NTNU
Norwegian University of
Science and Technology

Conditional Sampling from a Gamma Distribution given Sufficient Statistics

Marius Fagerland

Master of Science in Physics and Mathematics
Submission date: June 2016
Supervisor: Bo Henry Lindqvist, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Problem description

- Give an introduction to stochastic simulation in parametric distributions.
- Give an introduction to sufficiency of statistics in parametric models.
- Present and study algorithms for simulation of samples from conditional distributions of data given sufficient statistics, in particular the gamma distribution.
- Illustrate the theory by simulated examples.

NTNU, January 18, 2016

Bo Lindqvist
Supervisor

Preface

The motivation for this paper comes from my specialization project. From my specialization project, I could use the Gibbs sampler to study the gamma distribution and its sufficient statistics. This way I could use the Gibbs sampler to compare other sampling methods for the gamma distribution. Algorithm 1 became a particular interest because of the NTNU's previous research of this algorithm.

For guiding me through the process, I would like to thank my supervisor Bo Henry Lindqvist.

Abstract

This thesis is an analysis of conditional sampling from a gamma distribution given sufficient statistics. Several sampling algorithms are considered. An algorithm similar to direct sampling is discussed in particular. This algorithm uses parameter adjustments to meet conditions of sufficient statistics. However, this algorithm is influenced by a pivotal condition. How this condition affects algorithm 1 is presented. A Gibbs sampler is assumed to give correct samples, and will be used in comparison to the other samplers. Several data sets are used, and all of them follow the case of 3 data points.

Sammendrag

Denne oppgave er en analyse av betinget simulering av data fra en gammafordeling gitt suffisiente observatorer. Flere algoritmer for å generer data er anvendt. En algoritme som likner på direkte generering av data er sett på spesielt. Denne algoritmen justerer parameterne slik at kravene til suffisientene er møtt. En betingelse til denne algoritmen kalles for pivotal betingelsen. Denne oppgaven ser nærmere på hvordan denne betingelsen påvirker denne algoritmen. Gibbs sampler er en annen algoritme som er brukt. Denne er antatt å gi riktige data. For å teste de ulike algoritmene er flere datasett benyttet. Alle datasett har benyttet 3 datapunkter.

Contents

1 Introduction

In this paper sampling from a gamma distribution given sufficient statistics is studied. The gamma distribution is often used in lifetime analysis [14] and is a very flexible distribution. This distribution will be introduced later. First, the subject of sampling given sufficient statistics will be presented. Information about this topic can be found in my specialization project.

A statistic is a function that returns a summary of the data. Examples of this can be mean value and standard deviation of the sample. Let's assume our data is from a distribution depending on some parameters Θ. The statistics we are interested in are those who contain information about the parameters Θ. This leads us to sufficient statistics. A sufficient statistic is a statistic that captures all information about the parameters Θ and discards the rest. From [2] we have the following.

Definition 1 *(Sufficient statistic definition) A statistic $T(\boldsymbol{X})$ is sufficient statistic for Θ if the conditional distribution of sample $\boldsymbol{X}$ gives the value for $T(\boldsymbol{X})$ does not depend on Θ.*

The meaning behind this is that we can use the information from the statistics to generate new samples, and generate new samples with the same sufficient statistics. For instance, the sufficient statistics of gamma distribution are the sum and product of a sample. This will be shown later. By generating new samples given sufficient one can guarantee that the new samples will have the same sum and product as the original sample. In this paper several ways to simulate new samples from gamma distribution given sufficient statistics will be studied.

The main algorithm of interest is denoted algorithm 1. The specifics of this algorithm will be introduced later. This algorithm has a particular condition to be sure to generate samples from the correct distribution. The condition is called a pivotal condition and is as follows. The sufficient statistics $\tau(u,\theta)$ depends only on u through a function $r(u)$. The function $r(u)$ has a unique representation by solving $\tau(u,\theta) = t$. This condition will be studied to see if it has any effect on the gamma distribution. The theory for this can be found in paper [10].

To analyze the algorithms, several goodness of fit tests will be applied. Test statistics will be used to determine if the samples are from the correct distribution. However since the gamma distribution is flexible, there might not be easy to determine if samples are from the wrong distribution. Furthermore, a goodness of fit test will be used to determine differences in the samples between the different algorithms.

2 Introduction to theory

In this chapter the gamma distribution and its sufficient statistics will be introduced. Furthermore, the methodology of the different samples will be shown.

2.1 Gamma distribution

From [14] consider a number of events that follow a homogeneous Poisson process with rate parameter $1/\beta$. Denote the time intervals between the events $X_1,...,X_\alpha$. The total time of the events is given by

$$X = X_1 + \cdots + X_\alpha.$$

Then X is gamma distributed with parameters (α,β). The probability density function for the gamma distribution is given as follows.

$$f(x;\alpha,\beta) = \frac{x^{\alpha-1}}{\Gamma(\alpha)\beta^\alpha} e^{\frac{-x}{\beta}} \tag{2.1}$$

The function $\Gamma(\alpha)$ is called the gamma function and is given by

$$\Gamma(\alpha) = \int_0^\infty t^{\alpha-1} e^{-t} dt.$$

Furthermore $\alpha > 0$, $\beta > 0$ and $X > 0$. The equation 2.1 is also the definition of a gamma distribution when $\alpha > 0$ is not an integer. Next the sufficient statistic in a gamma distribution is found.

Sufficient statistics in gamma distribution

To find the sufficient statistics of a gamma distribution one can find the joint distribution and use the factorization theorem. For the gamma distribution the joint distribution becomes

$$f(x_1,...,x_n;\alpha,\beta) = \prod_{i=1}^{n} \frac{{x_i}^{\alpha-1}}{\Gamma(\alpha)\beta^\alpha} e^{-\frac{x_i}{\beta}}.$$

This expression can be rearranged to

$$\frac{e^{-\frac{\sum_{i=1}^n x_i}{\beta}}}{\Gamma(\alpha)^n \beta^{\alpha n}} \prod_{i=1}^{n} \frac{{x_i}^\alpha}{x_i}.$$

From the factorization theorem the sufficient statistics are $\left(\sum_{i=1}^{n} x_i, \prod_{i=1}^{n} x_i\right)$. By transforming the sufficient statistics they become $\left(\frac{1}{n}\sum_{i=1}^{n} x_i, \frac{\left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}}}{\frac{1}{n}\sum_{i=1}^{n} x_i}\right)$. This transformation can be done because of [1]. These sufficient statistics will make it easier for some of the samplers. With the sufficient statistics, we can generate samples from the distribution.

2.2 Algorithm 1

Algorithm 1 is the first algorithm to be used for generating samples. This algorithm is found in paper [8]. The general setup is that we have data X and sufficient statistics T. An assumption is a random vector U with known distribution. Furthermore consider functions $(\chi(\cdot,\cdot), \tau(\cdot,\cdot))$ such that

$$(\chi(U,\theta), \tau(U,\theta)) \overset{\theta}{\sim} (X, T).$$

For instance, the vector U could be from a uniform distribution between 0 and 1, while $\chi(U,\theta)$ is the inverse of the cumulative distribution and $\tau(U,\theta)$ is the sufficient statistics. This is the idea for algorithms 1. To estimate θ one could solve $\tau(U,\theta) = t$, where t is the values for sufficient statistics for a data set. An overview of algorithm 1 is shown in section 2.6. As mentioned in the introduction this algorithm has a condition to guarantee samples from right distribution. The condition is as follows. $\tau(u,\theta)$ is only dependent on u in a function $r(u)$. However, this condition does not need to be met to give samples from the right distribution. To simulate from a gamma distribution, a trick is used. It is easier to simulate from a gamma distribution where $\beta = 1$. Hence, we have that

$$Y = \frac{X}{\beta},$$

where X has the distribution in equation 2.1. Then a gamma sample can be found by first simulating Y and then set $X = \beta Y$. Now the $\chi(U,\theta)$ becomes

$$\chi(u,\alpha,\beta) = \beta F^{-1}(u;\alpha), \tag{2.2}$$

where

$$F(y;\alpha) = \frac{\gamma(y;\alpha,1)}{\Gamma(\alpha)}.$$

The function γ is the incomplete gamma function. The inverse function $F^{-1}(y;\alpha)$ cannot be found analytically and must be obtained numerically. Furthermore the $\tau(U,\theta)$ becomes

$$\tau_1(u,\alpha,\beta) = \frac{\beta}{n}\sum_{i=1}^{n} F^{-1}(u;\alpha). \tag{2.3}$$

$$\tau_2(u,\alpha,\beta) = \frac{\left(\prod_{i=1}^{n} F^{-1}(u;\alpha)\right)^{\frac{1}{n}}}{\frac{1}{n}\sum_{i=1}^{n} F^{-1}(u;\alpha)} \tag{2.4}$$

Since τ_1 and τ_2 is defined the pivotal condition can be studied. As mentioned in the introduction the pivotal condition is met when the sufficient statistics $\tau(u,\theta)$ depends only on u through a function $r(u)$. This can be shown as $\tau(u,\theta) = \tilde{\tau}(r(u),\theta)$. From equations 2.3 and 2.4 the pivotal condition is not met for a gamma distribution. To generate sample one would first solve the equation $\tau_2(\alpha) = t_2$ to estimate α. Furthermore, solve equation $\tau_1(\hat{\alpha},\beta) = t_1$ for β. Then use equation equation 2.2 to generate a sample. An overview of this algorithm can be found in section 2.6.

2.3 Weighted sampling, algorithm 2

This algorithm is very similar to algorithm 1, and is also found in paper [8]. However weights $W_t(u)$ for the vector U is proposed. This is to yield samples from the right distribution. The weights can be found by

$$W_t(u) = \pi\{\hat{\theta}(u,t)\}|det\partial_t\hat{\theta}(u,t)| = |\frac{\pi(\theta)}{det\partial_\theta\tau(u,\theta)}|_{\theta=\hat{\theta}(u,t)}.$$

The function $\pi(\theta)$ can be chosen freely. Some popular choices are a constant and Jeffreys prior. To generate a sample one would generate a sample V proportional to $W_t(u)f(u)$ and use this to get a sample from $\chi(V,\hat{\theta}(V,t))$. An overview of the algorithm can be found in section 2.6. For the Gamma distribution we have that

$$det\partial_\theta\tau(u,\theta) = \begin{vmatrix} \frac{\partial\tau_1}{\partial\alpha} & \frac{\partial\tau_1}{\partial\beta} \\ \frac{\partial\tau_2}{\partial\alpha} & \frac{\partial\tau_2}{\partial\beta} \end{vmatrix} = \left|\frac{\partial\tau_1}{\partial\beta}\frac{\partial\tau_2}{\partial\alpha}\right|$$

To find $\frac{\partial\tau_2}{\partial\alpha}$ a trick is proposed. The trick is as follows.

$$\frac{\partial\log\tau_2}{\partial\alpha} = \frac{\partial\tau_2}{\partial\alpha}\frac{1}{\tau_2}.$$

Furthermore we get

$$\frac{\partial\tau_2}{\partial\alpha} = \tau_2\frac{\partial\log\tau_2}{\partial\alpha}.$$

From equation 2.4 we get the equation

$$\log \tau_2 = \frac{1}{n}\sum_{i=1}^{n} \log F^{-1}(u_i, \alpha) - \log \frac{1}{n}\sum_{i=1}^{n} F^{-1}(u_i, \alpha).$$

Then $\frac{\partial \log \tau_2}{\partial \alpha}$ becomes

$$\frac{\partial \log \tau_2}{\partial \alpha} = \frac{1}{n}\sum_{i=1}^{n} \frac{h(u_i, \alpha)}{F^{-1}(u_i, \alpha)} - \frac{\sum_{i=1}^{n} h(u_i, \alpha)}{\sum_{i=1}^{n} F^{-1}(u_i, \alpha)},$$

where $h(u_i, \alpha)$ is the derivative of $F^{-1}(u_i, \alpha)$ with respect to α. The variables t_1 and t_2 is the observed values of τ_1 and τ_2. Finally $det\partial_\theta \tau(u, \theta)$ becomes

$$det\partial_\theta \tau(u, \theta) = \frac{t_1 t_2}{\beta} \frac{\partial \log \tau_2}{\partial \alpha}.$$

The function $\pi(\theta)$ can be chosen such that $t_1 t_2/\beta$ vanishes when calculating weights. Another addition to the π function is to set it to 0 for specific α-values. In our case we set $\pi = 0$ for $\alpha < 0.2$ and $\alpha > 200$. This is because the inverse gamma cumulative function becomes difficult for small α-values. This can be seen in figure 4.2. The upper limit is to narrow down the search limit for a numerical solution. This leads to weights as shown below.

$$W_t = \frac{1}{\frac{1}{n}\sum_{i=1}^{n} \frac{h(u_i,\alpha)}{F^{-1}(u_i,\alpha)} - \frac{\sum_{i=1}^{n} h(u_i,\alpha)}{\sum_{i=1}^{n} F^{-1}(u_i,\alpha)}}, \quad 0.2 \leq \alpha \leq 200.$$

This sampling algorithm is very similar to algorithm 1. In the u values are weighted and denoted v. To find the v values, a Metropolis-Hastings step is used. Since a uniform distribution is used to draw v values, the alpha value in Metropolis-Hastings is found by

$$\alpha = \min(1, \frac{W_t(u_{proposal})}{W_t(u_{current})}).$$

Then the rest is as with algorithm 1. A sample X is found by βY.

2.4 Gibbs sampler with Metropolis-Hastings step.

Since the values of the sufficient statistics cannot differ from the original data set, the sampler must generate new samples with the same values for the sufficient statistics. Given a sample with N points and number of sufficient statistics, one can draw three arbitrary points and draw a new value for one of the points given the two others. By setting one of the points, one can calculate the two others. This is where the Metropolis-Hastings step is used to draw a new value. Further, when this new value is drawn the two other values must be adjusted to fulfill the sufficient

statistics. This should be done several times for each sample. The reason for this is to make the samples independent.

Assume we have chosen three points (X_1, X_2, X_3) from our sample. It is these three points that are going to be updated. The idea is to draw a new value for one of them. To draw a new value we need to use the transformation of variables from appendix 6.2. The transformation is

$$(Z_1, Z_2, Z_3) = (\sum_{i=1}^{3} X_i, \prod_{i=1}^{3} X_i, X_3).$$

This will yield the probability density $f_{Z_1,Z_2,Z_3}(z_1, z_2, z_3)$. However we are interested in $f_{Z_3|Z_1,Z_2}(z_3|z_1, z_2)$, where $Z_3 = X_3$ is the variable drawn from this distribution. Fortunately from Bayes rule [2] the distribution becomes

$$f_{Z_3|Z_1,Z_2}(z_3|z_1, z_2) \propto f_{Z_1,Z_2,Z_3}(z_1, z_2, z_3).$$

Further we continue with transformation of variables. The determinant of the Jacobian matrix then becomes

$$|J| = X_3(X_2 - X_1).$$

We also have the following relations from the tranformation, where we have put $a = Z_1$ and $b = Z_2$

$$\begin{aligned} X_1 + X_2 + X_3 &= a \\ X_1 + X_2 &= a - X_3 \\ X_1 \cdot X_2 \cdot X_3 &= b \\ X_1 \cdot X_2 &= \frac{b}{X_3}. \end{aligned}$$

Then X_1 and X_2 are the roots of

$$X^2 - (a - X_3)X^2 + \frac{b}{X_3} = 0,$$

which are

$$X_{(1,2)} = \frac{(a - X_3) \pm \sqrt{(a - X_3)^2 - \frac{4b}{X_3}}}{2}.$$

From this the jacobian becomes

$$|J| = X_3\sqrt{(a - X_3)^2 - \frac{4b}{X_3}}.$$

From paper [12] the determinant of the inverse jacobian is

$$|J|^{-1} = \frac{1}{X_3\sqrt{(a-X_3)^2 - \frac{4b}{X_3}}}.$$

We also have that

$$f_{X_1,X_2,X_3}(x_1,x_2,x_3) = \prod_{i=1}^{3} \frac{{x_i}^{\alpha-1}}{\Gamma(\alpha)\beta^\alpha} e^{-\frac{x_i}{\beta}}., \quad 0 \leq x_i \leq a \text{ for all } i,$$

the density to draw new values for X_3 becomes

$$\Pi \propto \frac{f_{X_1,X_2,X_3}(x_1,x_2,x_3)}{X_3\sqrt{(a-X_3)^2 - \frac{4b}{X_3}}}.$$

This is where the Metropolis-Hastings step is used. The requirements for this density are listed below.

$$0 \leq X_3 \leq a \tag{2.5}$$

$$(a-X_3)^2 - \frac{4b}{X_3} \geq 0 \tag{2.6}$$

$$(a-X_3) + \sqrt{(a-X_3)^2 - \frac{4b}{X_3}} \leq a. \tag{2.7}$$

So when we draw from a proposal density like a uniform distribution, we need to make sure that the new X_3 fulfills these requirements. If the requirements are not met, then the probability of accepting this new value is zero. If the requirements are met then, the acceptance probability is defined as,

$$\tilde{\alpha} = \min\left(1, \frac{\Pi(X_{prop})}{\Pi(X_{curr})}\right). \tag{2.8}$$

From this we see that the term $f_{X_1,X_2,X_3}(x_1,x_2,x_3)$ will vanish. This is because of the sufficient statistics where sum and product should be the same for all samples. Which one of the data points who will be X_1, X_2 and X_3 can be chosen at random. An overview of the algorithm is section 2.6. A related algorithm for Gibbs sampling for the gamma distribution is given in [12]. The idea of considering three variables at a time is, however, apparently new.

2.5 Naive sampler

The idea behind this sampler is to use only the samples where the sufficient statistics match. It is a lot easier to simulate from a gamma distribution when the samples are independent of sufficient statistics. To create samples with the right distribution

one would only use samples where the sufficient statistics match the original data. This can be difficult in practice. To make this easier, we allow a small error in sum and product of the samples.

$$|\sum_{i=1}^{n} x_i - t_1| < \epsilon_1$$
$$|\prod_{i=1}^{n} x_i - t_2| < \epsilon_2,$$

where $\mathbf{x} = (x_1, ..., x_n)$ is the generated samples, t_1 and t_2 is the sum and product of original data. Finally ϵ_1 and ϵ_2 are the allowed errors. Even with well assigned ϵ values the acceptance rate for a sample might be very low. An overview of the algorithm can be found in 3.

2.6 Overview of the samplers

In this section, the overview of the different samplers is shown. They only show the very basics of each algorithm.

Algorithm 1 Generate new samples with algorithm 1.

Draw U from a known distribution.
Find θ such that $\tau(U, \theta) = t$.
A sample is given by $X_t(U) = \chi\{U, \hat{\theta}(U, t)\}$.

Algorithm 2 Generate new samples with algorithm 2.

Draw V from a distribution proportional to $W_t(u)f(u)$.
Find θ such that $\tau(V, \theta) = t$.
A sample is given by $X_t(V) = \chi\{V, \hat{\theta}(V, t)\}$.

Algorithm 3 Generate new samples with naive sampling.

Draw X from $Gamma(\alpha, \beta)$.
Accept sample if $|\sum_{i=1}^{n} x_i - t_1| < \epsilon_1$ and $|\prod_{i=1}^{n} x_i - t_2| < \epsilon_2$

Algorithm 4 Generate new samples with Gibbs sampler.

Draw 3 arbitrary indices from $\{1, 2, ..., n\}$
Caclulate sum a and product b of the corresponding X's.
Draw a proposal X_3 from $U[0, a]$.
Check requirements in equations 2.5, 2.6 and 2.7.
Calculate X_1 and X_2.
Calculate $\tilde{\alpha}$ from equation 4.2.
Accept with probability $\tilde{\alpha}$
Repeat from top desired times to make sample approximately independent.

3 Goodness of fit

In this chapter goodness of fit testing will be introduced. A goodness of fit is how well the observed data match a model. First test statistics will be introduced and how data fits a data model. Furthermore, a goodness of fit is presented to compare differences between algorithms.

3.1 Test statistics

A test statistic is a function to test a certain attribute. Most of this information can also be found in my specialization project. For a gamma distribution, the Cramer-von Mises test [15] can be used. This test judges the goodness of fit of a cumulative distribution to an empirical distribution. The test is as follows.

$$\omega^2 = \frac{1}{12n}\sum_{i=1}^{n}\left(\frac{2i-1}{2n} - F(x_i;\hat{\alpha},\hat{\beta})\right)^2,$$

where x_i values are ordered. Here $\hat{\alpha}$ and $\hat{\beta}$ are estimated from samples. One way to do this is to find the maximum likelihood estimators. How to find maximum likelihood estimators can be found in appendix 6.2. In R you an use the built in function $fitdistr$ to calculate maximum likelihood estimators. Documentation of this is shown in [6]. A test statistic is used in hypothesis testing. This is where you test two hypotheses against each other. The first hypothesis is denoted the null hypothesis and the second hypothesis denoted alternative hypothesis. The null hypothesis is assumed to be correct. In this case, the null hypothesis is that the original samples come from a gamma distribution. The alternative hypothesis is that the original samples do not come from a gamma distribution. A measurement of when to reject the null hypothesis is a p-value. The definition and calculation of a p-value are found in appendix 6.2. In this case, the null hypothesis is expected to hold. This is because we are studying samples with only 3 data points. Hence, this gives room for the gamma distribution model to fit the data points.

3.2 Expectation of a chosen function ϕ

To do goodness of fit testing one could calculate $E\{\phi(X)|T = t\}$. This can be used to compare the different algorithms with each other. The $\phi(X)$ is chosen to give a value for a given sample X. In our samples, we assume that each sample has 3 data points. Hence, ϕ functions involving 3 data points have been chosen. The first ϕ function to be tested is

$$\phi(X) = \frac{1}{3}\sum_{i=1}^{3} I(x_i > a), \tag{3.1}$$

where a is a chosen value. The second chosen ϕ function is

$$\phi(X) = I(\frac{x_1 x_2}{x_3} > a). \tag{3.2}$$

Since we have the product of x_1 and x_2 in the numerator, this ϕ function might not be a 1-1 function. A plot of this function given a data set is shown in figure 3.1. The third ϕ function is

$$\phi(X) = I\left(\left(\frac{x_1}{x_2}\right)^{x_3} > a\right). \tag{3.3}$$

A plot of this function given a data set is shown in figure 3.1. From the figure, we see that this function is almost a 1-1 function. This could make it easier to compare samples. To calculate $E\{\phi(X)|T = t\}$ one would calculate $\phi(X)$ for every sample and find the average value.

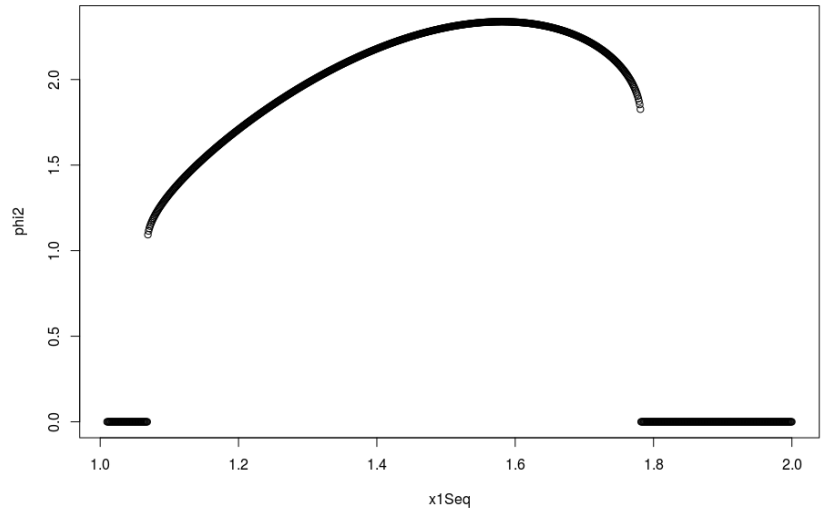

FIGURE 3.1: Plot of second ϕ function for x_1 values given sufficient statistics. Those values that are zero are invalid values. Data set shown in table 5.6.

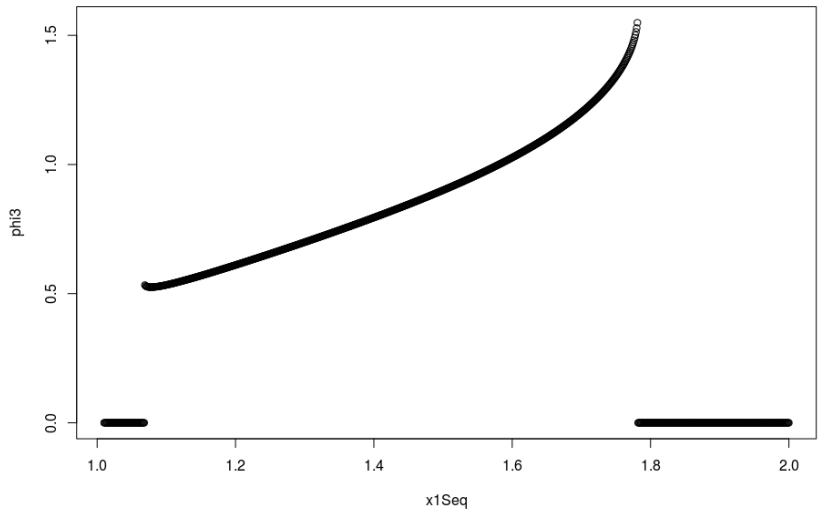

FIGURE 3.2: Plot of third ϕ function for x_1 values given sufficient statistics. Those values that are zero are invalid values. Data set shown in table 5.6.

4 Implementation

This chapter is about implementation and discussion about how to implement sections of the samplers.

4.1 Inverse of cumulative Gamma function.

The inverse of the cumulative gamma distribution cannot be found analytically and must be found numerically. Since we would want to find a value t such that

$$F^{-1}(u) = t,$$

for a given $u \in U[0,1]$ one can instead find t such that

$$F(t) = u.$$

In R programming it can be easier to use a minimizer function to minimize

$$|F(t) - u|, \tag{4.1}$$

since the minimum of this will be zero. This can be solved by other methods too. By keeping u fixed and plotting the inverse cumulative distribution with respect to α we get the figure shown in 4.1. Since this is a 1-1 function equation 4.1 only has one solution.

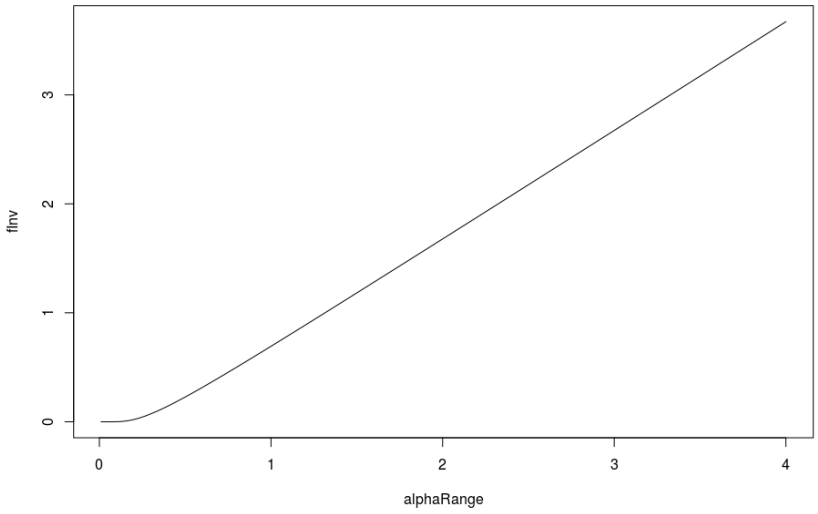

FIGURE 4.1: Plot of inverse cumulative distribution function with respect to α. This is plotted for $u = 0.5$.

4.2 Estimation of derivate of inverse gamma cumulative function.

To find the derivate of the inverse gamma cumulative function one can use first order forward difference. This is given as

$$f'(x) \approx \frac{f(x+h) - f(x)}{h},$$

where h is a small step forward. This scheme can be found in [7] and [16].The estimated derivate of the inverse gamma cumulative function for a fixed u is shown in figure 4.2.

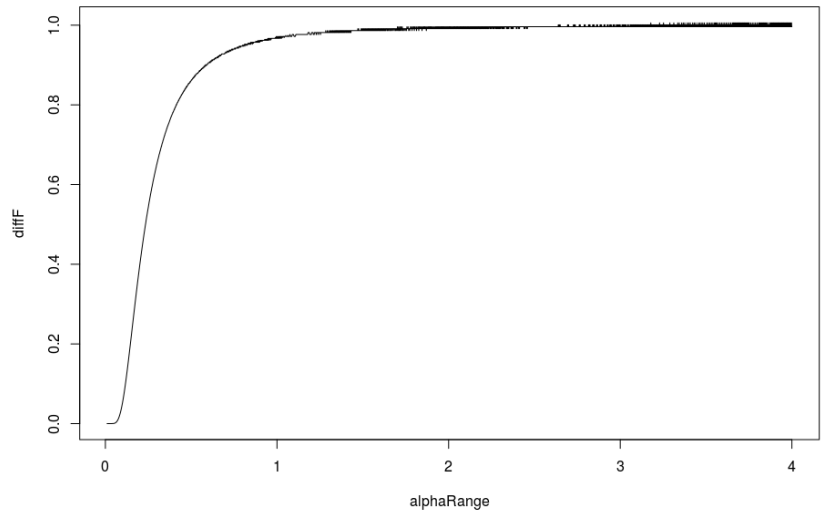

FIGURE 4.2: Plot of derivative of inverse cumulative distribution function with respect to α. This is plotted for $u = 0.5$.

4.3 Solving for alpha and beta.

To solve for alpha we have the same situation as the inverse of the cumulative Gamma function. From the figure 4.3 we see that the function τ_2 is a 1-1 function. Hence alpha can be found numerically. α can be found by minimizing

$$|\tau_2 - n\frac{\left(\prod_{i=1}^{n} F^{-1}(u_i, \alpha)\right)^{1/n}}{\sum_{i=1}^{n} F^{-1}(u_i, \alpha)}|, \tag{4.2}$$

and β can be found by

$$\beta = n\frac{\tau_1}{\sum_{i=1}^{n} F^{-1}(u_i, \alpha)}. \tag{4.3}$$

The alpha to be used in equation 4.3 is the one from equation 4.2

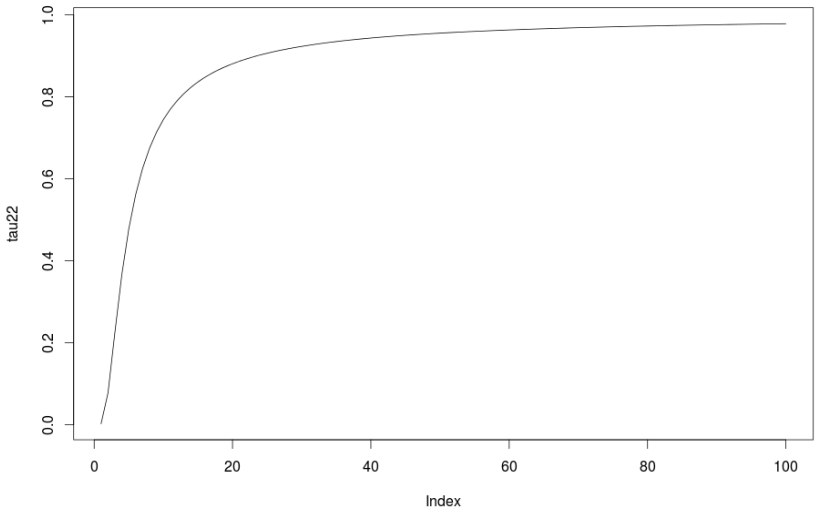

FIGURE 4.3: Plot of τ_2 function for α-values.

5 Results and discussion of samplers

In this chapter, the results from the different algorithms will be presented.The Gibbs algorithm is assumed to give right samples. This is because this algorithm as been used in my specialization project and a modified version in paper [12]. Thus, this algorithm will be the one to compare the other algorithms. The results have been split into several data sets. Each data set has 10^5 samples for each algorithm and for the ϕ functions in equations 3.1, 3.2 and 3.3.

5.1 Data sets

Some of the data sets have used a gamma distribution to generate data. One data set have been selected out of memory to try to create other results. And final data sets are from a ball bearing failure data set.

5.1.1 Data set 1

The first data set is shown in table 5.1. From this table we see that all the data points are below 1. This may cause sensitivity issues for the naive sampler. The reason for this is that the product of the sample becomes a low number. Hence, the ϵ value needs to be lower than usual. The results for this data set are given in tables 5.3, 5.4 and 5.5. From these tables we see that the algorithm 1 is the one closest to the Gibbs algorithm. For this data set algorithm 1 might not be influenced by not fulfilling the pivotal condition. The naive sampler gets close to the Gibbs sampler for some of the cases. The reason for the variance might be because of the ϵ-value. The variance in algorithm 2 may result from numerical approximations. The p-values for this data set are shown in table 5.2, and plots of the Cramer-von Mises test statistic are shown in figures 5.1, 5.2, 5.3 and 5.4. The figures are somewhat similar to each other.

Table 5.1: Data set 1 generated from a gamma distribution.

0.5772030	0.4340237	0.4212959

Table 5.2: P-values with Cramver-von Mises test statistic for data set 1.

Algorithm	Cramer-von Mises
Algorithm 1	0.155
Algorithm 2	0.178
Gibbs	0.155
Naive	0.153

TABLE 5.3: $E[\phi(X)|T = t]$ for data set 1 with ϕ as shown in equation 3.1 and $a = 0.5$

Algorithm	$E[\phi(X)\|T = t]$
Algorithm 1	0.4159
Algorithm 2	0.4087
Gibbs	0.4163
Naive	0.3974

TABLE 5.4: $E[\phi(X)|T = t]$ for data set 1 with ϕ as shown in equation 3.2 and $a = 0.5$

Algorithm	$E[\phi(X)\|T = t]$
Algorithm 1	0.4497
Algorithm 2	0.5000
Gibbs	0.4510
Naive	0.4372

TABLE 5.5: $E[\phi(X)|T = t]$ for data set 1 with ϕ as shown in equation 3.3 and $a = 1.0$

Algorithm	$E[\phi(X)\|T = t]$
Algorithm 1	0.5015
Algorithm 2	0.4985
Gibbs	0.5020
Naive	0.4999

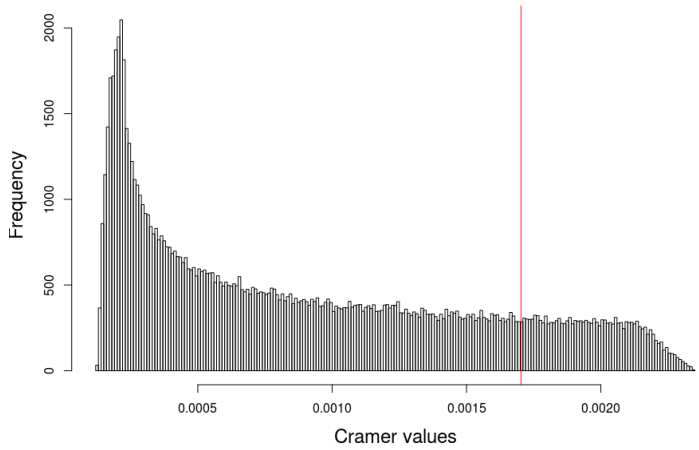

FIGURE 5.1: Plot of Cramer-Von Mises test statistics for naive sampler of data set 1. The vertical line is the observed Cramer-Von Mises value from original data set.

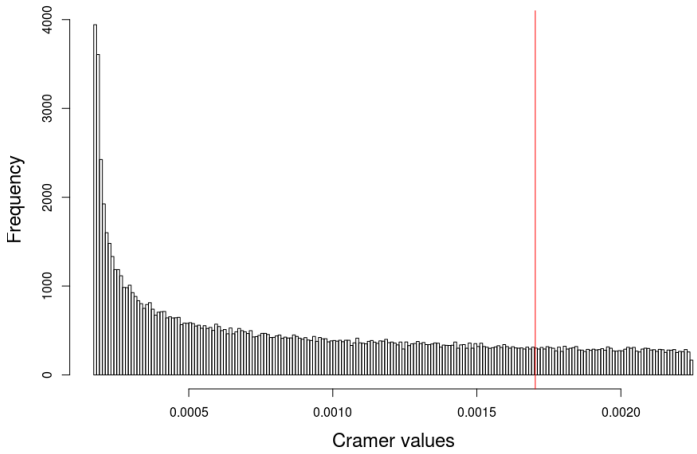

FIGURE 5.2: Plot of Cramer-Von Mises test statistics for Gibbs sampler of data set 1. The vertical line is the observed Cramer-Von Mises value from original data set.

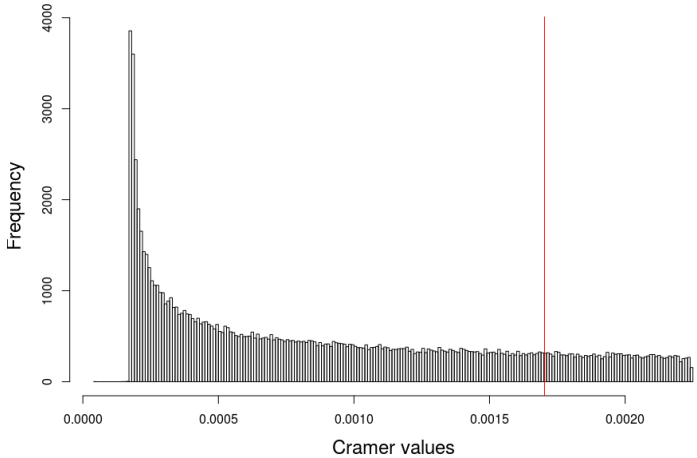

FIGURE 5.3: Plot of Cramer-Von Mises test statistics for algorithm 1 of data set 1. The vertical line is the observed Cramer-Von Mises value from original data set.

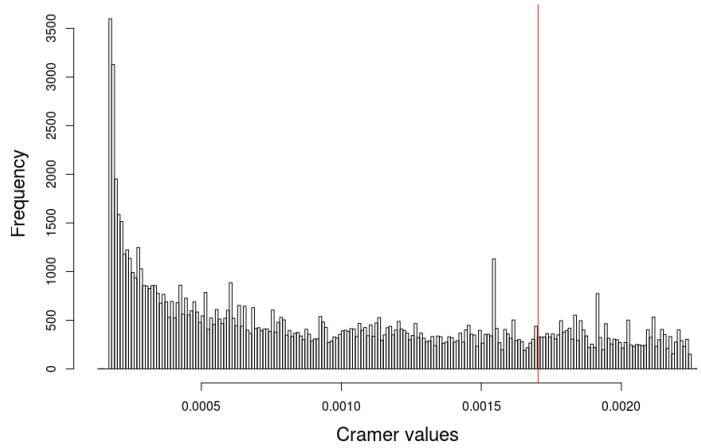

FIGURE 5.4: Plot of Cramer-Von Mises test statistics for algorithm 2 of data set 1. The vertical line is the observed Cramer-Von Mises value from original data set.

5.1.2 Data set 2

This is another data set from a gamma distribution. The data points are shown in table 5.6. For this data set, the data points are above 1. Hence, the ϵ-value might not be sensitive to the product of the data set. The results for this data set are shown in tables 5.8, 5.9 and 5.10. From these tables, we see that the naive sampler is much closer to the Gibbs sampler. As mentioned earlier the ϵ-value is not as sensitive for the product of data set. The variance for algorithm 2 in this data set is also high compared to the others. Algorithm 1 is for this data set also very close to the Gibbs sampler. However not as much as the naive sampler. The p-values are shown in table 5.7. All of the p-values are above the significance levels. Hence, the null hypothesis is kept. The p-values are very close to each other. It is only algorithm 2 which is lower than the rest. From figure 5.8, we see that algorithm 2 generates some extreme values.

TABLE 5.6: Data set 2 generated from a gamma distribution.

1.621813	1.059797	1.554334

Table 5.7: P-values with Cramver-von Mises test statistic for data set 2.

Algorithm	Cramer-von Mises
Algorithm 1	0.183
Algorithm 2	0.118
Gibbs	0.185
Naive	0.185

Table 5.8: $E[\phi(X)|T = t]$ for data set 2 with ϕ as shown in equation 3.1 and $a = 1.4$

Algorithm	$E[\phi(X)\|T = t]$
Algorithm 1	0.4953
Algorithm 2	0.5153
Gibbs	0.4963
Naive	0.4961

Table 5.9: $E[\phi(X)|T = t]$ for data set 2 with ϕ as shown in equation 3.2 and $a = 1.3$

Algorithm	$E[\phi(X)\|T = t]$
Algorithm 1	0.5294
Algorithm 2	0.5624
Gibbs	0.5336
Naive	0.5307

Table 5.10: $E[\phi(X)|T = t]$ for data set 2 with ϕ as shown in equation 3.3 and $a = 1.0$

Algorithm	$E[\phi(X)\|T = t]$
Algorithm 1	0.5024
Algorithm 2	0.5131
Gibbs	0.5008
Naive	0.5009

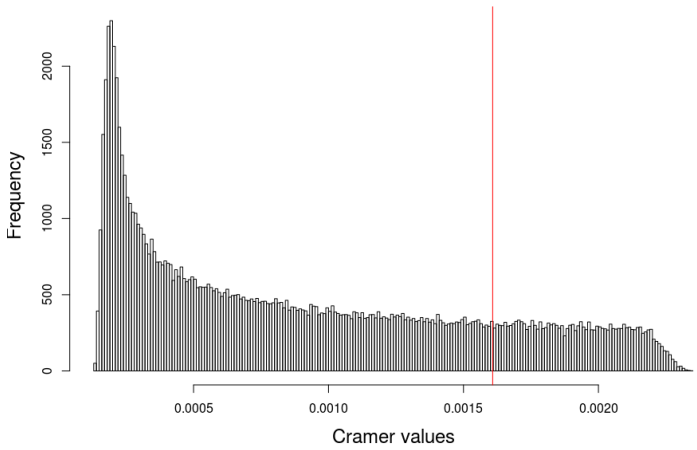

Figure 5.5: Plot of Cramer-Von Mises test statistics for naive sampler of data set 2. The vertical line is the observed Cramer-Von Mises value from original data set.

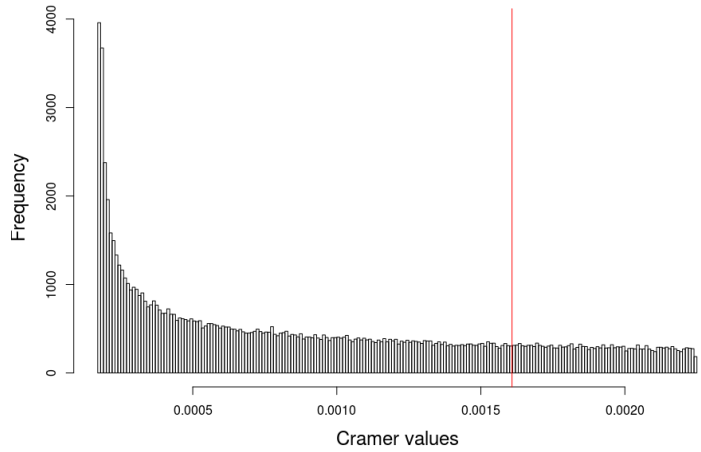

FIGURE 5.6: Plot of Cramer-Von Mises test statistics for Gibbs sampler of data set 2. The vertical line is the observed Cramer-Von Mises value from original data set.

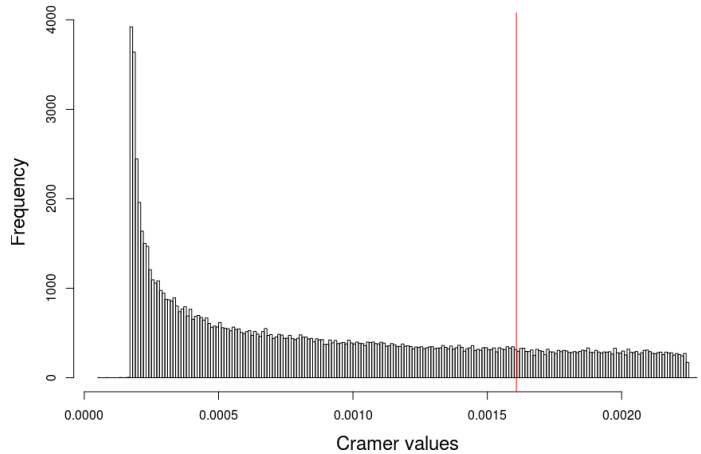

FIGURE 5.7: Plot of Cramer-Von Mises test statistics for algorithm 1 of data set 2. The vertical line is the observed Cramer-Von Mises value from original data set.

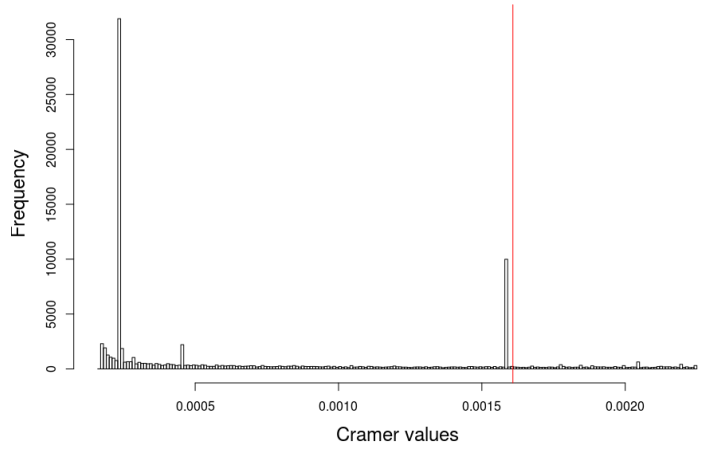

FIGURE 5.8: Plot of Cramer-Von Mises test statistics for algorithm 2 of data set 2. The vertical line is the observed Cramer-Von Mises value from original data set.

5.1.3 Data set 3

This data set consists of integer values from a uniform distribution between 0 and 20. The data set is shown in table 5.11. Because of the high variance in this data set, it creates a problem for the naive sampler. This makes the naive sampler slow and makes it hard to generate many samples. Because of this results from the naive sampler have not been added to the following results. Results for this data set are shown in tables 5.13, 5.14 and 5.15. As seen for the previous data sets algorithm 1 is again close to the Gibbs sampler. While algorithm 2 seems to vary how close it is to the Gibbs sampler. The p-values are shown in table 5.12. From the p-values, the null hypothesis is not rejected for this data set. All of the p-values are above significance level 5% and 10%. Here the test statistic plots are quite similar to each other.

TABLE 5.11: Data set 3

5	15	13

Table 5.12: P-values with Cramver-von Mises test statistic for data set 3.

Algorithm	Cramer-von Mises
Algorithm 1	0.271
Algorithm 2	0.255
Gibbs	0.269
Naive	

Table 5.13: $E[\phi(X)|T = t]$ for data set 3 with ϕ as shown in equation 3.1 and $a = 11.0$

Algorithm	$E[\phi(X)\|T = t]$
Algorithm 1	0.4652
Algorithm 2	0.4557
Gibbs	0.4658
Naive	

Table 5.14: $E[\phi(X)|T = t]$ for data set 3 with ϕ as shown in equation 3.2 and $a = 7.0$

Algorithm	$E[\phi(X)\|T = t]$
Algorithm 1	0.5678
Algorithm 2	0.5678
Gibbs	0.5691
Naive	

Table 5.15: $E[\phi(X)|T = t]$ for data set 3 with ϕ as shown in equation 3.3 and $a = 0.0118$

Algorithm	$E[\phi(X)\|T = t]$
Algorithm 1	0.6519
Algorithm 2	0.6647
Gibbs	0.6547
Naive	

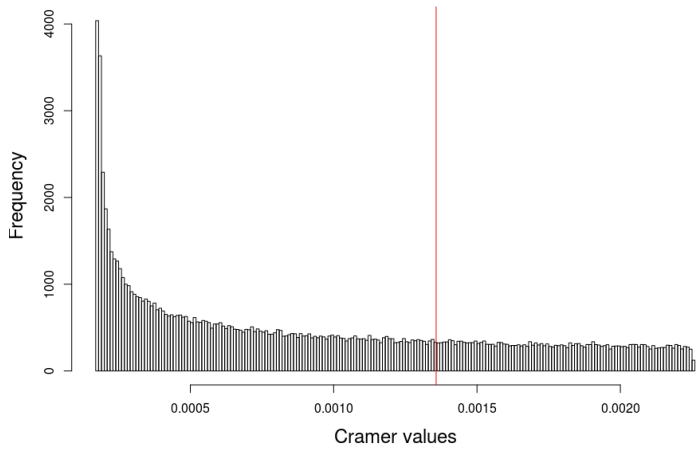

Figure 5.9: Plot of Cramer-Von Mises test statistics for Gibbs sampler of data set 3. The vertical line is the observed Cramer-Von Mises value from original data set.

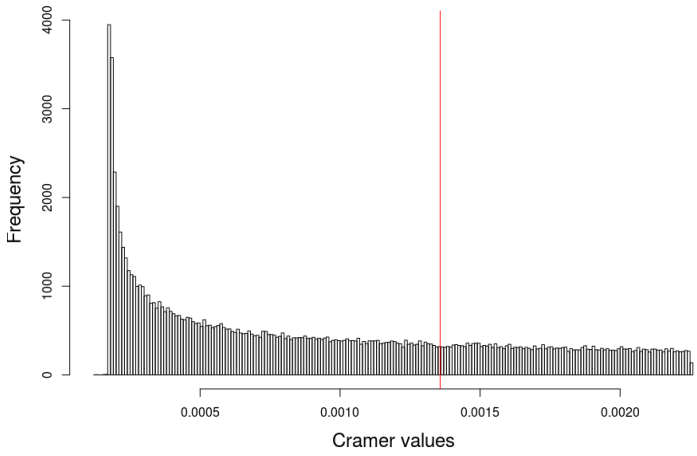

FIGURE 5.10: Plot of Cramer-Von Mises test statistics for algorithm 1 of data set 3. The vertical line is the observed Cramer-Von Mises value from original data set.

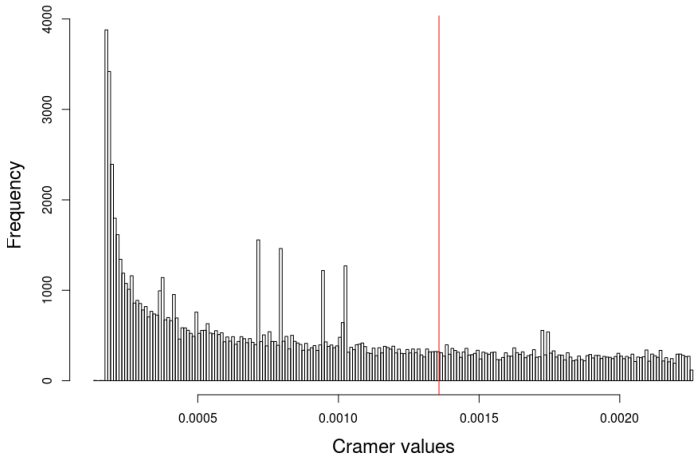

FIGURE 5.11: Plot of Cramer-Von Mises test statistics for algorithm 2 of data set 3. The vertical line is the observed Cramer-Von Mises value from original data set.

5.1.4 Data set 4

This data set retrieved from the ball bearing failure data set. The ball bearing failure data set can be found in [17]. Only three of the data points from the ball bearing data set have been chosen. This is because the focus of this paper is samplers with three data points. The data points chosen are shown in table 5.16. The results for

this data set is shown in tables 5.18, 5.19 and 5.20. From these tables we see that the results are very much the same as in data set 3. The p-values are shown in table 5.17.

Table 5.16: Data set 4 from ball bearing failure data.

17.88	28.92	33.00

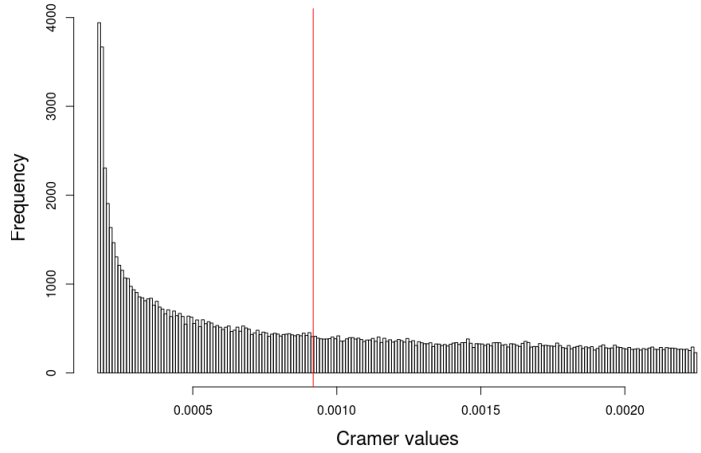

Figure 5.12: Plot of Cramer-Von Mises test statistics for Gibbs sampler of data set 4. The vertical line is the observed Cramer-Von Mises value from original data set.

Table 5.17: P-values with Cramver-von Mises test statistic for data set 4.

Algorithm	Cramer-von Mises
Algorithm 1	0.429
Algorithm 2	0.455
Gibbs	0.427
Naive	

TABLE 5.18: $E[\phi(X)|T = t]$ for data set 4 with ϕ as shown in equation 3.1 and $a = 27.0$

Algorithm	$E[\phi(X)\|T = t]$
Algorithm 1	0.4678
Algorithm 2	0.4995
Gibbs	0.4677
Naive	

TABLE 5.19: $E[\phi(X)|T = t]$ for data set 4 with ϕ as shown in equation 3.2 and $a = 21.0$

Algorithm	$E[\phi(X)\|T = t]$
Algorithm 1	0.5795
Algorithm 2	0.5694
Gibbs	0.5802
Naive	

TABLE 5.20: $E[\phi(X)|T = t]$ for data set 4 with ϕ as shown in equation 3.3 and $a = 0.0118$

Algorithm	$E[\phi(X)\|T = t]$
Algorithm 1	0.5946
Algorithm 2	0.6224
Gibbs	0.5978
Naive	

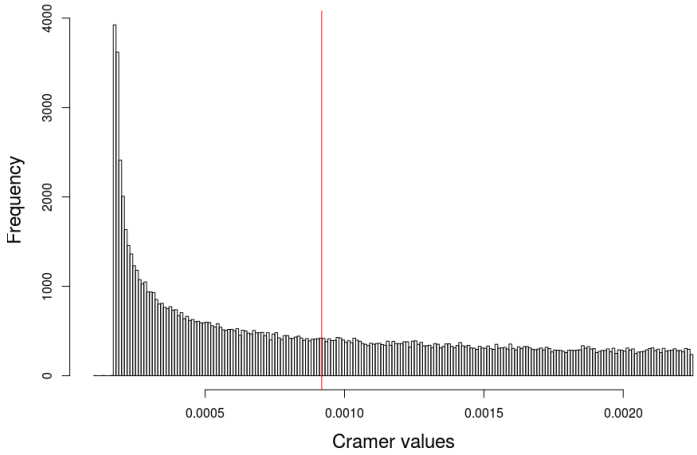

FIGURE 5.13: Plot of Cramer-Von Mises test statistics for algorithm 1 of data set 4. The vertical line is the observed Cramer-Von Mises value from original data set.

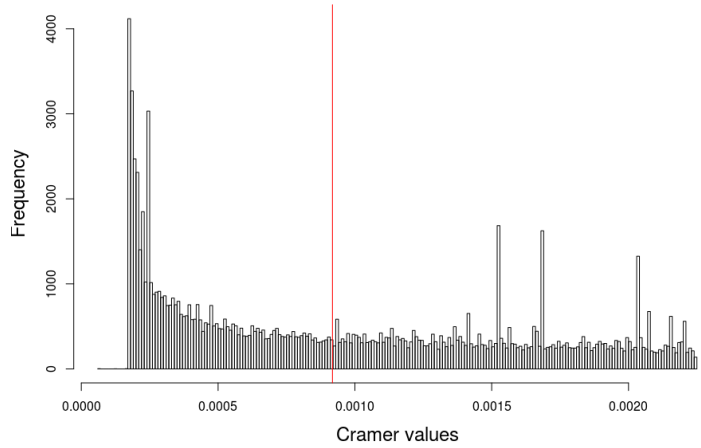

FIGURE 5.14: Plot of Cramer-Von Mises test statistics for algorithm 2 of data set 4. The vertical line is the observed Cramer-Von Mises value from original data set.

5.1.5 Data set 5

This data set is to see what happens when data points are close to each other. The reason for small values is to be able to use the naive sampler too. The data set is shown in table 5.21. Results of calculating $E\{\phi(X)|T = t\}$ are shown in tables 5.23, 5.24 and 5.25. The p-values for this data set are shown in table 5.22. The null hypothesis won't be rejected for this data set either.

TABLE 5.21: Data set 5.

0.40	0.42	0.43

TABLE 5.22: P-values with Cramver-von Mises test statistic for data set 5.

Algorithm	Cramer-von Mises
Algorithm 1	0.622
Algorithm 2	0.608
Gibbs	0.619
Naive	0.623

TABLE 5.23: $E[\phi(X)|T = t]$ for data set 5 with ϕ as shown in equation 3.1 and $a = 0.42$

Algorithm	$E[\phi(X)\|T = t]$
Algorithm 1	0.4373
Algorithm 2	0.4125
Gibbs	0.4375
Naive	0.4720

TABLE 5.24: $E[\phi(X)|T = t]$ for data set 5 with ϕ as shown in equation 3.2 and $a = 0.407$

Algorithm	$E[\phi(X)\|T = t]$
Algorithm 1	0.5777
Algorithm 2	0.4930
Gibbs	0.5763
Naive	0.5053

TABLE 5.25: $E[\phi(X)|T = t]$ for data set 5 with ϕ as shown in equation 3.3 and $a = 0.42$

Algorithm	$E[\phi(X)\|T = t]$
Algorithm 1	0.4664
Algorithm 2	0.4776
Gibbs	0.4653
Naive	0.4773

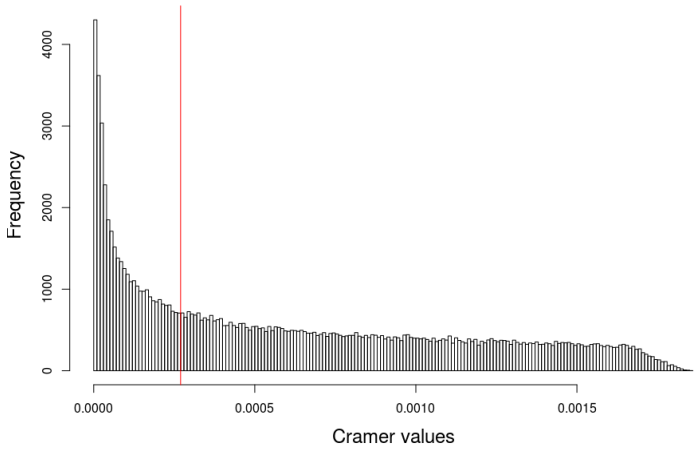

FIGURE 5.15: Plot of Cramer-Von Mises test statistics for naive sampler of data set 5. The vertical line is the observed Cramer-Von Mises value from original data set.

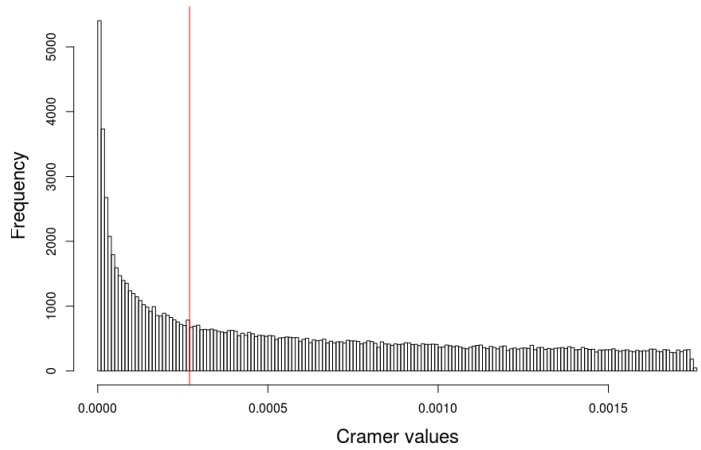

FIGURE 5.16: Plot of Cramer-Von Mises test statistics for Gibbs sampler of data set 5. The vertical line is the observed Cramer-Von Mises value from original data set.

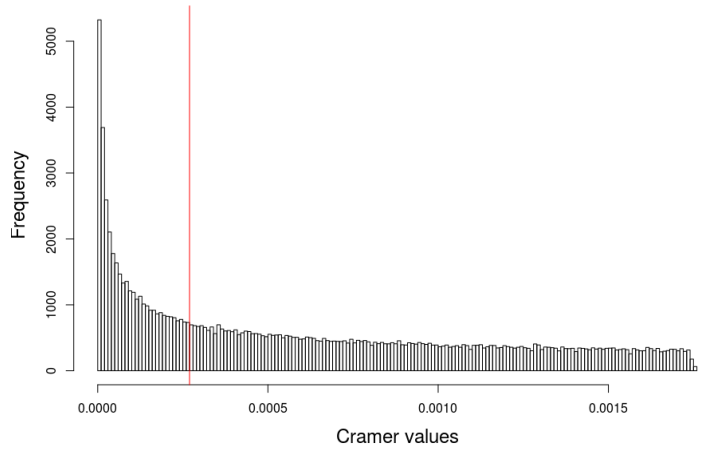

FIGURE 5.17: Plot of Cramer-Von Mises test statistics for algorithm 1 of data set 5. The vertical line is the observed Cramer-Von Mises value from original data set.

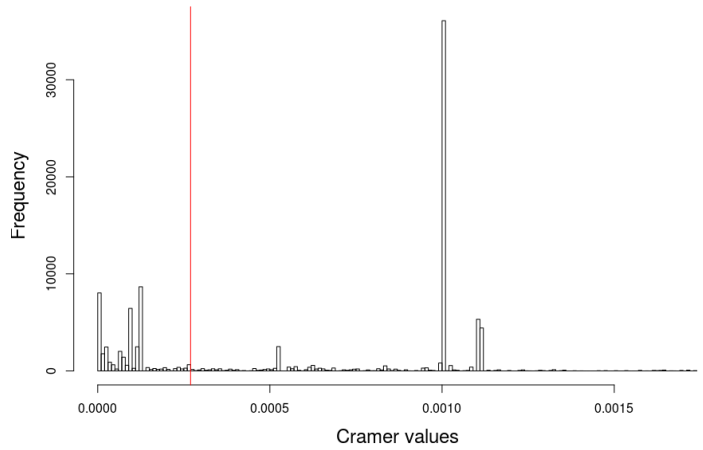

FIGURE 5.18: Plot of Cramer-Von Mises test statistics for algorithm 2 of data set 5. The vertical line is the observed Cramer-Von Mises value from original data set.

5.2 Discussion and comparison of samplers

All of the samplers gets high enough p-values to keep the null hypothesis for the data sets. This is expected because of the flexibility of the gamma distribution. Since a sample is only 3 data points, this makes it easier to keep the null hypothesis. Another note on the p-value results is that the algorithm 2 results vary compared to the other samplers. By studying the test statistics plots for the data sets, we see that the distribution of the test statistics values between algorithm 1 and Gibbs sampler are very close to each other. While algorithm 2 has somewhat the same shape but is different. Algorithm 2 seems to get some extreme distribution values. This is seen especially in data set 5. The naive sampler is close to algorithm 1 and the Gibbs sampler. However for the larger test statistic values they seem to differ. This may be the result of numerical error.

For the results presented regarding $E[\phi(X)|T = t]$ there are some characteristics in every data set. One of the characteristics is that algorithm 1 always is close to the Gibbs sampler. Hence, this may lead to that algorithm 1 generates samples with the right distribution or close to the right distribution. While algorithm 2 seems to vary some with the results. The reason for this would mostly go to the numerical approximation of the weights. First of all algorithm 2 has the numerical approximation of the inverse gamma cumulative. This is the same as in algorithm 1. Furthermore, there is a numerical approximation of the derivate of the inverse gamma cumulative. This will lead to an enhanced error in the calculation of weights. This might also lead to the error in the p-values. When it comes to the naive sampler,

this method is very sensitive to the size of the data points. For small data points, the ϵ-value must be very small. However, the number of options for data points are limited. Hence, it doesn't take a long time to find correct samples. For larger values the options for data points increases and the sampler takes significantly more time to run.

6 Concluding remarks

In this chapter the results of the samplers are summarized, and what further work could be done.

6.1 Performance of samplers

All of the samplers generate samples from the right distribution. However, the naive sampler works best for small data points. Algorithm 2 generates samples from the right distribution or close to the right distribution. When it comes to algorithm 1 it seems that the pivotal condition has little or no effect. Then algorithm 1 would be to preferred compared to algorithm 2. For the gamma distribution algorithm 1 and the Gibbs sampler would be recommended. This is for sampling for 3 data points.

6.2 Further work

Further work would be to look close at the naive sampler. For instance try to analyze more the effects of the ϵ-value for small and large data points. Further, investigate the performances of the samplers for data points greater than 3. The Gibbs sampler will become slower as larger thinning might be required. The subject of thinning can be found in [13]. This is done in my specialization project. The naive sampler is expected to become much slower. This is because of the increased number of options for a sample. For algorithm 1 and 2, the increased data points will cause the algorithms to become slower. This is because they have to calculate inverse cumulative distribution for each new data point. However, this is also an advantage compared to naive sampling. The speed loss compared to the naive sampler won't be great. One interesting point is to see how the increased data points affect the pivotal condition for algorithm 1.

One could also try other distributions to find examples where the pivotal condition is not met, and algorithm 1 generates samples from the wrong distribution. One example is the truncated exponential distribution. This is done in paper [11].

Bibliography

[1] E. B. Andersen. «Sufficient statistics and latent trait models».
In: *Psychometrika* 42.1 (1977), pp. 69–81 (cit. on p. 4).

[2] G. Casella and R. L. Berger. *Statistical inference*. Vol. 2.
Duxbury Pacific Grove, CA, 2002 (cit. on pp. 1, 7, 40, 41).

[3] G. Casella and E. I. George. «Explaining the Gibbs sampler».
In: *The American Statistician* 46.3 (1992), pp. 167–174 (cit. on p. 39).

[4] M. Fagerland. «Nonhomogenous Poisson process sampling with combined log-linear and power-law rate functions».
Specialization project. Departement of Mathematical Sciences, NTNU. 2015 (cit. on p. 39).

[5] W. R. Gilks, N. Best, and K. Tan.
«Adaptive rejection Metropolis sampling within Gibbs sampling».
In: *Applied Statistics* (1995), pp. 455–472 (cit. on p. 39).

[6] M. Kohl, P. Ruckdeschel, et al.
«R Package distrMod: S4 Classes and Methods for Probability Models».
In: *Journal of Statistical Software* 35.10 (2010), pp. 1–27 (cit. on p. 11).

[7] R. J. LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. Vol. 98. Siam, 2007 (cit. on p. 16).

[8] B. H. Lindqvist and G. Taraldsen.
«Conditional Monte Carlo based on sufficient statistics with applications».
In: *Advances in statistical modeling and inference. Essays in Honor of Kjell A. Doksum* (2007). Ed. by V. Nair, pp. 545–562 (cit. on pp. 4, 5).

[9] B. H. Lindqvist and G. Taraldsen. «Exact Statistical Inference for Some Parametric Nonhomogeneous Poisson Processes».
In: *Journal of Iranian Statistical Society* 12.1 (2013), pp. 113–126 (cit. on p. 41).

[10] B. H. Lindqvist and G. Taraldsen.
«Monte Carlo conditioning on a sufficient statistic».
In: *Biometrika* 92.2 (2005), pp. 451–464 (cit. on p. 1).

[11] B. H. Lindqvist, G. Taraldsen, M. Lillegård, and S. Engen.
«A counterexample to a claim about stochastic simulations».
In: *Biometrika* 90.2 (2003), pp. 489–490 (cit. on p. 35).

[12] R. A. Lockhart, F. J. O'Reilly, and M. A. Stephens.
«Use of the Gibbs sampler to obtain conditional tests, with applications».
In: *Biometrika* 94.4 (2007), pp. 992–998 (cit. on pp. 8, 19).

[13] A. E. Raftery and S. M. Lewis. «Implementing mcmc».
In: *Markov chain Monte Carlo in practice* (1996), pp. 115–130 (cit. on p. 35).

[14] M. Rausand and A. Høyland.
System reliability theory: models, statistical methods, and applications. Vol. 396.
John Wiley & Sons, 2004 (cit. on pp. 1, 3).

[15] M. A. Stephens. «Use of the Kolmogorov-Smirnov, Cramér-Von Mises and related statistics without extensive tables». In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1970), pp. 115–122 (cit. on p. 11).

[16] E. Süli and D. F. Mayers. *An introduction to numerical analysis*.
Cambridge university press, 2003 (cit. on p. 16).

[17] D. R. Thoman, L. J. Bain, and C. E. Antle.
«Inferences on the parameters of the Weibull distribution».
In: *Technometrics* 11.3 (1969), pp. 445–460 (cit. on p. 27).

Theory

In this chapter general theory needed are explained. Most of this theory is also gathered from my specialization project [4].

Finding sufficient statistics

To find sufficient statistics can be difficult, however one can use the factorization theorem to find the statistics. The theorem is as follows.

Theorem 1 *(Factorization theorem) Let $f(\boldsymbol{x}|\Theta)$ denote the joint pdf or pmf of a sample $\boldsymbol{X}$. A statistic $T(\boldsymbol{X})$ is a sufficient statistic for Θ if and only if there exist functions $g(t|\Theta)$ and $h(\boldsymbol{x})$ such that, for all sample points $\boldsymbol{x}$ and all parameter points Θ,*

$$f(\boldsymbol{x}|\Theta) = g(T(\boldsymbol{x})|\Theta)h(\boldsymbol{x}).$$

$h(\boldsymbol{x})$ should not be dependent on Θ. Then the remaining part will be $g(T(\boldsymbol{x})|\Theta)$ and from this we can see the sufficient statistic.

Tranformation of variables

For tranformation of variables in distribution the following theorem is presented.

Theorem 2 *Let X have pdf $f_X(x)$, let $Y = g(X)$, where g is a monotone function. Let $\mathcal{X} = \{x : f_X(x) > 0\}$ and $\mathcal{Y} = \{y : y = g(x) \text{ for some } x \in \mathcal{X}\}$. Suppose that $f_x(x)$ is continuous on $\mathcal{X}$ and that $g^{-1}(y)$ has a continuous derivative on $\mathcal{Y}$. Then the pdf of Y is given by*

$$f_Y(y) = \begin{cases} f_X(g^{-1}(y))\left|\frac{d}{dy}g^{-1}(y)\right| & y \in \mathcal{Y} \\ 0 & \textit{otherwise.} \end{cases}$$

Sampling with sufficient statistics

This section will go in depth of generating samples given sufficient statistics. The main focus will be a Gibbs sampler with a Metropolis-Hastings step. Information about this topic can be found in [5].

Gibbs sampler

The Gibbs sampler is Markov chain Monte Carlo (MCMC) sampler. A MCMC sampler is a way to sample from a distribution by constructing a Markov chain with spesific equilibrium. From [3] we have that the Gibbs sampler is used when direct sampling is difficult or cannot be done. This sampler samples from a conditional distribution. This can be done with sufficient statistics.

Metropolis-Hastings sampler

This is also a MCMC sampler where direct sampling is difficult. However this is used when the conditional posterior is unknown. Hence this uses a proportional density instead, and has an acceptance rejection step to determine if the new sample is from the desired distribution. Let X_{prop} be from proposal density $p(x)$. Then the acceptance probability is given as

$$\alpha = \min\left(1, \frac{\pi(x_{prop})p(x_{curr})}{\pi(x_{curr})p(x_{prop})}\right),$$

where $\pi(x)$ is the posterior distribution to sample from.

P-Value

A p-value can give the result of a hypothesis test. The following defintion and theorem can be found in [2]. First we are going to define a valid p-value.

Definition 2 *A p-value $p(\boldsymbol{X})$ is a test statistic satisfying $0 \leq p(\boldsymbol{x}) \leq 1$ for every sample point $\boldsymbol{x}$. Small values of $p(\boldsymbol{X})$ give evidence that H_1 is true. A p-value is valid if, for every $\theta \in \Theta_0$ and every $0 \leq \alpha \leq 1$,*

$$P_\theta(p(\boldsymbol{X} \leq \alpha)) \leq \alpha. \tag{.1}$$

Then a p-value is as follows.

Theorem 3 *Let $W(\boldsymbol{X})$ be a test statistic such that large values of W give evidence that H_1 is true. For each sample point $\boldsymbol{x}$, define*

$$p(\boldsymbol{x}) = \sup_{\theta \in \Theta_0} P_\theta(W(\boldsymbol{X}) \geq W(\boldsymbol{x})).$$

Then, $p(\boldsymbol{X})$ is a valid p-value.

The Θ_0 is the subset of the parameter space for the null model. P-values can also be defined by using sufficient statistics. A p-value is then defined as

$$p(\boldsymbol{x}) = P(W(\boldsymbol{X}) \geq W(\boldsymbol{x}) | S(\boldsymbol{X}) = S(\boldsymbol{x})), \tag{.2}$$

where $S(\boldsymbol{x})$ is a sufficient statistic under the null hypothesis. By this definition the p-value given a sufficient statistic is valid as shown below

$$P_\theta(p(\boldsymbol{X}) \leq \alpha) = \sum_x P(p(\boldsymbol{X}) \leq \alpha | S(\boldsymbol{X}) = s) P_\theta(S(\boldsymbol{X}) = s) \leq \sum_s \alpha P_\theta(S(\boldsymbol{X}) = s) = \alpha.$$

This result is for discrete $S(\boldsymbol{X})$. However for the continuous case the one can replace the sums with integrals.

For a two-sided hypothesis test the p-value is found by the equation below,

$$p(\boldsymbol{x}) = 2(\min(P(W(\boldsymbol{X}) \geq W(\boldsymbol{x})), P(W(\boldsymbol{X}) \leq W(\boldsymbol{x})))).$$

To calculate the p-value in equation .2 we refer to [9]. From this paper we have that the p-value can be estimated by

$$\hat{p} = \#\{W^* \geq W_{obs}\}/M,$$

where W^* is a test statistic for a sample, W_{obs} is the observed test statistic from original data and M is the number of samples.

Likelihood and estimators

In this section likelihood and its estimators are introduced. To summarize data the likelihood function can be used. The theory presented here is from [2]. The likelihood function is defined in the following definiton.

Definition 3 *Let $f(\boldsymbol{x}|\Theta)$ denote the joint pdf or pmf of the sample $\boldsymbol{X} = (X_1, ..., X_n)$. Then, given that $\boldsymbol{X} = \boldsymbol{x}$ is observed, the function Θ is defined by*

$$L(\Theta|\boldsymbol{x}) = f(\boldsymbol{x}|\Theta) \tag{.3}$$

is called the likelihood function.

By using this function one can find estimators for Θ. These estimators are the ones that maximizes the likelihood function. Furthermore these estimators are most likely. Hence the they are given the name maximum likelihood estimators. The formal definition is as follows

Definition 4 *For each sample point $\boldsymbol{x}$, let $\hat{\Theta}(\boldsymbol{x})$ be parameter value at which $L(\Theta|\boldsymbol{x})$ attains its maximum as a function of Θ, with $\boldsymbol{x}$ held fixed. A maximum likelihood estimator(MLE) of the parameter Θ based on a sample $\boldsymbol{X}$ is $\hat{\Theta}(\boldsymbol{X})$.*

The maximum of the likelihood can be found analytically or numerically. Analytically it is recommended to take the log of the likelihood function. This for easier differentiation. Hence it is recommended to solve

$$\frac{\partial \log L(\Theta|\boldsymbol{x})}{\partial \Theta_i} = 0, \quad i = 1, ..., k.$$

This will give the MLE of Θ.

R script

Here the R code for the master thesis is shown.

```
library(MASS)

calcWeight <- function(u, alpha) {
  # Calculates weight for given u and alpha.
  #
  # Args:
  #   u: A vector.
  #   alpha: A scalar.
  #
  # Returns:
  #   The weight value. A scalar.
  gammaInv = rep(0, length(u))
  diffGammaInv = rep(0, length(u))
  for(i in 1:length(u)) {
    gammaInv[i] = invGammaCumulative(u[i], alpha)
    diffGammaInv[i] = diffAlphaInvGammaCumulative(u[i], alpha)
  }
  pi = getPiValue()
  weight = pi/((1/length(gammaInv))*(sum(diffGammaInv/gammaInv)) - sum(
    diffGammaInv)/sum(gammaInv))
  return(weight)
}

getPiValue <- function() {
  # get value of pi function to be used in calculation of weights.
  #
  # Returns:
  #   A scalar value.
  if (piValue == "constant") {
    return(1)
  } else if (piValue == "jeffrey") {
    # Return jeffrey prior
    return(sqrt((1/(estAlpha[sampleIndex]^2)) + (1/(2-(exp(estAlpha[
    sampleIndex]) + exp(-estAlpha[sampleIndex]))))))
  } else if (piValue == "betaOption") {
    return(estBeta)
  } else if (piValue == "alphaOption") {
    return(estAlpha[sampleIndex])
  }
}

calcPhi <- function(u, alpha) {
  xValue = rep(0, length(u))
  for(i in 1:length(u)) {
    xValue[i] = invGammaCumulative(u[i], alpha)
  }
  return(calcPhiGivenX(xValue))
```

```
}

calcPhiGivenX <- function(x) {
  # Calc phi value for a vector x.
  #
  # Args:
  #   x: A vector of data.
  #
  # Returns:
  #  A scalar.

  if(phiOption == "probValueOption") {
    phiPoint = rep(0, length(x))
    for(i in 1:length(x)) {
      phiPoint[i] = getPhiValue(x[i])
    }
    return(sum(phiPoint)/length(phiPoint))
  } else if(phiOption == "x1x2divX3Option") {
    return(getPhiValue(x[1]*x[2]/x[3]))
  } else if(phiOption == "x1divx2powx3Option") {
    return(getPhiValue((x[1]/x[2])^x[3]))
  } else if(phiOption == "sinusfunction") {
    return(sin(x[1]) + sin(x[2]) + sin(x[3]))
  }
  return(-1)
}


getPhiValue <- function(xValue) {
  # Calculates phi for an element of an x vector.
  #
  # Args:
  #   xValue: A scalar value.
  #
  # Returns:
  #
  return(xValue > probValue)
}

optimInvGammaCumulative <- function(u, alpha) {
  # Use optim function to find inverse of cumulative distribution
  result = optim(c(10.1), invGammaAbsFunction, u=u, alpha=alpha, lower=0,
     upper=100,  method="Brent")
  return(result)
}

invGammaAbsFunction <- function(x, u, alpha) {
  return(abs(pgamma(x, shape=alpha, scale=1) - u))
}

invGammaCumulative <- function(u, alpha) {
  # Finds the inverse of the cumulative gamma.
  #
```

```
  # Args:
  #   u: Scalar value.
  #   alpha: Scalar value.
  #
  # Returns:
  #   A scalar. The inverse value given u and alpha.
  x = 0
  stepSize = 0.1
  tolerance = 0.00001
  direction = 1
  integralValue = 0

  while(abs(integralValue - u) > tolerance){
    x = x + direction*stepSize
    if(x<0) {
      x = 0
      direction = 1
    }

    if(method == "integrate") {
      integralv = integrate(gammaDensity, 0, x)
      integralValue = integral$valuev
    } else if(method == "pgamma") {
      integralv = pgamma(x, shape=alpha, scale=1)
      integralValue = integralv
    }


    # Going left and pass the point
    if ((u > integralValue) && (direction == -1)) {
      stepSize = stepSize/2
      direction = 1
    }

    # Going right and pass the point
    if ((u < integralValue) && (direction == 1)) {
      stepSize = stepSize/2
      direction = -1
    }
  }
  return(x)
}

diffAlphaInvGammaCumulative <- function(u, alpha) {
  # Derivative of gamma distribution with respect to alpha.
  #
  # Args:
  #   u: Scalar between 0 and 1.
  #   alpha: Scalar larger than 0.
  #
  # Returns:
  #   Scalar value. Derivative at point alpha.
  firstPoint = invGammaCumulative(u,alpha)
```

```
  secondPoint = invGammaCumulative(u, alpha + alphaHStep)
  return((secondPoint - firstPoint)/alphaHStep)
}

calcDerivateFunction <- function(u, alphaValue) {
  # Analytically find the derivative of cumulative inverse.
  #
  # Args:
  #   u: Scalar value between 0 and 1.
  #   alphaValue: Scalar value.
  #
  # Returns:
  #   A scalar value.
  largeFInv = invGammaCumulative(u, alphaValue)
  integralPart = integrate(integralFunction, 0, largeFInv)
  return((digamma(alphaValue)*u - integralPart$value)*gamma(alphaValue)
    /((largeFInv^(alphaValue - 1))*exp(-largeFInv)))
}

integralFunction <- function(y) {
  # Function to be integrated.
  return(log(y)*(y^(alphaValue - 1))*exp(-y)/gamma(alphaValue))
}

optimfindAlpha <- function(u, s2) {
  # Use optim function fo find alpha value.
  if((calcValueTau2(u, alphaUpperBound) < s2) || (calcValueTau2(u,
    alphaLowerBound) > s2)) {
    return(-1)
  }
  solution = optim(c(0.1), optimFunction, u=u, lower=alphaLowerBound,
    upper=alphaUpperBound, method="Brent")
  return(solution$par)
}

optimFunction <- function(alpha, u) {
  # Function minimize to find alpha.
  return(abs(s2-calcValueTau2(u, alpha)))
}

findBeta <- function(s1, u, alphaValue) {
  # Calculates beta
  #
  # Args:
  #   s1: Scalar value.
  #   u: Vector of values between 0 and 1.
  #   alphaValue: Scalar value.
  #
  # Returns:
  #   A scalar value.
  largeFInv = rep(0, length(u))
  for(i in 1:length(u)) {
    largeFInv[i] = invGammaCumulative(u[i], alphaValue)
```

```
  }
  return(s1*length(u)/(sum(largeFInv)))
}

calcValueTau2 <- function(u, alphaValue) {
  largeFInv = rep(0, length(u))
  for(i in 1:length(u)) {
    largeFInv[i] = invGammaCumulative(u[i], alphaValue)
  }
  return(length(u)*((prod(largeFInv))^(1/length(u)))/sum(largeFInv))
}

calcValueTau2Method2 <- function(x) {
  return(length(x)*((prod(x))^(1/length(x)))/sum(x))
}

gibbsSampling <- function(xInit) {

  # Used for thinning
  NUM_ITERATIONS = 5000
  xCurrent = xInit
  for(i in 1:NUM_ITERATIONS) {
    randomXpos = sample(length(xCurrent), size=3)
    sumX = xCurrent[randomXpos[1]] + xCurrent[randomXpos[2]] + xCurrent[
    randomXpos[3]]
    prodX = xCurrent[randomXpos[1]]*xCurrent[randomXpos[2]]*xCurrent[
    randomXpos[3]]
    x1 = runif(1)*sumX
    if(isValidX1Proposal(x1, sumX, prodX)) {
      roots = findRoots(x1, sumX, prodX)
      x2 = roots[1]
      x3 = roots[2]

      # A extra check for illegal values.
      if(is.nan(x2) || is.nan(x3)){
        print((x1^3 - 2*sumX*x1^2 + (sumX^2)*x1 - 4*prodX))
      }
      xProposal = xCurrent
      xProposal[randomXpos[1]] = x1
      xProposal[randomXpos[2]] = x2
      xProposal[randomXpos[3]] = x3
      alphaMetHastings = findAlphaMetHastings(c(xCurrent[randomXpos[1]],
    xCurrent[randomXpos[2]], xCurrent[randomXpos[3]]), c(xProposal[
    randomXpos[1]], xProposal[randomXpos[2]], xProposal[randomXpos[3]]))
      acceptProb = runif(1)
      if(acceptProb <= alphaMetHastings) {
        xCurrent = xProposal
      }
    }
  }
  return(xCurrent)
}
```

```
isValidX1Proposal <- function(x1, sumX, prodX) {
  return((x1^3 - 2*sumX*x1^2 + (sumX^2)*x1 - 4*prodX) > 0)
}

findRoots <- function (x1, sumX, prodX) {
  root1 = ((sumX - x1) + sqrt( (sumX - x1)^2 - 4*prodX/x1 ))/2
  root2 = ((sumX - x1) - sqrt( (sumX - x1)^2 - 4*prodX/x1 ))/2
  return(c(root1, root2))
}

findAlphaMetHastings <- function(xCurrent, xProposal) {
  piProp = 1/(xProposal[1]*sqrt((sum(xProposal) - xProposal[1])^2 - 4*
    prod(xProposal)/xProposal[1] ))
  piCurrent = 1/(xCurrent[1]*sqrt((sum(xCurrent) - xCurrent[1])^2 - 4*
    prod(xCurrent)/xCurrent[1] ))
  return(min(1, piProp/piCurrent))
}

findGammaAlphaMetHastings <- function(xCurrent, xProposal) {
  piProp = 1/(xProposal[1]*sqrt((sum(xProposal) - xProposal[1])^2 - 4*
    prod(xProposal)/xProposal[1] ))
  piCurrent = 1/(xCurrent[1]*sqrt((sum(xCurrent) - xCurrent[1])^2 - 4*
    prod(xCurrent)/xCurrent[1] ))
  if(is.nan(piCurrent)) {
    print(xCurrent)
  }
  return(min(1, piProp/piCurrent))
}


cramerVonMisesValueTest <- function(x, alpha, beta) {
  # Calculates the value for a Cramer-von Mises test.
  #
  # Args:
  #   x: A vector sample.
  #
  # Returns:
  #   A scalar value.
  x = sort(x)
  cramerSum = 0
  for(i in 1:length(x)) {
    cramerSum = cramerSum + (((2*i - 1)/(2*length(x))) - pgamma(x[i],
    shape=alpha, rate=beta))^2
  }

  cramer = (1/(12*length(x)))*cramerSum

  return(cramer)
}

findGammaMLE <- function(x){
  solution = optim(c(1,1), negativeLogLikelihoodGamma, x=x)
  return(solution$par)
```

```
}

negativeLogLikelihoodGamma <- function(par, x) {
  # Calculates the negative log-likelihood for a gamma distribution.
  #
  # Args:
  #   par: A vector of size 2. First element is alpha and second element
    is beta.
  #   x: Data to calculate log-likelihood from. A vector.
  #
  # Returns:
  #  The log-likelihood value. A scalar
  alpha = par[1]
  beta = par[2]
  logLikelihood = -((alpha - 1)*sum(log(x)) - (1/beta)*sum(x) - length(x)
    *log(gamma(alpha)) - alpha*length(x)*log(beta))
  return(logLikelihood)
}

calcAveragPhiValueForData <- function(mydata) {
  sumData = sum(mydata)
  prodData = prod(mydata)
  tolerance = 0.03
  minValue = min(mydata) - 2*tolerance
  maxValue = max(mydata) + 2*tolerance
  sampleNumber = 1
  NUM_ITERATIONS = 10000
  sumPhi = 0
  while(sampleNumber <= NUM_ITERATIONS) {
    x = runif(3, max = sumData)
    if((abs(sum(x) - sumData) < tolerance) && (abs(prod(x) - prodData) <
    tolerance)) {
      sumPhi = sumPhi + calcPhiGivenX(x)
      sampleNumber = sampleNumber + 1
      print(sampleNumber)
    }
  }
  return(sumPhi/(sampleNumber-1))
}

algorithm2Sampling <- function(NUM_ALG2_SAMPLES) {
  cramerNum = 0
  cramerStat = rep(0, NUM_ALG2_SAMPLES)
  vCurr = runif(NUM_POINTS)
  alphaCurr = optimfindAlpha(vCurr, s2)
  while(alphaCurr==-1) {
    vCurr = runif(NUM_POINTS)
    alphaCurr = optimfindAlpha(vCurr, s2)
  }
  piCurr = calcWeight(vCurr, alphaCurr)
  phiSum = 0

  for(i in 1:NUM_ALG2_SAMPLES) {
```

```
    print(i)
    vProp = runif(NUM_POINTS)
    alphaProp = optimfindAlpha(vProp, s2)
    piProp = 0
    if(alphaProp != -1) {
      piProp = calcWeight(vProp, alphaProp)
    }
    alphaMetHastings = min(1, piProp/piCurr)
    uProb = runif(1)
    if(uProb <= alphaMetHastings) {
      vCurr = vProp
      alphaCurr = alphaProp
      piCurr = piProp
    }
    betaCurr = findBeta(s1, vCurr, alphaCurr)
    xSample = rep(0, length(vCurr))
    for(j in 1:length(vCurr)) {
      xSample[j] = betaCurr*invGammaCumulative(vCurr[j], alphaCurr)
    }
    phiSum = phiSum + calcPhiGivenX(xSample)
    cramerStat[i] = cramerVonMisesValueTest(xSample, mleAlpha, mleBeta)
    if(cramerStat[i] >= cramerObs) {
      cramerNum = cramerNum + 1
    }
  }

  hist(cramerStat,breaks=200, main="", xlab="Cramer values", cex.lab=1.5)
  abline(v = cramerObs, col="red")
  alg2sampCramer <<- cramerStat

  print((cramerNum/NUM_ALG2_SAMPLES))
  alg2pvalue <<- (cramerNum/NUM_ALG2_SAMPLES)
  return(phiSum/NUM_ALG2_SAMPLES)
}

algorithm1Sampling <- function(NUM_ALG1_SAMPLES) {
  phiSum = 0
  cramerNum = 0
  cramerStat = rep(0, NUM_ALG1_SAMPLES)
  for(i in 1:NUM_ALG1_SAMPLES) {
    print(i)
    u = runif(NUM_POINTS)
    alphavalue = optimfindAlpha(u, s2)
    while(alphavalue == -1) {
      u = runif(NUM_POINTS)
      alphavalue = optimfindAlpha(u, s2)
    }
    betavalue = findBeta(s1, u, alphavalue)
    xSample = rep(0, length(u))
    for(j in 1:length(u)) {
      xSample[j] = betavalue*invGammaCumulative(u[j], alphavalue)
    }
    phiSum = phiSum + calcPhiGivenX(xSample)
```

```
    cramerStat[i] = cramerVonMisesValueTest(xSample, mleAlpha, mleBeta)
    #print(cramerStat[i])
    if(cramerStat[i] >= cramerObs) {
      cramerNum = cramerNum + 1
    }
  }
  print("here")
  print((cramerNum/NUM_ALG1_SAMPLES))
  hist(cramerStat,breaks=200, main="", xlab="Cramer values", cex.lab=1.5)
  abline(v = cramerObs, col="red")
  alg1sampCramer <<- cramerStat
  alg1pvalue <<- (cramerNum/NUM_ALG1_SAMPLES)
  return(phiSum/NUM_ALG1_SAMPLES)
}

naiveSampling <- function(myData, tolerance, mleAlpha, mleBeta) {
  NUM_NAIVE_SAMPLES = NUM_SAMPLES
  sumData = sum(myData)
  prodData = prod(myData)
  sampleNumber = 0
  sumPhi = 0
  iterations = 0
  cramerNum = 0
  cramerStat = rep(0, NUM_NAIVE_SAMPLES)
  while(sampleNumber<NUM_NAIVE_SAMPLES) {
    x = rgamma(3, shape = mleAlpha, rate = mleBeta)
    iterations = iterations + 1
    if((abs(sum(x) - sumData) < tolerance) && (abs(prod(x) - prodData) <
    tolerance)) {
      sumPhi = sumPhi + calcPhiGivenX(x)
      sampleNumber = sampleNumber + 1
      cramerStat[sampleNumber] = cramerVonMisesValueTest(x, mleAlpha,
    mleBeta)
      if(cramerStat[sampleNumber] >= cramerObs) {
        cramerNum = cramerNum + 1
      }
      print(sampleNumber)
    }
  }

  hist(cramerStat,breaks=200, main="", xlab="Cramer values", cex.lab=1.5)
  abline(v = cramerObs, col="red")

  acceptRate = sampleNumber/iterations
  averagePhi = sumPhi/sampleNumber
  print((cramerNum/NUM_NAIVE_SAMPLES))
  naivePvalue <<- (cramerNum/NUM_NAIVE_SAMPLES)
  naiveCramers <<- cramerStat
  return(c(acceptRate, averagePhi))
}

# Specific variables.
alpha = 1
```

```
beta = 1
hStep = 0.01
alphaHStep = 0.01
method = "pgamma"
NUM_SAMPLES = 100000
NUM_POINTS = 3
alphaUpperBound = 200
alphaLowerBound = 0.05
# Pi is used in calculation of weights
# Options are:
#   "constant"
#   "betaOption"
#   "jeffrey"
#   "alphaOption"
piValue = "constant"

phi = rep(0, NUM_SAMPLES)
# Phi options:
# x larger than a: "probValueOption"
# x1 times x2 div x3: "x1x2divX3Option"
# x1 div x2 pow x3: "x1divx2powx3Option"
phiOption = "x1x2divX3Option"
# Phi is the prob that X>probValue. In raport this is just denoted "a".
probValue = 0.42

# Data generation options:
# pgamma generated: "pgamma"
# Bo data: "bo"
# Custom data: "custom"
# Custom data2: "custom2"
# ...
dataGenOption = "custom4"



# Generate data
gammaData = 0
if(dataGenOption == "pgamma") {
  gammaData = rgamma(NUM_POINTS, shape=alpha, scale = beta)
} else if(dataGenOption == "bo") {
  NUM_POINTS = 6
  alphaUpperBound = 1.2
  alphaLowerBound = 0.8
  gammaData = c(4.399, 1.307, 0.085, 0.7910, 0.2345, 0.1915)
} else if(dataGenOption == "custom") {
  gammaData = c(0.5772030, 0.4340237, 0.4212959)
} else if(dataGenOption == "custom2") {
  gammaData = c(1.621813, 1.059797, 1.554334)
} else if(dataGenOption == "custom3") {
  gammaData = c(5, 15, 13)
} else if(dataGenOption == "ball1") {
  gammaData = c(17.88, 28.92, 33.00)
} else if(dataGenOption == "ball2") {
```

```
  gammaData = c(41.52, 42.12, 45.60)
} else if(dataGenOption == "ball3") {
  gammaData = c(48.40, 51.84, 51.96)
} else if(dataGenOption == "ball4") {
  gammaData = c(54.12, 55.56, 67.80)
} else if(dataGenOption == "ball5") {
  gammaData = c(68.64, 68.64, 68.88)
} else if(dataGenOption == "ball6") {
  gammaData = c(84.12, 93.12, 98.64)
} else if(dataGenOption == "ball7") {
  gammaData = c(105.12, 105.84, 127.92)
} else if(dataGenOption == "custom4") {
  gammaData = c(0.40, 0.42, 0.43)
} else if(dataGenOption == "custom5") {
  gammaData = c(0.72, 0.72, 0.85)
}

#17.88 28.92 33.00 41.52 42.12 45.60 48.40 51.84
#51.96 54.12 55.56 67.80 68.64 68.64 68.88 84.12
#93.12 98.64 105.12 105.84 127.92 128.04 173.40

hist(gammaData)
# Calculation of statistics
s1 = sum(gammaData)/NUM_POINTS
s2 = NUM_POINTS*((prod(gammaData))^(1/NUM_POINTS))/sum(gammaData)
# Log-likelihood
mleEstimators = fitdistr(gammaData, "gamma")
mleAlpha = mleEstimators$estimate[1]
# Beta is the rate parameter, opposite of definition of gamma distr in
    report.
mleBeta = mleEstimators$estimate[2]
# Observed cramer-von mises
cramerObs = cramerVonMisesValueTest(gammaData, mleAlpha, mleBeta)


# Naive sampler. Uses one tolerance value for both sufficient statistics
    in naive sampling.
tolerance = 0.0001
# These variables will be set in naiveSampling function.
naiveCramers = rep(0,100000)
naivePvalue = 0

naiveSampler = naiveSampling(gammaData, tolerance, mleAlpha, mleBeta)

# Plot of cramer-von mises from naive sampling
hist(naiveCramers,breaks=200, main="", xlab="Cramer values", cex.lab=1.5)
abline(v = cramerObs, col="red")


# Gibbs sampling
NUM_GIBBS_SAMPLES = NUM_SAMPLES
xSample = gammaData
phiGibbs = rep(0, NUM_GIBBS_SAMPLES)
```

```
gibbsObslargerWObs = 0
cramerNum = 0
cramerStatGibbs = rep(0, NUM_GIBBS_SAMPLES)
for(i in 1:NUM_GIBBS_SAMPLES) {
  xSample = gibbsSampling(xSample)
  phiGibbs[i] = calcPhiGivenX(xSample)
  if(phiGibbs[i] >= wObs) {
    gibbsObslargerWObs = gibbsObslargerWObs + 1
  }
  cramerStatGibbs[i] = cramerVonMisesValueTest(xSample, mleAlpha, mleBeta
    )
  #print(cramerStat)
  if(cramerStatGibbs[i] >= cramerObs) {
    cramerNum = cramerNum + 1
  }
  print(i)
}
gibbsS1 = sum(xSample)/NUM_POINTS
gibbsS2 = NUM_POINTS*((prod(xSample))^(1/NUM_POINTS))/sum(xSample)
gibbsPvalue = gibbsObslargerWObs/NUM_GIBBS_SAMPLES
averagePhiGibbs = sum(phiGibbs)/NUM_GIBBS_SAMPLES
## P-values
gibbsPValue = cramerNum/NUM_GIBBS_SAMPLES

hist(cramerStatGibbs,breaks=200, main="", xlab="Cramer values", cex.lab
    =1.5)
abline(v = cramerObs, col="red")

# These variables will be set in algorithm 1 and 2 functions.
alg2sampCramer = rep(1, 100000)
alg1sampCramer = rep(1, 100000)
alg2pvalue = 0
alg1pvalue = 0

system.time({alg1Results2 = algorithm1Sampling(NUM_SAMPLES)})
system.time({alg2Results2 = algorithm2Sampling(NUM_SAMPLES)})

hist(alg1sampCramer,breaks=200, main="", xlab="Cramer values", cex.lab
    =1.5)
abline(v = cramerObs, col="red")

hist(alg2sampCramer,breaks=200, main="", xlab="Cramer values", cex.lab
    =1.5)
abline(v = cramerObs, col="red")


# The next commented section is for multi-threading algorithm 1 and 2. P-
    values and cramer-von mises will not be saved in variables.
# Only expected phi values will be saved.

#library("parallel")
#library("foreach")
#library("doParallel")
```

```
#
#cl = makeCluster(detectCores() - 1)
#registerDoParallel(cl, cores = detectCores() - 1)
#workers = 10
#stime = system.time({
#res = foreach(i=1:workers,
#         .combine = rbind) %dopar% {
#           try({
#             result1 = algorithm1Sampling(100000/workers)
#           })
#         }
#})
#stopCluster(cl)
#
#alg1Results3 = (sum(res[,1]))/workers
#
#cl = makeCluster(detectCores() - 1)
#registerDoParallel(cl, cores = detectCores() - 1)
#workers = 10
#stime = system.time({
#  res = foreach(i=1:workers,
#                .combine = rbind) %dopar% {
#                  try({
#                    result1 = algorithm2Sampling(100/workers)
#                  })
#                }
#})
#stopCluster(cl)
#
#alg2Results3 = (sum(res[,1]))/workers
#

# Plot of phi functions
sumData = 4.23
prodData = 2.67
x1Seq = seq(1.01, 2, 0.001)
phi2 = rep(0,length(x1Seq))
phi3 = rep(0,length(x1Seq))
for(i in 1:length(x1Seq)) {
  if(isValidX1Proposal(x1Seq[i], sumData, prodData)) {
    roots = findRoots(x1Seq[i], sumData, prodData)
    phi2[i] = phi2Function(x1Seq[i], roots[1], roots[2])
    phi3[i] = phi3Function(x1Seq[i], roots[1], roots[2])
  }

}

phi2Function <- function(x1, x2, x3) {
  return((x1*x2/x3))
}

phi3Function <- function(x1, x2, x3) {
  return((x1/x2)^x3)
```

```
}

plot(x1Seq, phi2)
plot(x1Seq, phi3)

# Plot of F inverse
u = 0.5
alphaRange = seq(0.01, 4, 0.001)
fInv = rep(0, length(alphaRange))
for(i in 1:length(fInv)) {
  fInv[i] = invGammaCumulative(u, alphaRange[i])
}
plot(alphaRange, fInv, type="l")

diffF = rep(0, length(fInv))
for(i in 1:length(diffF)) {
  diffF[i] = diffAlphaInvGammaCumulative(u, alphaRange[i])
}

plot(alphaRange, diffF, type="l")


save.image(file="scen33.RData")
```