smoothing of the `calibrat` data set)

For this exercise we load in the `calibrat` data set and apply a loess smooth to it. We then compute the residuals of this loess smooth model with the data itself. Due to the range of the values of the `counts` and `tsh` variables a direct application of the toolbox function `loess` results in a large number of matrix conditioning warnings. To attempt to overcomes this, as in Exercise 7.7, we standardize each variable by subtracting the mean and dividing that variables standard deviation. This is done via options in the MATLAB function `load_calibrat`. Unfortunately this preprocessing does not remove all the warning messages but does seem to result in reasonable smooths. The results from this procedure are plotted in Figure 90. In all cases we see that the loess curves seem to model the data quite well. In this data set we again see the phenomena where if the value of α is too small the loess algorithm "pays too much attention to the data and not enough to the trend". The residual plots show that in general there is more error in the loess fit at the left end of the domain (small values of x) than at the right end of the domain.

These plots can be generated by running the MATLAB script `prob_7_8.m`.

Exercise 7.10 (the convex hull of some BPM topics)

For this problem we will plot the convex hull of the data from Example 7.8 using the MATLAB command `convhull` and compare these results with the polar smooth presented there.

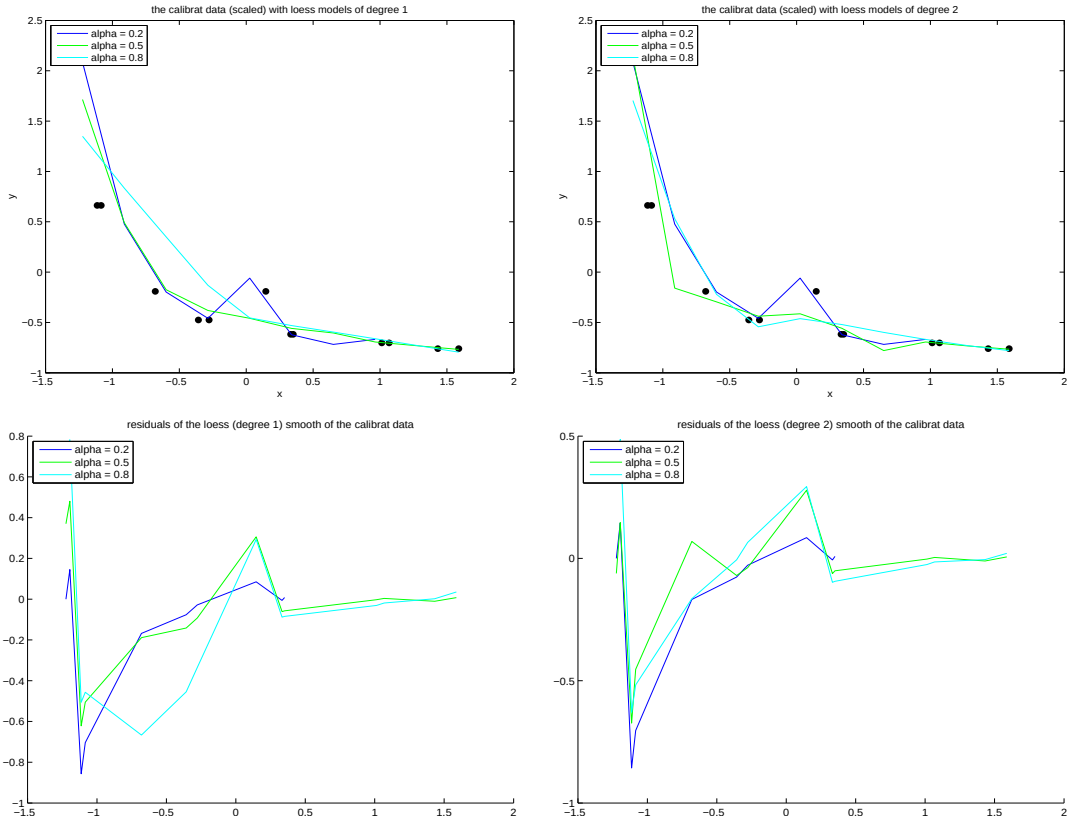

Figure 90: The loess smooths and the residuals plots that result when running the experiment presented in Exercise 7.8. **Top:** Polynomial smooths for $\lambda = 1$ (left) and $\lambda = 2$ (right). **Right:** Residuals of the loess smooths for corresponding value of λ.

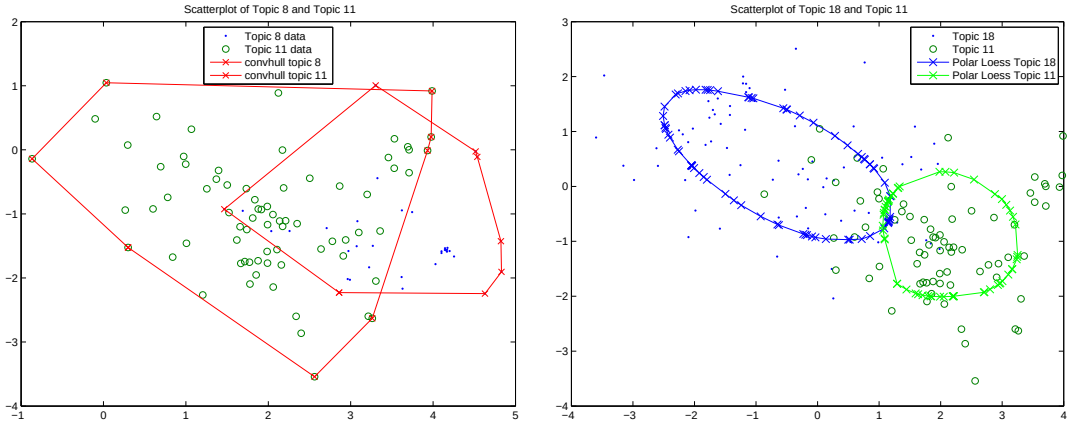

Figure 91: **Left:** The convex hull produced by the `convhull` command for topics 8 and 11. **Right:** The class boundary produced by the `polarloess` command for topics 18 (Oklahoma City bombing) and 11 (North Korea).

When we run the code in the MATLAB script `prob_7_10.m` we produce the plot shown in Figure 91 (left). From that plot we see that using the convex hull as a measure of a classes location can be sensitive to outliers. This is especially true with regard to topic 8 where the convex hull procedure predicts a grouping of points that clearly overstates the true domain for this topic.

Exercise 7.11 (more examples of polar smoothing)

For this exercise we repeat Example 7.8 (polar smoothing) but with different news topics. We choose the news topics with numerical indices of 11 and 18 which correspond to topics on North Korea and the Oklahoma City bombing since both of the these topics seem to have quite a few data points. When we run the MATLAB script `prob_7_11.m` we call the MATLAB command `polarloess.m` from the EDA toolbox we obtain the plots in Figure 91 (right). From this plot (by eye) we see that the polar loess command does a very good job at denoting the central region of the specified data clusters and is more resilient to outliers than is the convex hull (see Exercise 7.10).

Exercise 7.12 (smoothing some gene expression data sets)

For this exercise we selected the `leukemia` data set and from this selected a specific gene index 34 and performed a loess smooth of the gene responses across patients/experiments. In addition we used the EDA toolbox function `loessenv` to assess the variation in the gene response. When the MATLAB script `prob_7_12.m` is run we obtain the Figure 92 where we see that taking α too small will result in very profiles of gene response that are too oscillatory.

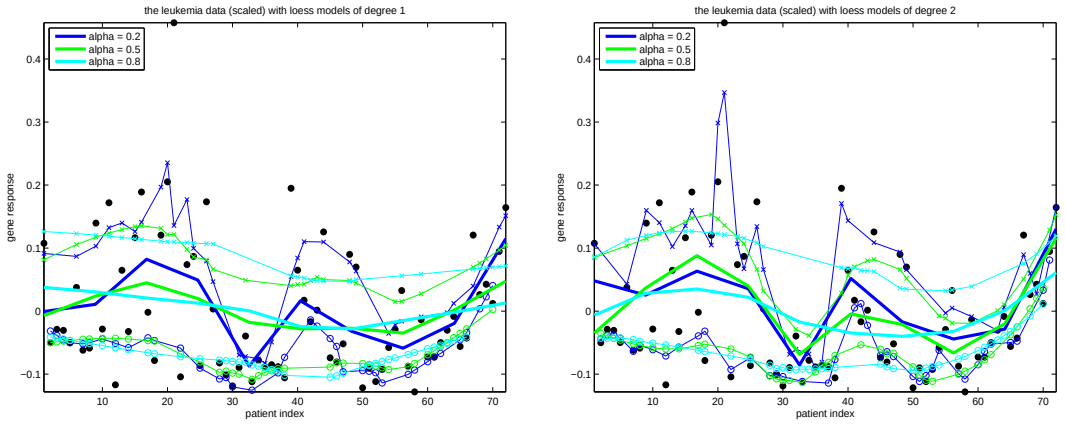

Figure 92: The robust loess smooth and loess envelops for the thirty-fourth gene of the `leukemia` data set. **Left:** For various values of α and $\lambda = 1$ (first order polynomial smooths). **Right:** For various values of α and $\lambda = 2$ (second order polynomial smooths).

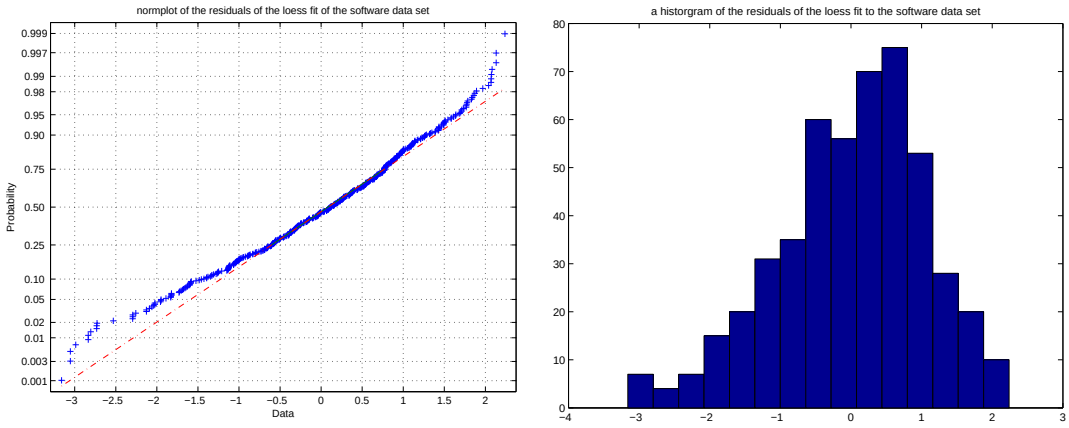

Figure 93: A study of the residuals of the loess fit to the `software` data set. **Left:** A qq-plot of these residuals. **Right:** A histogram of these residuals. Note that from these plots the residuals look to be reasonably approximated as a normal distribution.

Exercise 7.13 (the residuals of the `software` data smooth)

For this exercise, by repeating the steps found in Example 7.4 we will obtain the loess smooth and from this smooth the residuals for the `software` data set. We will display these residuals in two ways. The first with the `normplot` MATLAB command and then as histogram of using the MATLAB command `hist`. These procedures are performed in the MATLAB script `prob_7_13.m`. When that script is run it produces the plots shown in Figure 93. From this plot we see that we have indications that the distribution of the residuals is normal.

Exercise 7.14 (the residuals of the `votfraud` and `calibrat` data sets)

For this exercise we repeat the steps in Exercise 7.13 (residual plots) on the data sets suggested in Exercise 7.6 (`votfraud`) and Exercise 7.8 (`calibrat`). We used Sturges' rule

$$k = \lceil 1 + \log_2(n) \rceil \,. \tag{6}$$

to specify the number of bin's k to use for the histogram. Because of the small sample size in these data sets (small n) Sturges' rule correspondingly predicts only a few bins (small k).

Part (a): For the `votfraud` data set running the MATLAB code in `prob_7_13_vot.m` we obtain the results displayed in Figure 94 (top). We see that the residual do indeed look quite normally distributed.

Part (b): For the `calibrat` data set when we run the MATLAB code in `prob_7_13_cal.m` we obtain the results in Figure 94 (bottom). Again we see that the residual (except for a few outliers) look to be normally distributed.

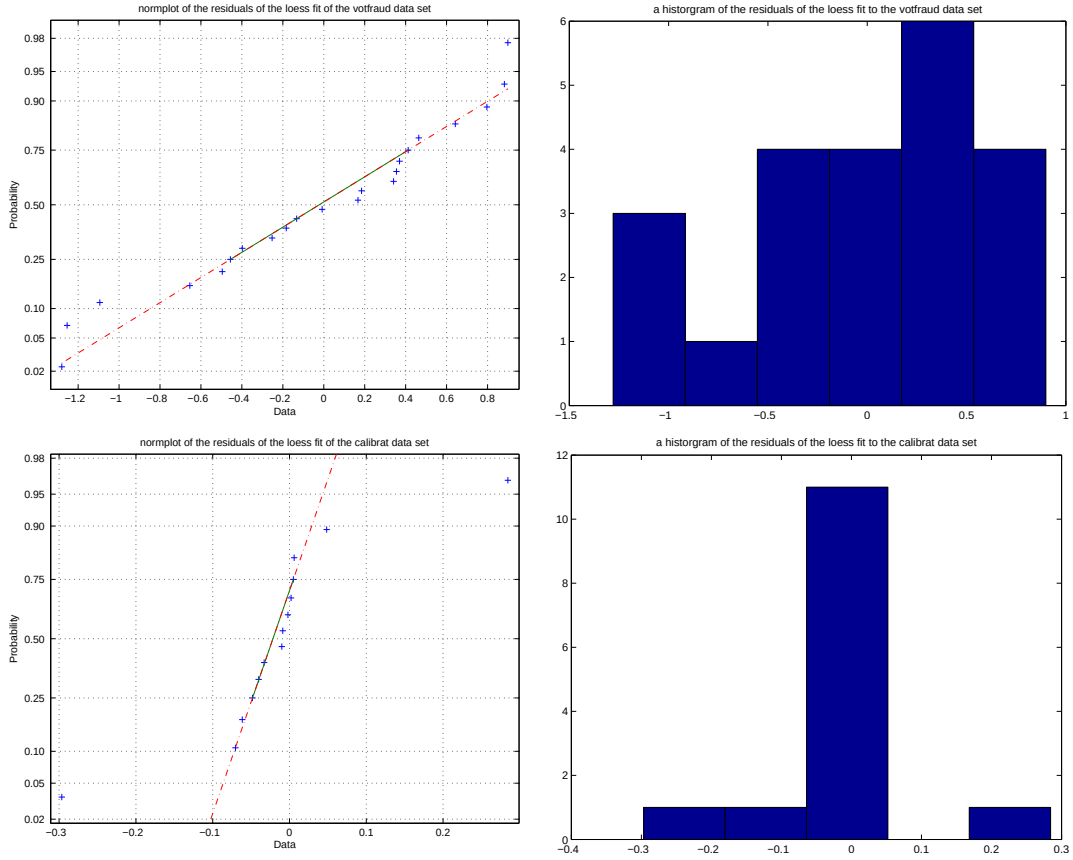

Figure 94: A study of the residuals of the loess fit to the `votfraud` and `calibrat` data sets. **Top:** For the `votfraud` data set we present a qq-plot of the loess residuals (left) and a histogram of the loess residuals (right). Note that with the results displayed here the residuals look to be normally distributed.
Bottom: For the `calibrat` data set we present a qq-plot of the loess residuals (left) and a histogram of the loess residuals. Note again that with the results displayed here the residuals look to be normally distributed, with the presence of a few outliers.

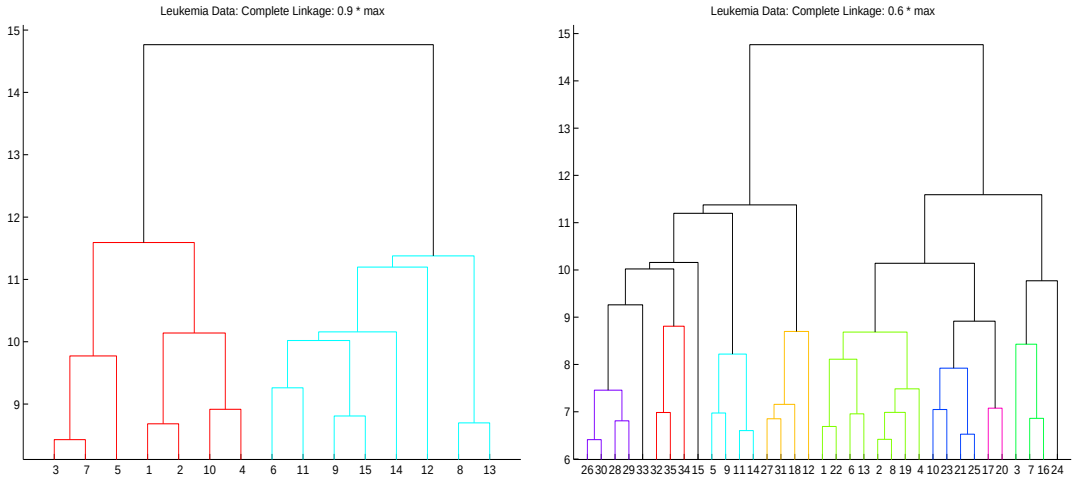

Figure 95: Some examples using the option `colorthreshold` with the `dendrogram` function. **Left:** With a specification of 0.9 of the maximum linkage length. **Right:** With a specification of 0.6 of the maximum linkage length.

Chapter 8: Visualizing Clusters

Exercise Solutions

Exercise 8.1 (the option `colorthreshold` to the function `dendrogram`

When the `colorthreshohld` option is enabled (with an additional scalar threshold) the dendrogram drawn by MATLAB will color all nodes uniquely whos linkage is less than the input arguments threshold. If no argument is provided MATLAB will select a threshold based on 70% of the maximum linkage. An example at running the `dendrogram` command with this option can be found in the MATLAB script `prob_8_1.m` where we apply agglomerative clustering to the `leukemia` data set. Two representative examples obtained when running the `dendrogram` function with two different arguments to the `colorthreshold` argument are given in Figure 95. This option makes it very easy to visually see the clusters that result when one cuts a dendrogram at a given linkage value.

Exercise 8.2 (differences between various tree visualization techniques)

A **dendrogram** very clearly provides a visual representation of agglomerative clustering. The length of the edges represent the distances between the merged data clusters represented by the data points beneath the given dividing line. A **treemap** begins with a rectangle and proceeds to divide it into sub-regions the size of each is proportional to the number of points in that cluster. This representation does a good job of displaying the relative cluster sizes that results when thresholding a clustering algorithm at a fixed level. A **rectangle** visualization

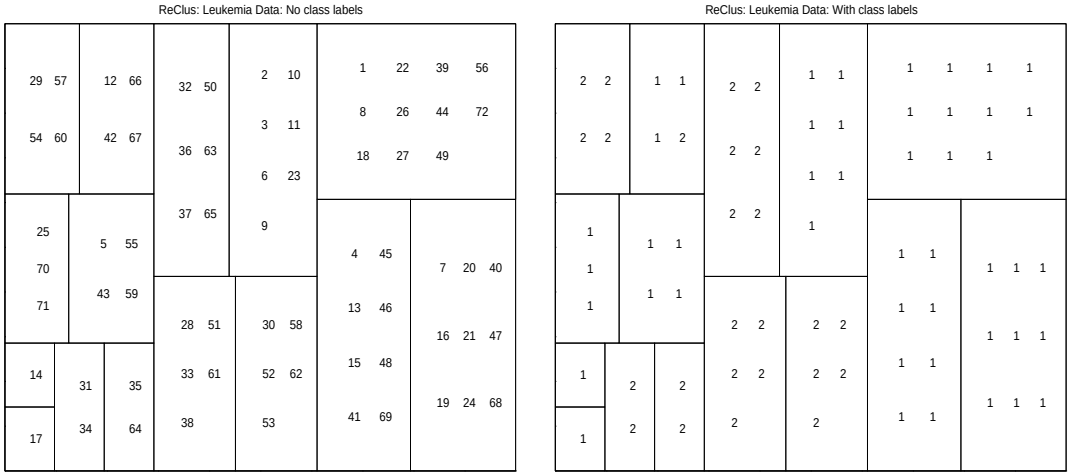

Figure 96: Some examples of `reclus` plots on the `leukemia` data set. **Left:** Plotted without class labels. **Right:** Plotted with class labels. Note from this second plot we can see that these clusters do a good jobs at delineating regions that contain only one type of leukemia.

plot is a way to draw one fixed rectangle which is then subdivided as the number of clusters changes. The benefit of this procedure is that the glyphs don't need to be redrawn as the number of clusters change. The **ReClus** visualization method allows the display of the results from non-hierarchical clustering methods like k-means and model-based clustering.

Exercise 8.3 (a ReClus plot for the `leukemia` data set)

See the MATLAB script `prob_8_3.m` where we use the EDA toolbox function `reclus.m` on the `leukemia` data set. The results obtained when we run this code are shown in Figure 96. Note that the default `reclus.m` function provided with the text had a bug when passed two arguments, this error was fixed and the corrected version of `reclus.m` placed on the web site associated with these notes.

Exercise 8.4 (specifying the number of clusters by distance)

When we plot the dendrogram of the `leukemia` data set "by eye" we see that a good dividing line between the two major clusters would be the value 13.0. We can then call the EDA toolbox function `rectplot.m` with the option `dis` and that parameter value. This procedure is implemented in the MATLAB script `prob_8_4.m`. When this script is run we obtain the results shown in Figure 97 shown here with the cancer type labeling.

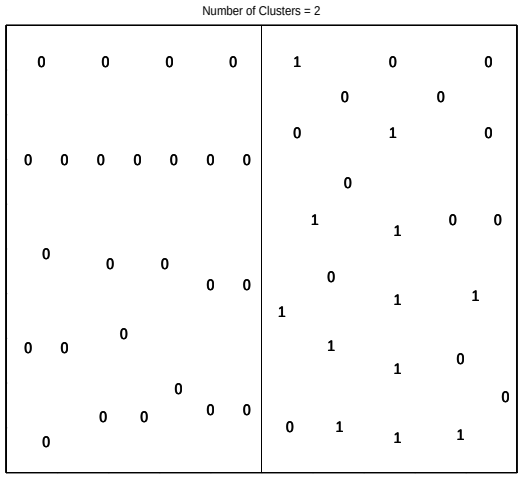

Figure 97: A rectangle plot of the `leukemia` data set and specifying two clusters. The two cancer types are labeled as a 0 or a 1.

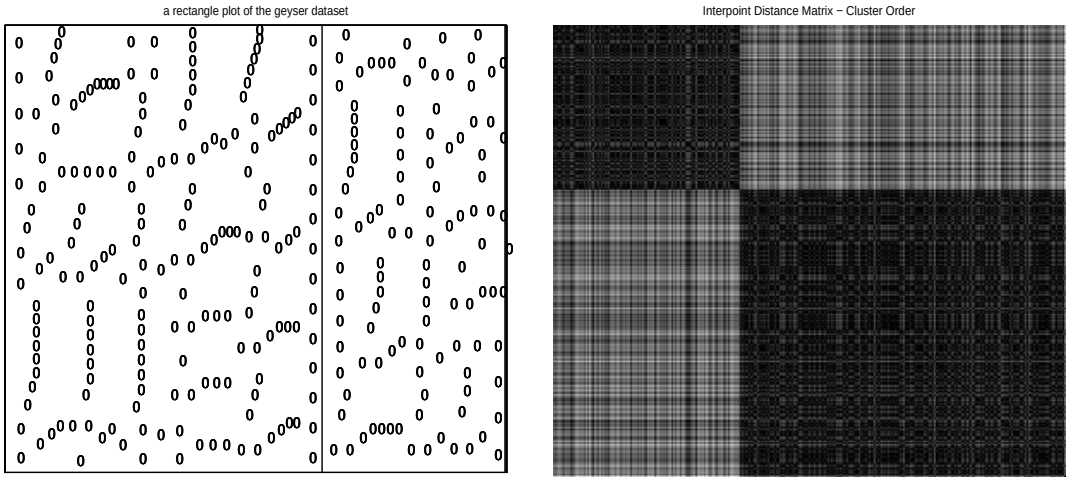

Figure 98: Some examples of data clustering visualization on the `geyser` data set. **Left:** A rectangle plot with each point labeled with a 0 for simplicity of presentation. **Right:** A distance image.

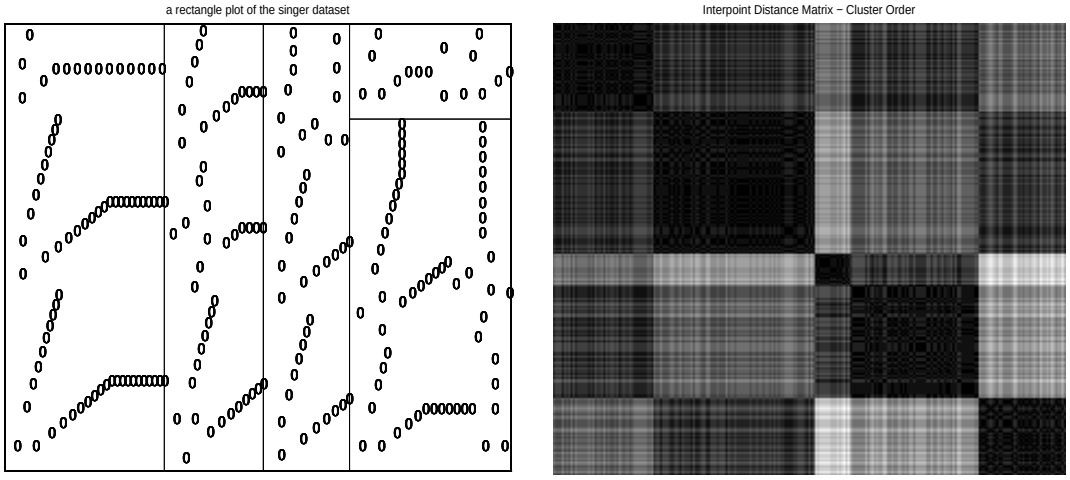

Figure 99: Some examples of data clustering visualization on the `singer` data set. **Left:** A rectangle plot with each point labeled with a 0 for simplicity of presentation. **Right:** A distance matrix.

Exercise 8.5 (clustering visualization applied to various data sets)

Part (a): For this part of the problem we apply various visualization tools on agglomerative clustering results on the `geyser` data set. Two of the results when we run the MATLAB script `prob_8_5_geyser.m` can be see in Figure 98.

Part (b): For this part we apply various visualization tools on agglomerative clustering results on the `singer` data set. Two of the results when we run the MATLAB script `prob_8_5_singer.m` can be see in Figure 99. In these figures the number of clusters is specified at 5. This number corresponds to a possible number of common groupings of singers in a choir; namely the classes soprano, alto, tenor, bass, or baritone. From the distance matrix in Figure 99 (right) this appears to be a reasonable number of clusters components.

Part (c): For this part we apply various visualization tools on agglomerative clustering results on the `skulls` data set. Two of the results when we run the MATLAB script `prob_8_5_skulls.m` can be see in Figure 100. This is an example data set where there does not appear to be a strong clustering. At most there are possibly two clusters and that is the number of clusters presented here.

Part (d): For this part we apply various visualization tools on agglomerative clustering results on the `sparrow` data set. Two of the results when we run the MATLAB script `prob_8_5_sparrow.m` can be see in Figure 101.

Part (e): For this part we apply various visualization tools on agglomerative clustering results on the `oronsay` data set. Two of the results when we run the MATLAB script `prob_8_5_oronsay.m` can be see in Figure 102. For these plots we have assumed 6 clusters.

Part (f): For this part we apply various visualization tools on agglomerative cluster-

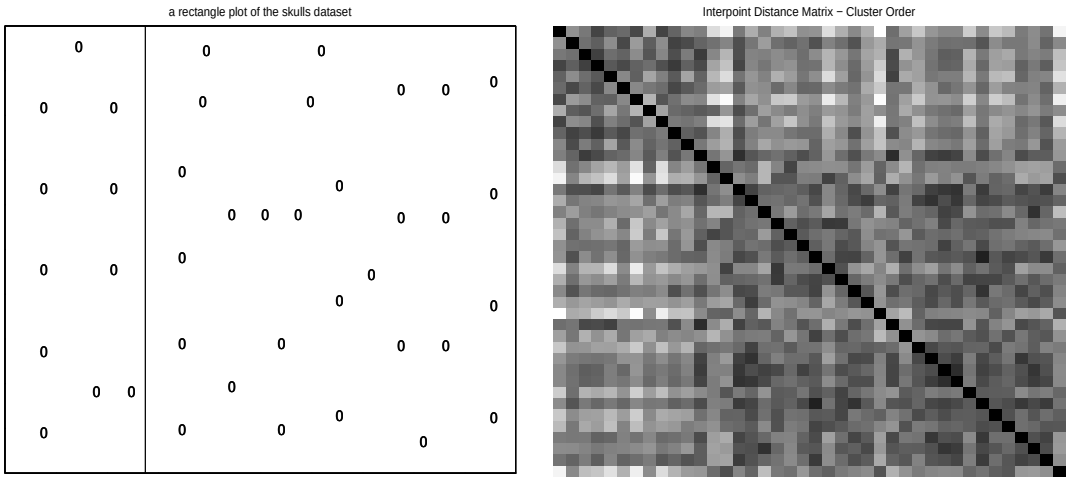

Figure 100: Some examples of data clustering visualization on the `skulls` data set. **Left:** A rectangle plot with each point labeled with a 0 for simplicity of presentation. **Right:** A distance matrix.

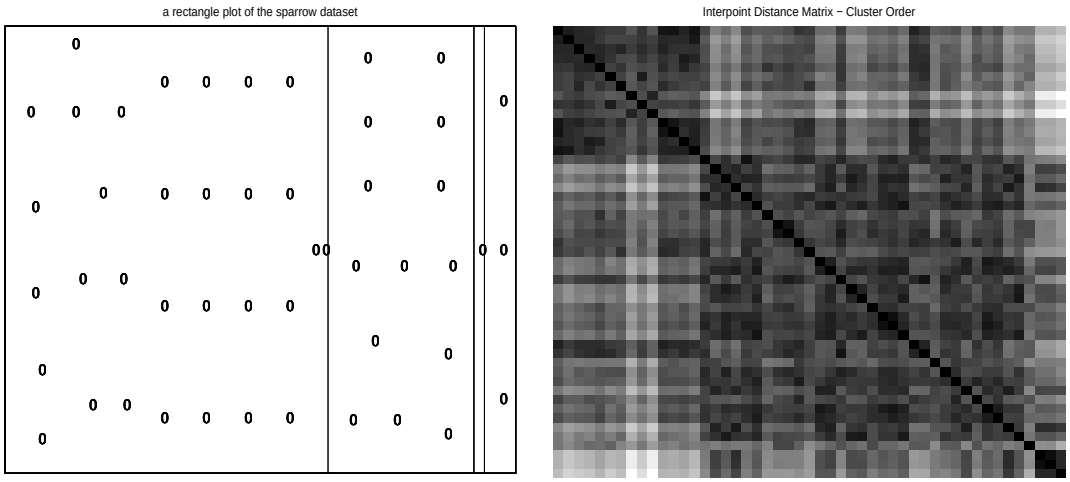

Figure 101: Some examples of data clustering visualization on the `sparrow` data set. **Left:** A rectangle plot with each point labeled with a 0 for simplicity of presentation. **Right:** A distance matrix.

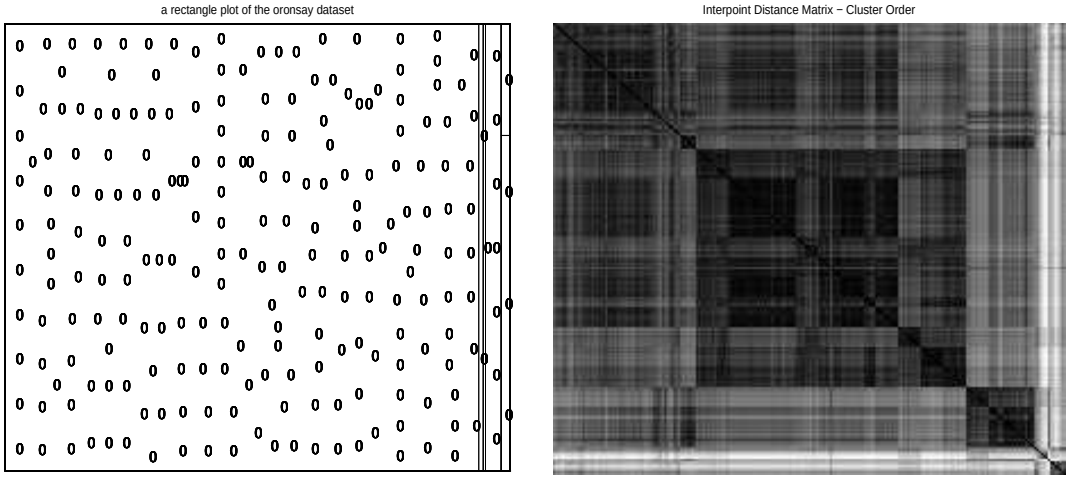

Figure 102: Some examples of data clustering visualization on the `oronsay` data set. **Left:** A rectangle plot with each point labeled with a 0 for simplicity of presentation. **Right:** A distance matrix.

ing results on the `lungB` data set. Four of the results when we run the MATLAB script `prob_8_5_lungB.m` can be see in Figure 103. Note that these plots lend credibility to the fact that this data possesses clusters. The image plot in Figure 103 (lower right) is quite compelling in that regard.

Exercise 8.6 (clustering of some BPMs)

For this problem we apply the ISOMAP dimensionality reduction procedure on the `L1bpm` data set. This is the same experiment done in Example 8.7 but rather than restricting to only two topics here we explicitly don't restrict the number of topics found in the `L1bpm` data set. We then apply Ling's method to produce a data image. Specifically in this problem we produce two images; one with randomly ordered data and another with the data ordered so that data points in the same cluster are adjacent in the resulting image. The randomly ordered matrix is shown in Figure 104 (left) while when we order the data points according to their cluster labeling we obtain the image shown in Figure 104 (right). This problem is worked in the MATLAB script `prob_8_6.m`.

Exercise 8.7 (a image plot of the `iris` data set)

For this problem we apply Ling's method to produce an image plot by ordering the data points so that points who are in the same cluster are nearer to each other. This problem is implemented in `prob_8_7.m`. When that script is run it produces the image plot shown in Figure 105. Note clustering is very strongly present.

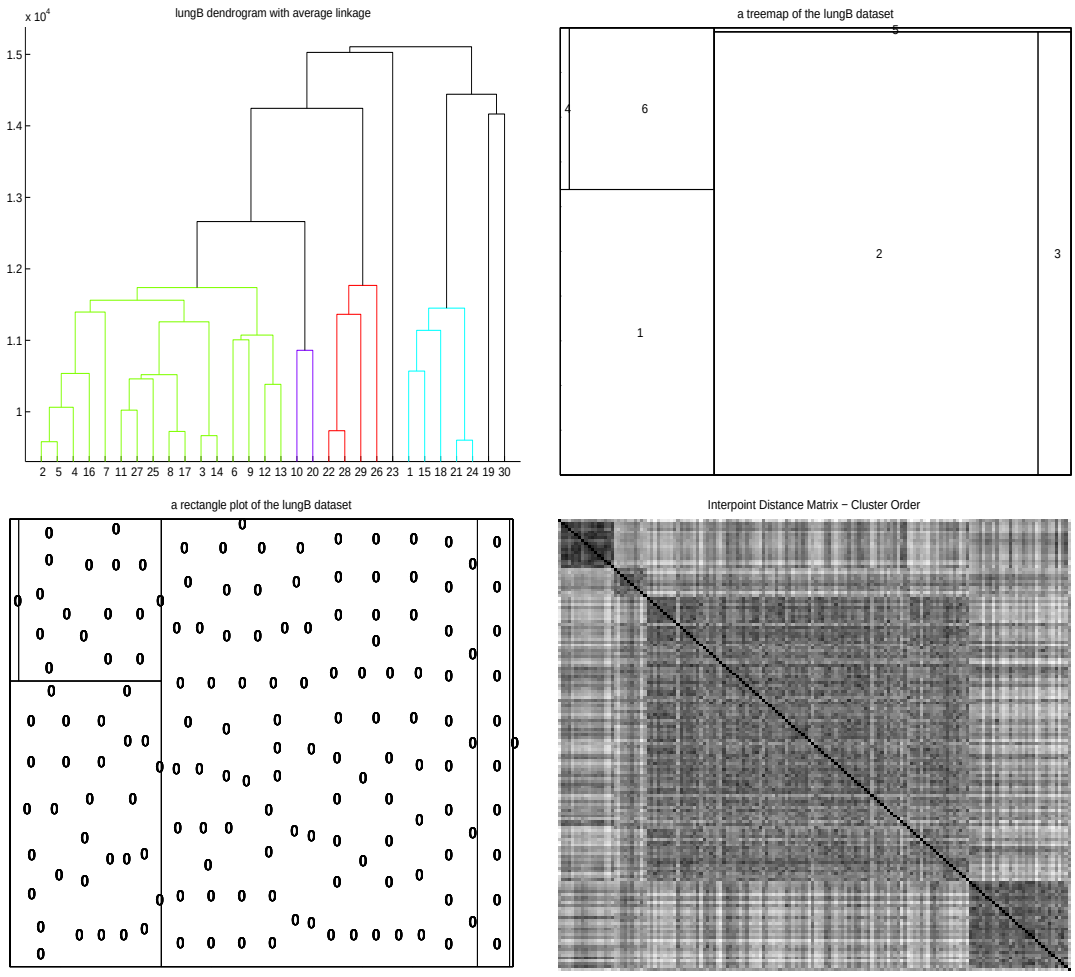

Figure 103: Some examples of data clustering visualization on the `lungB` data set. **Left Top:** A dendrogram of this data with clusters thresholded at the value of $1.2\,10^4$ and the resulting clusters colored. **Right Top:** A tree map of the resulting five clusters. **Left Bottom:** A rectangle plot with each point labeled with a 0 for simplicity of presentation. **Right Bottom:** The resulting distance matrix. Note the very dark corner of data points in the upper left hand corner.

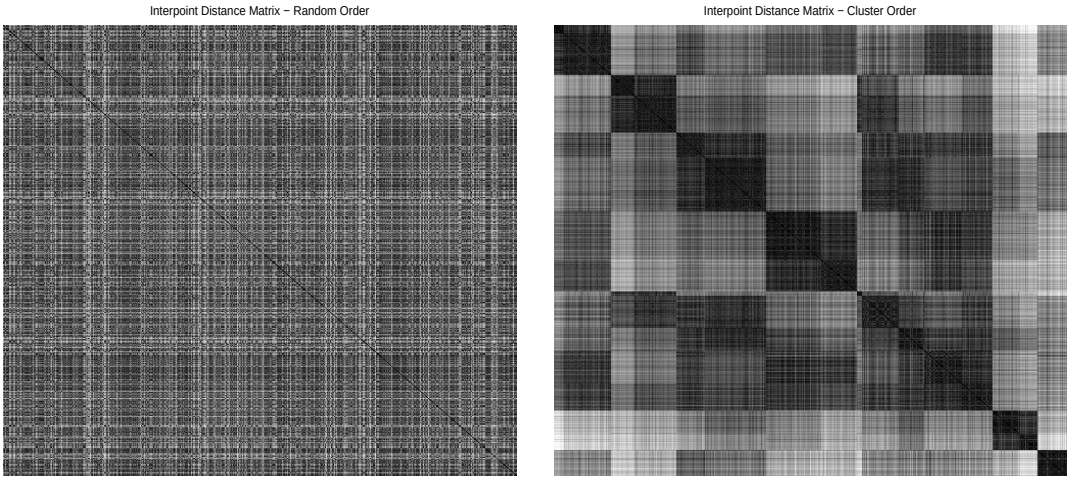

Figure 104: Two data images of the `L1bpm` data set. **Left:** With random ordering of the data points. **Right:** With ordering of the data such that points in neighboring clusters are closer together.

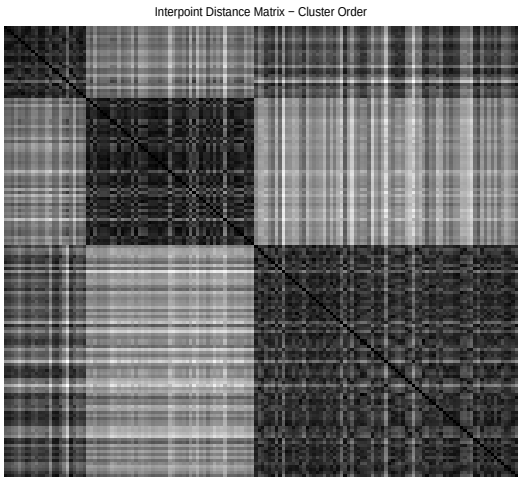

Figure 105: The cluster sorted data images of the `iris` data set.

Exercise 8.12 (the data image of the `iris` data set)

Based on the books Figure 8.9 it appears that the third and fourth feature would be best for classifying iris. These two features show a strong contrast in value when we move from species to species as observed from the fact that they change value significantly as we consider different iris species.

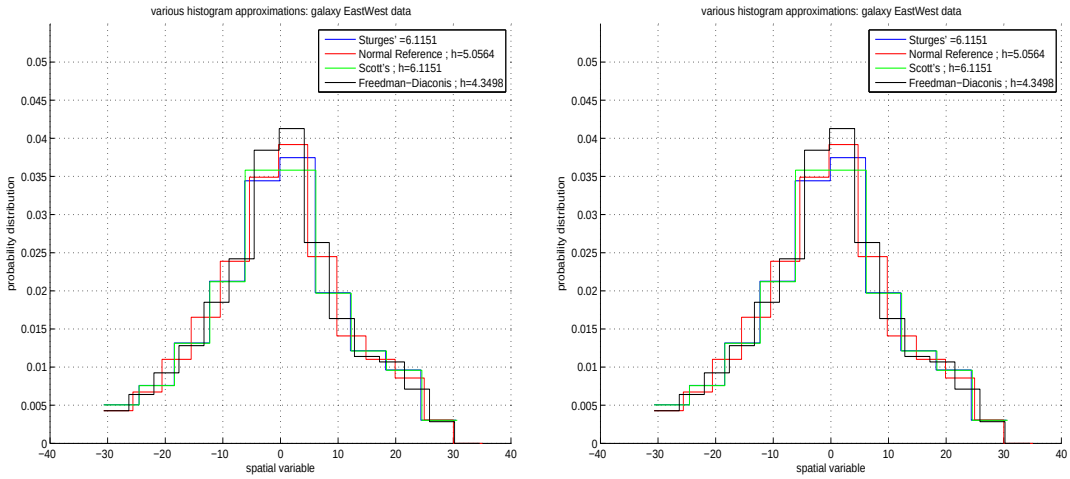

Figure 106: Some examples using the various optimal bin width rules presented in Exercise 9.3. **Left:** Density histograms of the `galaxy` data set (specifically the `EastWest` component). **Right:** Density histograms of the `forearm` data set.

Chapter 9: Distribution Shapes

Exercise Solutions

Exercise 9.1 (the number of histogram bins)

See the MATLAB script `prob_9_1.m` where we plot a frequency histogram and a relative frequency histogram of the `galaxy` data set. One has to select a value for the number of bins that finds a balance in the bias-variance trade off that is present when fitting a model with parameters (like the number of bins) using a specific set of data. When we run the above code the frequency histogram indicates that this data appears somewhat normally distributed but with heavier tails.

Exercise 9.2 (the number of histogram bins with the `forearm` data)

See the MATLAB script `prob_9_2.m` where we plot a frequency histogram and a relative frequency histogram of the `forearm` data set. The same issues in selecting the number of bins mentioned in Exercise 9.1 appear here also.

Exercise 9.3 (a comparison of various bin width rules)

For this exercise we implement four bin sizing methods given a one-dimensional data set with n elements, a population standard deviation given by σ, and a sample standard deviation

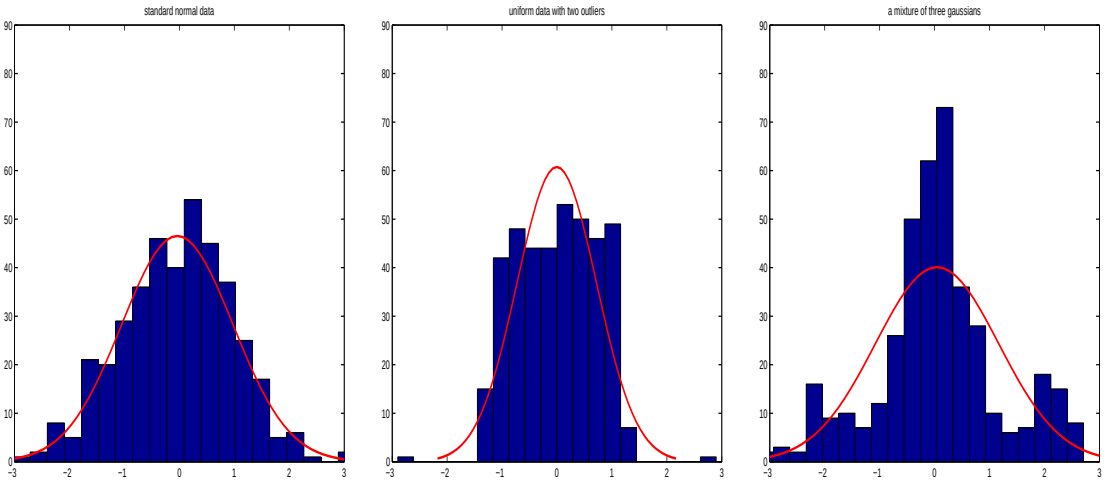

Figure 107: Some examples using the MATLAB function `histfit` function to plot normal densities on some "toy" data sets. **Left:** Data actually generated from a normal distribution. **Center:** Data generated from a uniform distribution with some outlying data points. **Right:** Data generated from a mixture of normal distributions. Note that each of these data sets has an almost identical boxplot. See Figure 9.8 from the book.

given by s. The four rules we compare are Sturges's rule given by Equation 6 above, followed by the normal reference rule which specifies a uniform bin width given by $\hat{h}^*$ of

$$\hat{h}^* = \left(\frac{24\sigma^3\sqrt{\pi}}{n} \right)^{1/3} , \tag{7}$$

Scott's rule which specifies the bin widths of

$$\hat{h}^* = 3.5sn^{-1/3} , \tag{8}$$

and finally the Freedman-Diaconis Rule which specifies bin widths $\hat{h}^*$ given by

$$\hat{h}^* = 2\,\mathrm{IQR}\,n^{-1/3} . \tag{9}$$

Here in the Freedman-Diaconis rule the expression IQR is the inter-quartile range a robust method at computing a measure of distribution spread defined as the difference between the 75% quantile and the 25% quantile. Note that we are assuming that the `forearm` and `galaxy` data sets are not sufficiently skewed or heavy tailed in that we don't need to apply Scott's correction factor to these widths $\hat{h}^*$ for skewed data [10]. In the MATLAB script `prob_9_3.m` we implement density histograms using each of the above four rules. When this script is run it produces plots as shown in Figure 106. Empirically, we see that the various bin width rules seem to produce almost identical looking density histograms.

Exercise 9.6 (the MATLAB function `histfit`)

In example 9.2 we construct three artificial data sets each of which has approximately the same mean and standard deviation. For each data set we then call the MATLAB function

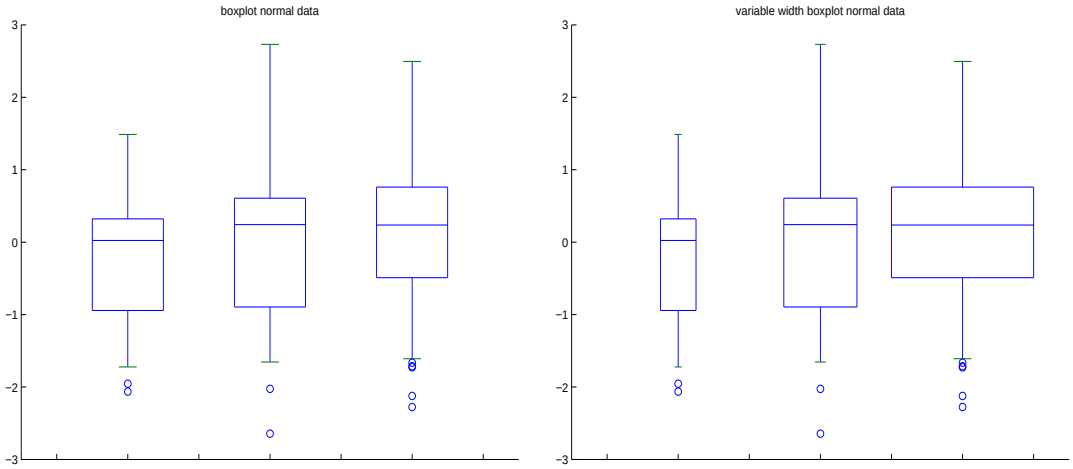

Figure 108: Some examples using the EDA toolbox function `boxp` function to plot boxplots and variable width boxplots. **Left:** Box plots of normal data with different numbers of points 30, 50, and 100. **Right:** A variable width box plot of the same data.

`histfit` on the data which plots a histogram of the data and then overlays a plot of a normal distribution with matched mean and standard deviation on top of the individual histograms. This is implemented in the MATLAB script `prob_9_6.m`. When this script is run it produces the three plots shown in Figure 107. Notice how the normal curve best fits the data generated from a normal distribution and appears to be a poor approximation to the densities found in the other two case.

Exercise 9.7 (using the `surf` plot)

See the MATLAB script `prob_9_7.m` for an implementation of this problem. The requested density is plotted as a surface plot at the end of that script.

Exercise 9.8 (quartiles of the `geyser` data set)

See the MATLAB code `prob_9_8.m` for an implementation of this exercise. Note that to plot the density histogram we used the MATLAB function `cshistden.m` provided in the computational statistics toolbox [8]. This function computes and plots a histogram based density estimate corresponding to the data presented in the `geyser` data set. The observed quantile do correspond to observed values on both the boxplot and the density histogram that are produced when the script is run.

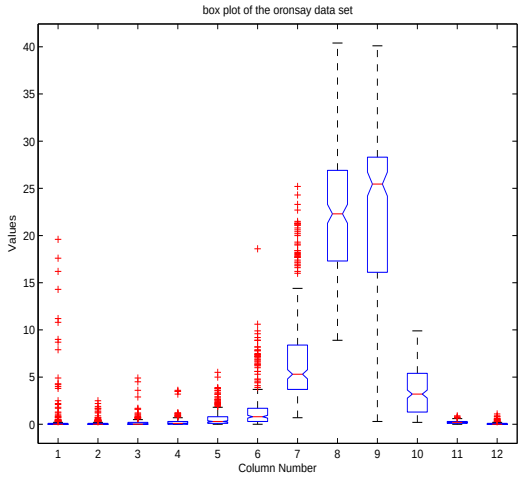

Figure 109: Box plots of the features of the `oronsay` data set. Notice the large number of potential outliers.

Exercise 9.9 (examples of variable width boxplots)

For this exercise we use the EDA toolbox function `boxp` with the argument `vw` (for "variable width") to generate a variable width box plot for some standard normal data. This exercise is implemented in the MATLAB script `prob_9_9.m`. When that script is run it produces the two plots shown in Figure 108.

Exercise 9.10 (using the `histfit` function)

For this problem, rather than generate random data we will use data from the `galaxy` data set; specifically the `EastWest` subdata set which we know from previous studies is approximated well with a normal distribution. In the MATLAB script `prob_9_10.m` we load the `galaxy` data set and apply the `histfit` function. When that script is run the resulting normal fit is quite good.

Exercise 9.12 (histograms of some similar data)

The box plot and box-percentile plot for this artificial data set is computed and displayed in example 9.6 from the book. Histograms for the three specific data sets for this example are constructed in Exercise 9.6 and shown in Figure 107 above.

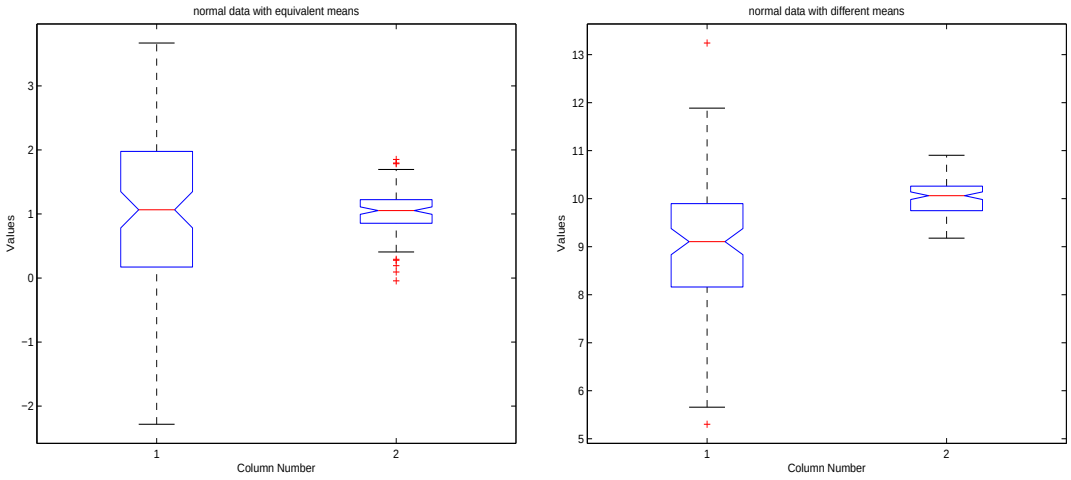

Figure 110: **Left:** Box plots with notches for normal data with the same means (both 1). **Right:** Box plots with notches for normal data with *different* means (one is 9 and the other is 10). Notice that the notches in the second example don't overlap, indicating that there is a significant difference between the means.

Exercise 9.13 (options to `boxplot`)

In the MATLAB script `prob_9_13.m` we use the MATLAB function `boxplot` to plot a boxplot of the `oronsay` data set. The resulting plot with the option `notches` set to `on` is shown in Figure 109. The notches are nice to plot because they provide a robust estimate of the uncertainty in the value of the medians. Boxes that have their notches non-overlapping have medians that differ at the 5% significance level. Notice that with this data set a great number of the individual features have significant outliers.

Exercise 9.14 (the `notches` option to `boxplot`)

In the MATLAB script `prob_9_14.m` we generate two sets of random variables both from a joint Gaussian distribution. In the first set the means of the two random variables are the same (both are 1). While in the second set they are different (one is 9 and the other is 10). We then compute boxplots (with notches) of the two sets of variables. The results from this experiment are shown in Figure 110.

Exercise 9.15 (plotting `notches` with the `oronsay` data set)

See the Exercise 9.13 where this is done.

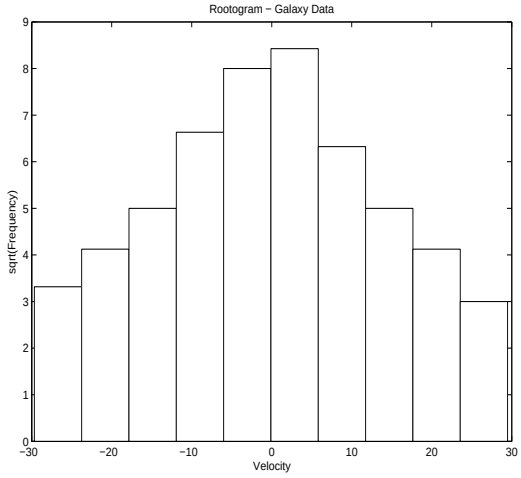

Figure 111: A `rootogram` of the `galaxy` data set.

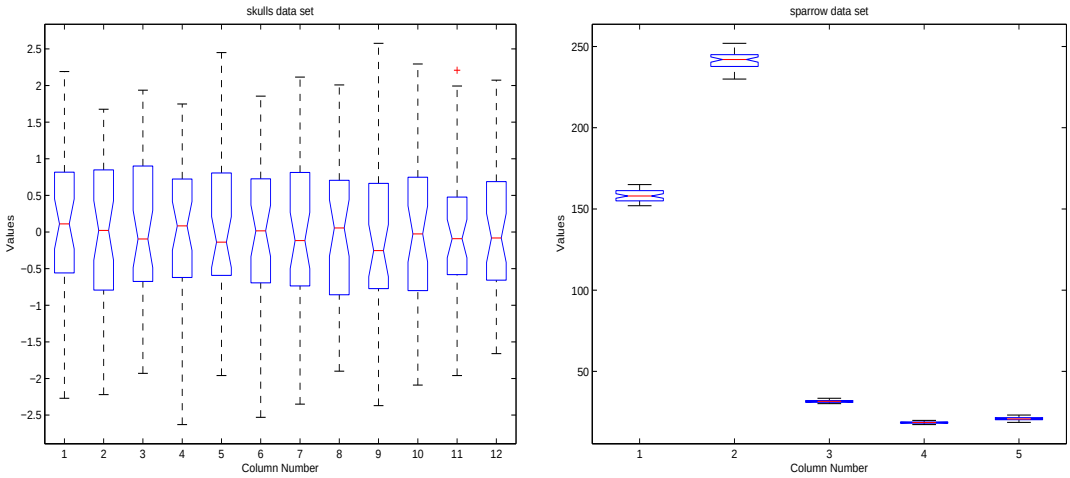

Figure 112: **Left:** Box plots with notches for the `skulls` data set. **Right:** Box plots with notches for the `sparrow` data set.

Exercise 9.19 (a rootogram)

See the MATLAB script `prob_9_19.m` where we compute a `rootogram` using the MATLAB function `hist`. The result from this experiment is shown in Figure 111

Exercise 9.21 (boxplots on various data sets)

Part (a): See the MATLAB script `prob_9_21_skulls.m` for this exercise. When this is run it produces the plot shown in Figure 112 (left).

Part (b): See the MATLAB script `prob_9_21_sparrow.m` for this exercise. When this is

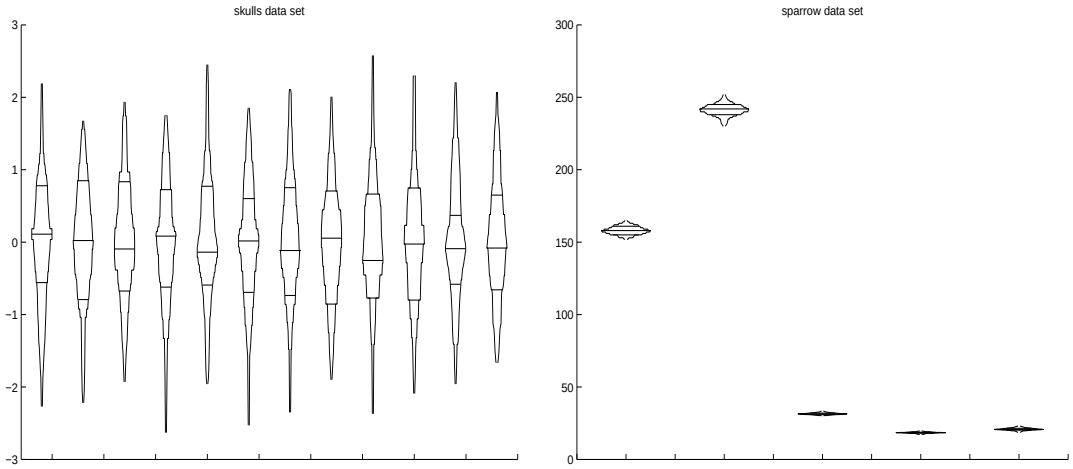

Figure 113: **Left:** Box-percentile plots of the `skulls` data set. **Right:** Box-percentile plots the `sparrow` data set.

run it produces the plot shown in Figure 112 (right).

Exercise 9.22 (variations on boxplots for various data sets)

For this exercise we will use the box-percentile plot technique on a few data sets.

Part (a): See the MATLAB script `prob_9_22_skulls.m` for this exercise. When this is run it produces the plot shown in Figure 113 (left).

Part (b): See the MATLAB script `prob_9_22_sparrow.m` for this exercise. When this is run it produces the plot shown in Figure 113 (right).

Exercise 9.23 (qq-plots of exponential data)

See the MATLAB script `prob_9_23.m` for this exercise. When this is run it produces the plot shown in Figure 114. Notice how the qq-plot of exponential data (left) has sufficient curvature present in the tails (at the far ends of the plot) that the normal distribution (right) does not. This is a good indication that a normal model is *not* appropriate for the given data.

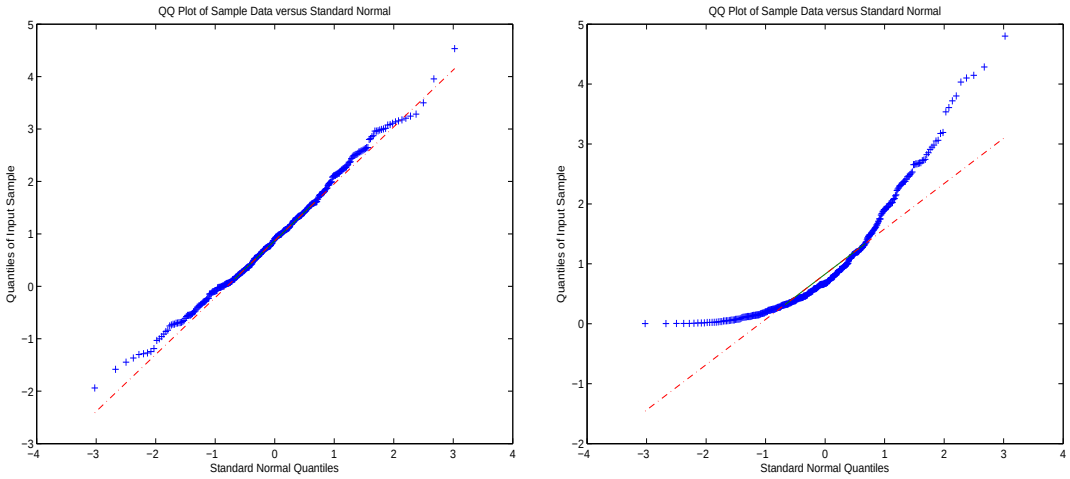

Figure 114: **Left:** The qq-plot of data that *is* normally distributed. **Right:** The qq-plot of data that is exponentially distributed. Notice how the normal data lies almost exactly on the underlying line, while the exponential data posses significant deviation at the ends of the plot.

Chapter 10: Multivariate Visualization

Exercise Solutions

Exercise 10.2 (using the `glyphplot` function)

For this problem we use the MATLAB function `glyphplot` to duplicate Figures 10.1 and 10.2 from the book. See the MATLAB script `prob_10_2.m` for its implementation. When this script is run we plot the Chernoff faces and a star diagram for the `cereal` data set.

Exercise 10.3 (using the `gscatter` function)

For this problem we plot the `oronsay` data set with the data points colored according to their `midden` labeling using the `gscatter` function. See the MATLAB script `prob_10_3.m` for an implementation of this problem. Note that the `gscatter` function is very similar to the function `plot_labeled.m` created for this book.

Exercise 10.5 (a scatterplot matrix of the `oronsay` data set)

See the MATLAB script `prob_10_5.m` for an implementation of this problem. When that script is run we obtain a scatterplot matrix of `oronsay` data set using the MATLAB command `plotmatrix`. Because there are so many features in the `oronsay` data set ($p = 12$) viewing this scatterplot matrix is somewhat difficult.

Exercise 10.6 (using the `hexplot` function)

For this problem we use the `hexplot` function. This function takes a second argument, `N`, that represents the approximate number of bins in the variable that has the *larger* range. The effect of changing this value is what this problem attempts to explore. This problem is implemented in the MATLAB script `prob_10_6.m`. When that script is run by increasing the `N` value one can observe the effect of decreasing the bin width. For this type of plot increasing `N` causes the hexagons to become much more point like.

Exercise 10.8 (a scatter plot of the `hamster` data set)

From a visual inspection using the `plotmatrix` function and the cross hairs function available in the MATLAB figure window see that the outlying data point discussed is located with

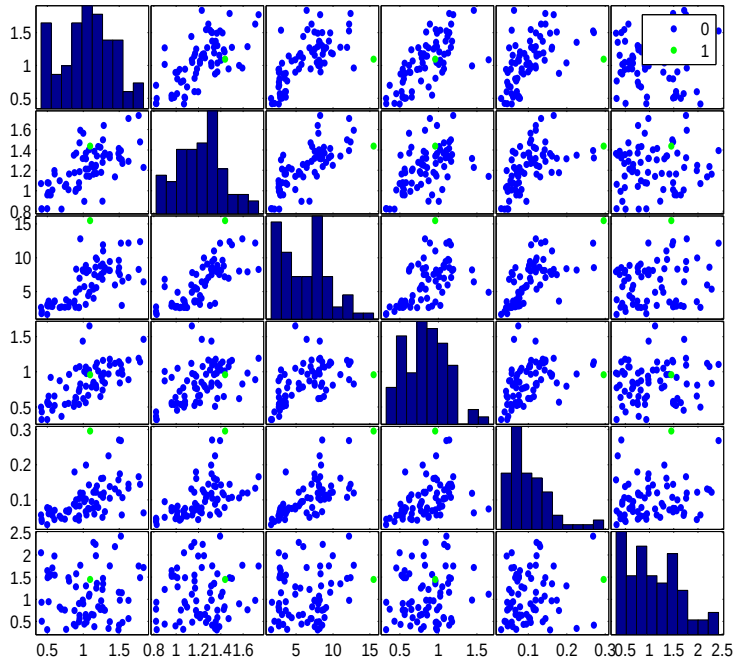

Figure 115: Using the technique of linking to highlight a specific data point in the `hamster` data set. The "special" data point is green while the other points are plotted in blue.

an liver variable value around 15.49 and a spleen variable value of 0.2959. Plotting a scatter plot of the liver variable by itself we find this data point has index $i = 53$. We can then label this point differently from the others and emphasis this using the `gplotmarix` function. The enables one to see this outlying data point in each of the other projections. This problem is implemented in the MATLAB script `prob_10_8.m`. When that script is run one of the the results is presented in Figure 115. In that figure the outlying point is plotted in green, while the other points are plotted in blue.

Exercise 10.9 (a dot chart of the `brainweight` data set)

This problem is implemented in the MATLAB script `prob_10_9.m`. When it is run it produces the requested dotplot. We plot the animal name on the vertical axis and the brainweight value on the horizontal axis.

Exercise 10.11 (correlation viewed in parallel plots)

For this problem we randomly generate two data sets. The first has a correlation coefficient of $+1$ while the second data set has a correlation coefficient of -1. Specifically, for this problem we generate vectors **z1** and **z2** containing n random variables. In MATLAB the command to do this is the following

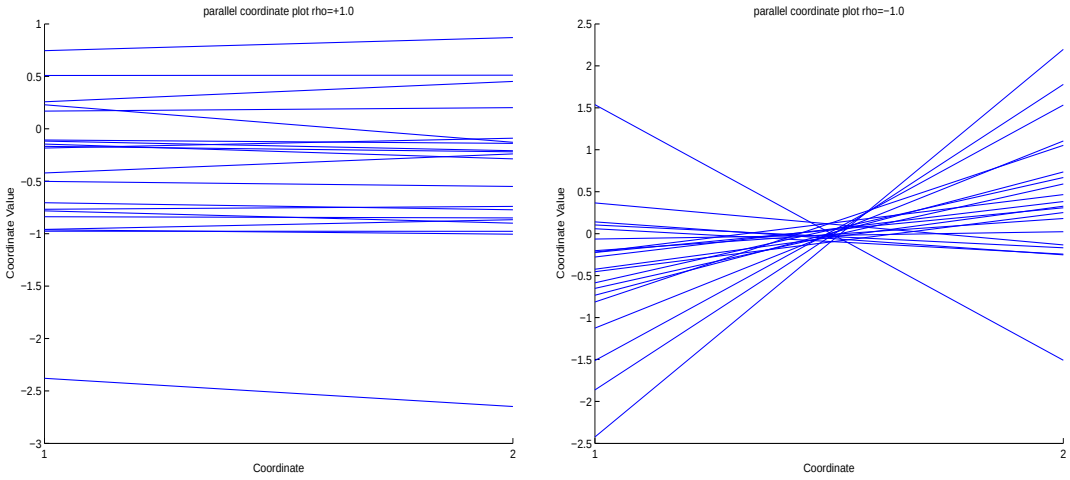

Figure 116: **Left:** A parallel plot of perfectly correlated data. Note how horizontal the lines connecting the points are. **Right:** A parallel plot of perfectly anti-correlated data. Note how "diagonal" the lines connecting the points are.

```
z1 = randn(1,n);
z2 = rho * z1 + sqrt( 1 - rho^2 ) * randn(1,n);
```

This procedure will generate draws of two random variables **z1** and **z2** that have a correlation coefficient of **rho**. We then plot this data using parallel coordinates. When $\rho = +1$ (the data is completely correlated) we get the plot shown in Figure 116 (left) and when $\rho = -1$ (the data is completely anti-correlated) we get the plot shown in Figure 116 (right). Notice that when $\rho = +1$ the parallel plot shows many horizontal looking likes, while when $\rho = -1$ the parallel plot shows many "crossed" lines. Code to run this experiment can be found in the MATLAB script `prob_10_11.m`.

Exercise 10.12 (parallel plots of clustered data)

For this problem we generate two-dimensional data to plot with the toolbox command `parallelcoords`. The first coordinate of our data set is from a Gaussian with zero mean and unit variance. The second coordinate is generated from two different Gaussians. The first has a mean of -2 and the second has a mean of $+2$. Both have unit variance. When we plot this data in a parallel plot we get the Figure 117 (left). Code to run this experiment can be found in the MATLAB script `prob_10_12.m`. Notice the clusters easily seen in the second dimension.

Exercise 10.13 (a `dotchart` of the `playfair` data set)

This problem is implemented in the MATLAB script `prob_10_13.m`. When this is run it produces the plot shown in Figure 117 (right).

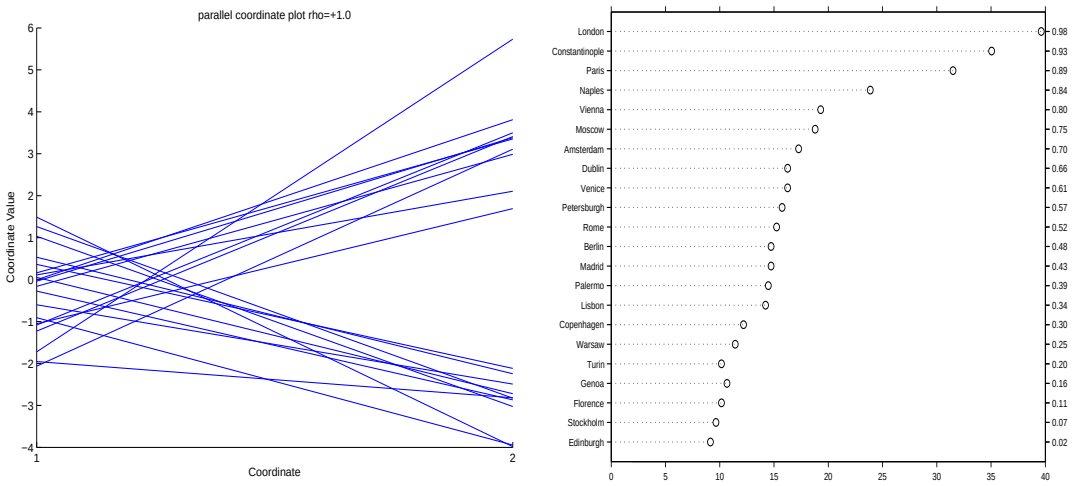

Figure 117: **Left:** A parallel coordinate plot for the data in Exercise 10.12. **Right:** A dotchart of the `Diameter` sub data in the `playfair` data set with the `CD` (cumulative distribution) option.

Exercise 10.15 (a coplot with the dependence on tensile strength)

Since the first variable presented to the `coplot` MATLAB routine is the variable we conditioning on we need to put tensile strength first in the data matrix. When this is done running the MATLAB script `prob_10_15.m` produces the desired coplot.

References

[1] J. D. Banfield and A. E. Raftery. Model-based Gaussian and non-Gaussian clustering. *Biometrics*, 49:803–821, 1993.

[2] G. Celeux and G. Govaert. Gaussian parsimonious clustering models. *Pattern Recognition*, 28(5):781–793, May 1995.

[3] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience Publication, 2000.

[4] C. T. F. and C. M. A. A. *Multidimensional Scaling*. Chapman and Hall, 2001.

[5] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

[6] J. L. Friedman and J. W. Tukey. A projection persuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, C-23:881–889, 1974.

[7] L. Kaufman and P. J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. John Wiley and Sons, New York, 1990.

[8] W. Martinez. *Computational statistics handbook with MATLAB*. Chapman & Hall/CRC, 2001.

[9] W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.

[10] D. W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization (Wiley Series in Probability and Statistics)*. Wiley-Interscience, September 1992.

[11] G. Strang. *Linear Algebra and Its Applications*. Brooks/Cole, 3 edition, 1988.