A Solution Manual for: Computational Statistics Handbook with MATLAB by Wendy L. Martinez and Angel R. Martinez.

John L. Weatherwax*

August 8, 2011

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 novel data sets to use to explore the techniques discussed in the book on. 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[1] but in fact they *do not* and can require some modifications or a different approach altogether to be make 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 computational statistics.

Book Review

With the wide distribution of very powerful personal computers in todays research environment it is imperative that todays practicing statistician be able to understand, implement, and use computational statistical techniques that would have been considered impractical

*wax@alum.mit.edu

[1]e.g. Chapter 9 Problem 9.2 when trying to cluster the `household` data

previously. An example of a very general and powerful technique in this class is the bootstrap, which allows the statistician to extract very useful statistics (like the standard error) by basically performing the same statistical estimation repeatedly. In the applied domain, this technique is certainly not used in as many places as it could be. Simply *recognizing* that we can obtain an estimate of the error on any statistic (no matter how complicated), represents a powerful step in ones understanding and usually allows a number of interesting analysis to be considered. This is just one example of the many useful techniques covered in this excellent book. I would recommend it to anyone looking to learn some very powerful and practical computational statistical techniques. In addition, there is ample MATLAB code, the usage of which is demonstrated in the examples and exercises, available for the user to use on his/her specific problems.

I've found that often simply reading a text is not sufficient for mastery of the presented material. To ensure that I was able to receive the maximal benefit from this excellent book, I worked and implemented a great number of the computational problems and exercises. Notes and codes I developed as I worked through the book can be found here:

`http://waxworksmath.com/Authors/G_M/Martinez/CSHWM/martinez.html`

It is hoped that if other students of this book have questions on the material perhaps these note may provide insight and or directions for further thought. Working them has certainly provided me with a much larger arsenal of tools that can be brought to bear on my statistical projects.

Chapter 2: Probability Concepts

Exercise Solutions

Exercise 2.4 (the Laplacian density)

Lets begin by showing that the given f is a density function. We will integrate and see if we obtain unity. We have

$$\begin{aligned}\int_{-\infty}^{\infty} f(x)dx &= \int_{-\infty}^{0} \frac{\lambda}{2}e^{\lambda x}dx + \int_{0}^{\infty} \frac{\lambda}{2}e^{-\lambda x}dx \\ &= \frac{\lambda}{2}\left(\frac{e^{\lambda x}}{\lambda}\Big|_{-\infty}^{0} + \frac{\lambda}{2}\left(\frac{e^{-\lambda x}}{(-\lambda)}\Big|_{0}^{\infty}\right.\right. \\ &= \frac{1}{2}(1) - \frac{1}{2}(-1) = 1\,,\end{aligned}$$

so the required normalization condition is satisfied.

Part (a): The cummulative distribution function $F(\cdot)$ for a Laplacian density is given by

$$F(x) = \int_{-\infty}^{x} \frac{1}{2}\lambda e^{\lambda\xi}d\xi = \frac{1}{2}\, e^{\lambda\xi}\Big|_{-\infty}^{x} = \frac{1}{2}e^{\lambda x} \quad \text{for} \quad x < 0\,,$$

and

$$F(x) = \frac{1}{2} + \int_{0}^{x} \frac{1}{2}\lambda e^{-\lambda\xi}d\xi = \frac{1}{2} + \frac{\lambda}{2}\,\frac{e^{\lambda\xi}}{(-\lambda)}\Big|_{0}^{x} = 1 - \frac{1}{2}e^{-\lambda x} \quad \text{for} \quad x > 0\,,$$

Chapter 3: Sampling Concepts

Notes on the Text

The book presents the empirical estimate of a distributions p-th quantile $\hat{q}_p$ by finding the index j and returning $X_{(j)}$ such that

$$\hat{q}_p = X_{(j)} \quad \text{where} \quad \frac{j-1}{n} < p \leq \frac{j}{n} \quad \text{for} \quad j = 1, 2, \cdots, n\,. \tag{1}$$

Simple code in C++ to compute this index j is given by

```
int pthQuantileIndexJ(double p, int n){

  // j is a "one" based index i.e. v[1] is the first element of the array v[]
  int j;
  if( p==0. ){  // p==0 is a special case
    j = 1;
  }else{
    j = static_cast<int>(ceil(n*p));
  }
  return j-1; // return a "zero" based index

}
```

Exercise Solutions

Exercise 3.1

See the code `prob_3_1.m` for Matlab code to perform these experiments. We see that all distributions are centered on the mean 0 and that the variance decreases as the sample size increases. In addition, the empirical distribution appears Gaussian in agreement with the central limit theorem.

Exercise 3.2

See the code `prob_3_2.m` for Matlab code to perform these experiments. We see that all distributions are centered on the mean of 0.5 and that the variance decreases as the sample size increases. In addition, the empirical distribution appears Gaussian in agreement with the central limit theorem.

Exercise 3.3

Since the mean square error estimator can be written in terms of the sum of the variance of the estimator and the square bias of the estimator. In equation

$$\mathrm{MSE}(T) = \mathrm{V}(T) + \mathrm{Bias}(T)^2 \,.$$

Since we are told that these estimators are unbiased we know that $\mathrm{Bias}(T) = 0$ for both. Thus $\mathrm{MSE}(T) = \mathrm{V}(T)$. The better estimator would be the one with the smaller mean square error (equivalently variance) which in this case is T_1 and is called the more efficient estimator. The relative efficiency of estimator T_1 to T_2 is given by

$$\mathrm{eff}(T_1, T_2) = \frac{\mathrm{MSE}(T_2)}{\mathrm{MSE}(T_1)} = \frac{\mathrm{V}(T_2)}{\mathrm{V}(T_1)} = 2 \,.$$

Exercise 3.4

See the code `prob_3_4.m` for Matlab code to perform these experiments. As the sample size gets larger the empirical coefficient of skewness and kurtosis converge to their true theoretical values.

Exercise 3.5

See the code `prob_3_5.m` for Matlab code to perform these experiments. As the sample size gets larger the empirical coefficient of skewness and kurtosis converge to their true theoretical values.

Exercise 3.6

See the code `prob_3_6.m` for Matlab code to perform these experiments. The cumulative distribution function for a uniform random variable is linear. This behavior can be seen by running this code.

Exercise 3.7

See the code `prob_3_7.m` for Matlab code to perform these experiments. The cumulative distribution function for a normal random variable has a sort of "s" shape. This behavior can be seen by running this code. The value of the empirical cumulative distribution function evaluated at the smallest sample value should be somewhat near zero. The value of the empirical cumulative distribution function evaluated at the largest sample value should be somewhat near one.

Exercise 3.8

The quartiles are the values $q_{0.25}$, $q_{0.5}$, and $q_{0.75}$ such that

$$p = \int_{-\infty}^{q_p} f(x)dx\,,$$

in otherwards the points q_p are the limit of the density function such that the total integral is equal to p. For the quartiles, the point $q_{0.5}$ can be approximated as the sample median, the point $q_{0.25}$ can be approximated as the median of the points that are *less* than the median (over the entire sample) and the point $q_{0.75}$ can be approximated at the median of the points that are *greater* than the median (again over the entire sample). See the code `prob_3_8.m` for Matlab code to perform these experiments.

Exercise 3.9

We can approximate the quantiles $\hat{q}_p$ with a random sample from a given distribution from the order statistics $X_{(j)}$ from the sample. Here j is selected such that

$$\frac{j-1}{n} < p < \frac{j}{n}\,.$$

Thus after sorting the random sample of data, to compute the quantile $\hat{q}_p$ we compute the index floor(np) and extract this order statistics from the sorted random sample. See the code `prob_3_9.m` for Matlab code to perform these experiments.

Exercise 3.10

Empirical estimates of any given quantile are calculated in the Matlab code `sample_quantiles.m`. Running the code

```
sample_quantiles( randn(1e6,1), [ 0.01, 0.333, 0.5, 0.666, 0.99 ] )
```

shows some approximate quantile values for the standard normal distribution.

Exercise 3.11

To facilitate this exercise we can generate the sample quantiles using linear interpolation in the Matlab function `sample_quantiles_linInterp.m`. Experiments with an increasing number of samples can be found in the file `prob_3_11.m`. As the number of samples goes up the estimates indeed improve.

Exercise 3.14

The Matlab command

```
norminv( [ 0.25 0.5 0.75 ] )
```

gives the *theoretical* quantiles for the standard normal. The same thing can be approximated using code from this chapter and running the command

```
sample_quantiles( randn(1e3,1), [ 0.25, 0.5, 0.75 ] )
```

or

```
sample_quantiles_linInterp( randn(1e3,1), [ 0.25, 0.5, 0.75 ] )
```

The results are quite similar to the theoretical values and improve as we add increase the number of samples.

Exercise 3.15

The quartile coefficient of skewness is code in the Matlab function `quartile_coef_skewness.m`. A simple example of running this code is given by the call

```
quartile_coef_skewness( randn(1e3,1) )
```

Exercise 3.16

In the Matlab file `cmpt_ml_var_bias.m` for a function that computes the empirical bias for many random samples using the maximum likelihood estimator of the variance. The Matlab file `prob_3_16.m` drives this code and plots histograms of the bias that results from many Monte-Carlo trials. In Figure 1 we present the result from running this driver code. On the left is a plot of the empirical bias when running many experiments with the maximum likelihood estimator of the variance. On the right is the same set of experiments performed with an unbiased estimator. These results show that the biased estimator is systematically shifted away from zero. To derive these results we used samples consisting of 10 samples.

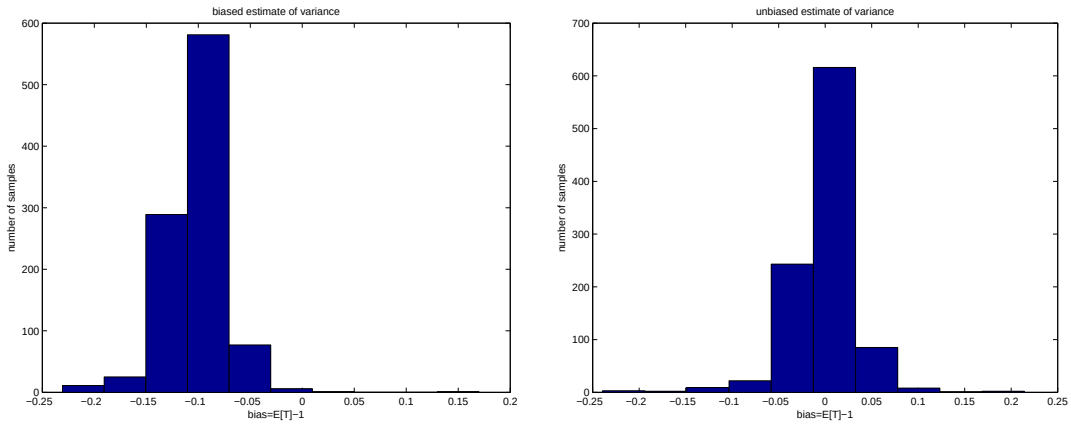

Figure 1: A comparison between a biased and an unbiased estimator of the variance. **Left:** using a biased variance estimator. **Right:** Using an unbiased variance estimator. Note the constant shift to the left in the left most plot that results from using a biased estimator.

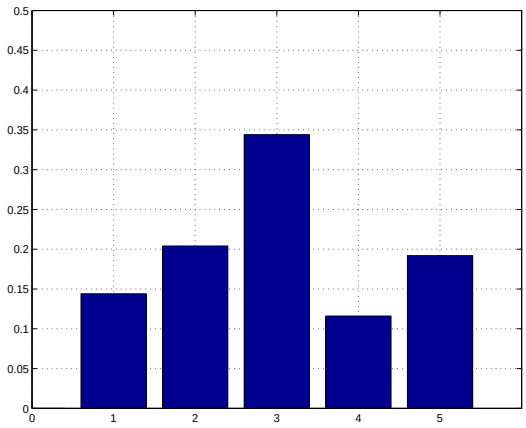

Figure 2: The empirical probabilty mass for each random variable when using the discrete form of the acceptance rejection method. These empirical probabilites match quite well the true probabilities.

Chapter 4: Generating Random Variables

Exercise Solutions

Exercise 4.1

Skipped, but as the sample size gets larger the relative frequencies of each discrete value will converge to the probabilities of occurance.

Exercise 4.2

See the code `disc_acc_rej.m` for an implementation of a discrete form of the acceptance rejection method for generating non-uniform discrete random varialbes from uniform discrete random variables. Then see the code `prob_4_2.m` for an implementation of how to call this function. There we generate 10^6 discrete random numbers distributed according to the given discrete distribution and then compare the empirical probabilities of each variable with the true distributions. See the Figure 2 for the output from running this code.

Exercise 4.3

This is implemented in the Matlab function `disc_inv_tranf.m` and is driven by the Matlab function `prob_4_3.m`.

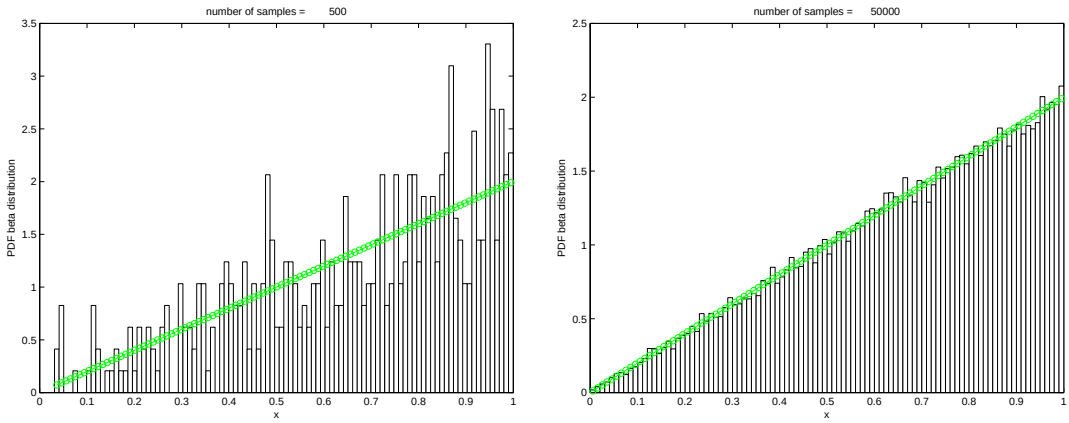

Figure 3: A comparison between the empirical probability density function for a beta random variable (with parameters $\alpha = 2$ and $\beta = 1$, and the true expression (which is $2x$). **Left:** with an empirical density generated from 500 samples. **Right:** with an empirical density generated from 50000 samples.

Exercise 4.7

See the Matlab code `rubinstein_beta.m` and the driver code `prob_4_7.m` to run the requested experiment.

Exercise 4.8

The requesed code is the the Matlab file `prob_4_8.m`. When this code is run it shows that of the total number of random draws generated `irv+irej` approximatly one half of them are rejected.

Exercise 4.9

The requesed code is the the Matlab file `prob_4_9.m`. When this code is run to generate 500 samples the comparison of the empirical probability density function and the true probability density function is shown in Figure 3 (left). When this code is run to generate 50000 samples the comparison is shown in Figure 3 (right).

Exercise 4.10

Skipped since this is very much like Exercise 4.2.

Exercise 4.11

Using the Matlab function `chi2pdf` these plots are produced in the routine `prob_4_11.m`.

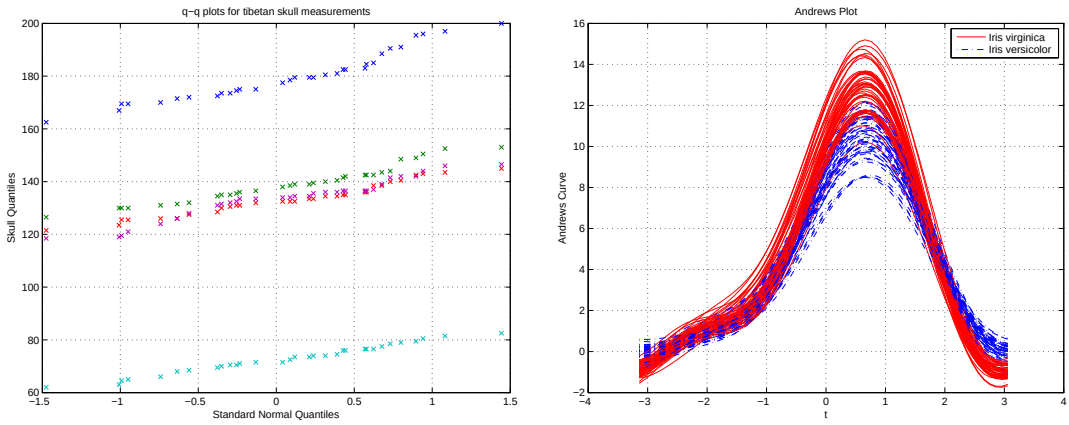

Figure 4: **Left:** Q-Q plots of the tibetian skull data. The fact that each curve is so linear implies that it maybe well modeled with a Gaussian distribution. **Right:** The Andrews plots for the *Iris virginica* and the *Iris versicolor* species of iris produced from Exercise 5.5.

Chapter 5: Exploratory Data Analysis

Exercise Solutions

Exercise 5.1

See the Matlab file `prob_5_1.m` for this exercise.

Exercise 5.2

See the Matlab file `prob_5_2.m` for this exercise.

Exercise 5.3

See the Matlab file `prob_5_3.m` for this exercise. We generate random variates to use in comparing with the tibetian data. The q-q plot is presented in Figure 4 (left). There one can see that the skull measurements would be very accurately approximated with a Gaussian distribution.

Exercise 5.4

See the Matlab file `prob_5_4.m` for this exercise. This code plots both the planar "slice" through the trivariate normal distribution and the curvy slice using the peaks function.

Exercise 5.5

See the Matlab file `prob_5_5.m` for this exercise. The Andrews plots for the *Iris virginica* and *Iris versicolor* data are shown in Figure 4 (Right). In that plot these two species don't seem to be as separated as the *Iris setosa* and *Iris virginica* species. They look more difficult to classify with these features. In fact the *naming* conventions might indicate that the botanist found the *setosa* species significantly different than the *virginica* and the *versicolor* species. This seems to be verified in the Andrews plots.

Exercise 5.6

See the Matlab file `prob_5_6.m` for this exercise. We can tell that the locations (center points) of the two distributions are different because the center of the "line" is shifted left or right from zero depending on the values of the means of the two normals. We can tell that the scales of the two Gaussians are different because the quantiles for the x and y axis are larger or smaller than what they would be for the standard normal. For example the standard normal has a 99.9% of the probability less than the value of 3. While a Gaussian with a standard deviation of σ has this same probability less than the value of 3σ.

Exercise 5.7

See the Matlab file `prob_5_7.m` for this exercise. There we randomly select an order for the features and plot the corresponding parallel plot for all *three* species of iris. The results of running this code is show in Figure 5 (left). In that figure one can see that the species *versicolor* and *virginica* are rather similar in each coordinate value, but the species *setosa* is different than the other two in the first and third features. This suggests that these two features might be the best to us for classification.

We note that permuting the axis does not affect the results we obtain but only the vertical location of the individual feature indices.

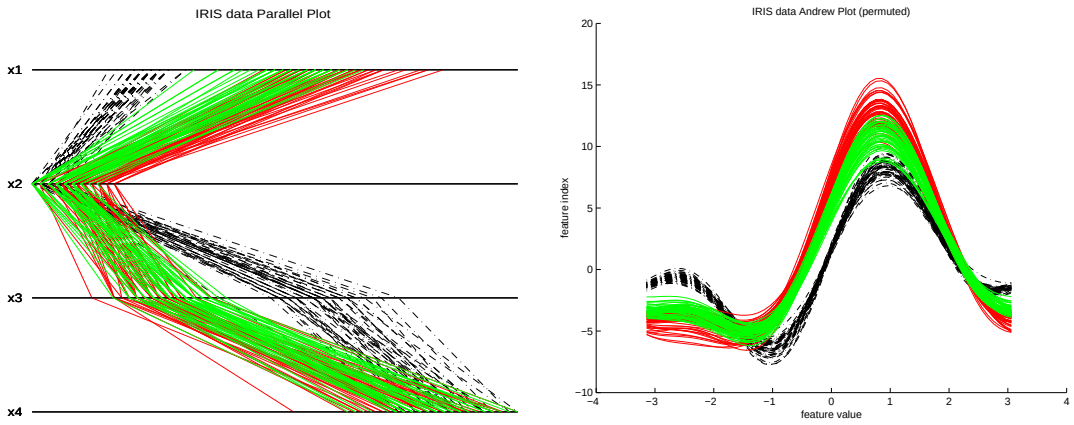

Figure 5: **Left:** The result from plotting the three species of iris using a parallel plot. The species *setosa* is plotted in black, *virginica* is plotted in red and the species *versicolor* is plotted in green. **Right:** The result from plotting the three species of iris using a Andrews plot. Each run of the code `prob_5_8.m` uses a *different* permutation of the feature indices and so may yield a slightly different plot than that shown here. The species *setosa* is plotted in black, *virginica* is plotted in red and the species *versicolor* is plotted in green.

Exercise 5.8

See the Matlab file `prob_5_8.m` for this exercise. The result from one particular run is shown in Figure 5 (right). As discussed earlier the species *setosa* appears easier to classify than the other two species.

Exercise 5.10

See the Matlab file `prob_5_10.m` for this exercise. When this script is run one can easily see there looks to be easy separation between the clusters of insects. It appears that a linear boundary (planes) would work to separate the classes.

Exercise 5.11

See the Matlab file `prob_5_11.m` for this exercise.

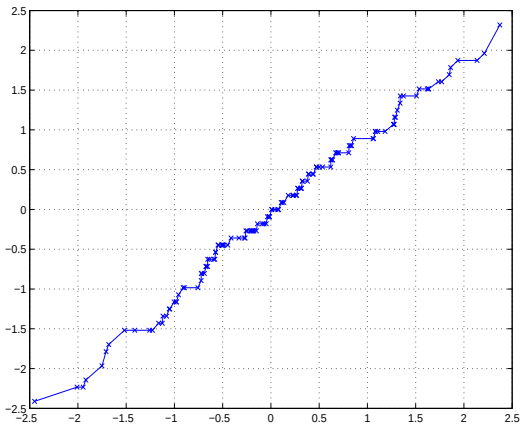

Figure 6: A Q-Q plot of the order statics for the sphered forearm data v.s. the quantiles of the standard normal. The fact that the resulting curve is linear is a strong indication that this data is normally distributed.

Exercise 5.12

That these two vectors are orthogonal can be seen by taking their dot product. We find that

$$\begin{aligned} \alpha(t)^t\beta(t) &= \frac{2}{d}(\sin(\omega_1 t)\cos(\omega_1 t) - \cos(\omega_1 t)\sin(\omega_1 t) + \cdots \\ &+ \sin(\omega_{d/2} t)\cos(\omega_{d/2} t) - \cos(\omega_{d/2} t)\sin(\omega_{d/2} t)) = 0\,. \end{aligned}$$

Since there are an even number of components they all cancel in pairs and the dot product is zero.

Exercise 5.14

See the Matlab file `prob_5_14.m` for this exercise. When this is run it plots two alpha level contour of the trivariate standard normal. As expected the isosurfaces are spheres.

Exercise 5.15

See the Matlab file `prob_5_15.m` for this exercise. To better see if this data is distributed via a standard normal one can sphere the points i.e. compute new variables s_i from the given x_i by subtracting the mean and dividing by the standard deviation. That is

$$s_i = \frac{x_i - \hat{\mu}}{\hat{\sigma}}\,.$$

When we do this and plot the order statistics for the given data the plot we obtain is given in Figure 6.

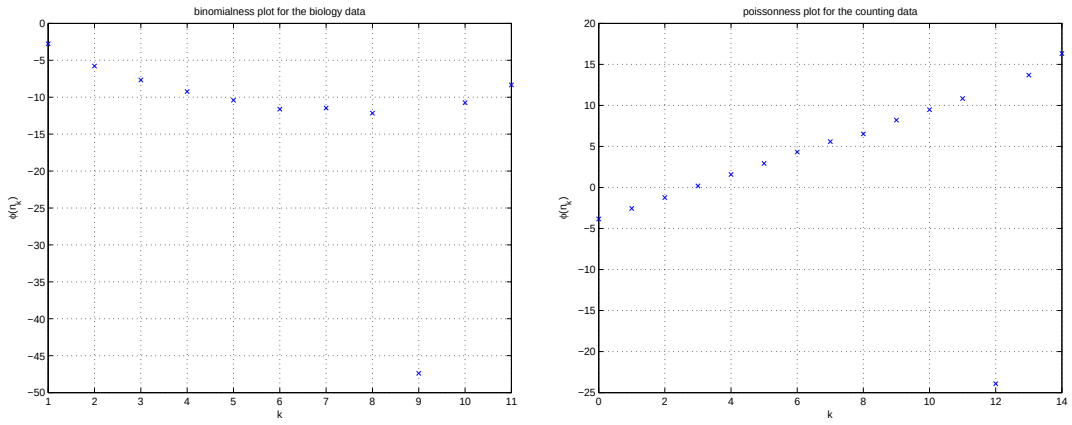

Figure 7: **Left:** The Binomialness plot for the `biology` data set. Note that the degree of linearity is not that great. **Right:** The Poissonness plot for the `counting` data set.

Exercise 5.16

See the Matlab file `prob_5_16.m` for this exercise. When this code is run the resulting stem and leaf plot appears rather skewed. This skewed distribution also might have a small second mode at larger values.

Exercise 5.17

See the Matlab file `prob_5_17.m` for this exercise. When this code is run the resulting plot is shown in Figure 7 (left). The fact that this plot is not very linear indicates that the given data is *not* distributed with a binomial distribution.

Exercise 5.18

See the Matlab file `prob_5_18.m` for this exercise. When this script is run it produces a plot that is shown in Figure **??** (right). There one can see the very strong linear relationship that exists gives a strong indication that this data maybe Poisson distributed.

Exercise 5.19

See the Matlab file `prob_5_19.m` for this exercise. When this plot is run it generates box plots for each of the three species of iris for each of the four features. These plots are shown in Figure 8.

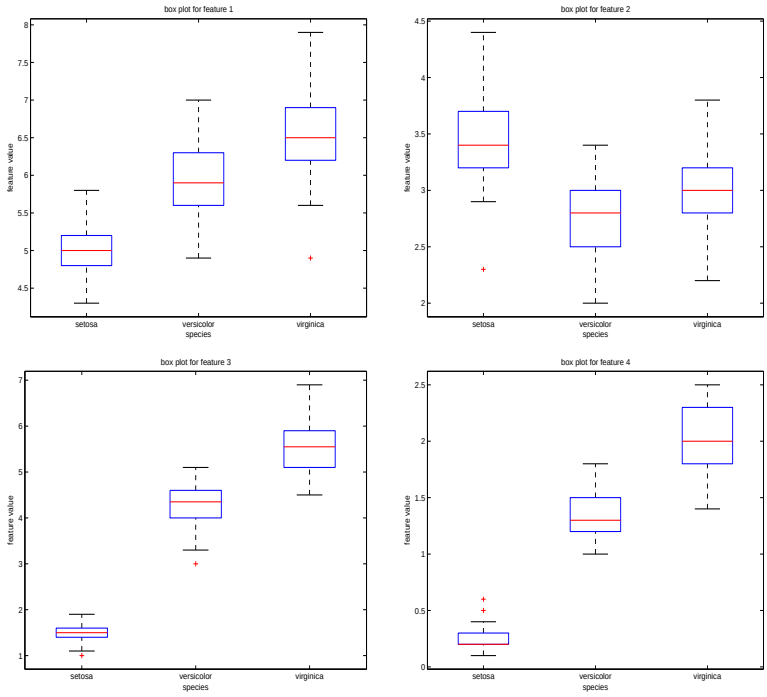

Figure 8: Box plots for the four features of the iris data.

Exercise 5.20

See the Matlab file `prob_5_20.m` for this exercise. When this script is run it produces the boxplot that is shown in Figure 9 (left). One can see immediately that the normal patients have a much smaller value (and tighter distribution) of features when compared to the diabetic patients.

Exercise 5.21

See the Matlab file `prob_5_21.m` for this exercise. Running this code produces the interesting plot shown in Figure 9 (right).

Exercise 5.22

See the Matlab file `prob_5_22.m` for this exercise. We basically run the computational statistics commands written for this book that do projection pursuit for exploratory data analysis (PPEDA). When we attempt four "revisions" with the data we obtain the plots in Figure 10. The revision are shown clockwise from top to bottom. To highlight how well this routine performs at finding structure in the data we color the data corresponding to "genuine" money in green and that corresponding to forged money in red. What is very apparent is that for the first projection PPEDA does a very nice job of finding a projection that separates the

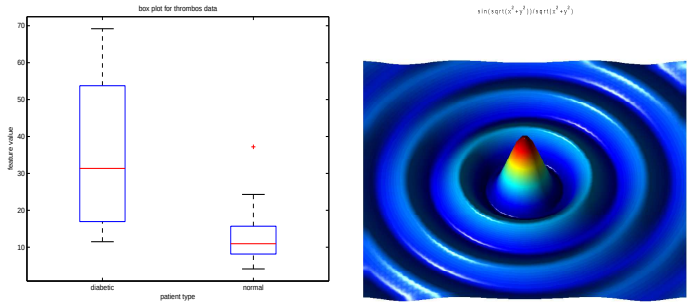

Figure 9: **Left:** The boxplot for the `trombos` data set. **Right:** An example of some of the shading options in Matlab for three dimensional graphics.

two classes very nicely. The other projections seem to be informative also.

Exercise 5.23

See the Matlab file `prob_5_23.m` for this exercise. When this script is run we use the Matlab command `plotmatrix` to generate scatter plots of all features against all others. The sphered data does not look much different than the non-sphered data. When the project pursuit routine is run quite a few clusters can be observed. The first four are shown in Figure 11

Exercise 5.24

See the Matlab file `prob_5_24.m` for this exercise. An initial study of the data do not show much clustering of the classes. An initial run of the projection pursuit algorithm for exploratory data analysis PPEDA (with its default parameters) did not give clusters in the data that provided much discrimination potential.

To further study this problem I requested that the project pursuit algorithm search longer to find with the number of random starts to 50 and the number of steps to be taken with no increase in our clustering metric to 50. With these parameter settings I was not able to observe much clustering. See the Figure 12 for the first four projections.

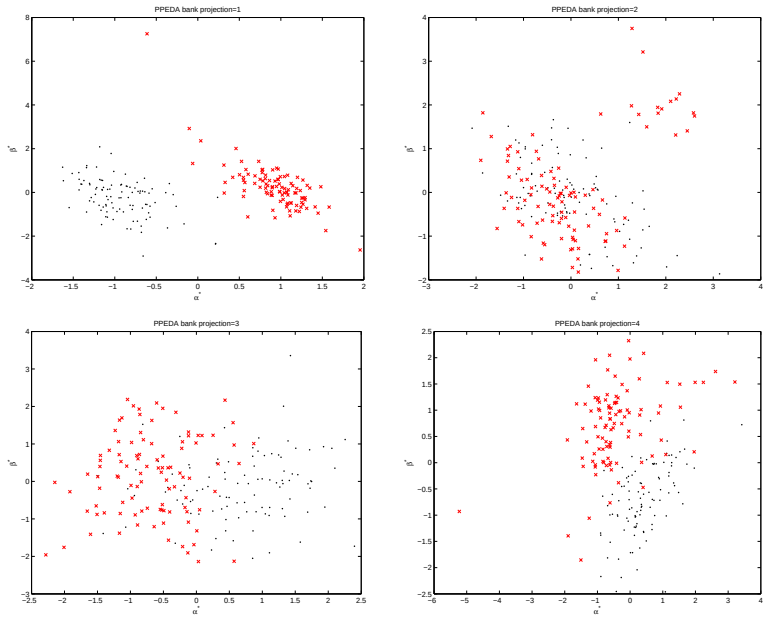

Figure 10: Four projections produced by using PPEDA on the bank data. The first projection does the best job at separating the clusters. The third and fourth projection is quite good also. The red dots correspond to forged money, while the black dots correspond to genuine money.

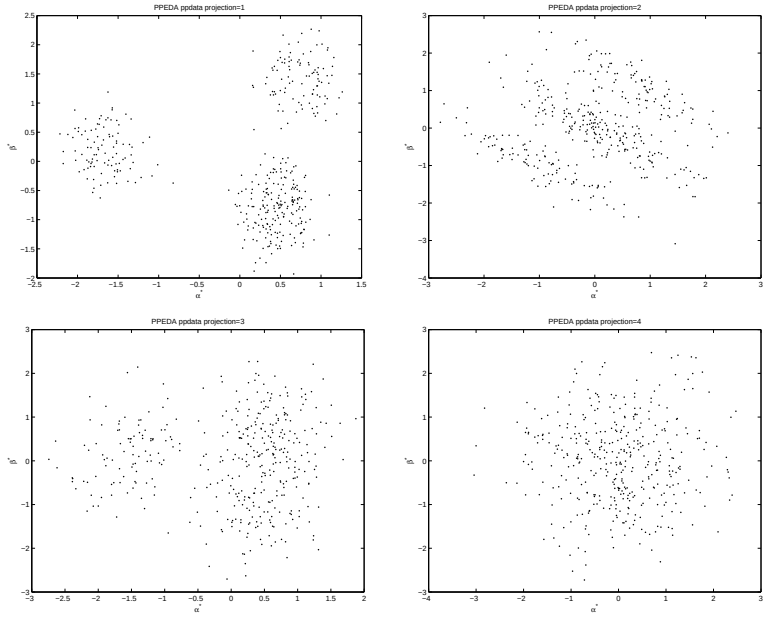

Figure 11: Four projections produced when using PPEDA for the synthetic data given in the book. Notice that the first, second, and third do a nice job of clustering the data.

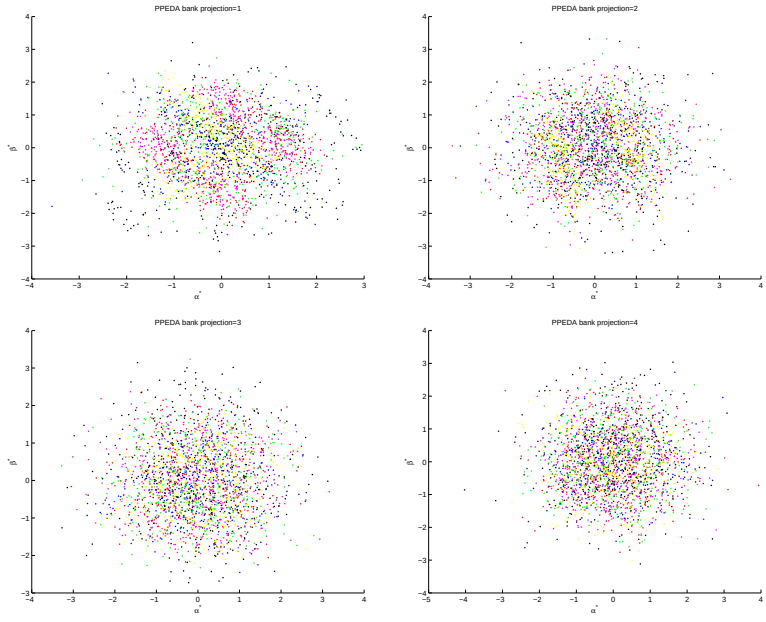

Figure 12: Four projections produced when using PPEDA for the "posse" data given in the book. None of these projections is particularly strong at representing clusters in this data.

Chapter 6: Monte Carlo Methods for Inferential Statistics

Exercise Solutions

Exercise 6.1

See the Matlab file `prob_6_1.m` for this exercise. When we change the models standard deviation to 5 from 15 it makes it much more likely that if the sample statistic is *larger* than the expected 45 then the population has *changed* its mean travel time. In this case the sample mean is located 4.4 standard deviations from the mean. This happens with such a small probability that the Matlab function `norminv( -4.4, 0, 1 )` can't compute it. Thus this value of $\bar{x}$ is not consistent with the null hypothesis.

Exercise 6.2

We recall the two types of errors that can happen in hypothesis testing with a simple description that will hopefully help remember them. If we think of the hypothesis H_0 as the tried-and-true "accepted truth" and the alternative hypothesis H_1 as the current new idea. A type I error happens when one incorrectly rejects the status quo hypothesis H_0 and decides instead to believe in the new idea or H_1. On the other hand, a type II error happens when one incorrectly rejects the new idea H_1 and instead maintains once believe in the old idea, H_0. We can perhaps relate these mathematical ideas to the way some people may act or develop as they age. As a young person it is often the case that when starting something new you think that the old way of doing something is outdated and wrong. If we label the old way of operating H_0 and the new way H_1 it might postulated that early on, young people are appt to make a type I mistakes in that they might argue against the old method H_0 claiming that it is no longer valid and that the new method H_1 is the correct way. On the other hand it might be argued that old people are appt to make type II mistakes. Meaning that when a new idea is presented (a H_1 hypothesis) they are appt to *not* believe or accept that idea and to instead stay with the status quo H_0. If H_0 is not correct they will have made a type II error.

For this problem, as the mean of the true distribution of the travel times μ increases our measured statistic $z_0 = \frac{\bar{x}-\mu_0}{\sigma_X/\sqrt{n}}$, also increases because the data used in computing $\bar{x}$ increases. Thus the probability that z_0 falls in a region generated by H_0 decreases. Thus the probability that we make a type II error (fail to reject H_0) decreases. The numeric probability that we make a type II error (as a function of the true distribution mean μ) is the probability that the statistic $\bar{x}$ derived from H_1 is less than our hypothesis testing threshold leading to us to accept H_0. This probability is equal to the cumulative distribution function for a normal (with mean μ) evaluated at evaluated at our critical values.

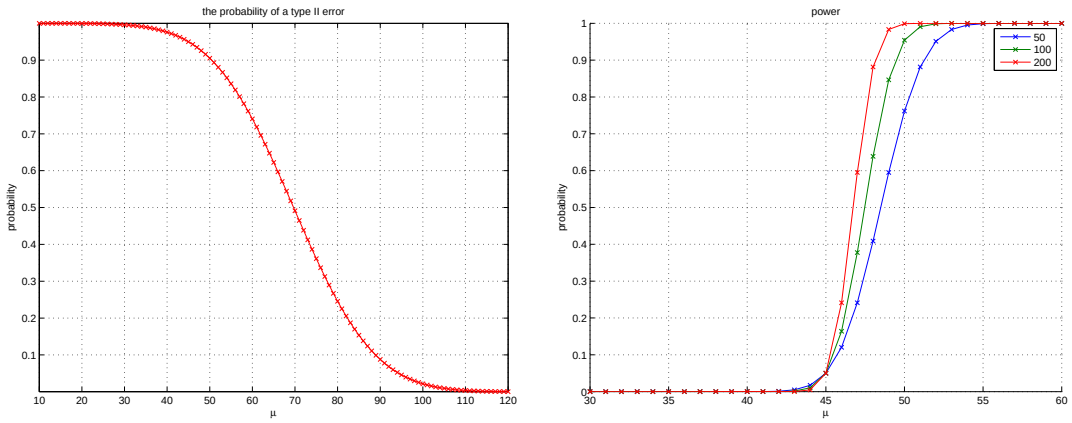

Figure 13: **Left:** The probability of a type II error as a function of the mean of the hypothesis H_1. As the mean decreases, our measurement statistics decreases, and we are almost certain that the hypothesis H_0 will be accepted and a type II error. In the same way as the mean increases, our measurement statistic increases we are almost certain that the hypothesis H_0 will be rejected avoiding a type II error. **Right:** The power (the probability we don't make a type II error) of our hypothesis test for several value of n. We see the curve steepening as the number of samples increases.

See the Matlab file `prob_6_2.m` for this exercise. When this code is run it produces a plot that can be seen in Figure 13 (left). See the figures caption for a discussion. This is plot the complement (one minus) of the power.

Exercise 6.3

For the p-value approach to hypothesis testing for the given statistic observed we compute the probability of obtaining this sample under the assumption that H_0 is true. If this value is "too small" we reject H_0. The p-value is the probability you are still correct if you maintain your believe in H_0 or the probability that our observed statistic was produced randomly from H_0. If we consider the p-value from this example we find it to be given by 0.07. An alpha threshold of 0.1 would state that if the probability of observing our statistic under H_0 is less than this amount we reject the H_0 hypothesis. Since in this case that is true we would reject the H_0 hypothesis. See the Matlab file `prob_6_3.m` for this exercise.

Exercise 6.4

From our study of the statistics of the sample mean, we recall that the sample mean itself has a mean of that of the true distribution μ_0, and a variance given by $\frac{\sigma_0}{\sqrt{n}}$. As n increases this variance shrinks to zero. The smaller our variance the more powerful we expect our test to be at avoiding type II errors (failing to reject H_0). In the Matlab file `prob_6_4.m` we compute the power of our test for the three sample n values suggested in the text. The

resulting plot is shown in Figure 13 (right).

Exercise 6.5

Here we are interested in the hypothesis test

$$\begin{aligned} H_0 &: \mu = 454 \\ H_1 &: \mu \neq 454\,, \end{aligned}$$

for which we will need to use a two tailed test. We will do Monte-Carlo simulation to perform our hypothesis test in this case. To do this we assume we know the exact model (i.e. the mean and standard deviation) under which data was generated from the H_0 hypothesis and generate a large number of "samples" M (each "sample" is consists of n points) from H_0. For each sample we compute the mean, which in this case is our test statistic. We then have M example values of test statistics under the hypothesis H_0. Next we pick a significance level α, the probability that we don't reject H_0 given that H_1 is true. This α is the probability we make a type I error. With this α we then compute the $\alpha/2$ and $1-\alpha/2$ quantiles. If our original sample falls *outside* of this region, we can reject H_0 in favor for concluding H_1.

In implementing the procedure above, since we have samples of our statistic from H_0 we need to compute the empirical quantiles, which we do with the Matlab function `csquantiles`. When we do this we find these values to be

$$\begin{aligned} \hat{q}_{\alpha/2} &= -1.6444 \\ \hat{q}_{1-\alpha/2} &= +1.6691\,. \end{aligned}$$

Since our observed statistic is -2.5641, we reject the hypothesis H_0 in favor for the hypothesis H_1. These calculations are done in the Matlab file `prob_6_5.m`.

Exercise 6.6

See the Matlab file `prob_6_6.m` for calculations of the approximate type I error α with a larger number M of bootstrap samples. One sees that the more bootstrap samples one uses the better the approximation to the true type I error. This result is to be expected.

Exercise 6.7 (the parametric bootstrap)

The parametric bootstrap procedure is the following algorithm/procedure:

- Given a random sample $\mathbf{x} = (x_1, x_2, \cdots, x_n)$, calculate our function (statistic) of these variables $\hat{\theta}$. Here x_i can be a scalar or a vector quantity.

- Estimate the parameters $\mathcal{P}$ for the distribution that did (or might have) generated these n feature measurements $\{x_i\}_{i=1}^n$. Thus we now assume that we know the distribution function ($p(x|\mathcal{P})$) of the features.
- Sample from this distribution $p(x|\mathcal{P})$, n times to get enough data to compute our statistic of interest $\hat{\theta}^{*b}$.
- Repeat the previous two steps B (the number of bootstraps) times.
- This gives B estimates of your statistic θ. From the empirical distribution of these B samples one can obtain an estimate of any desired characteristic of your statistic θ. Common quantities of interest are the standard error, the bias, the confidence interval etc.

To estimate the standard error we take the standard deviation of the above bootstrap generated distribution of our statistic θ. The bias of our estimator is estimated using the initial sample $\mathbf{x}$ (to compute $\hat{\theta}$) and then using all the bootstrap replicas as

$$\hat{\text{bias}}_B = \hat{\bar{\theta}^*} - \hat{\theta}\,.$$

Here $\hat{\bar{\theta}^*}$ is the bootstrap estimated mean and $\hat{\theta}$ individual sample estimate (using our original sample $\mathbf{x}$) of our function $\hat{\theta} = T = t(x_1, x_2, \cdots, x_n)$. The bootstrap percentage interval is obtained from the bootstrap generated distribution of our statistic θ. We specify the probability of making a type I error α (failing to reject the null hypothesis) and compute the interval $\alpha/2$ confidence interval

$$(\hat{\theta}_B^{\alpha/2}, \hat{\theta}_B^{1-\alpha/2})\,.$$

Where the bootstrap quantiles $\hat{\theta}_B^{\alpha/2}$ are determined empirically, using something like the computational statistics toolbox function `csquantiles` or the Matlab function `quantile`.

See the Matlab file `parm_bootstrap.m` for an implementation of the parametric bootstrap and `prob_6_7.m` for an example using this function with the `forearm` data to compute parametric bootstrap estimates of the standard error of the skewness, kurtosis, and standard percentile interval.

Exercise 6.8 (the bootstrapped confidence interval)

I choose to do this problems based on the variance function `var` rather than the standard central moment. From the `forearm` data we get a value of the variance given by 1.25546. The empirical bootstrapped confidence interval (computed here) with $\alpha = 0.1$ is given by $(1.03746, 1.47049)$. The *parametric* bootstrap computed confidence interval is determined in the previous problem and is given by $(1.01545, 1.52213)$. The parametric estimate is wider since when we generate the bootstrap samples from a Gaussian, we could obtain data points that are larger than the ones we would see by considering only sampling with replacement.

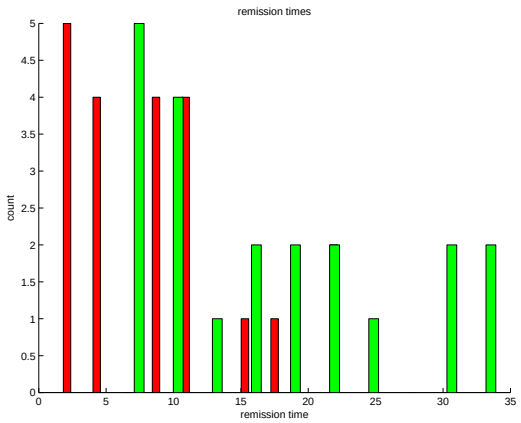

Figure 14: Histogram plots of the data in the `remiss` data set. The red bars correspond to the control group while the green bars correspond to the experimental group.

The empirical bootstrapped α confidence interval can be computed with the Matlab function `bootstrp_sci.m`, and this function is called in the script `prob_6_8.m`. Here α is the probability of making a type I error by rejecting the hypothesis that our sample statistic is based on the given distribution. Intuitively the "size" of the returned confidence interval is based on this value, smaller values of α give rise to larger confidence intervals.

Exercise 6.9 (the bootstrap confidence interval for the sample mean)

Using the bootstrap estimator we can compute confidence intervals for an statistic (function) we can compute using the set of n data points. This is very advantageous because it can be difficult or impossible to derive the theoretical probabilistic distribution of the given statistics. In the case of the *sample* mean we do know the theoretical probability distribution when the individual samples x_i are drawn from a normal distribution with population mean μ_0 and variance σ_0^2. In this case the sample mean of n points is also normally distributed with a mean μ_0 (the same as the population mean) and a variance equal to σ_0^2/n. Given an α (the probability of type I error), we can compute the theoretical quantile for the sample mean by computing the $\alpha/2$ and $1-\alpha/2$ quantiles of a normal with the above mean and variance. This is demonstrated in the Matlab script `prob_6_9.m`. With 1500 bootstrap replications the empirical bootstrap estimate of the confidence interval is given by $(18.64929, 18.95357)$ while the theoretical confidence interval on the sample mean when we have 140 samples (the number of samples in the `forearm` data set) is given by $(18.64638, 18.95791)$ the two are amazingly close.

Exercise 6.10 (Monte Carlo hypothesis tests on the difference of the means)

For this data since we are not given (or told) the probability distributions that generated it we have to do some exploratory data analysis to decide what a reasonable density would be.

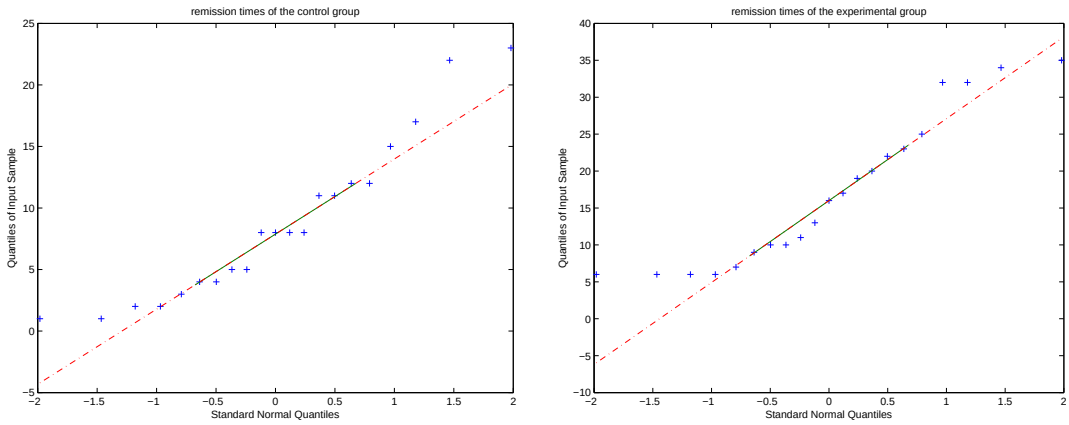

Figure 15: The QQ plots of the remission data in `remiss`. **Left:** The QQ plot for the control group. **Right:** the QQ plot for the experimental group. Notice that in both cases the data is approximately linear from which we conclude that a normal approximation is not a bad one to take.

We begin by plotting a histogram of the data in Figure 14. From that figure it looks like the experimental distribution has longer remission times (on average)

To set up a hypothesis test we need to determine probability densities for these two sets of data. We begin by assuming a normal distribution. In Figure 15 we present the QQ plots for both the control group and the experimental group. In both cases a Gaussian approximation seems reasonable.

Based on the results of earlier we will test the hypothesis that the average remission time of the experimental data is greater than that of the control group. Thus in the language of hypothesis testing we have that

$$\begin{aligned} H_0 &: \mu_E = \mu_C \\ H_1 &: \mu_E > \mu_C \,. \end{aligned}$$

If we define our statistic t to be the sample mean of the remission times for members of the experimental class and assume that we *know* the mean of the control group (estimated from the data this is $\mu_C = 8.66$) our hypothesis test becomes

$$\begin{aligned} H_0 &: \mu_E = 8.66 \\ H_1 &: \mu_E > 8.66 \,. \end{aligned}$$

The value of our initial statistic on the experimental remission times is given by $t_0 = 17.09$. Now to use the p-value method for hypothesis testing we need to determine the probability that under hypothesis H_0 we would get a value of our statistic this large or larger. That is we need to determine $P_{H_0}(T \geq t_0) = P_{H_0}(T \geq 17.09)$. We do this in the Matlab file `prob_6_10.m`. We can do this with the bootstrap method. We randomly draw 10^6 samples with replacement from the given data set `control`. We evaluate the mean of each bootstrapped sample and estimate the p-value by counting the number of times that the bootstrap sample had a value greater than or equal to the observed t_0. When we run this

code we find that from H_0 we *never* get a sample with mean value greater than or equal to t_0. Thus we take p-value to be zero and corresponding have a probability of making a type I error (rejecting H_0) of zero. We can safely conclude that the mean of the second population is greater than the first.

Exercise 6.11

These calculations are done in the Matlab file `prob_6_11.m`. In the case of the lsat scores we see that the sample estimates, with the bootstrap estimated standard error, generally contain of the population variance.

The same calculation for the gpa data does not seem to match as well in that the bootstrap estimated confidence interval of the variance does not aways include the true variance (it does however come very close).

Exercise 6.12

We will use as our statistic. the correlation coefficient between the lsat scores and the grade point averages (gpa). That function is implemented in the utility function `cross_corr_fn.m`. To test for statistical significance in the Matlab file `prob_6_12.m` we compute the $\alpha = 0.1$ (or 90 percent) confidence interval on the cross correlation between the lsat and gpa data. We select a sample of size fifteen and use the bootstrap with 1000 bootstrap replications to derive our confidence interval. If the lower bound of this confidence interval is greater than zero we can conclude that the sample cross correlation is statistically significant (to within making a type I error of 0.1).

Exercise 6.13

This is implemented in the Matlab file `prob_6_13.m`. Since the median is a non-smooth statistic from the discussion in the book we expect that the bootstrap to not perform well. How that manifests itself in this problem is that the location of the population parameter seems to fall on the *edges* of the confidence interval. In addition, the addition of more samples does not seem to decrease the size of or improve the confidence interval in a significant way. I would expect that as we take more and more samples our confidence interval would shrink smoothly surrounding the the population median. The confidence intervals do seem to shrink as we take more samples but in a non-smooth way.

Exercise 6.14

We state that since the data for this exercise is waiting times, we expect that it is distributed with an exponential distribution. Using the Matlab tool `dfittool`, we see that an exponential distribution fits quite well with a mean value of 437.21. To get an appropriate confidence interval using the Matlab function created for this chapter `bootstrp_sci.m` we generate a 90% confidence interval of $(358.62097, 530.89516)$. These calculations are done in the Matlab code `prob_6_14.m`.

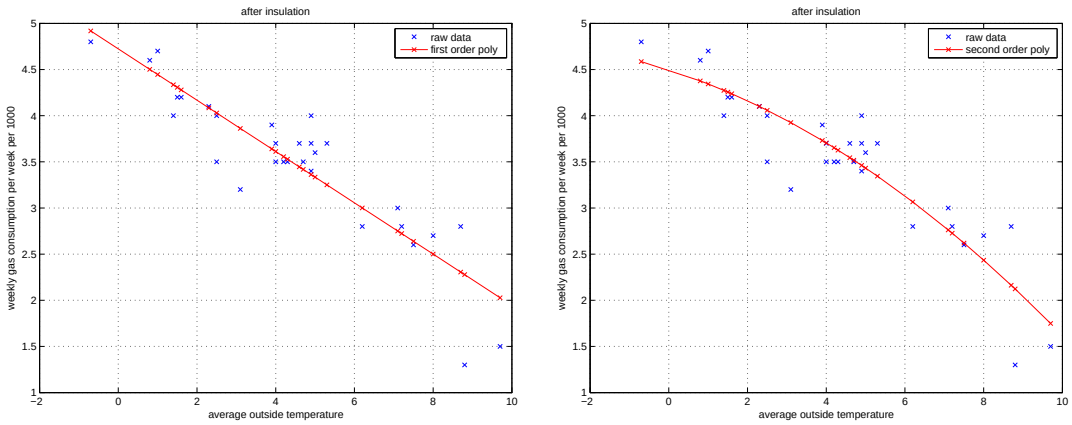

Figure 16: **Left:** A scatter plot of the raw data and the first order polynomial model. **Right:** The raw data and a second order polynomial model.

Chapter 7: Data Partitioning

Exercise Solutions

Exercise 7.1 and 7.2 (cross-validation)

See the Matlab file `prob_7_1_N_2.m` for this exercise. The data for the gas consumption before insulation is plotted in Figure 16. Visually this looks like the relationship between the two variables is linear. Performing cross-validation with $K = 1$ and with $K = 4$ we find that the estimated prediction error for the first order polynomial (a line) are given by

$$0.08394 \quad \text{and} \quad 0.09383\,.$$

When we run n fold cross-validation we obtain estimated prediction errors for three polynomial models (first, second, and third order polynomials) of

$$0.08394, 0.08079, 0.08567\,.$$

Thus the second order polynomial is slightly better at fitting this data than the first order polynomial (but not by much). When we run cross-validation with $K = 4$ the results are very similar. That the second order polynomial is not much different than the linear fit is also verified when we look at the coefficients found for a second order fit. They are given by (from highest degree to lowest degree)

$$-0.0098 - 0.31636.7682\,,$$

Where we see that the $O(x^2)$ term has a coefficient that is $O(10^{-2})$ the next smaller term. When the linear and quadratic fits are plotted on type of the raw data you can see that the two curves are almost indistinguishable.

When the data on the gas usage after insulation is plotted much of the above analysis is the same. The main difference is that with hold one out cross validation the estimated prediction

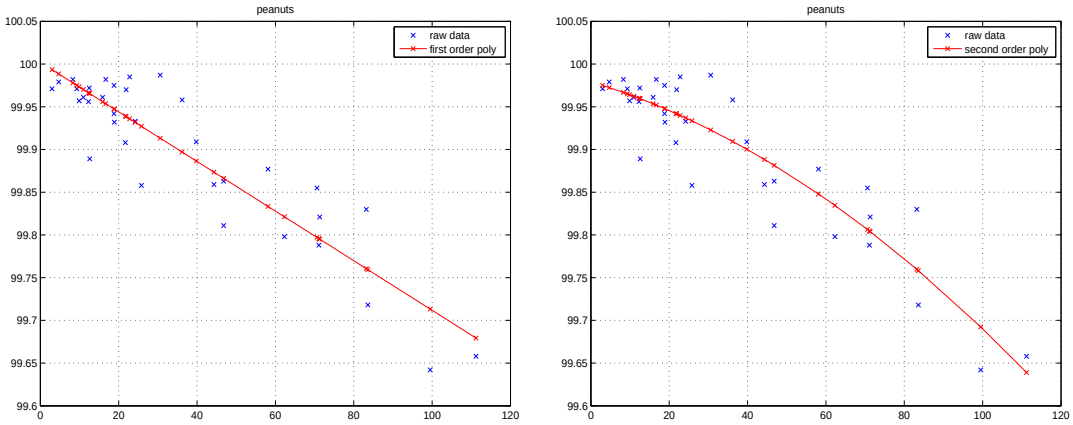

Figure 17: **Left:** A scatter plot of the raw data and the first order polynomial model for the peanuts data set. **Right:** The raw data and a second order polynomial model.

errors for the three polynomial models (first, second, and third order polynomials) are given by

$$0.14137, 0.13875, 0.13769\,,$$

where each polynomial model seems about the same as far as preditive accuracy goes. The results for $K = 4$ are different. For $K = 4$ gives estimated prediction errors for our three polynomials are given by

$$0.12497, 0.48132, 0.46143\,.$$

Here the fact that the magintude of the linear predictor is so much smaller than the other ones suggests that on smaller data sets the linear model is better than the other two. From this fact (and the assumption that we are working in a case where we have less data than we would like) it seems reasonable to conclude that the linear model is the best one.

Exercise 7.3 (modeling the amount of alfatoxin in peanuts)

See the Matlab file `prob_7_3.m` for this exercise. When we perform leave one out cross validated error for polynomial models of degree one, two, and three we find estimated errors of

$$0.00168, 0.00159, 0.00184\,,$$

while the estimated error with $K = 4$ partitions of the data (we have roughly 8 data points to test with) gives estimated model errors of

$$0.00159, 0.00135, 0.00198\,.$$

Both sets of estimated errors indicate that we should take the second order polynomial for the best fitting model. In all cases the model discrepency is not that great. Both models are plotted in Figure 17

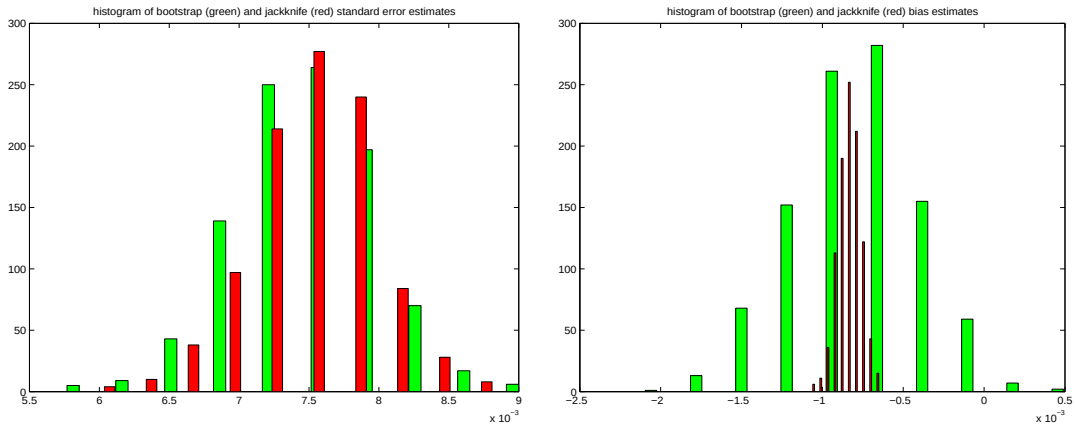

Figure 18: A comparison between the bootstrap and jackknifed estimated standard confidence intervals **Left** and bias **Right** in estimating the second moment. This histograms show that the distribution of standard confidence intervals generated by the two procedures is quite similar. The distribution of the bias is different in that it appears that the the bootstrap has a much wider distribution of possible values while the jackknife produces much smaller biases.

Exercise 7.4 (normal data)

When samples are drawn from a normal distribution with a standard deviation σ the standard error of the sample mean (assuming n samples) is given by

$$\frac{\sigma}{\sqrt{n}},$$

when we draw our samples from the standard normal $\sigma = 1$ and the above simplifies to $\frac{1}{\sqrt{n}}$. This exercise is performed in the Matlab file `prob_7_4.m`. In general the bootstraped and the jacknifed estimate of the standard error are very close to the theretical estimate.

Exercise 7.5 (uniform data)

The procedures for computing the bootstrap and the jackknife estimates of the standard error for the sample mean don't change when the distribution changes. This is a very powerful benifit of using parameter free methods to derive confidence bounds on our desired statistics. For the standard uniform distribution (with range $[0, 1]$) the variance is given by

$$\sigma^2 = \frac{1}{12}.$$

As suggested, we use $\sigma/sqrtn = \frac{1}{\sqrt{12n}}$ as an estimate of the standard error for the sample mean for this problem. When the Matlab file `prob_7_5.m`, is run we see that both the bootstraped and jackknifed estimate of the standard error are quite close to this value.

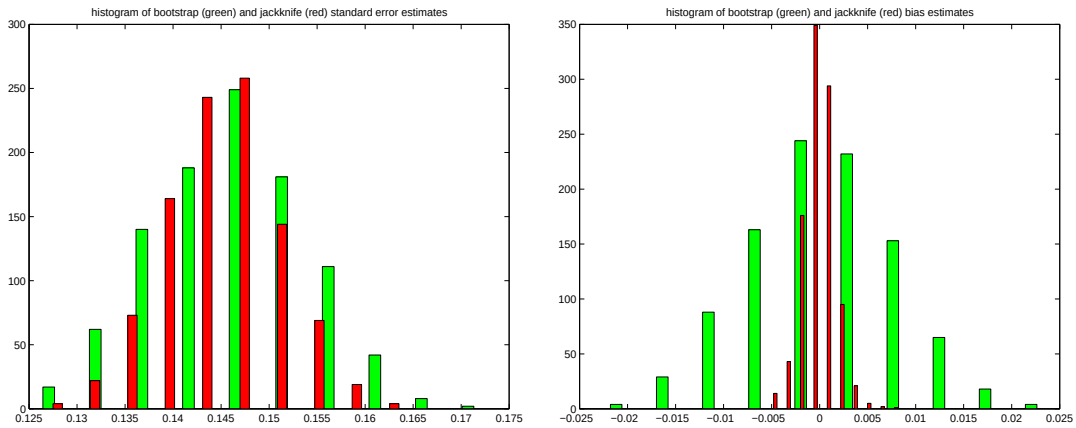

Figure 19: A comparison between the bootstrap and jackknifed estimated distributions of the standard confidence intervals **Left** and bias **Right** in estimating the skewness. These histograms show that the distribution of the standard confidence interval generated by the two procedures is quite similar. The distribution of the bias is different in that the bootstrap appears to have a much wider distribution of possible values than the jackknife produces.

Exercise 7.6 (a MC study of estimating the standard error and bias)

When we run the code in the Matlab file `prob_7_6.m` we see two things. The first is that the distribution of the standard errors produced by the bootstrap procedure and the jackknife procedure are very similar. The histogram in Figure 18 **Left** shows the distribution of the standard confidence interval of the second moment statistics $\text{mom} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$, where $\bar{x}$ is the sample mean for both the bootstrap and jackknife procedures. In Figure 18 **Right** we show the distribution of the bias of the second moment when computed using the bootstrap and the jackknife procedure.

Exercise 7.7 (a MC study of estimating the sample coefficient of kurtosis)

The Matlab code `prob_7_7.m` compares the distributions of the standard error on the sample estimate of *skewness* when using the bootstrap and the jackknife procedures. In Figure 19 **Left** we plot the two distributions of the standard error. From that we can see that the distribution of standard errors for both procedures are very similar. In Figure 19 **Right** we plot a histogram of the distributions of bias for both procedures. Again we see that the jackknife is much more tightly distributed about the origin giving the conclusion that the jackknife procedure is less biased that the bootstrap procedure in computing the skewness.

The histograms in Figure 20 **Left** shows the distribution of the standard confidence interval on the kurtosis statistic for both the bootstrap and jackknife procedures. We see that again both procedures generate very similar distributions. In Figure 20 **Right** we show the distribution of the possible bias of the kurtosis statistic computed using the bootstrap and the jackknife procedure.

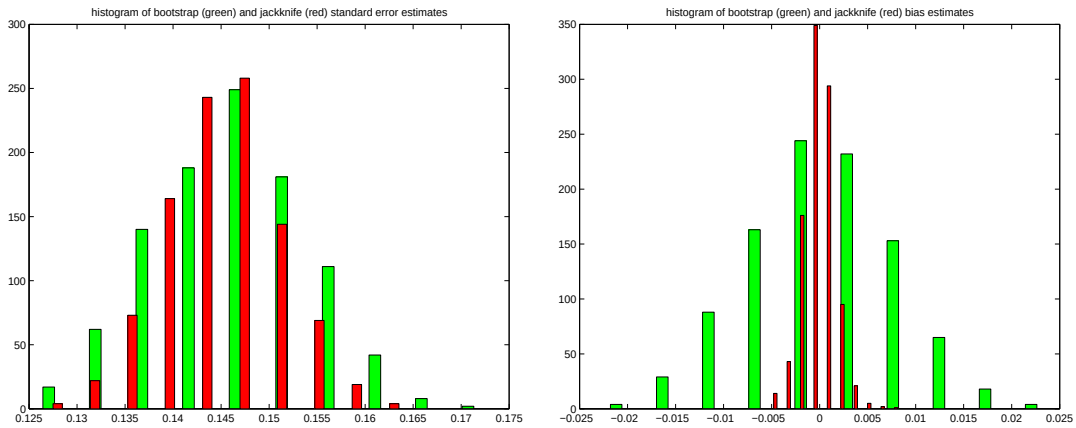

Figure 20: A comparison between the bootstrap and jackknifed estimated standard confidence intervals **Left** and bias **Right** in estimating the sample kurtosis. In the left figure, the histograms show that the distribution of the standard confidence interval generated by the two procedures is quite similar. In the right figure the distribution of the bias from the bootstrap has a much wider distribution of possible values than the jackknife procedure.

Exercise 7.8 (jackknife replicates of the median)

This exercise is implemented in `prob_7_8.m`. When we compute the jackknifed replicates of the median of the `lsat` data (a component of the entire `law` dataset) we have only *two* unique values that of 587 and 598. The jackknife estimate of the standard error is 4.5 with an estimated bias of zero.

When we compute the jackknifed replicates of the median of the `gpa` data we have only *one* unique value of 3.15 giving zeros for the jackknifed estimate of the standard error and bias. Obvisouly with so few unique samples in the jackknifed estimtes will be very poor.

Exercise 7.9 (estimates of the standard error)

This exercise is done in the Matlab file `prob_7_9.m`. Both the bootstrap and the jackknife give similar results for the standard error. The bootstrap estimate (with $B = 400$ bootstrap replicas) of the bias is -0.5 while the corresponding jackknife estimate is 0.0. Increasing the number of bootstrap replicas causes the the bias estimated by the bootstrap procedure to decrease.

Exercise 7.10 (the bootstrap-t interval)

This exercise is done in the Matlab file `prob_7_10.m`, where we will use three functions from the computational statistics toolbox. The three functions are `csbootint.m` to compute the bootstrap-t interval, `csbooperint.m` to compute the bootstrap percentile interval, and

`csbootbca.m` to compute the BC_a interval. When this code is run we find that the

Chapter 8: Probability Density Estimation

Exercise Solutions

Exercise 8.3 (Monte Carlo estimation of the MSE for the histogram)

I found the definition of the MSE of an estimator given in the book a bit confusing. The MSE of a function estimator $\hat{f}$ is defined as

$$\text{MSE}[\hat{f}(x)] = E[(\hat{f}(x) - f(x))^2] \,.$$

What is confusing about this is what the expectation in the above expression is to be taken with respect to. One can determine what this expectation should be taken with respect to with some simple reasoning. The idea is that $\hat{f}$ is a function that is determined from a *specific* sample of data and perhaps parameters of the method. The important point is that it is *different* for every data sample (set of n points). Thus the expectation should be taken with respect to all samples (of a given size) we could obtain from our given population. In the case of this problem the function $\hat{f}$ is an estimate of the true underlying population probability density obtained from a frequency histogram (appropriately normalized) with a fixed bin width h.[2]

To perform a Monte-Carlo simulation we repeatedly draw $n = 100$ univariate random normals, construct a histogram based probability density function approximation from them i.e. construct $\hat{f}$, and compute $(\hat{f}(x) - f(x))^2$. For this exercise we can do this since we know the specific form of the density function $f(\cdot)$. Since it is a univariate Gaussian we have that

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \,.$$

For the first part of this problem we hold the the number of points n *constant* and compute a Monte-Carlo estimate of the MSE error (as a function of x) for two values of h one "small" and one "large". As an example of what we expect to obtain consider if our h is taken too "small". In that case we expect to not have many samples in each bin (one can imagine an extreme case where there is only one point) in each bin so that the PDF estimate of ν_k/nh would be very large and greatly over estimate the true value of the PDF. If we have h too "large" then the coarseness of the discretization itself will result in a poor estimator for $\hat{f}$ by taking our approximation constant over large regions. To perform an experiment demonstrating this effect, we compute 10 Monte-Carlo estimates of the PDF computed with a fixed value of h and then take the mean and standard deviation of all PDF estimates. We plot the mean PDF estimate and the one standard deviation "confidence interval" of this PDF estimate. In addition, for each of the Monte-Carlo estimated PDF's we can compute the MSE error function $(\hat{f}(x) - f(x))^2$ as a function of x. We can average all the Monte-Carlo MSE estimates to get a total average estimate as a function of x. In Figure **??** we present

[2] One could denote the dependence of $\hat{f}$ on the parameters used in determining it by writing $\hat{f}(x) = \hat{f}(\{x_i\}, n, h)(x)$, to denote that the functional estimate we obtain depends on such things like the specific samples x_i, the number of samples n, and the chosen bin width h.

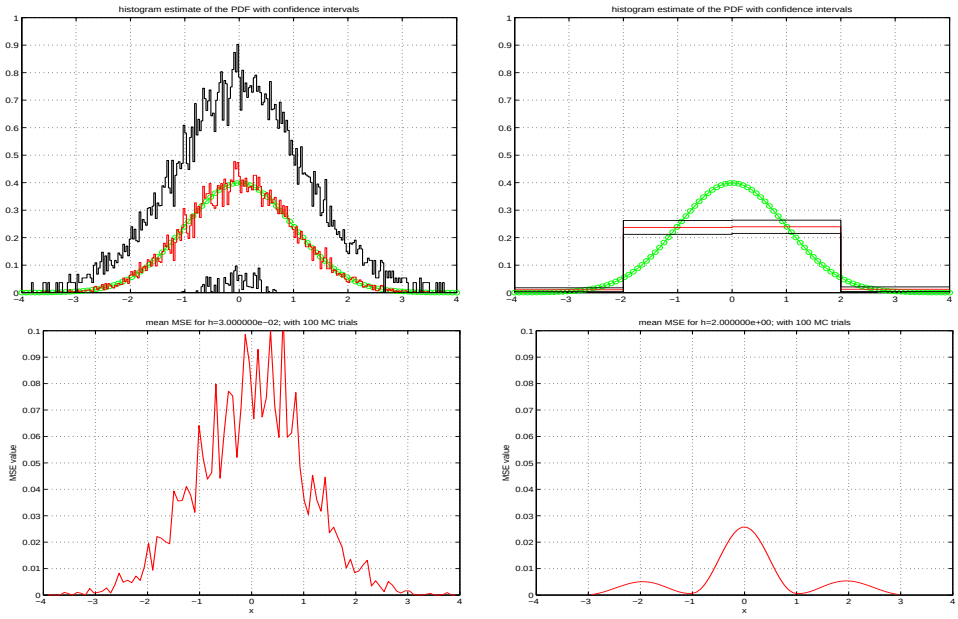

Figure 21: **Top:** Plots of the average histogram PDF. The true PDF for our standard normal is plotted in green on both. **Bottom:** Plot of the Monte-Carlo estimated MSE as a function of x.

four plots of this type for two values of h. The first row of plots contains Monte-Carlo based histogram based PDF estimates (and confidence intervals) for $h = 0.003$ and $h = 2.0$. The bottom row contains the Monte-Carlo estimate of the corresponding MSE error as a function of x.

Some comments are in order. For small values of h we see that the mean PDF estimate is rather close to the true value (shown in green). The one standard deviation estimates are much wider however. When we use a small h our estimator $\hat{f}$ has a low bias but a high variance. The case of large h gives poor estimates of the true PDF but the variance of the estimates are much tighter. In this case the estimator $\hat{f}$ has a high bias but a low variance. These two effects are at the root of the bias-variance trade-off. We want to pick an approximate $\hat{f}$ has that has a low bias which in this case taking h small but not such a small h that we end up with a large variance that the approximation $\hat{f}$ is tailored specifically to the observed data.

Figure 21 also show the average MSE as a function x for both approximations. In these two cases the two MSE are of the same order of magnitude even though the h values are rather different. Note that the MSE error is largest in the center of the distribution (nearer the mean) and smaller in the tails of the distribution.

If the value of h is too small for a given number of data points then increasing the number of points will improve the PDF estimate since with more data there is a better chance that each bin will have more samples in it. With an h that is too large increasing the number of samples will not help. The problem is that for a given h one does not necessary know the number of data points required to get the optimal estimate. The optimal h is dependent on

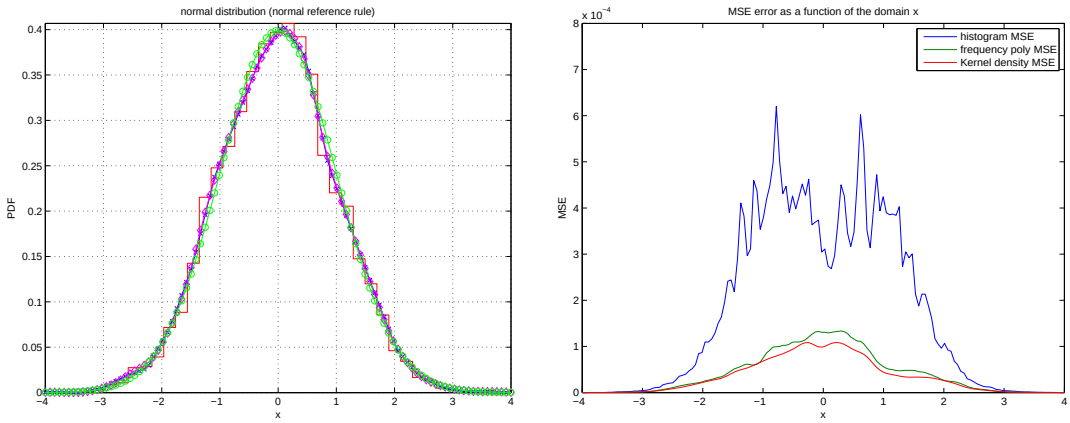

Figure 22: **Left:** Plots of three PDF estimates of a Gaussian. The true density is plotted in green. **Right:** Plots of Monte-Carlo estimates of the MSE as a function of x for each of the estimation techniques.

the number of samples simply modifying n or h independently is generally not optimal.

Changing the error function to the absolute error $|\hat{f}(x) - f(x)|$ changes the magnitudes but does not change any of the conclusions above. These experiments are performed in the Matlab script `prob_8_3.m`.

Exercise 8.4 (using the normal references rules Part I)

For this problem we draw samples from a normal distribution and use the suggested normal reference rules to specify the bin widths for estimating the probability density estimate PDF using a histogram, a frequency polygon, and a kernel density estimate. In Figure 22 **Left** we present overlays of the three density estimate compared to the true Gaussian density estimate.

Plots of an Monte-Carlo estimate of the mean square error for each of the three methods as a function of the spatial domain are plotted in Figure 22 **Right**. From this plot it appears that the kernel density estimate has a smaller mean square error as a function of x. If we integrate across the domain in x to compute the *mean* square error we find that The errors for the histogram, frequency polygon, and the kernel density estimate are given by

```
0.01676, 0.00707, 0.00543,
```

respectively. This again shows that the kernel density estimate has a smaller error on average. These calculations are performed in the Matlab file `prob_8_4.m` and `norm_ref_rule_hist.m`.

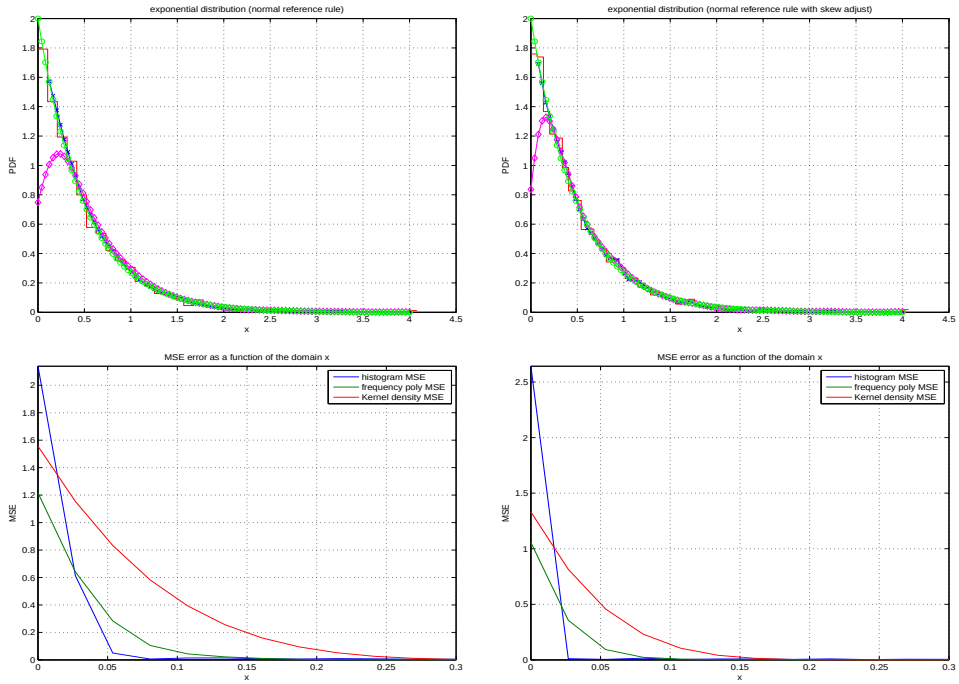

Figure 23: **Top Left:** Plots of three PDF estimates of a exponential distribution with a mean of one-half. For each PDF estimate we select the bin width h using the normal reference rule (without and skewness factor). **Top Right:** Plots of three PDF estimates of a exponential distribution with a mean of one-half. For each PDF estimate we select the bin width h using the normal reference rule **with** the skewness factor. The true density is plotted in green in both cases. **Bottom Left:** Plots of Monte-Carlo estimates of the MSE as a function of x for each of the estimation techniques with the normal reference rule. **Bottom Right:** Plots of Monte-Carlo estimates of the MSE as a function of x for each of the estimation techniques using the normal reference rule and a skewness factor.

Exercise 8.5 (using the normal references rules Part II)

For this problem we draw our samples from a exponential distribution with a mean taken at 1/2. Since this distribution is skewed and we expect to show that when using the normal reference rules naively (that is without introducing a skewness factor to the bin size) is *not* optimal when compared using a skewness factor to modify the normal reference rule for bin sizes. To do this we repeat the exercise in 8.4 above with the source of data given by an exponential distribution. These results are presented in Figure 23.

We wrote two versions of code to test these ideas. The first version `prob_8_5_no_skew.m` generates data from an exponential distribution and approximates the probability density function using the normal reference rule. The second version `prob_8_5_with_skew.m` does the same but apply a skewness correction factor as appropriate given that the data is highly skewed. In all cases the skewness factor is less than one and decreases the bin size relative to the normal reference rule. Under infinite data, all things being equal the decreased bins size should provide better approximations to the true probability distribution. Because of this fact perhaps the results below should be scaled to represent this fact. In both cases

we compute our density estimates using 5000 data points[3] and initialize the Matlab random seeds so that the *same* random samples will be processed by both algorithms. The three probability density estimates compared are histograms, frequency polygons, and a kernel density estimate for the specified value of h in both cases. The top row in Figure 23 shows the densities in both cases compared with the truth. It has to be noted that the kernel density estimate in both cases is significantly worse than the other two estimates near $x = 0$ due to the discontinuous nature of the exponential distribution there (it has a finite value $x = +\epsilon$ and is zero for $x = -\epsilon$). This feature makes it very difficult for the kernel density estimates to fit to the true density. This affect can be seen in both plots.

We next perform an Monte-Carlo estimate of the MISE $E\left[\int(\hat{f}(x) - f(x))^2 dx\right]$ for each of the density estimate techniques. For 100 Monte-Carlo random draws of data from an exponential distribution I computed three density estimates (using the above techniques), I then computed the MSE function $(\hat{f}(x) - f(x))^2$ using these density estimates. To get a fair representation of the functional value of this MSE function in interpolated it to a fixed grid and finally integrated to complete the square error calculation. The average of each of our 100 Monte-Carlo estimates is then taken.

We find that integrated mean square errors for each of the three density methods is given by

```
hist=   1.33060; freqPol=   0.04667; kernel=   0.11766
hist=   2.08791; freqPol=   0.02783; kernel=   0.06314
```

From these numbers we see the general trend that the skewness adjustment produces probability density estimates that are better than not using such an adjustment. In addition, we see the difficulty that the kernel density methods have at approximating discontinuous densities in compared with the frequency polygon approach. We also see that both the frequency polygon and the kernel density estimates outperform the pure histogram approach. Something we saw in the earlier problem. Finally, we present some plots of the Monte-Carlo average mean square error $(\hat{f}(x) - f(x))^2$ as a function of x plotted in the lower half of Figure 23.

Note that in producing these plots we used the Matlab routines `norm_ref_rule_hist.m` and `norm_ref_rule_w_skew_hist.m` to derive histogram estimate of the given input data.

Exercise 8.6 (different bin widths for histograms)

This exercise is done in `prob_8_6.m` where we load the `snowfall` data set is loaded and histograms are computed for a variety of bin widths h. From the nature of the data (derived from a natural phenomena) we expect the data to be Gaussian. Applying Sturges rule to this data gives a bin width of 14.8 while using the Freedman-Diaconis rule gives an optimal bin width of 17.0. When we plot this data for various bin widths we see that when the bin

[3] in both cases increasing the number of samples results in an improved density estimate of the true density

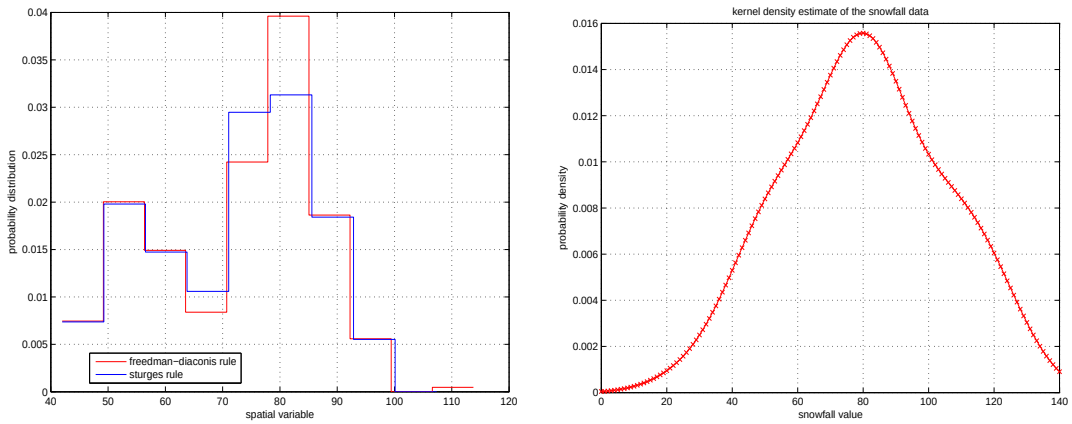

Figure 24: **Left:** Plots two version of the histogram approximation to the density if the `geyser` data. The first method uses the Freedman-Draconis rule to select the histogram width. The second method uses Sturges' rule to specify the number of bins (equivalently the bin width). On this data set the two results are very similar. **Right:** The kernel density estimate of the probability density function for the `snowfall` data set.

width is small we have very "spiky" looking histogram (an estimator with a low bias but high variance) while when we pick large values for our histogram bin width we obtain a very smooth estimate (an estimator with a high bias but with low variance). These larger values of h (near 15) do indeed produce histograms that look like a Gaussian with a large variance.

Exercise 8.7 (bin widths from the Freedman-Diaconis rule)

We implement this exercise in the Matlab file `prob_8_7.m`. When this script is run the plot is produces is placed in Figure 24 (left). We see that for this dataset both rules, the Freedman-Draconis rule and Sturges' rule both produce very similar looking histograms. The bin widths produced by the two methods are quite similar, with the Freedman-Draconis rule giving $h_{\text{FD}} = 7.17$, and Sturges' rule giving $h_{\text{S}} = 7.26$.

Exercise 8.10 (a frequency polygon of the iris data)

This exercise is worked in the Matlab file `prob_8_10.m`. When it is run we use the computational statistics toolbox function `csfreqpoly.m` to compute the frequency polygon for each pairing of the input features. From this we see that some features have multi-modal distributions. For example the third and fourth features are multi-modal.

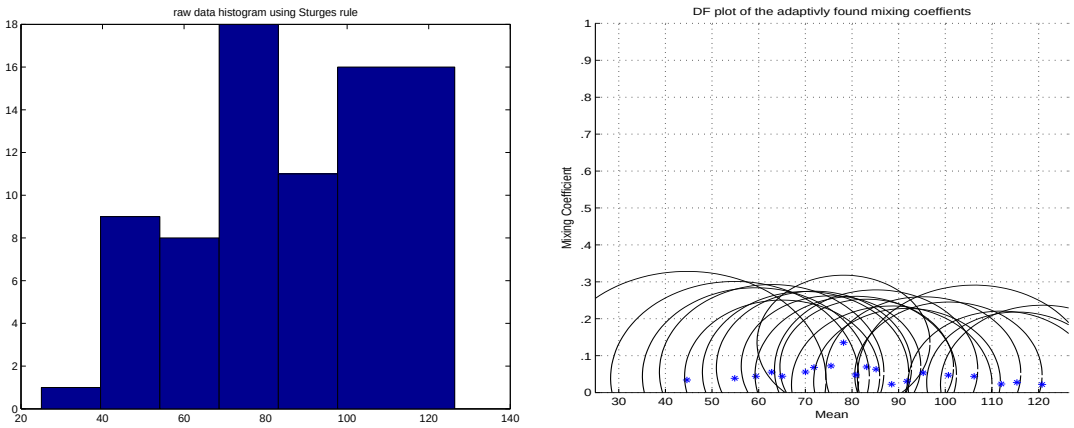

Figure 25: **Left:** Plots of the histogram of the data in the `snowfall` dataset. **Right:** A dF plot of the mixture components.

Exercise 8.11 (evaluating a kernel density estimate)

We present the code to perform the evaluation of a kernel density by looping over the evaluation points (rather than looping over the number of kernels) in the Matlab code `prob_8_11.m`. When this code is run it produces a kernel density estimate for the `snowfall` data set that looks like that shown in Figure **??** (right).

Exercise 8.14 (adaptive density modeling)

In the Matlab script `prob_8_14.m` we implement the required adaptive mixture density modeling, followed by a refinement of these parameters using the expectation maximization (EM) rule. Some discussion about the adaptive mixture density modeling procedure. When we run the above code we notice that with the adaptive component algorithm presented in the book, the *order* in which the data is presented to the algorithm matters and influence the results one obtains. Thus there maybe an optimal ordering of the datum for this algorithm such that would result in the fewest number of clusters. In addition, there is only one pass through the data so that there is no way to reduce the number of clusters after they are created. These properties are well seen in this example. In Figure 25 **Left** we plot a histogram (using Sturges' rule) from which we see that we don't expect a large number of mixture components. We find however that when we run the cluster algorithm provided in the book we get $N = 20$ mixture components. A dF plot of the mixture components is presented in Figure 25 **Right**. From that plot we see a great number of mixture components have means that are very close together and we would expect that a more compact clustering could be found.

In the second half of the above code we implement the expectation maximization procedure using the number of clusters found earlier. Running the code as presented in the book we found that the EM algorithm would evolve some of the mixture components towards zero. To mitigate this effect we remove the mixture coefficients if they fall below a given threshold

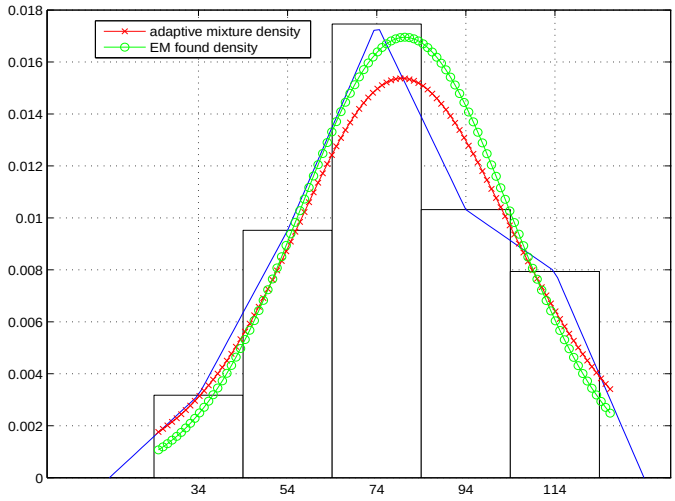

Figure 26: A comparison of three density estimation techniques. The first is a *direct* application of the adaptive mixtures procedure. The second is the output of the adaptive mixtures but with coefficients refined using the EM algorithm. The third is a frequency polygon.

with the following Matlab code

```
% check for mixture coefficients that shrink to zero (and remove them):
%
inds = find(mix_cof<=1e-9);
if( any(inds) )
  fprintf('removing at least the %d mixture coefficient ... \n',inds(1));
  mix_cof(inds) = [];
  mu(inds)      = [];
  var_mat(inds) = [];
  c             = c-length(inds);
end
```

With this code in-place the EM algorithm ended up removing all but *one* mixture component that had all of the weight showing that indeed a single Gaussian may not be a bad model for this data. In Figure 26 we plot the original mixture density (with twenty) components, the single Gaussian (found using the EM algorithm), and a frequency polygon. Both the single Gaussian and the mixture model do a good job representing the density.

Exercise 8.15 (density estimate of the `geyser` data)

This exercise is implemented in the Matlab script `prob_8_15.m`. When this script is run it estimates the mixture density for the raw data, generates data from this density, and then recomputes another density estimate from these generated data. We can then compute the integrated square error between these two estimated PDF's. We find the integrated square

error to be $5.845e-05$, showing that the two estimated densities are quite close. In fact when we plot the two densities they visually appear very similar. This result improves as we take more samples.

Exercise 8.16 (finite mixtures viewed as a kernel estimate)

If the kernel density estimate is viewed as a finite mixture model the mixture components, means, and variances are given by (using the variables of the kernel density estimate)

$$p_i = \frac{1}{h} \quad \mu_i = x_i \quad \sigma_i^2 = h\,.$$

Exercise 8.17 (adaptive mixtures)

This exercise is implemented in the Matlab script `prob_8_17.m`. When this is run some interesting things are found. From the initial data set x, we compute a permuted dataset exactly as suggested in the book. When we run the adaptive mixture density estimation procedure (with a maximum of 25 density components) each iteration returns approximately seven modes. This is obviously greater than the number used to generate the data which was three. The two densities that result are shown in Figure **?? Left**. From this plot it is apparent that the density might have fewer number of components that found in the adaptive mixture components. In addition, we see the phenomena where the densities *parameters* can be different but the resulting densities are the very similar. The ISE between these two curves is negligible (the same holds for the IAE). If we directly use the EM algorithm (provided in the Matlab function `csfinmix`) with the results from the adaptive mixture algorithm as the initial guess (with its seven components) the resulting densities are shown in Figure **?? Right**. Remember that a *different* ordering of the input data was specified to each application of the EM algorithm. This highlights a potential difficulty of the EM algorithm to converge to a local minimum. Since the number of data points generated to fit to was relatively small at 100, and the EM algorithm was seeded with an incorrect number of mixture components each run of the EM algorithm generated a slightly different densities which in this case do not look like the densities that generated the data. In fact, you can see from the "spike" like behavior of the resulting density in Figure **??** that the EM algorithm is fitting mixture components to *specific* data points. The EM algorithm is finding a solution that it too specific to the particular data samples generated and is over fitting the data. This is the symptom of having too many explanations for the observed facts.

In a very general way we have 100 observed facts and a combination of 7 mixture components to explain them. This is too many, and has resulted in an overfit density. This problem can be mitigated in several ways. By *increasing* the number of *data points*, you will have less fear that applications of the EM iterations will produce a density that is overfit to the given data set. This can also be achieved by *decreasing* the *number of mixture components*.

If after viewing the resulting density from the adaptive mixture modeling we set the `maxterms`

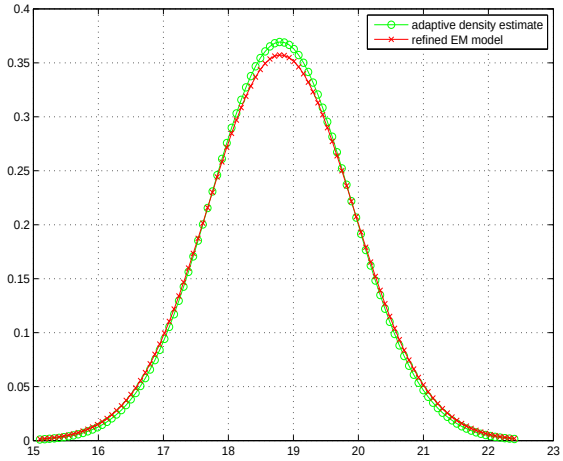

Figure 27: Density estimates for the `forearm` data.

variable at three. With this modification out initial condition to the EM algorithm consists of only three modes and the EM algorithm is able to find a more globally representative density.

Exercise 8.21 (the `forearm` data)

Many of the comments in Exercise 8.17 apply here as well in the specification of the parameter `maxterms`. When we specify value of `maxterms` at *one* (the expected number of modes) and refine the estimated mixture parameters using the EM algorithm we get the results plotted in Figure 27. See the Matlab file `prob_8_21.m` for the code.

Exercise 8.22 (the `elderly` data)

See the Matlab file `prob_8_22.m` for the code to perform this exercise. This is very similar to Exercise 8.21 above. The densities produced did not appear to have different modes at least not very significant ones.

Exercise 8.23 (the `nfl` data)

See the Matlab file `prob_8_23.m` for the code to perform this exercise. When we plot a scatter plot of the x_1 values against the x_2 values we see a strong linear dependence. The corresponding probability density function for the ordered pairs (x_1, x_2) has a strong diagonal direction as expected.

Chapter 9: Statistical Pattern Recognition

Exercise Solutions

Exercise 9.2 (clustering based on expenditures)

See the Matlab file `prob_9_2.m` for this exercise. We will apply two clustering methods to this data. The first will be agglomerative clustering and the second will be an application of k-means clustering. We begin by combining the individual measurements into a single data set. We have twenty measurements of men and twenty measurements of woman, thus if we consider the first set of measurements to be composed of men a "good" clustering would cluster the first twenty points together and the second twenty points together.

Since the measurements in the `household` data set correspond to things that are most likely not in the same units we should normalized the measurements before attempting to use them for classification. We can use the Matlab function `zscore` to perform this normalization. Without any other information, we will perform agglomerative clustering using the Matlab function `linkage` and the Euclidean distance as the similarity measure with single linkage. The Matlab function call used to compute the dendrogram

```
[H,T] = dendrogram(z);
```

Where `z` is the output from the `linkage` command. The dendrogram that results from this clustering is shown in Figure 28 (Left). The Matlab function dendrogram returns a number of "clusters" where each cluster can contain more than one point. For example the command

```
find(T==24)
```

returns that the 24th "cluster" contains all of the points $24, 28, 31, 33, 36, 38, 39$. Most "clusters" contain only one point however.

From this dendrogram we see that there are indeed clusters but maybe not the ones expected. It seems that the data clusters $3, 11, 5, 16, 20, 15, 10$ and 17 (each containing only one point) represent more unique people in that they don't cluster at the same distance as the others. These points excluded, there seem to be two main groupings. Except for the 24th "cluster" the first is the Matlab cluster points $9, 18, 1, 7, 12, 2, 4, 6, 8$ and 19 and the second is the points $21, 27, 25, 28, 24, 26, 13, 22, 23, 29$ and 30. Unfortunately, these discovered "clusters" don't seem to correspond strongly to two natural classification of men and women.

We next applied the k-means clustering procedure with random initial clusters and *two* clusters. In Figure 28 (Right) we plot the standardized first two features from our data set. We plot the true men with black squares, the clustered men with red squares, the

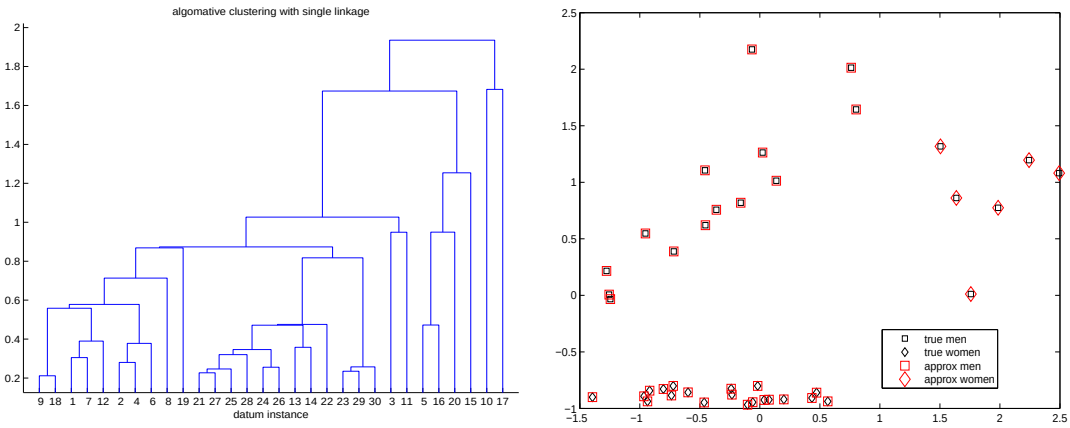

Figure 28: **Left:** The dendrogram produced via the Matlab command `dendrogram` (an agglomerative clustering procedure) when applied to the `household` data. **Right:** The corresponding clusters produced by the k-means clustering algorithm.

true women with black diamonds, and the approximate women with red squares. We see that the clustering produces clusters that while they maybe close in a Euclidean distance do not necessary correspond to the true class labels. This is the same results found with the agglomerative clustering algorithm above. These results might be understood by recognizing that even though we think that cluster men and woman should be an easy task, this may not be true when the features given are not biologically related.

Exercise 9.3 (a classification tree from the `household` dataset

See the Matlab file `prob_9_3.m` for this exercise. When this script is run with parameters that should result in a rather over fit tree `maxn=1`, we see that the resulting classification tree has only *two* leaves. These classification tree procedure choose to split along the second dimension with a split value (relative to the standardized) measurement of $s_2 = -0.42$. This split value is quite good in that when we then compare the classification accuracy of this one split tree we find it classifies with one-hundred percent accuracy! This might seem in contradiction to the results found in Exercise 9.2 where both clustering algorithms had difficulty extracting the expected clustering of men and women. This disparity of results can be attributed to a couple of factors. The most important is the recognition that these are indeed *different* algorithms. The tree classification procedure can easily perform splits (and effectively separate classes) along coordinate axis which is more difficult for the agglomerative and the k-means procedure to perform. Both of these two later procedure use the Euclidean distance as a similarity metric. This effectively treats all features as equivalent with depending on the problem may not be desired. The `household` data set is one where this property may not perform optimally.

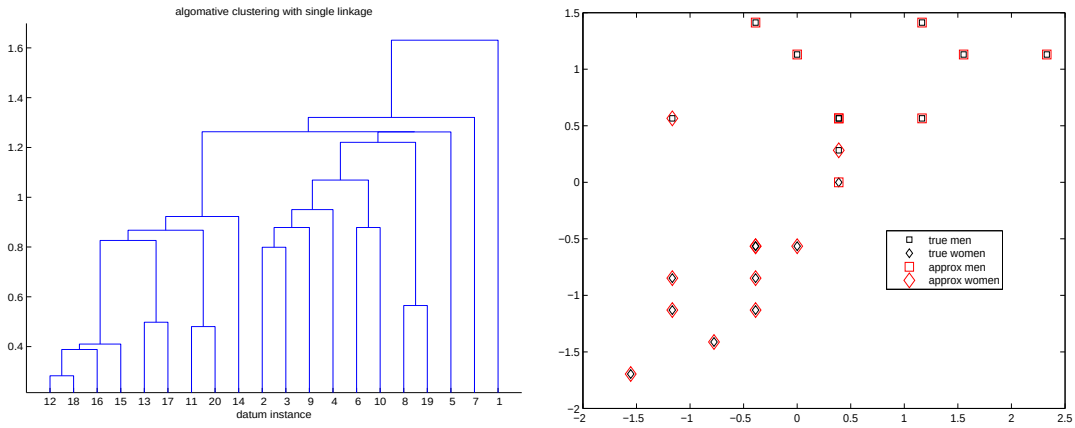

Figure 29: **Left:** The dendrogram produced via the Matlab command `dendrogram` (an agglomerative clustering procedure) when applied to the `measure` data. **Right:** The corresponding clusters produced by the k-means clustering algorithm.

Exercise 9.4 (clustering based on measurements)

See the Matlab file `prob_9_4.m` for this exercise. when we run the agglomerative clustering algorithm we get the dendrogram shown in Figure29 (Left). In this case each "cluster" contains only one point and see that at the highest level we have approximately 10 data points in each major grouping. We see that the data points 7 and 1 are different that the others in that they are the last data points brought into a cluster.

We next applied the k-means clustering procedure with random initial clusters and *two* clusters. In Figure 29 (Right) we plot the standardized first two features from our data set. As earlier, we plot the true men with black squares, the clustered men with red squares, the true women with black diamonds, and the approximate women with red squares. We see that in this case we see that the two clusters corresponds quite well with the physical labels of men and women. This is a good result in that it enforces our expectation of what we should find. In order to determine what Matlab used to label "men" we choose the label Matlab selected for the samples with the largest value of the third feature. For this data set that is the seventh sample.

Exercise 9.5 (the `cluster` and `inconsistent` Matlab functions)

See the Matlab code `prob_9_5.m` for the code to perform this exercise. We use the Matlab function `linkage` to generate the hierarchical cluster tree for the sample data set give in the text. We then use the Matlab function `cluster` to extract the clusters and plot each point in a color that depends on its cluster. These routines work much as one would imagine, clustering the first three points together and the last two points together.

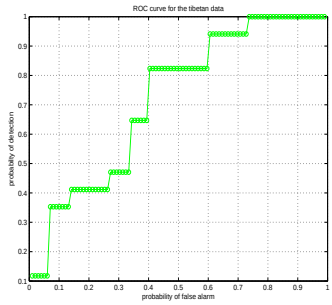

Figure 30: The ROC curve produced using the routine `csrocgen` on the `tibetan` data.

Exercise 9.6 (the `tibetan` data)

See the Matlab code `prob_9_6.m` for the code to perform this exercise. Rather than compute the ROC curve for this data set in detail[4], we will use the computational statistics toolbox function `csrocgen`.

To begin this exercise we recall that we are told that the first 17 data points correspond to the first class and the remaining 15 points correspond to the second class. From earlier exercises we know that distribution the individual features of the `tibetan` data is well approximated by a normal distribution. From this information we will build a Bayesian classifier using a normal density as an approximation to the class conditional density. This is exactly the density estimation technique used in the `csrocgen` procedure. When we run the code above we get the ROC curve shown in Figure 30.

Exercise 9.7 (comparing the performance of two classifier)

When we run the Matlab code `prob_9_7.m` it calls two subroutines. The first `prob_9_7_mdG.m` which uses hold one out cross-validation to estimate the probability of correct classification when we classify using a multidimensional Gaussian as an approximation to the probability density for each class. The second code `prob_9_7_prD.m` uses a product kernel to represent the class-conditional probabilities. Running this script we get an estimate of the probability of correct classification of 0.995 for the multidimensional Gaussian and a probability of correct classification of 0.895 for the product kernel. These results indicate that the multidimensional Gaussian is probably a better classifier.

Exercise 9.8 (ROC analysis on the `bank` data)

In this exercise we will study the ROC curve analysis on the `bank` data under two different classifiers. The Matlab code `prob_9_8.m` calls two subroutines. The first, `prob_9_8_mdG_ROC.m`,

[4]An example of performing this with two different feature estimation techniques is given in the Matlab codes associated with Exercise 9.8 **??**

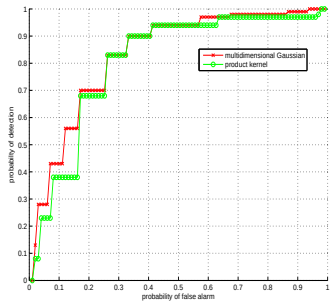

Figure 31: A comparison of the ROC curve produced when using a multidimensional Gaussian and a product kernel to estimate the density of the third feature of the `bank` data.

uses hold one out cross-validation to estimate the ROC curve for the `bank` data while the second `prob_9_8_prD_ROC.m`, uses a product kernel representation of the probability density in each class. In producing the ROC curve in both cases we need to perform the following steps:

- Specify a range for the probability of false alarm.
- Compute likelihood ratios (using leave-one-out cross-validation) for all of the non-target samples.
- For each of of the desired probabilities of false alarm p_{fa}, determine the threshold in likelihood ratio space that corresponds to this p_{fa} i.e. using appropriate quantiles of the likelihood ratios computed earlier.
- Using hold-one-out cross-validation compute the likelihood ratios for each member of the target class.
- For each of the thresholds computed above, compute the probability of detection p_d.
- Plot the probability of false alarm v.s. probability of detection.

All of these steps are implemented in the above Matlab files. When we execute these commands with *all* features we obtain a "perfect" ROC curve i.e. one that is able to maintain very large probability of correct classification while at the same time maintaining a very small probability of false alarm. To make the problem more difficult we will remove some features and recompute the ROC curve then. For example the *third* feature appears to be reasonable informative and will be used for comparison purposes here. When the Matlab scripts above are run we obtain the ROC curves shown in Figure 31. When we consider these results we see that the two classifiers are rather similar but the multidimensional Gaussian classifier has a higher probability of detection for smaller probabilities of false alarm. The multidimensional Gaussian has an area under the ROC curve of 0.81, while the product density has an area under the ROC curve of 0.79.

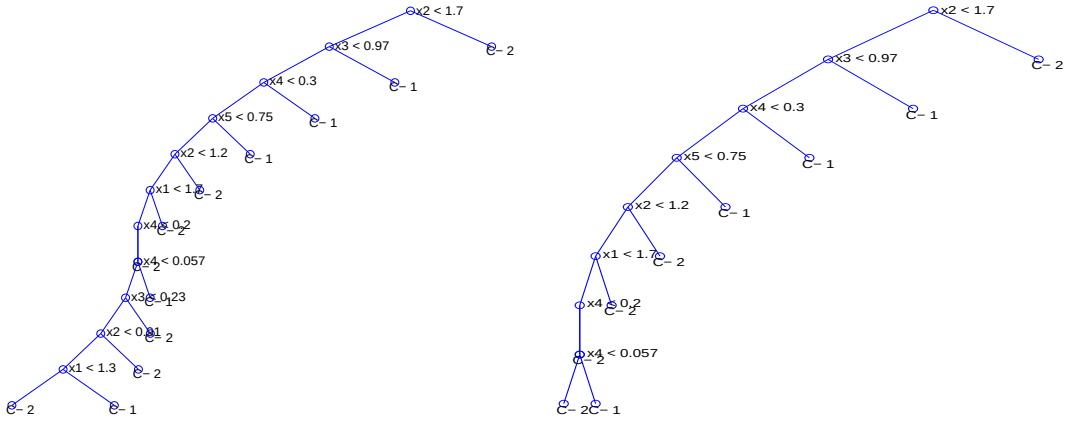

Figure 32: **Left:** The initial unpruned tree for the `bank` data and **Right:** The pruned tree that has the smallest classification error on an independent test set.

Exercise 9.9 (a classification tree for the `bank` data)

See the Matlab file `prob_9_9.m` for the code to perform this exercise. When develop a classification tree on the entire `bank` dataset we find that only one splitting (on the sixth feature) is required to optimally split the data. Since this produces a tree with only two branches we remove the sixth feature and obtain a classification tree that need considerably more splits to classify this data set. Randomly dividing the data into two sets (with 50 samples per class) the largest decision tree is shown in Figure 32 (left)[5]. We then prune this large tree using the Computational Statistics Toolbox function `csprunec`. Then using the remaining samples as an independent data set we classify each sample using each of the pruned trees above and record the probability of misclassification. We then select our optimal tree to be the one with the minimum classification error. This remaining tree is plotted if Figure 32 (right).

Exercise 9.10 (clustering the `bank` data)

In the Matlab script `prob_9_10.m` we first perform agglomerative clustering and then k-means clustering on the `bank` data. In Figure 33 **Left** we see the results from applying a single link agglomerative clustering algorithm to our data. One thing to note is that the Matlab `dendrogram` command can produce a large number of data points "under" a given leaf in the dendrogram. For example in this application the leaf 1 has 80 data points under it. The leaf 16 has 84 data points represented by its label. We can plot the number of points in each dendrogram cluster by issuing the Matlab command

```
figure; hist(T,length(unique(T)));
```

[5]Note that we have standardized the data using the Matlab function `zscore` before any computing the decision tree

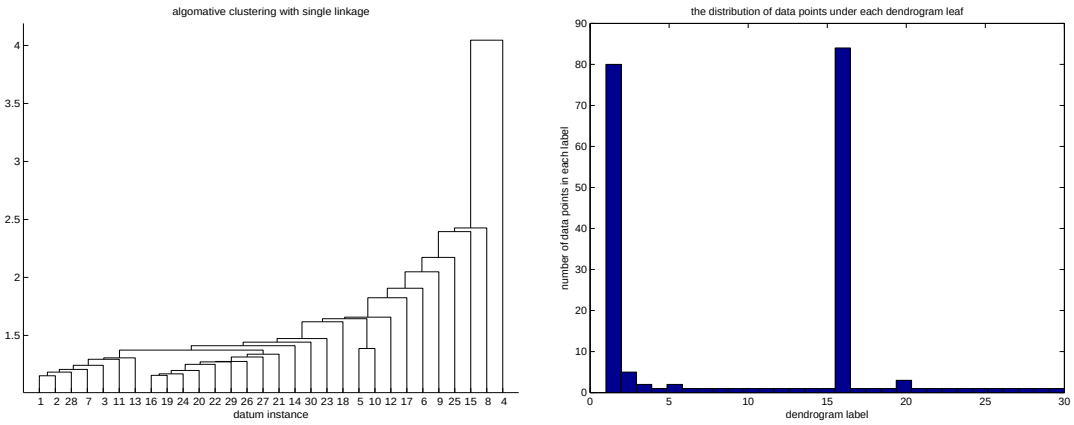

Figure 33: Results of hierarchal clustering on the `bank` data set.

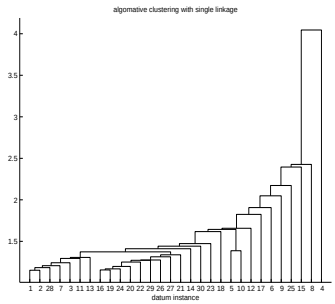

Figure 34: Results of k-means clustering on the `bank` data set.

This plot is plotted in Figure 33 **Right** where we see that there are two clear labels that contain a majority of the points and represent the two predominate clusters.

We next apply k-means clustering to the `bank` data. Since we know that there are two classes we will desire a clustering that contains only two classes. The plot of the resulting clusters is shown in Figure 34 along with the correct clustering using the known truth. When we compute how well our clustering does we see that the k-means method associates the correct class with a estimated probability of 0.96 which is quite good.

Exercise 9.11 (Monte Carlo study of the probability of error)

This exercise is performed in the Matlab file `prob_9_11.m`. There we generate a histogram of the probability of correct classification (the probability of misclassification would be one minus this result). The resulting histogram is plotted in Figure 35. From there we see that this probability is approximately normally distributed.

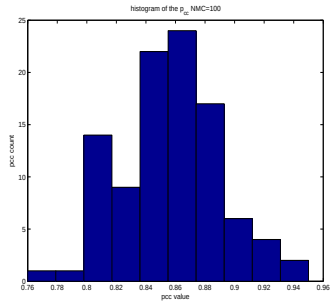

Figure 35: A histogram of the probability of correct classification for a number of Monte-Carlo runs.

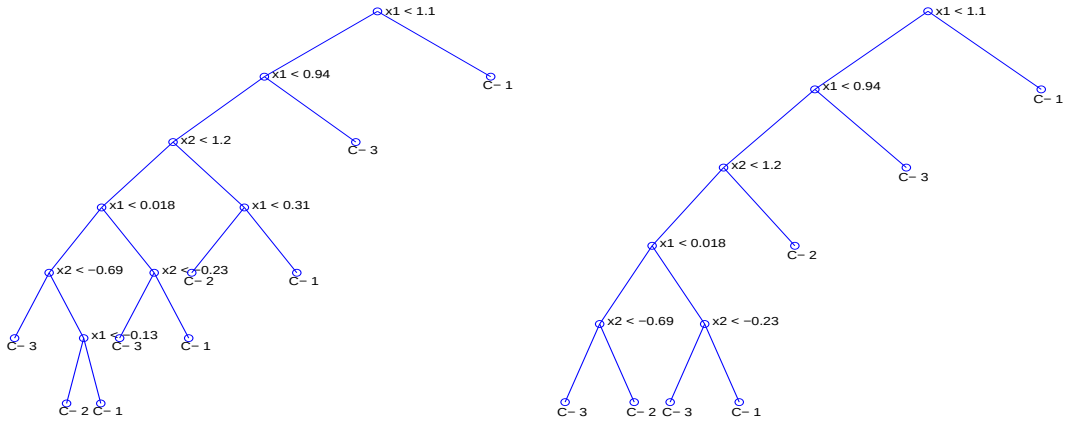

Figure 36: The initial (left) and final (right) classification trees for the `flea` data.

Exercise 9.13 (a classification tree the `flea` data)

In this problem we load `flea` data and then develop a large classification tree on this data. We then train a mixture model using all of the `flea` features with a call to the Computational Statistics Toolbox function `csfinmix`. Using this probabilistic model we can then generate additional data samples to prune our initial tree with. In many ways this is like the *parametric bootstrap*. We then use the "independent sample" data generated from our mixture model to calculate the classification error for each of the given pruned trees. We find that the largest tree has the minimum error. We then add to this error estimate the estimate of standard error for this tree. Finally, we pick the least complex tree that has a classification error below this value. For this example that turns out to be the *third* tree. In Figure 36 (left) we show the initial tree and in Figure 36 (right) we show the pruned find tree (corresponding to the third index).

The calculations for this problem can be found in the Matlab file `prob_9_13.m`.

Exercise 9.15 (classifying *versicolor* and *virginica*)

In this problem we are asked to classify the classes `versicolor` and `virginica` using *all* four of the features. When we classified these two species of iris using only the first two features we obtained a classification rate of 0.68. Using more features (all other things being equal) should improve our classification accuracy. When we run the Matlab code `prob_9_15.m` which uses cross-validation with $k = 1$ to estimate the probability of correct classification we see that this is indeed true. Using all four features we have an estimated probability of correct classification of 0.96.

Exercise 9.16 (classifying *virginica* and *setosa*)

In this problem we are asked to develop Bayesian classifiers and estimate the probability of misclassification using cross-validation on the `virginica` and the `setosa` species of iris. Since these species are more distinct we would expect that on only using two features we should be able to classify them relatively easily. This is implemented in the Matlab file `prob_9_16.m` where we see that with only two features (the same as used when attempting to classify `virginica` and `versicolor` that we obtain an estimated probability of classification of 0.99.

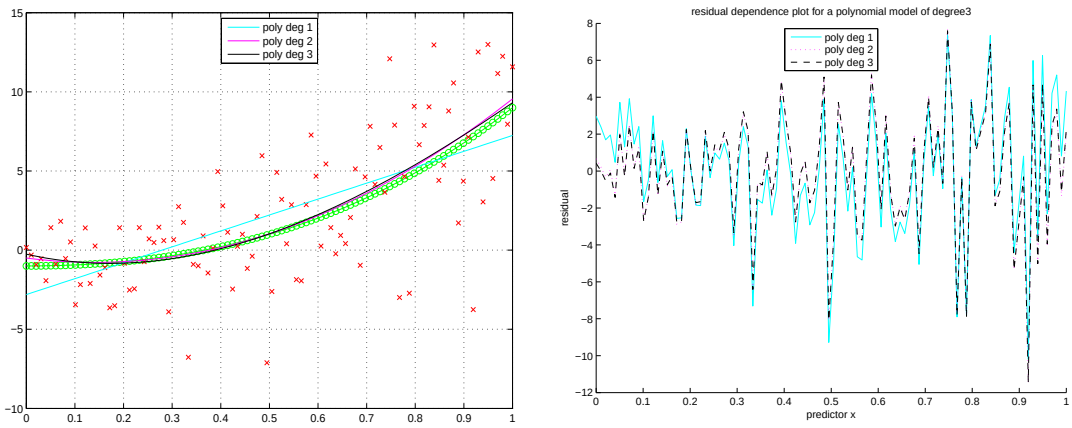

Figure 37: **Left:** The raw data and three polynomial models of degree one, two, and three aimed at approximating it. **Right:** The residual dependence plot for polynomial models of degree one, two and three. In this case the data is generated from a cubic polynomial with a variance that *varies* with respect to the x variable as $20x + 1$.

Chapter 10: Nonparametric Regression

Exercise Solutions

Exercise 10.1 (a non-constant variance)

See the Matlab file `prob_10_1.m` for this exercise. In this problem we have data generated from a polynomial model corrupted with noise whos variance is spatially dependent. Specifically, at a point x the noise's variance is given by $20x+1$. We attempt to fit three polynomial models to this data and observe the resulting residuals. In Figure 37 (left) we show the raw data (drawn with red x's), the true polynomial distribution (drawn with green circles), and the three polynomial models. We see that that the quadratic and cubic models appear very similar. In Figure 37 (right) we show the residual dependence plots for each of the three models. We see that the residuals of the quadratic and cubic polynomials are almost identical and the residual of the linear term is of a magnitude greater than the previous two. In addition, we see that as x increases the magnitude of the residual terms grows. This last observation is consistent with the fact that our noise variance grows as x increases. This would also indicate that the model of a constant variance is not appropriate for this data set.

Exercise 10.2 (a constant variance)

See the Matlab file `prob_10_2.m` for this exercise. Since we know that the true model for this data is generated from (a third degree polynomial) we expect that model to fit this data quite well. We attempt to fit polynomial models of degrees one, two, and three to the given

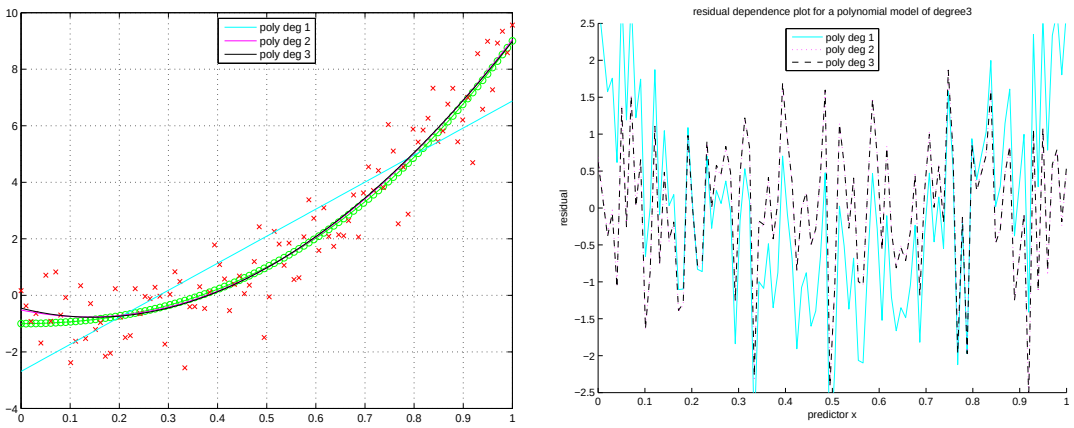

Figure 38: **Left:** The raw data and three polynomial models of degrees one, two and three aimed at approximating it. **Right:** The residual dependence plot for polynomial models of degree one, two and three. In this case the data is generated from a cubic polynomial with a variance that is *constant* with respect to the x variable.

noised data. In Figure 38 (left) we show the results for each of these three fits. The thing to notice is that the linear polynomial fits the worst while the quadratic and cubic polynomial fit approximately the same.

In Figure 38 (right) we show the results of the predictor residual plots for each of the polynomial models. In this figure we see that in general the residuals from the linear model are somewhat larger than the residuals from the quadratic and the cubic models. This implies that the the linear model may not be as good as the other two. We also see that the linear model has residuals that are larger in the middle (near $x = 0.5$) and on the extremes (where $x = 0$ and $x = 1$). These two areas correspond to exactly where the linear model diverges most from the curvature of the true cubic model. These results help emphasis that the quadratic and cubic models are better for this data set.

Exercise 10.3 (the distribution of the residuals)

See the Matlab file `prob_10_3.m` for this exercise. For this exercise we will use the same experimental setup found in Exercise 10.1. To better demonstrate the point of this exercise we will *increase* the number of locations where we sample our x axis on. Specifically we increase the number of points from 100 to 1000. This has the effect of increasing the number of samples we draw from our noise distribution and correspondingly allows us to draw more complete conclusions about the distribution of the residuals under each our models. When the above Matlab file is run, several plots are generated for each of the polynomial models. The first plot is a histogram of all residuals. These show longer tails than one would expect from normal distributions indicating that the variance may not be distributed normally. The second plots are qq-plots for each of the polynomial models and are shown in Figure 39. These plots show a divergence of the residual for each model with the normal distribution for large positive deviances from the mean. Again this gives an indication that the normal

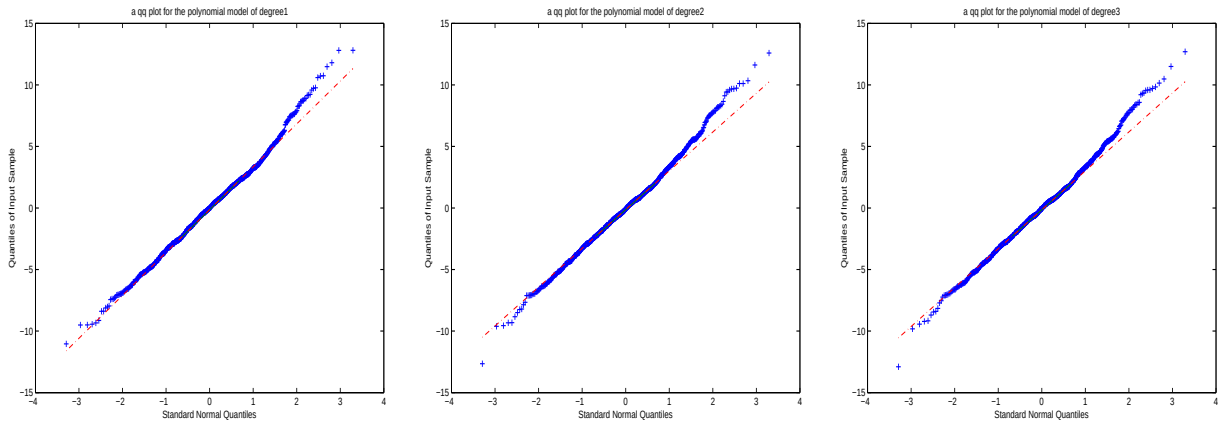

Figure 39: The qq-plots of the residuals for three polynomial models. The plots (from left to right), correspond to polynomial models of degrees one, two, and three. Notice the divergence from a normal model at the extreme positive values of the residuals.

distribution is not fully exact. Finally in Figure 40 we present boxplots of the residuals of the three polynomial models. Again the large number of red "cross" points above the box give an indication that a normal model is not fully appropriate.

Exercise 10.4 (residual smoothing)

See the Matlab file `prob_10_4.m` for this exercise. Following the residual smoothing procedure we first fit a preliminary model to the data, here a third degree polynomial. We then compute the residuals using $\epsilon_i = Y_i - \hat{Y}_i$, where the predicted variable $\hat{Y}_i$ is obtained from the polynomial model. Finally we perform loess smoothing on the residual and add this *correction* to the cubic polynomial model. This is similar to *stagewise additive refinement*, where we fit models to the remaining error (the residual) obtained from the additive sequence of models previously estimated. The addition (over the base polynomial model) can be observed in Figure 41

Exercise 10.5 (experiments with polynomial models)

See the Matlab file `prob_10_5.m` for this exercise. When we run this code we produce plots of the raw data and the associated polynomial models along with plots of the residuals associated with each model. In general the residuals *decrease* as the order of the polynomial increases this is true mainly because we evaluating the residuals at the *same* points where we are computing the polynomial approximations. As higher order polynomial can be constructed to pass exactly through more points it is not surprising that the residuals decrease. We should plot the residual function at point that were not used in the calculation of the model polynomial. We then would expect to see that very higher order polynomials provides a worse fit than lower ordered ones. Even taking this into consideration it appears the tenth

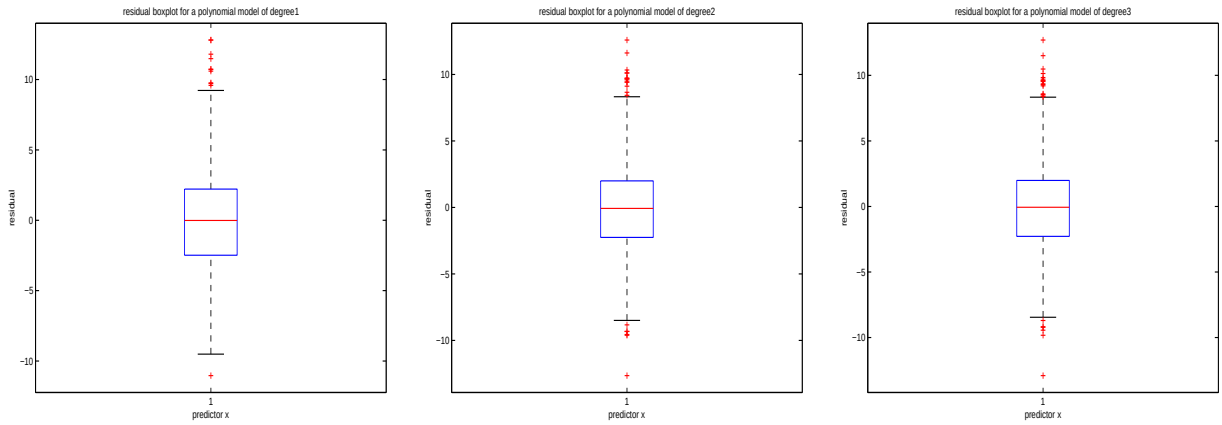

Figure 40: Boxplots of the residuals for three polynomial models. The plots (from left to right), correspond to polynomial models of degrees one, two, and three. Again notice that we actually obtain more residuals with a large value than would be predicted by a normal model.

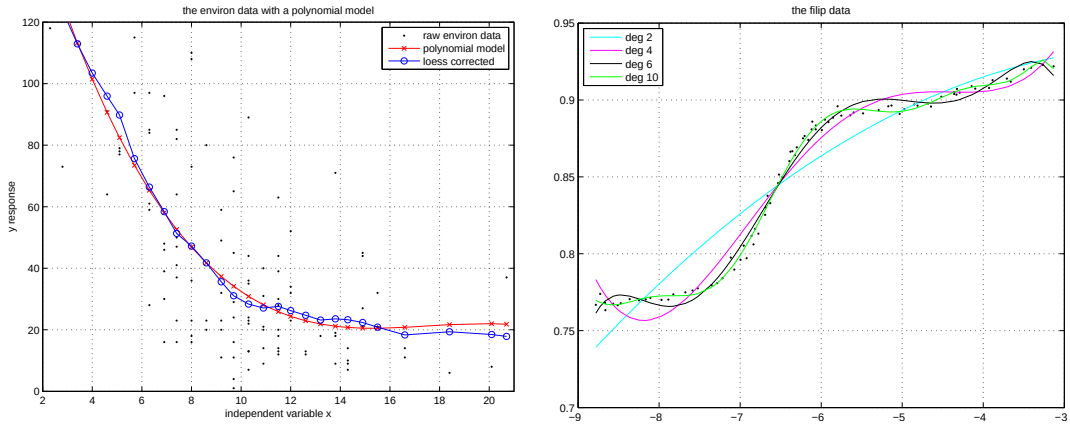

Figure 41: **Left:** The loess corrected model for the `environ` data. **Right:** The `filip` data with four polynomial models.

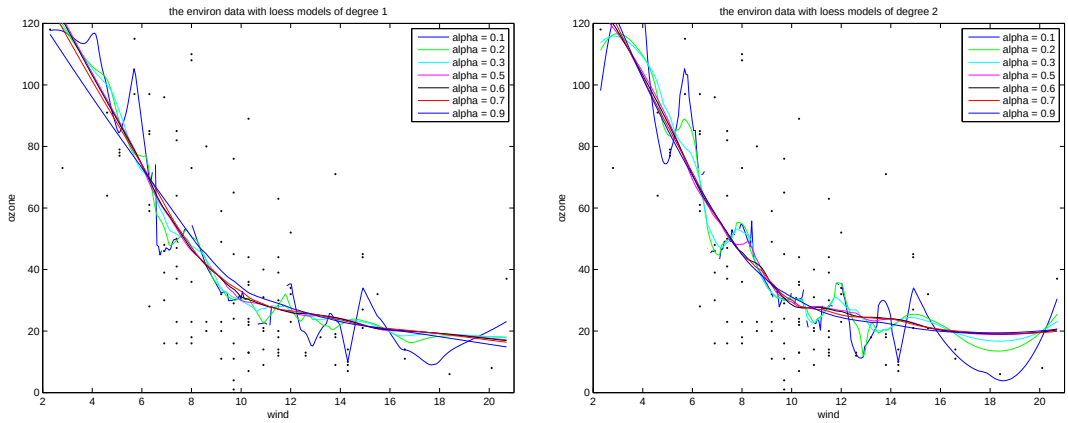

Figure 42: **Left:** Loess local smooths of the `environ` data using degree $d = 1$ polynomials (lines) and various values of α. **Right:** Loess local smooths of the `environ` data using degree $d = 2$ polynomials (quadratics) and various values of α.

order polynomial fits the data quite will. See Figure 41 for a plot of the model and the four polynomial models.

Exercise 10.6 (using polytool)

See the Matlab file `prob_10_6.m` for this exercise, which is simply a driver script for the `polytool` Matlab function call. The `polytool` lets one experiment fitting data with polynomials of different degrees.

Exercise 10.7 (loess with various values for α)

See the Matlab file `prob_10_7.m` for this exercise. We plot loess estimate of the `environ` data for various parameters. Remembering that a loess smooth of a data set is obtained by fitting a low degree polynomial close to points. For this exercise we will choose local polynomials of degree one and two. The number of local points (for each point where we want to evaluate the loess smooth at) is determined by a certain fraction of the total number of points in the data set. If n is the total number of data points and α is a number such that $0 < \alpha < 1$, then the number of points used in each local smooth is given by $O(\alpha n)$. We expect that when α is "small" we are fitting our local curves to specific details of the specific values of the given data set. This can be seen empirically using the experiments presented in `prob_10_7.m`. When this script is run two figures are produced. The first is reproduced in Figure 42 (left) and corresponds to using linear polynomials locally. The second is reproduced in Figure 42 (right) and corresponds to using quadratic polynomials locally. In both cases we see that as we make α smaller and smaller the loess smooths become much more oscillatory effectively, these local fits are begin fit to the noise inherent our data rather than the actual signal.

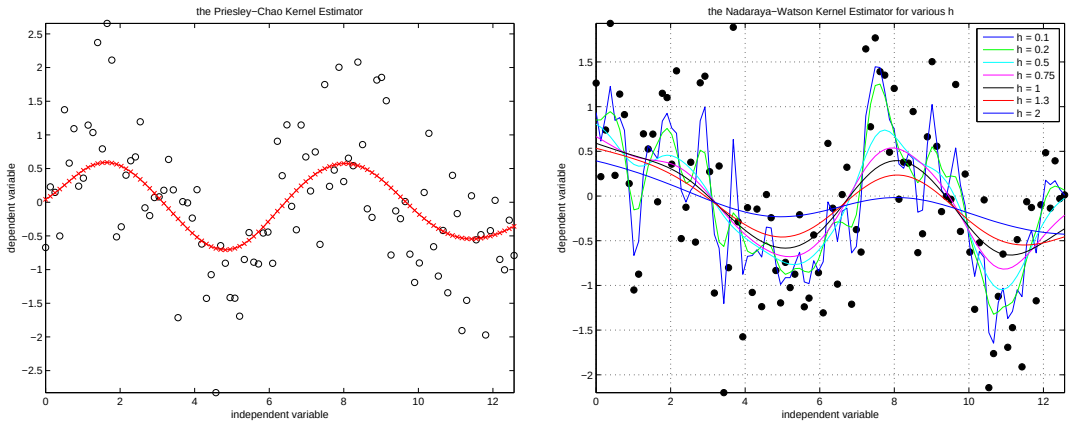

Figure 43: **Left:** A demonstration of the Priestley-Chao estimator used to estimate a sinusoidal pattern in noisy data. **Right:** The Nadaraya-Watson Kernel estimator as a function of h (the kernel width).

Exercise 10.8 (the Priestley-Chao estimator)

The Priestley-Chao estimator is a local regression estimator based on the use of kernels to do the regression. For this exercise we will repeat Example 10.6 but instead using the Priestley-Chao estimator. In the Matlab script `prob_10_8.m` we duplicate the Matlab code from that exercise where we add noise to a sinusoidal signal and attempt to "reconstruct" the signal using the Priestley-Chao estimator. Running the above code give the plot in Figure 43 (left)

Exercise 10.9 (experiments with various value of h)

See the Matlab file `prob_10_9.m` for this exercise. When this script is run we produce the plot shown in Figure 43 (right). Again we see the trade-off between representation and generalization that comes with changing the size of h. Smaller h fit the data too closely while large value of h misses general trends.

Exercise 10.10 (loess on the `human` data)

See the Matlab file `prob_10_10.m` for this exercise. As in Exercise 10.7 we perform loess smoothing on the `human` data. In this case we combine the `female` and the `male` measurements into one matrix and perform loess on the entire data set. One thing to notice about this data set is that it is not very large and it is rather noisy, both of which give the conclusion that many data points (a large value of α) is appropriate for this data. When we look at the results from running this script (shown in Figure 44) we indeed see that larger values of α provide better global fits.

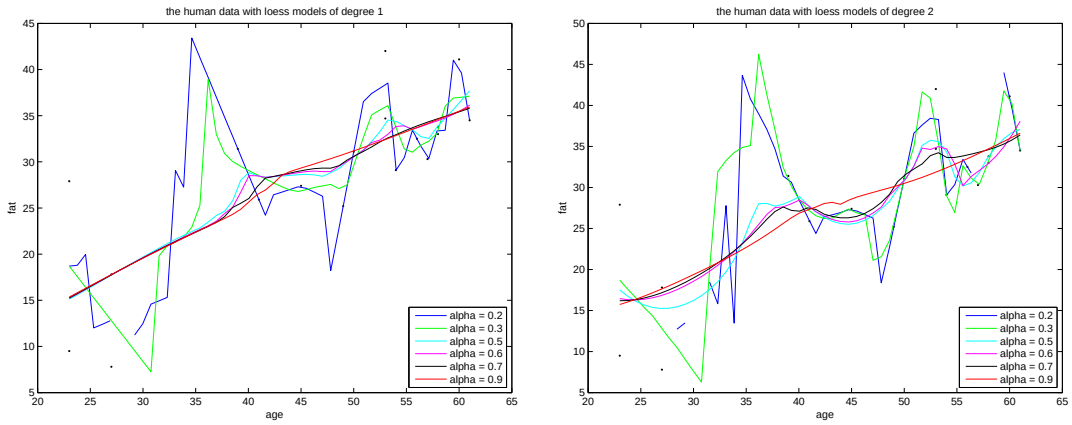

Figure 44: **Left:** Loess local smooths of the `human` data using degree $d = 1$ polynomials (lines) and various values of α. **Right:** Loess local smooths of the `human` data using degree $d = 2$ polynomials (quadratics) and various values of α.

Exercise 10.11 (loess on the `anaerob` data)

See the Matlab file `prob_10_11.m` for this exercise. As in Exercise 10.7 we perform loess smoothing on the `anaerob` data. One thing to notice is about this data set is that while it may not have a large number of data points (only 53) it does not seem to very noisy. Thus both local and global fits should perform equally well when performing a loess smooth with this data. Thus almost any reasonable value of α works well. This can been seen when we run this code (the result of which is presented in Figure 45).

Exercise 10.12 (a regression tree on the `brownlee` data)

We desire to use cross-validation to pick the optimal sized tree for the regression task of predicting the stack loss given independent variables that are related to the operational inputs of an ammonia producing plant. This exercise is done in the Matlab script `prob_10_12.m` where we split the input data into two parts and train on the first half and pick the optimal tree using cross-validation on the second half of the data. When we do this we find that we develop three regression trees with a mean square error of each tree on the testing set given by

$$0.7111\,, 0.8877\,, 0.7398\,.$$

The minimum mean square error is given by the *first* tree. Computing the standard error of this tree we obtain a value of 0.5130. The minimum complexity tree that is less than the sum of the mse and the standard error 1.2241 would be the third tree and that would be the optimal one to return.

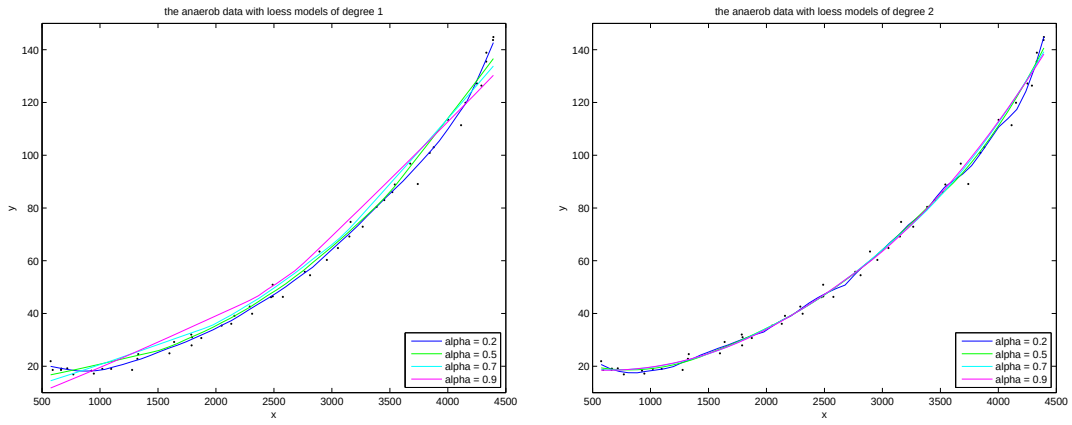

Figure 45: **Left:** Loess local smooths of the `anaerob` data using degree $d = 1$ polynomials (lines) and various values of α. **Right:** Loess local smooths of the `anaerob` data using degree $d = 2$ polynomials (quadratics) and various values of α. Note that all loess models work approximately the same. Since this data set has very little noise the loess models with small values of α (fits that are more local) do a better job.

Exercise 10.13 (a regression tree on the `abrasion` data)

We desire to use cross-validation to pick the optimal sized tree for the regression task of predicting the abrasion loss given independent variables that are hardness and tensile strength on particular metals of interest. This exercise is done in the Matlab script `prob_10_13.m` where we split the input data into two parts and train on the first half and pick the optimal tree using cross-validation on the second half of the data. When we do this we find that we develop seven regression trees with a mean square error of each tree on the testing set given by (in order of *decreasing* tree complexity)

$$6.3272\,, 6.3774\,, 5.4224\,, 5.5645\,, 5.0481\,, 4.9152\,, 7.4069\,,$$

these should be multiplied by 1000 to give appropriate units. From these we see that the minimum mean square error is given by the *sixth* tree. Computing the standard error of this tree we obtain a value of 1231. The optimal tree in this case would be the minimum complexity tree that is less than the sum of the mse and the standard error or 6146 would be the sixth tree again and that would be the optimal tree to return.

Exercise 10.14 (a loess smooth of the `helmets` data)

See the Matlab file `prob_10_14.m` for this exercise, where we perform loess on the `helmets` dataset. We use the Computational Statistics Toolbox function `csloessenv`. This function has an option to perform robust loess if desired. Given the rather noisy acceleration data around a time of 35 we expect that the α parameter which determine how many data points we should use in each local fit should be rather small (implying a more global fit). Experimenting we find that for a local first order polynomial fit $\alpha = 0.3$ seem to work well as

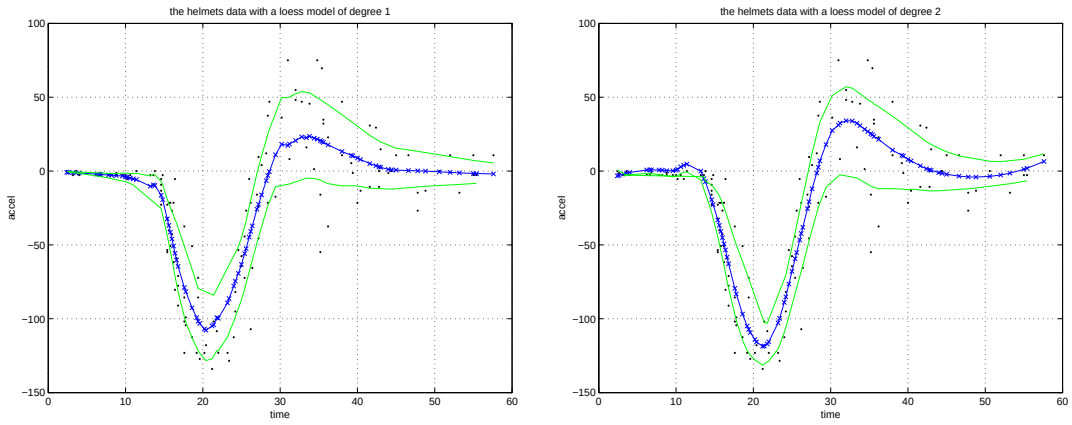

Figure 46: **Left:** Loess local smooth and smoothed envelope for the `helmets` data using local degree $d = 1$ polynomials (lines). **Right:** Loess local smooth and smoothed envelope for the `helmets` data using local degree $d = 2$ polynomials (quadratics).

seen in Figure 46 (left). It didn't seem that the robust version of loess was that much better than the non-robust version. Using second degree polynomials gives very similar results.

Exercise 10.15 (kernel regression on the `helmets` data)

The code for this exercise is in the Matlab script `prob_10_15.m`. In this script we compute a Nadarya-Watson estimator which is a kernel smoother given by

$$f_{\text{NW}}(x) = \frac{\sum_{i=1}^{n} K_h(X_i - x) Y_i}{\sum_{i=1}^{n} K_h(X_i - x)} .$$

Here $K_h(\cdot)$ is a kernel function with a "width" parameterized by the variable h. For this exercise we use a Gaussian for the kernel given by

$$K_h(x) = \frac{1}{\sqrt{2\pi}h} e^{-\frac{1}{2}\left(\frac{x}{h}\right)^2} .$$

The result of applying the Nadarya-Watson estimator (with $h = 2$) is shown in Figure 47.

Exercise 10.16 (regression trees on the `boston` data)

The code for this exercise is in the Matlab script `prob_10_16.m`. When that code is run we first grow the regression tree on the `boston` data set using Breiman et al. rule of thumb where by we grow the regression (on half of the data) tree until a each leaf node contains five or fewer data samples. This initial tree has 163 nodes and is quite large (relative to the pruned tree we will find below). We then prune this overly dense tree obtaining *seventy-four* pruned trees. Using each of these trees with the second half of our data we then determine the mean square error (MSE) each tree achieves. We find that the mean square errors for

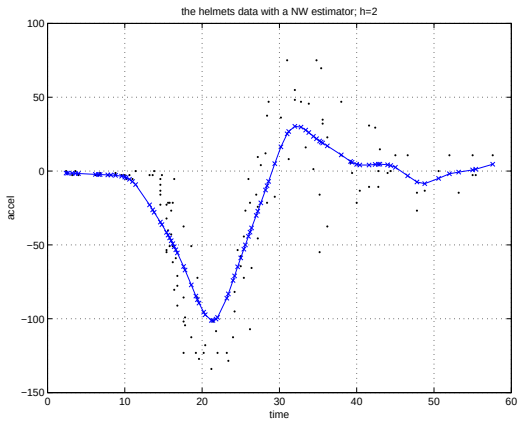

Figure 47: The `helmet` data with a Nadarya-Watson kernel estimator.

each of the regression tress is shown in Figure 48 (left). Also drawn on that figure are the minimum of the mean square errors drawn as a red line and this minimum incremented by the standard error of the tree that achieves this minimum error. We then select as an optimal tree the tree that is of minimum complexity that has a MSE *less* than this value (the MSE pulse its standard error). This corresponds to the tree with index 68. This tree is plotted in Figure 48 (right). Note that the tree selected by cross-validation in contrast has far fewer nodes.

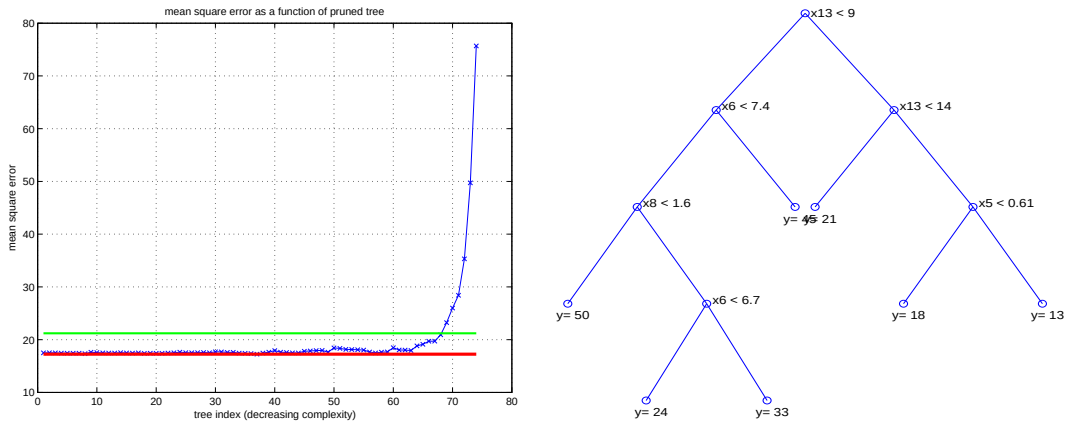

Figure 48: **Left:** The mean square error corresponding to each pruned tree on the testing data set for the `boston` data. Drawn as horizontal lines are the minimum mean square error (MSE) and the minimum mean square error m plus one standard error of this quantity $\hat{\text{se}}$. While the minimum MSE is located at index one (the most complex tree) the least complex tree with MSE less than $m + \text{se}$ is located at index 70. **Right:** The minimum complexity tree selected by cross-validation.

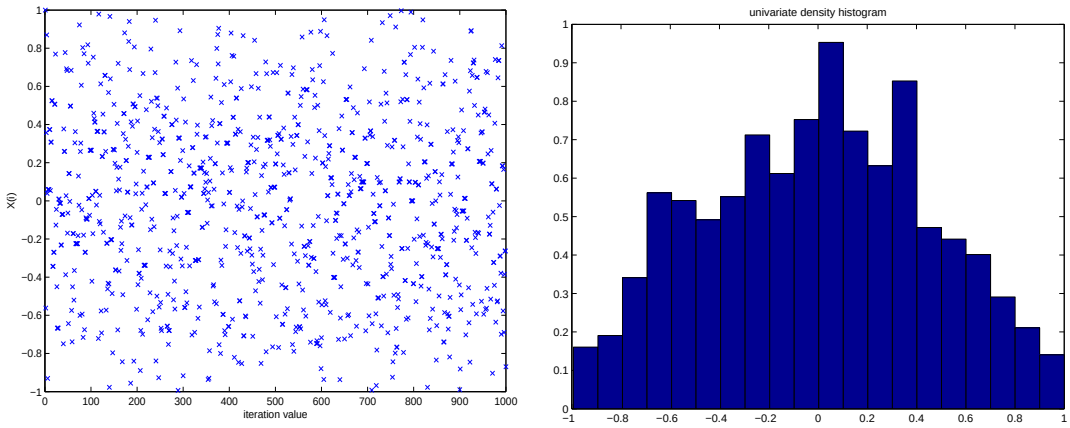

Figure 49: **Left:** Data samples from the Metropolis-Hastings chain when sampling from a von Mises distribution. Note in this case the support of the proposal distribution $[-1, +1]$ is *smaller* than the support of the true distribution $[-\pi, +\pi]$ and we don't expect to correctly represent the desired target distribution. **Right:** The corresponding density histogram of the samples from the chain.

Chapter 11: Markov Chain Monte Carlo Methods

Exercise Solutions

Exercise 11.1 (drawing samples from the von Mises distribution)

See the Matlab file `prob_11_1.m` for this exercise. To simulate from the von Mies distribution using the random-walk version of the Metropolis-Hastings sampler we will draw variates Y from our proposal distribution q, which in this case is the uniform distribution. Since this distribution is constant it will cancel in the calculation of α (the rejection threshold for the new point). That is, when we calculate α as

$$\alpha = \min\left\{1, \frac{\pi(Y)q(X_t|Y)}{\pi(X_t)q(Y|X_t)}\right\}, \tag{2}$$

we recognized that π is our *target* distribution which in this case is the von Mises distribution, while q is the proposal distribution which in this case is uniform and therefore cancels itself from the top and bottom of the above expression. Thus the above expression simplifies to

$$\alpha = \min\left\{1, \frac{\pi(Y)}{\pi(X_t)}\right\},$$

where Y is drawn from the target distribution and X_t is the previous state in the chain. The chain history and a density histogram are shown in Figure 49. One difficulty with simulating from this distribution, as suggested, is that the natural range of our variable is $-\pi \leq x \leq \pi$ but when using a uniform distribution for the proposal distribution we never get data samples outside of $[-1, +1]$ and thus never get samples as large as we should from

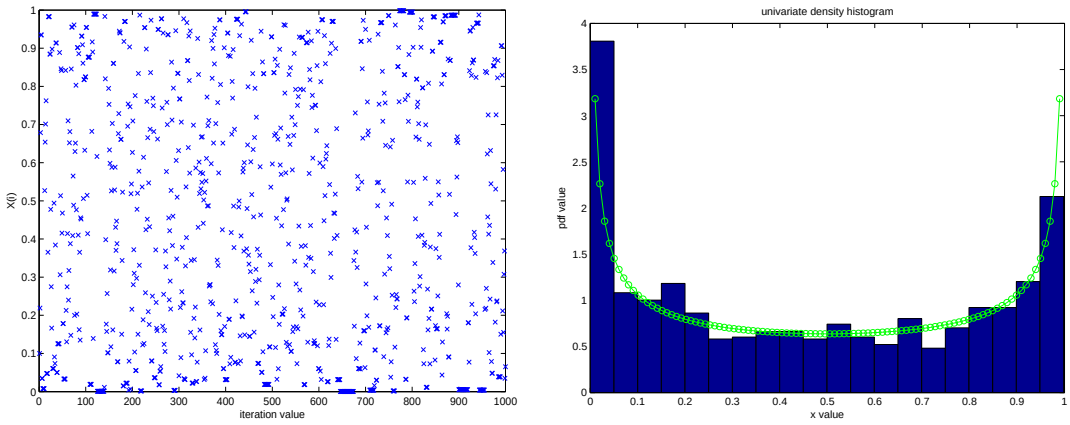

Figure 50: **Left:** Data samples from the Metropolis-Hastings chain when simulating from a beta distribution. **Right:** The corresponding density histogram of the samples from the chain. The PDF of a beta distribution with parameters $(\alpha, \beta) = (0.5, 0.5)$ is plotted as an overlay in green.

the given distribution. The support of our proposal distribution is smaller than the support of the target distribution. Thus we will not accurately represent the true distribution we desire to sample from. This can be fixed/corrected by sampling from a distribution with a *larger* support, say the uniform distribution over $[-4, +4]$. In this case our Monte-Carlo Metropolis-Hasting samples would converge to the correct distribution.

Exercise 11.2 (drawing samples from the beta distribution)

See the Matlab file `prob_11_2.m` for this exercise. We will simulate from the beta distribution using the random-walk version of the Metropolis-Hastings (MH) algorithm as in Exercise 11.1 above. We begin by recognizing that the beta distribution, with parameters α and β, has a probability density function given by

$$\pi(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1} .$$

The natural domain for this distribution is $[0, 1]$ so we will take as a proposal distribution q a uniform distribution over this same range. Now in the random walk Metropolis-Hastings sampler we will draw variates Y from our proposal distribution q and the probability that we accept this proposed point Y with probability α via Equation 2. A nice thing about using the Metropolis-Hastings algorithm to draw samples from $\pi(\cdot)$ is that we don't have to explicitly calculate the leading coefficients. For example in the case given here the probability of accepting the given MH step α becomes

$$\alpha = \min\left\{1, \frac{Y^{\alpha-1}(1-Y)^{\beta-1}}{X_t^{\alpha-1}(1-X_t)^{\beta-1}}\right\} .$$

If we take $\alpha = \beta = 0.5$ we obtain a particularly interesting looking probability density function. Running the above script we obtain a plots of the particular chain samples and a density histogram as shown in Figure 50.

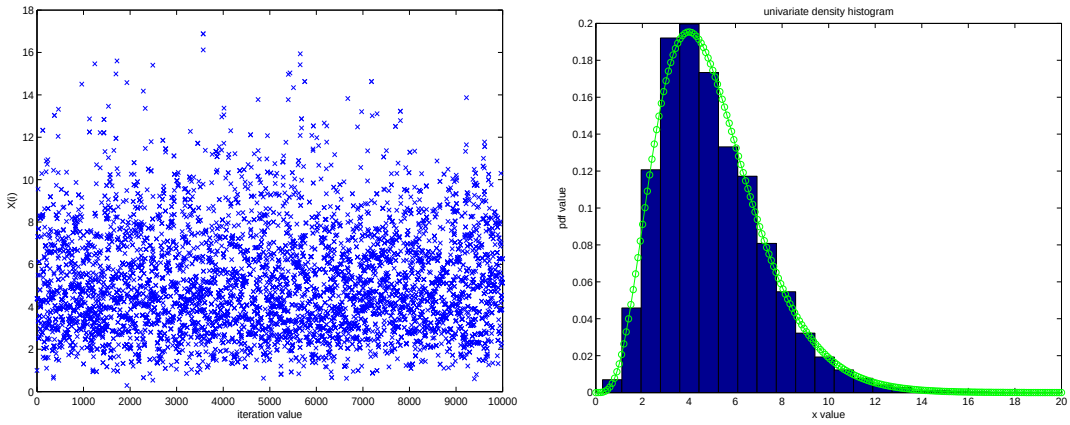

Figure 51: **Left:** The Metropolis-Hastings chain of samples when simulating from a gamma distribution. **Right:** The corresponding density histogram of the samples from the chain. The PDF of a gamma distribution with parameters $(\alpha, \beta) = (5, 1)$ is plotted as an overlay in green.

Exercise 11.3 (drawing samples from the gamma distribution)

See the Matlab file `prob_11_3.m` for this exercise. We will simulate from the gamma distribution using the random-walk version of the Metropolis-Hastings (MH) algorithm as in Exercise 11.1. We begin by recognizing that one representation of the gamma distribution, with parameters α and β, has a probability density function given by

$$\pi(x) = \left(\frac{\beta^\alpha}{\Gamma(\alpha)} \right) x^{\alpha-1} e^{-\beta x} .$$

The natural domain for this distribution is $[0, +\infty]$ so we will take as a proposal (candidate) distribution, q a uniform distribution over a relatively large range say $[0, 50]$. Now in the random walk Metropolis-Hastings sampler we will draw variates Y from our proposal distribution q and the probability that we accept this proposed point Y happens with a probability α via Equation 2. As in Exercise 11.2 a nice thing about using the Metropolis-Hastings algorithm to draw samples from $\pi(\cdot)$ is that we don't have to explicitly calculate the leading coefficients for the distribution π. For example, in the case given here the probability of accepting the given MH step α becomes

$$\alpha = \min \left\{ 1, \frac{Y^{\alpha-1} e^{-\beta Y}}{X_t^{\alpha-1} e^{-\beta X_t}} \right\} = \left(\frac{Y}{X_t} \right)^{\alpha-1} e^{-\beta(Y - X_t)} .$$

If we take $\alpha = 5$ and $\beta = 1$ we obtain a particularly interesting (and characteristic) looking probability density function. Running the above script we obtain plots of the particular chain samples and a density histogram as shown in Figure 51.

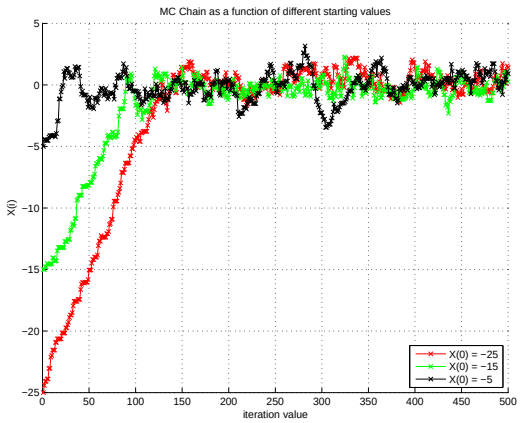

Figure 52: Data samples from the Metropolis-Hastings chain when simulating from a standard normal with three different starting locations -25, -15 and -5. Note that the number of steps required to be generating samples from $\pi(\cdot)$ increases the further we are in the tail of $\pi(\cdot)$ when we start.

Exercise 11.4 (generating standard normal random variates)

See the MATLAB script `prob_11_4.m` for this exercise. In this exercise we are to pick various values of the chain starting value X_0 and observe the affect this has on the length of the burn-in period. In this problem we expect that by specifying an initial value for our Monte Carlo (MC) chain that is significantly far from the mean of the distribution we are trying to approximate it will take a longer time for the MC chain to obtain samples that are from the desire distribution. Thus the experiment we will consider here is to start the chain with several points different from the mean of zero, and observe qualitatively when our MC chain seems to be generating samples from a standard normal. For starting points of -25, -15 and -5 (representing various distances from zero) we have a plot of the MC variates shown in Figure 52. In that figure we see that the burn in parameter should increase depending on how far the initial value of the chain $X(0)$ is from zero. If we start further away from the mean of the standard normal (say the point -25) it takes more iterations of the Metropolis-Hastings chain to generate samples from our desired distribution. When we look at Figure 52 we see that qualitatively when $X(0) = -25$ it takes around 125 iterations before our chain seems to be drawing samples from a standard normal. When $X(0) = -15$ this number is around 75 while when $X(0) = -5$ it is 25.

Exercise 11.5 (using the Gibbs sampler)

In the specification of the Gibbs sampler we assume that we know the conditional distributions and desire to sample from the marginal distribution

$$f(x_{,1}) = \int f(x_{,1}, x_{,2}) dx_{,2} \,.$$

The algorithm for the Gibbs sampler in the bivariate case is given by

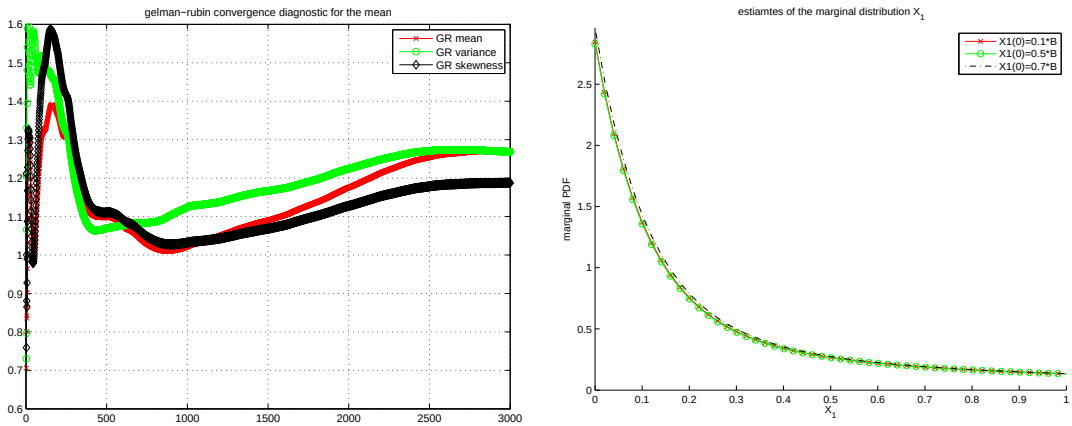

Figure 53: **Left:** A plot of the Gelman-Rubin convergence diagnostic for the mean, variance, and the skewness statistics (using three chains) as a function of the chain index. **Right:** A plot of the approximations to the marginal distribution of $x_{,2}$. Three plots are shown one for each chain (representing each initial condition).

- Generate at starting point $X_0 = (X_{0,1}, X_{0,2})$ and set $t = 0$.
- Generate the first coordinate of X_{t+1} i.e. $X_{t+1,1}$ from the conditional distribution

$$f(X_{t+1,1}|X_{,2} = x_{t,2}) \,.$$

- Generate the second coordinate of X_{t+1} i.e. $X_{t+1,2}$ from the conditional distribution

$$f(X_{t+1,2}|X_{,1} = x_{t+1,1}) \,,$$

  where we are using the value of the sample drawn in the previous step.
- Set $t = t + 1$ and continue

Once we have a significant number of samples X_t by following the above procedure we can compute an approximate marginal distribution of $x_{,1}$ i.e. $\hat{f}(x_{,1})$ using

$$\hat{f}(x_{,1}) = \frac{1}{k-m} \sum_{i=m+1}^{k} f(x_{,1}|X_{i,2} = x_{i,2}) \,, \tag{3}$$

where $f(x_{,1}|x_{,2})$ is the known conditional distribution evaluated at the $X_{,2}$ samples $i = m+1, m+2, \cdots, k$ produced from the Gibbs sampler and m is the "burn-in" time. Now from the problem specification the distribution of $x_{,1}$, given $x_{,2}$, is exponential with a mean of $1/x_{,2}$ and the distribution of $x_{,2}$, given $x_{,1}$, is exponential with a mean of $1/x_{,1}$. To prevent the exponential random variables generated from becoming too large we limit the output of each exponential random draw to be *less than* B. By doing this we provide the desired truncated exponential behavior.

For the MATLAB code to implement this problem see the MATLAB script `prob_11_5.m`. When this code is run it produces two plots shown in Figure 53. In Figure 53 (left) we plot

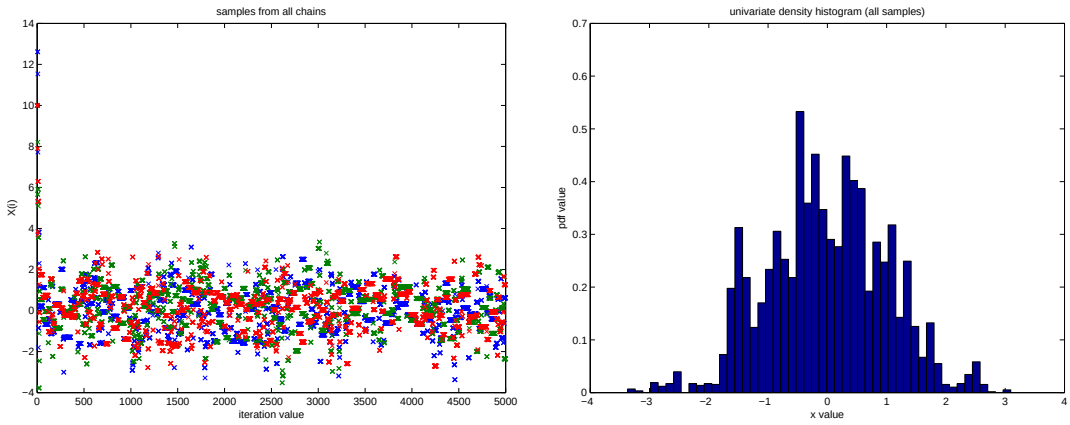

Figure 54: **Left:** The trajectories of each coordinate from of our sample. **Right:** The density histogram corresponding the first coordinate $x_{,1}$.

the Gelman-Rubin convergence diagnostic for the mean when we run three Monte-Carlo (MC) chains with three different starting points. Since all data samples are confined to be in the domain $[0, B]$ we take the starting points to be spaced uniformly inside that range. Specifically we take them to be $0.2B$, $0.5B$ and $0.7B$. We see that after about 400 iterations the diagnostic begins to flatten towards 1.1 demonstrating convergence.

In Figure 53 (right) we plot the marginal distributions produced from of the chains (each with a different initial start). We see that the three chains (after excluding a burn-in period) produce quite similar marginal distributions.

Exercise 11.6 (higher dimensional Metropolis-Hasting sampling)

In this problem specification, the target distribution we are trying to draw samples from is a tridimensional standard normal and we will use a proposal distribution $q(Y|X_t)$ given by a tridimensional normal centered on the previous chain iterate X_t but with a "larger" covariance. That is

$$Y \sim \mathcal{N}(y; X_t, 9I)\,.$$

See the MATLAB script `prob_11_6.m` for this exercise. When this script is run it produces several plots, two of which can be seen in Figure 54. In that figure, we plot the sample trajectories for each component of our vector X_t and a density histogram of the first component X_1 of our generated sample. I choose to generate 5000 samples and used a burn-in of 500 samples. The density histogram in Figure 54 (right) is looks qualitatively like a normal distribution.

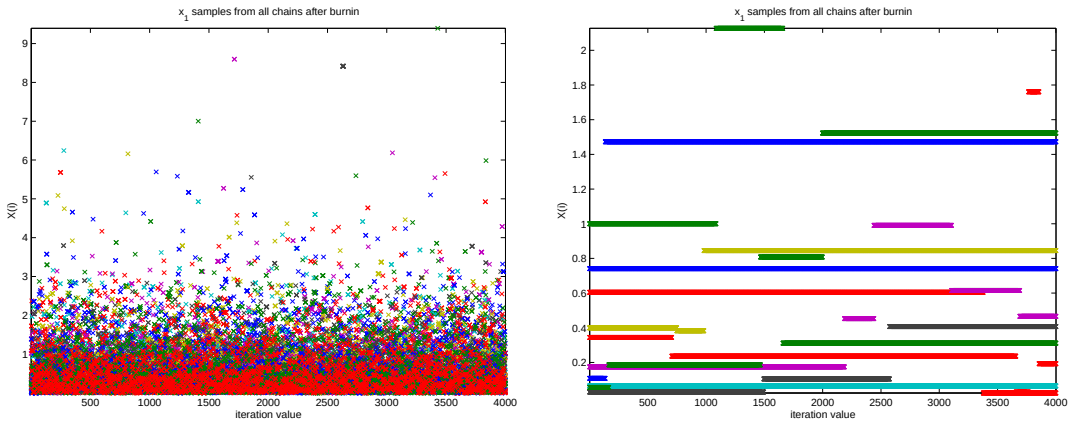

Figure 55: **Left:** The samples X_1 generated from our MC chains when applying Gibb's sampling to this problem and using three independent exponential distribution with parameter one for the proposal distributions. **Right:** The samples X_i when a poor proposal distribution is used. Note that we have a large number of iterations where the proposed point Y is *not* selected. This lack of selection of the proposed point Y signifies that the proposal distribution is a poor one and many iterations with a corresponding large value for the burn-in time will be required to draw samples correctly using it.

Exercise 11.7 (generating samples from a complicated density)

For this problem one could use Gibbs sampling by computing the required marginal distributions from the given joint distribution, $f(x_{,1}, x_{,2}, x_{,3})$, using Bayes' rule. For example $f(x_{,1}|x_{,2}, x_{,3})$ is computed as

$$f(x_{,1}|x_{,2}, x_{,3}) = \frac{f(x_{,1}, x_{,2}, x_{,3})}{f(x_{,2}, x_{,3})} ,$$

with $f(x_{,2}, x_{,3}) = \int f(x_{,1}, x_{,2}, x_{,3}) dx_{,1}$. In these expressions the normalization constant C in the definition of the joint f will cancel and we don't need to explicitly know its numerical value. Since this procedure would require computing several integrals it seems easier to take an alternative approach by using the Metropolis-Hastings algorithm directly with a proposal distribution given by three *independent* exponential random variables $x_{,1}$, $x_{,2}$, and $x_{,3}$. Here we have chosen an exponential random distribution for Y_i based on the algebraic form of the expression $f(x_{,1}, x_{,2}, x_{,3})$. To be able to guarantee that our independent draws of $y_{,i}$ have a large enough support, we take them to be exponential random variables with a parameter 1. That is, the probability density for $Y_{,i}$ will be taken to be

$$f_{Y_{,i}}(y) = e^{-y} \quad \text{for} \quad i = 1, 2, 3 .$$

In this case our proposal distribution $q(Y|X_t)$ is independent of X_t and this Monte Carlo (MC) method is called the independence sampler [2]. With this expression for the distribution of Y we see that the probability of accepting a given step in the chain α, is given

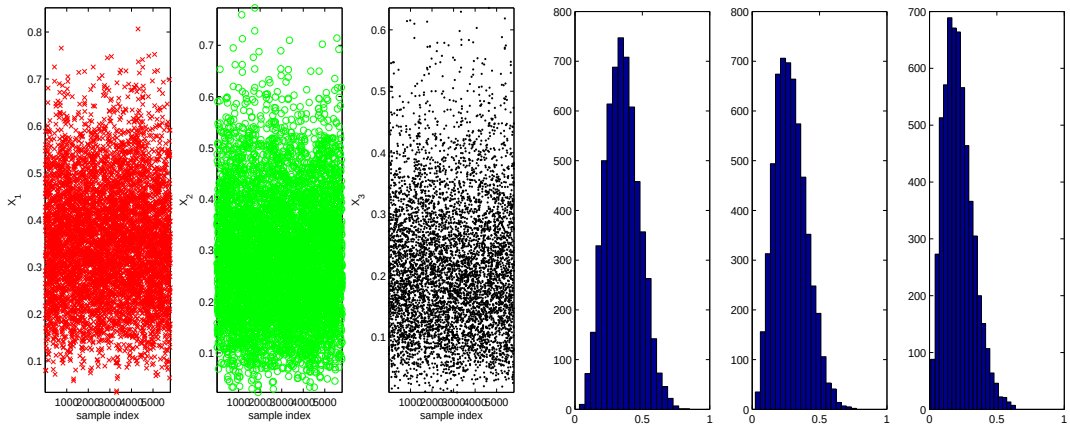

Figure 56: **Left:** The samples X_i drawn when using the Gibbs algorithm on this problem. **Right:** Frequency histograms of the samples drawn.

by

$$\begin{aligned}
\alpha &= \min\left\{1, \frac{\pi(Y)q(X_t|Y)}{\pi(X_t)q(Y|X_t)}\right\} = \min\left\{1, \frac{\pi(Y)q(X_t)}{\pi(X_t)q(Y)}\right\} \\
&= \min\left\{1, \frac{e^{-(y_1+y_2+y_3+y_1y_2+y_1y_3+y_2y_3)}e^{-(x_1+x_2+x_3)}}{e^{-(x_1+x_2+x_3+x_1x_2+x_1x_3+x_2x_3)}e^{-(y_1+y_2+y_3)}}\right\} \\
&= \min\left\{1, e^{-(y_1y_2+y_1y_3+y_2y_3-x_1x_2-x_1x_3-x_2x_3)}\right\} .
\end{aligned}$$

When we do this, after enough iterates, we are generating samples from $f(x_{,1}, x_{,2}, x_{,3})$, so taking the product of the components of these samples for form the product $X_1X_2X_3$ and averaging, we can compute an estimate $E[X_1X_2X_3]$. This is done in the MATLAB script `prob_11_7.m` and we find that using a chain length of 5000 with a burn-in of 1000 that $E[X_1X_2X_3] \approx 0.08787$. The samples drawn from this procedure are shown in Figure 55 (left).

To verify that we have done everything correctly we repeat the above exercise but with a proposal distribution q that generates independent uniform variates over a significantly large range say $[0, L]$. Doing this gives $\alpha = \min\{1, \frac{\pi(Y)}{\pi(X_t)}\}$ and we can compute $E[X_1X_2X_3]$ for various values of L. If we take $L = 5$ (a relatively small value) we obtain $E[X_1X_2X_3] \approx 0.087799$, in close agreement with the earlier calculation. If we take a uniform distribution that is too "large" say $L = 20$ our MC iterations may not converge and we will obtain a very poor approximation to our expectation of $E[X_1X_2X_3] \approx 0.129317$. The samples drawn from this poor proposal distribution are shown in Figure 55 (right). The fact that over many steps in the value from the proposal distribution for X_1 is *not* accepted gives one the indication that this proposal distribution is quite poor.

Exercise 11.8 (more Gibbs sampling)

In Gibbs sampling we generate elements from the joint distribution $f(x_{,1}, x_{,2}, x_{,3})$ by sampling repeatedly from the conditional distributions $f(x_{,1}|x_{,2}, x_{,3})$, $f(x_{,2}|x_{,1}, x_{,3})$, and $f(x_{,3}|x_{,1}, x_{,2})$. For this exercise we perform the following procedure

- Generate at starting point $X_0 = (X_{0,1}, X_{0,2}, X_{0,3})$ and set $t = 0$.
- Generate the first coordinate of X_{t+1} i.e. $X_{t+1,1}$ from the conditional distribution

$$f(X_{t+1,1}|X_{,2} = x_{t,2}, X_{,3} = x_{t,3}) = (1 - x_{t,2} - x_{t,3})Q\,,$$

  with $Q \sim B(5, 2)$.
- Generate the second coordinate of X_{t+1} i.e. $X_{t+1,2}$ using the data point $X_{t+1,1}$, found above, from the conditional distribution

$$f(X_{t+1,2}|X_{,1} = x_{t+1,1}, X_{,3} = x_{t,3}) = (1 - x_{t+1,1} - x_{t,3})R\,,$$

  with $R \sim B(4, 2)$.
- Generate the third coordinate of X_{t+1} i.e. $X_{t+1,3}$ using the data points $X_{t+1,1}$ and $X_{t+1,2}$ from the conditional distribution

$$f(X_{t+1,3}|X_{,1} = x_{t+1,1}, X_{,2} = x_{t+1,2}) = (1 - x_{t+1,1} - x_{t+1,2})S\,,$$

  with $S \sim B(3, 2)$.
- Set $t = t + 1$ and continue

Here the notation $B(\alpha, \beta)$ means a draw from the beta distribution with parameters α and β. This exercise is done in the MATLAB script `prob_11_8.m`. When this script is run it produces the plots shown in Figure 56.

Exercise 11.9 (simulating Bayesian learning)

For this problem we are told that $P(\theta|D)$ is known (up to a normalization constant) as

$$P(\theta|D) \propto \frac{1}{1+\theta^2} \exp\left\{-\frac{n(\theta - \bar{x})^2}{2}\right\}\,,$$

where $\bar{z}$ is the average of $n = 20$ standard normals. To draw samples from $P(\theta|D)$ using a Metropolis-Hasting type algorithm we need to select a proposal distribution $P(Y|X_t)$. For this problem we'll select a uniform distribution with a range given by the minimum and maximum of the 20 initial random draws. See the MATLAB script `prob_11_9.m` for this exercise. When this script is run it produces several plots, two of which can be seen in Figure 57. There we plot a density histogram of the samples generated and the Gelman-Rubin convergence diagnostic. Taking the mean of all of our generated samples from $P(\theta|D)$ we obtain an approximate mean of 0.2026 which is very close to the initial mean of the samples of 0.2158.

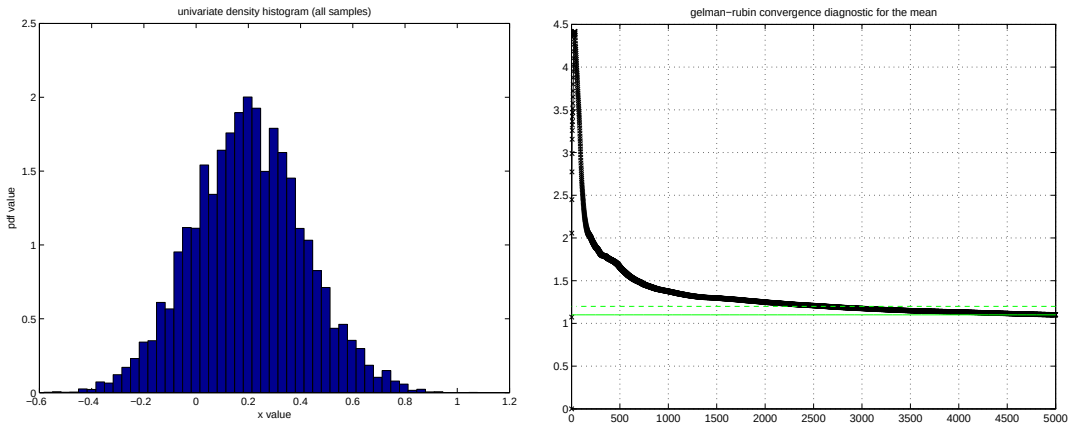

Figure 57: **Left:** A density histogram of the samples drawn from the Metropolis-Hasting algorithm designed for this problem. **Right:** The Gelman-Rubin convergence diagnostic for the mean (this is computed using five MH chains).

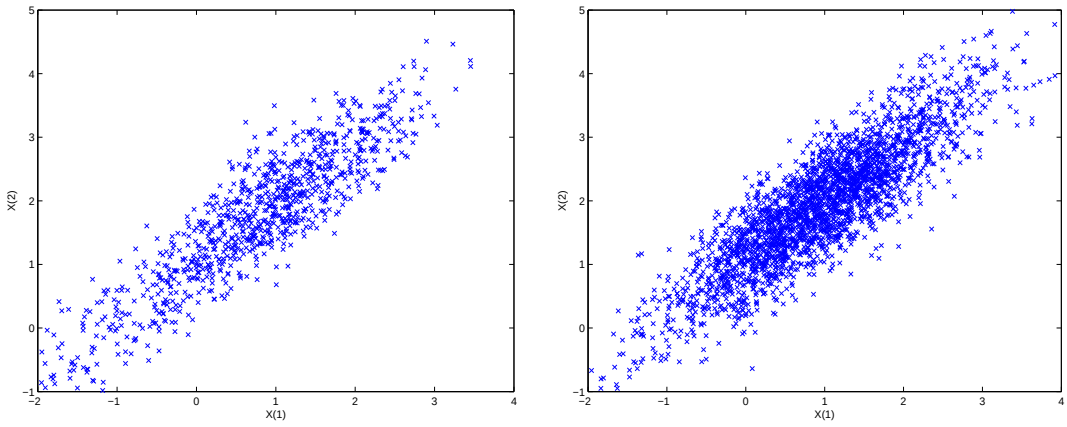

Figure 58: **Left:** The scatter plot generated when using an independence sampler with a proposal distribution such that $Y = X_t + Z$ and the increment random variable Z generated from a bivariate uniform distribution. **Right:** The scatter plot generated when using an independence sampler with a proposal distribution such that $Y = X_t + Z$ and the increment random variable Z generated from a bivariate normal distribution.

Exercise 11.11 (the random walk Metropolis-Hasting sampler)

If we have a proposal distribution q satisfies $q(Y|X) = q(|X - Y|)$ then the Metropolis-Hasting algorithm is called the Random-Walk Metropolis algorithm. For our random walk to be given by $Y = X_t + Z$ with Z a bivariate uniform we need $q(Y|X)$ to be the probability density function (PDF) of a bivariate uniform random variable *centered* on the point X_t. Then the code for this exercise follows in much the same way as Example 11.3 from the book but with the variate Y generated as (in MATLAB notation)

```
y(1) = unifrnd(X(i-1,1)-0.5*sig,X(i-1,1)+0.5*sig);
y(2) = unifrnd(X(i-1,2)-0.5*sig,X(i-1,2)+0.5*sig);
```

here `sig` is the desired width of the uniform distribution around X_t. In this case the probability of acceptance α, is given as $\alpha = \min\left\{1, \frac{\pi(Y)}{\pi(X_t)}\right\}$, with $\pi(\cdot)$ the desired bivariate normal distribution with parameters

$$\mu = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \text{and} \quad \Sigma = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}.$$

This exercise is implemented in the MATLAB script `prob_11_11.m`. When this script is run it produces the plot shown in Figure 58 (left). This plot matches very well that presented in Example 11.4 from the book.

Exercise 11.12 (more random walk Metropolis-Hasting sampling)

This exercise is implemented in the MATLAB script `prob_11_12.m`. When this script is run it produces the plot shown in Figure 58 (right). Again this plot matches very well that presented in Example 11.4 from the book.

Exercise 11.13 (generating samples from the log-normal distribution)

In simulating using the independence sampler we specify a proposal distribution such that $q(X|Y) = q(X)$, or one whos proposed samples, Y, do not depend on the previous state of the chain.

$$\alpha(X_t|Y) = \min\left\{1, \frac{\pi(Y)q(X_t|Y)}{\pi(X_t)q(Y|X_t)}\right\} = \min\left\{1, \frac{\pi(Y)q(X_t)}{\pi(X_t)q(Y)}\right\}.$$

For this problem we desire to sample from a log-normal distribution which has a density function that is proportional to the following function

$$\pi(x) = \frac{1}{x}\exp\left\{-\frac{(\ln(x))^2}{2}\right\},$$

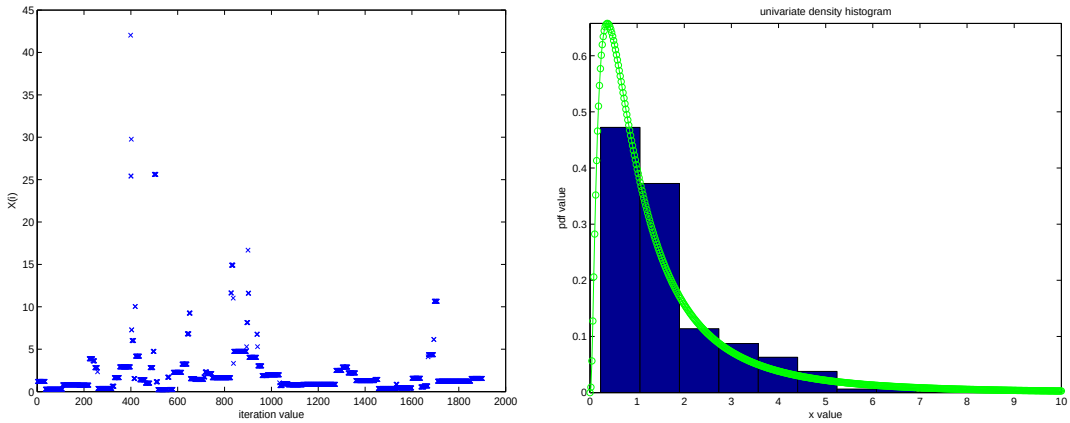

Figure 59: **Left:** Data samples from the Metropolis-Hastings chain when drawing samples from a log-normal distribution. **Right:** The corresponding density histogram of the samples from the chain. The PDF of a log-normal distribution with parameters $(\mu, \sigma) = (0, 1)$ is plotted as an overlay in green.

To do this we will specify a proposal distribution $q(X)$ given by a a Gamma distribution. It is know that when using the independence sampler that care needs to be taken that the proposal distribution have heavier tails than the desired density [1]. Guided by this we will sample from a Gamma distribution with parameters $\alpha = 1$ and $\beta = 2$. This procedure is implemented in the MATLAB script `prob_11_13.m`. When this is run it produces the plots shown in Figure 59.

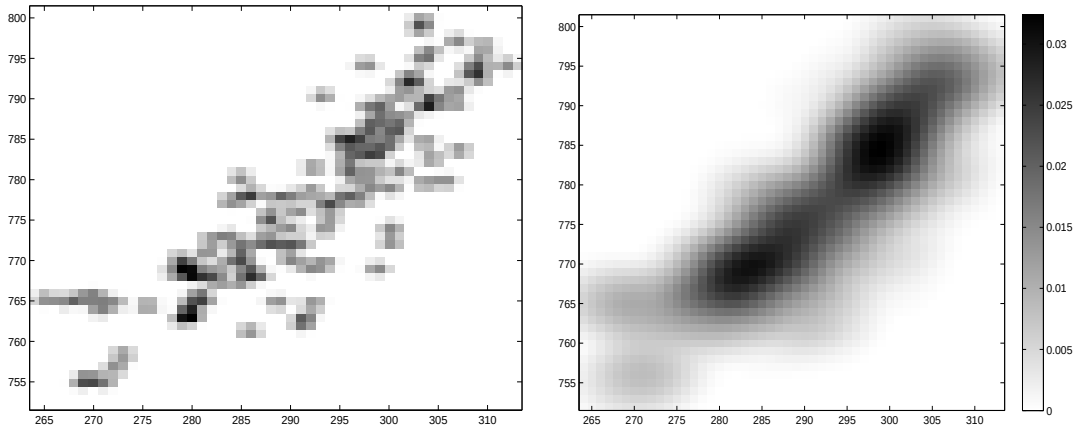

Figure 60: **Left:** The estimated spatial event density function for the `uganda` dataset using a bandwidth of $h = 100$. **Right:** The estimated spatial event density function for the `uganda` dataset using a bandwidth of $h = 500$. These represent two "extremes" of the bias-variance trade off.

Chapter 12: Spatial Statistics

Exercise Solutions

Exercise 12.1 (plotting labels on points)

See the MATLAB file `prob_12_1.m` for this exercise. When this is run it produces a labeled dot plot for the `okwhite` dataset as requested.

Exercise 12.2 (estimating the first order intensity with various bandwidths)

See the MATLAB file `prob_12_2.m` for this exercise. When this script is run, the derived two-dimensional estimate of the spatial intensity using a bandwidth of $h = 100$ is shown in Figure 60 (left) while the same thing estimated for $h = 500$ is shown in Figure 60 (right). Here again, we notice the standard bias-variance trade-off when estimating density functions from samples. A kernel width that is "too small" (like $h = 100$) will produce a very sample dependent result (low bias but high variance), while a kernel width that is "too large" (like $h = 500$) will produce an estimate that is too smooth (high bias but low variance).

Exercise 12.3 (a dot map for the `bodmin` data)

See the MATLAB file `prob_12_3.m` for this exercise. When this script is run (now shown here) we obtain a dot map for the `bodmin` dataset. From this plot it *does* appear that a

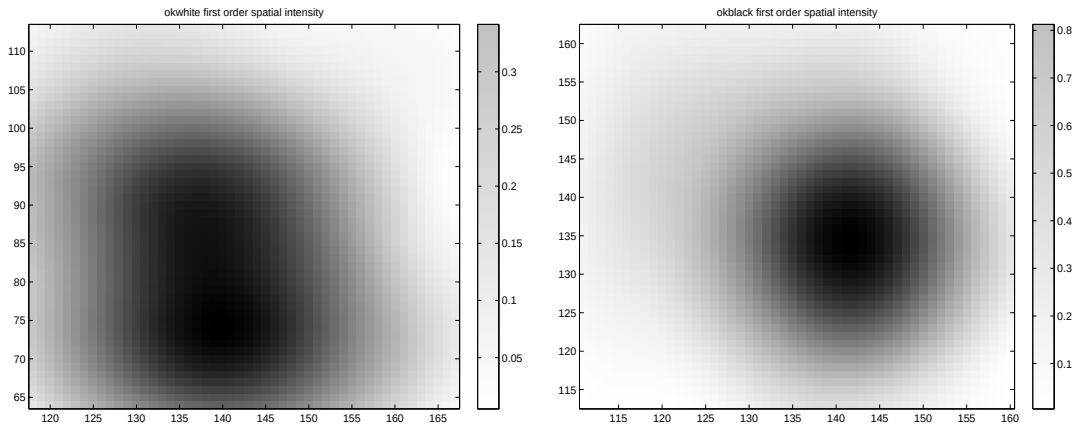

Figure 61: **Left:** The estimated spatial event intensity using a kernel bandwidth of $h = 100$ for the `okwhite` dataset. **Right:** The estimated spatial event intensity using a kernel with a bandwidth of $h = 100$ for the `okblack` dataset.

clustering process is at work.

Exercise 12.4 (using `csgetregion`)

See the MATLAB file `prob_12_4.m` for this exercise. When this is run the user can experiment with the `csgetregion` code. This routine provides a very nice way of interactively selecting a bounding region from a given two dimensional region.

Exercise 12.5 (estimating first-order and second-order properties)

This exercise is implemented in the MATLAB script `prob_12_5.m`. When that script is run, we plot the estimated spatial intensity as an image using MATLAB's `imagesc` command. From the produced plots, shown in Figure 61, one can see that the location of thefts for white offenders and black offenders appear at different locations. These first order properties are estimated using the computational statistics toolbox function `csintkern`.

To estimate the second order properties of this data set we will compute an estimate of the event-event nearest neighbor distribution function $\hat{G}(x)$. In 62 (left) we plot estimates of the event-event distribution functions for both the `okwhite` and the `okblack` datasets. In this figure we see that the `okwhite` curve seems to rises more slowly than the corresponding curve for the `okblack` curve. This implies that the `okblack` theft events appear to be more clustered than the corresponding events for `okwhite` dataset.

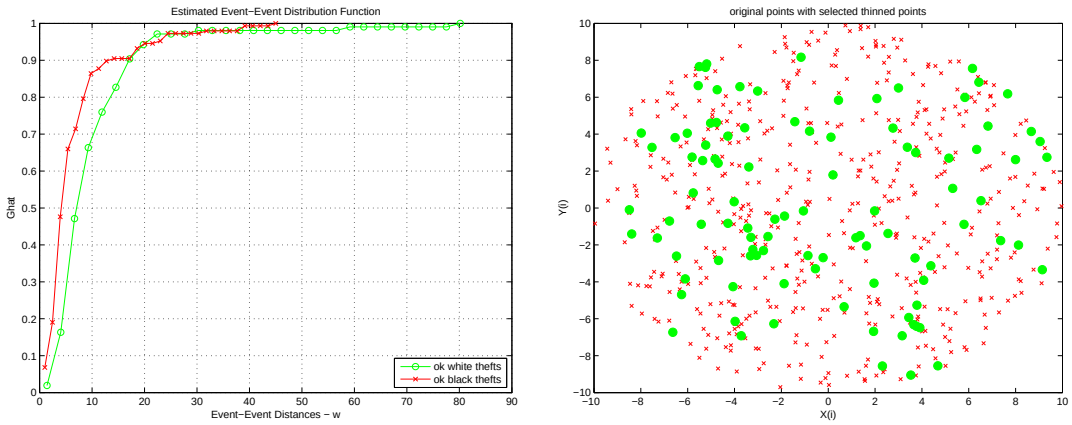

Figure 62: **Left:** The estimated event-event distribution functions for the Oklahoma City data. **Right:** The original data points and the selected thinned points corresponding to the desired inhibition process.

Exercise 12.6 (an inhibition process via thinning)

To generate an inhibition process using the thinning approach we first generate a large number of data points according to a homogeneous Poisson process. These events are then "thinned" by deleting all pairs of events that are closer than δ (the inhibition distance). This problem can be found in the MATLAB script `prob_12_6.m`. When this script is run it produces the plot shown in Figure 62 (right). There we see both the original Poisson data points along with the points selected using the thinning process. All of the selected points are at least $\delta = 0.01$ from each other.

Exercise 12.7 (the point-event nearest neighbor distribution for the `bodmin` data)

The point-event nearest neighbor event distribution is denoted, $F(x)$, and under the complete spatial randomness (CSR) model it has an analytic expression given by

$$F(x) = P(X \leq x) = 1 - e^{-\lambda \pi x^2}, \quad \text{for} \quad x > 0\,.$$

In this problem, we will assess if our data points are generated according to such a process, using a Monte Carlo test. We begin by selecting a random *initial* sampling of points, $\mathcal{P}$, in our domain R to use in computing the point-event distances with. Using these, we can compute an initial estimate of F called $\hat{F}_{\text{Obs}}(x)$. We then generate B bootstrap estimates of $\hat{F}_{\text{CSR}}(x)$. To do this we generate B examples of n "events" at random and using the the initial sampling of points found earlier, $\mathcal{P}$, compute $\hat{F}_b(x)$ (the estimated point-event distribution under CSR events) for each. Using these B bootstrap replicas of our CSR process (at every x) we compute the mean value of all our bootstrap replicas and the minimum and maximum values. We will call the mean value $\hat{F}_{CSR}(x)$. We then plot $\hat{F}_{\text{Obs}}(x)$ *against* $\hat{F}_{CSR}(x)$. If the data follows a CSR process the resulting plot should be a straight line.

This problem is implemented in the MATLAB script `prob_12_7.m`. When this script is run

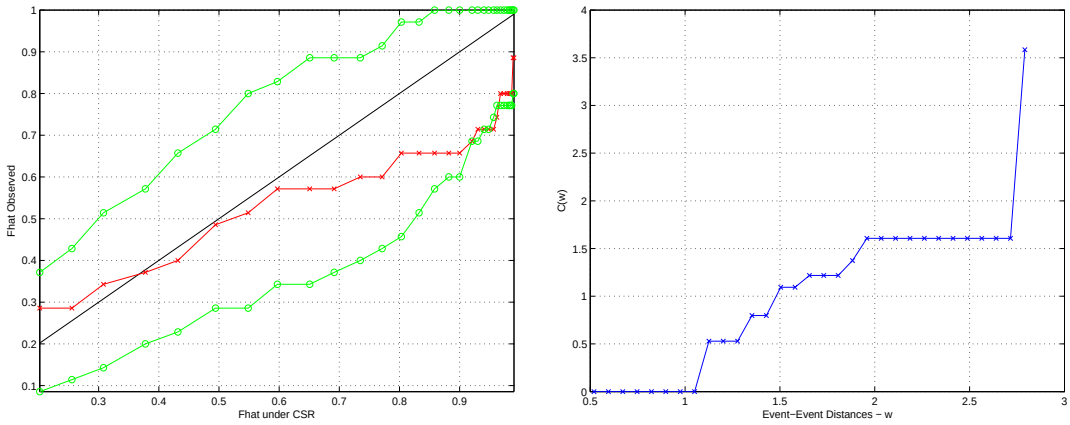

Figure 63: **Left:** A plot of $\hat{F}_{CSR}(x)$ against $\hat{F}_{\text{Obs}}(x)$. **Right:** The empirical event-event cumulative distribution function transformed as specified by Equation 4.

the resulting plot is shown in Figure 63 (left). From this plot we see a rather straight line (initially at least) which might indicate that the `bodmin` data might follow a CSR process.

Exercise 12.8 (another test for departure from the CSR model)

Equation 12.12 is

$$C(w) \equiv \left\{ -\frac{\log(1-\hat{G}(w))}{\hat{\lambda}\pi} \right\}^{1/2} . \tag{4}$$

In this expression $\hat{\lambda}$ is an estimate of the intensity of the spatial point process and $\hat{G}$ is an estimate of the event-event nearest neighbor distance distribution function. The spatial intensity can be estimated here as $\hat{\lambda} = \frac{n}{r}$ with n the number of events in a certain region of area r. If a plot of w v.s. $C(w)$ gives a straight line then we expect our process to be very similar to a CSR point process. A significant departure from a straight line would provide the opposite interpretation.

This exercise is done in the MATLAB script `prob_12_8.m`. When that script is run we obtain the plot shown in Figure 63 (right). Since there are only thirty five points in the `bodmin` dataset, we can't expect our estimates to be incredibly statistically significant. In particular we see that there are no event-event distances smaller than about 1.0 and all but one are smaller than about 2.0. In between these two limits of w the data is rather linear. This implies that the `bodmin` dataset maybe CSR and does not indicate evidence for a departure from CSR. This is in agreement with the conclusion in Example 12.7, but in contradiction to the conclusion in Example 12.5.

Exercise 12.9 (an alternative test-statistic for a CSR process)

Equation 12.22 is

$$P(\hat{F}_{\text{Obs}}(x) > U(x)) = P(\hat{F}_{\text{Obs}}(x) < L(x)) = \frac{1}{B+1}\,,$$

but as discussed in the text, it maybe better to use the statistic

$$T = \max_X |\hat{F}_{\text{Obs}}(x) - \hat{F}_{\text{CSR}}(x)|\,.$$

Our hypothesis test for assessing the departure from a CSR model using this statistic would then be

- H_0: the data we observe is from a CSR process.
- H_1: the data we observe is *not* from a CSR process.

To use the Monte Carlo tests from Chapter 6 we begin by selecting a random *initial* sampling of points, $\mathcal{P}$, in our domain R to use in computing the point-event distances with. From these we can compute an initial estimate of F called $\hat{F}_{\text{Obs}}(x)$. We then generate B bootstrap samples of *events* under the CSR model. This entails generating new CSR located "events". We can then use the initial sampling of points, $\mathcal{P}$, found earlier, to compute $\hat{F}_b(x)$ the estimated point-event distribution for every bootstrap replica. Using these B bootstrap samples $\hat{F}_b(x)$ we can compute B values of T using

$$T_b = \max_X |\hat{F}_{\text{Obs}}(x) - \hat{F}_b(x)|\,,$$

to obtain an empirical sampling of the statistic T. One expects that if $T > 0$ we should reject the hypothesis H_0 in favor of H_1. Here the expression $T > 0$ is taken to be an inequality in statistical terms. Using our B bootstrap samples we can construct an approximation to the distribution function for the random variable T. Our statistical test for this problem is then to select a value for t^* (depending on a desired probability of making a Type I error) and reject H_0 if $\bar{T} > t^*$, where $\bar{T}$ is the mean value of all samples T_b samples.

Exercise 12.10 (clustering in a Poisson cluster process)

This exercise is done in the MATLAB script `prob_12_10.m`. In this script we first generate data according to a Poisson cluster process. We then perform the processing outlined in Exercise 12.9 discussed above. Based on that discussion, we would like to compute the statistics of the variable T. To do this we draw $B = 100$ random "variables" $\hat{F}_b(x)$. For each each these samples, we compute the T statistic as explained above. When the above script is run it produces a density estimate for T as shown in Figure 64 (left). The mean of this distribution is located at 0.3757, while the mean minus three standard deviations is given by 0.2872. Since this is significantly different than zero we conclude that our process is *not* CSR and possibly represents a cluster process.

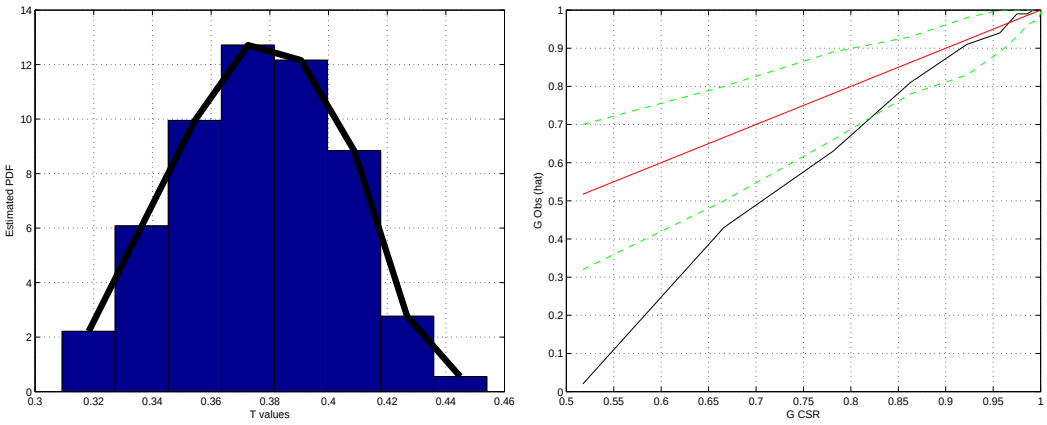

Figure 64: **Left:** The empirical density function for the T statistic computed via Monte Carlo simulation. **Right:** A plot of the observed empirical event-event CDF against one for a CSR process. The 45 degree line is shown in red.

Exercise 12.11 (regularity in a inhibition process)

This exercise can be found in the MATLAB script `prob_12_11.m`. We begin by producing data from an inhibition process and then compare the empirical event-event cumulative distribution function with the known true one under the CSR model.

This exercise is based on the fact that we can assess departure from a CSR model by considering the empirical point-event F or event-event G distribution functions against *known* representations of these distribution functions under the CSR model. If we estimate the empirical nearest neighbor event-event distribution $\hat{G}$ and plot it against an estimate of the theoretical distribution function under CSR we expect to observe a straight line if our process is indeed CSR. To avoid all ambiguity, our procedure is then is to plot the theoretical event-event distribution function under CSR as the *dependent* variable and the observed empirical event-event CDF as the *independent* variable. We could of course do the same thing for the point-event distribution function F. In each case if our data process is indeed CSR we expect the independent variable to exactly predict the dependent variable and the resulting plot should be close to a straight line. If the empirical observed point-event distribution function, $\hat{F}(x)$, lies above or below the 45 degree line different interpretations of the point process results.

- If $\hat{F}(x)$ lies *below* the 45 degree line we have evidence for a *clustering* process.
- If $\hat{F}(x)$ lies *above* the 45 degree line we have evidence for a *regularity* process.

While for the empirical event-event distribution function $\hat{G}(x)$ the opposite interpretation holds. That is

- If $\hat{G}(x)$ lies *below* the 45 degree line we have evidence for a *regularity* process.

- If $\hat{G}(x)$ lies *above* the 45 degree line we have evidence for a *clustering* process.

When the above MATLAB script is run it produces the plot shown in Figure 64 (right). From this plot we see that the estimated $\hat{G}(x)$ lies *below* the 45 degree line which gives us an indication that our process is a regularity process which is consistent with how the points were generated in that points "too close" together were not allowed.

References

[1] W. R. Gilks. *Markov Chain Monte Carlo In Practice*. Chapman and Hall/CRC, 1999.

[2] T. L. Markov chains for exploring posterior distributions. *The Annals of Statistics*, 22:1701–1762, 1994.