# Computational Methods in Statistics and Operations Research

SOR3500

Worksheet Report

Marc Ferriggi (286397M)

February 19, 2018

# Contents

# List of Figures

# List of Tables

# Listings

# Chapter 1

# Worksheet 1

## 1.1 Introduction

This worksheet focuses on pseudo-random number generation. In particular, in Section 1.2 (Task A), Cholesky Factorisation shall be used to simulate multivariate random variables and in Section 1.3 (Task B), the accept/reject method shall be used to simulate from a univariate standard normal distribution.

For Task A, a trivariate normal distribtion with mean $\boldsymbol{\mu}$=(1,0,-0.5)' and variance covariance matrix $\boldsymbol{A} = \begin{pmatrix} 3 & 1 & 0.3 \\ 1 & 1.5 & 0.5 \\ 0.3 & 0.5 & 2 \end{pmatrix}$ will be considered. The Cholesky factorization of the variance covariance matrix will be used with the ultimate aim of simulating 10,000 values from this trivariate normal distribution from first principle. The algorithm will then be verified by calculating the relevant means, variances and covariances.

For Task B, a standard normally distributed random variable will be simulated. This will be done by taking the positive half of the standard normal, simulating from an exponential distribution and using the exponential density as a majorising function, and finally accepting/rejecting the point via generalized rejection sampling. The decision on whether the simulated value takes a positive or negative sign will be decided with probability 0.5 by generating a uniform value and comparing with a 0.5 threshold. 10,000 values will be simulated using this method and the majorising constant will

be chosen in a way that maximizes the acceptance rate.

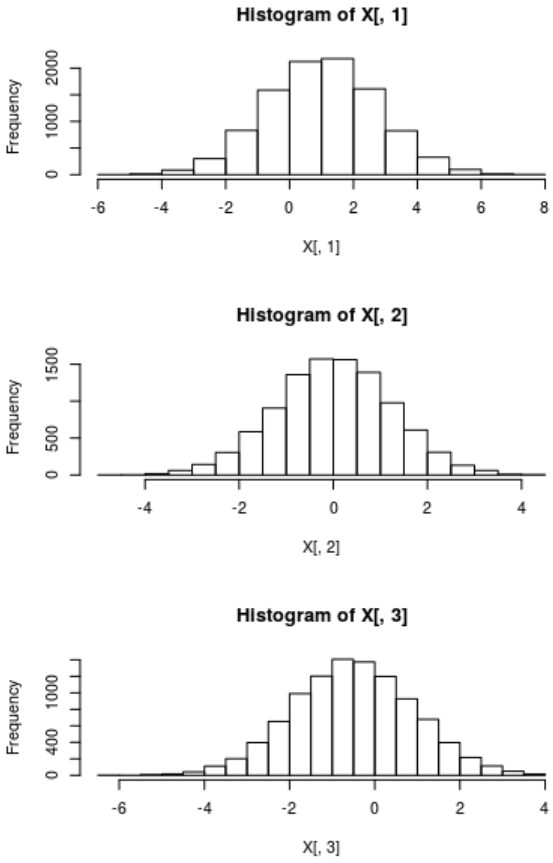
## 1.2 Task A

In order to generate the vector $\boldsymbol{X}$ from a trivariate normal distribution with mean $\boldsymbol{\mu}$=(1,0,-0.5)', variance covariance matrix $\boldsymbol{A} = \begin{pmatrix} 3 & 1 & 0.3 \\ 1 & 1.5 & 0.5 \\ 0.3 & 0.5 & 2 \end{pmatrix}$, we shall make use of the Cholesky Factorisation Theorem. By the properties of a variance covariance matrix, we know that $\boldsymbol{A}$ is symmetric and positive definite. Thus, we can let $\boldsymbol{A} = \boldsymbol{CC'}$ where $\boldsymbol{C}$ is worked out in R using the 'chol' function. The code used to perform Cholesky Factorisation of variance covariance matrix $\boldsymbol{A}$ can be seen in listing 6.1.

In order to check that the code in listing 6.1 gave us the correct result, 'C%*%C' was entered into the terminal and the output gave us back the original variance covariance matrix. The code which can be seen in listing 6.2 was then used to simulate the 10,000 values generated from this particular trivariate normal distribution.

The resultant plots generated in lines 18-20 in listing 6.2 can be seen in figure 1.1. These were used to ensure that each generated random number does in fact result in a multivariate normal distribution.

Figure 1.1: Histogram of generated Xi's



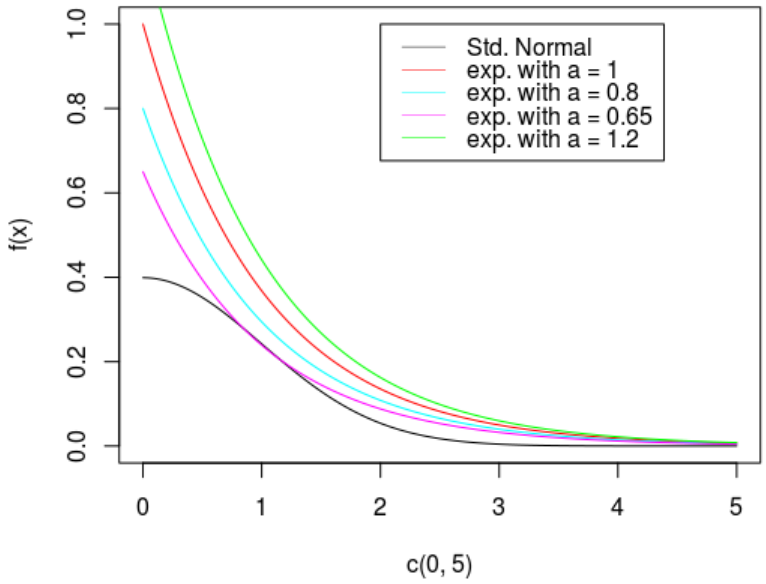

In order to work out the variance covariance matrix of our generated random numbers, the code that can be seen in listing 6.3 was used. This is based on the theory provided by [1].

The results of this exercise (when using the command set.seed(2)) were as follows: generated variance covariance matrix $\boldsymbol{V} = \begin{pmatrix} 4.08 & 1.02 & -0.20 \\ 1.02 & 1.51 & 0.50 \\ -0.20 & 0.50 & 2.26 \end{pmatrix}$ and generated mean vector $\hat{\boldsymbol{\mu}} = \begin{pmatrix} 1.019735 \\ 0.01156303 \\ -0.5041943 \end{pmatrix}$.

## 1.3 Task B

Initially, figure 1.2 was generated using the code in listing 6.4 to decide on an appropriate value for the majorising constant a. It would seem from the plot that letting a = 0.8 would be a good start for this exercise.

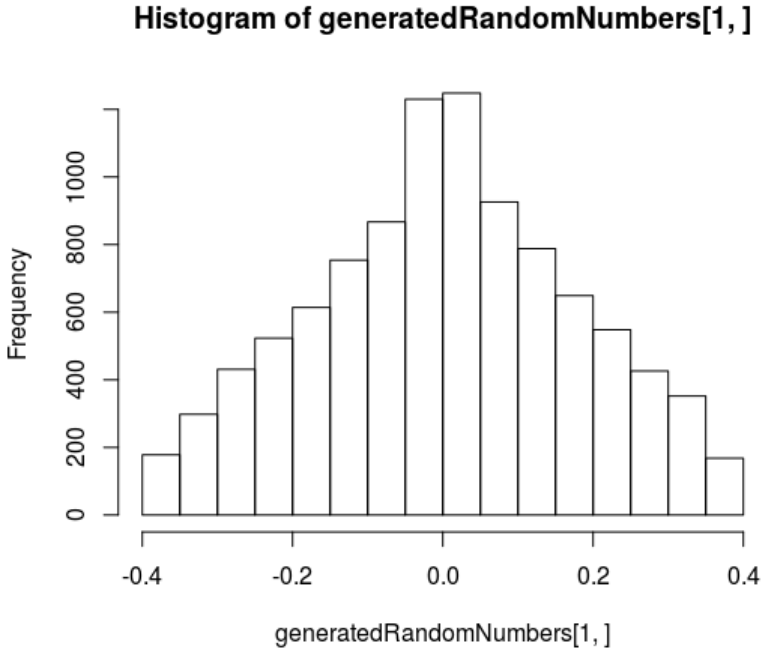
Figure 1.2: Exponential Distributions with different values for a




The code in listing 6.5 used the accept-reject method to simulate values from a standard normal distribution. This was done using the following steps:

1. x was simulated from exponential distribution with parameter $\lambda = 1$.
2. y was generated from the uniform distribution on the interval $(0, ae^{-x})$.
3. The point y was accepted if $y \leq \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$.
4. If y was accepted then a uniform random number between 0 and 1 was generated. If the number was less than 0.5, then y was given a negative value, else its value remained the same.

Figure 1.3: Resulting Histogram




The code in listing 6.5 was then modified in order to incorporate a loop that would repeat the exercise for different values of M to see which value seems to give the most efficient solution. This altered code can be seen in listing 6.6. When printing the values for counts, it would seem that the most efficient solution would be to let M be as small as possible but ensuring that the values of $Me^{-x} \geq \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ at all times. In this case, M=0.68.

## 1.4 Conclusion

The exercises of this worksheet show examples of methods on how to generate pseudo-random numbers in R given that the only in built function for random number generation is that which gives a uniformly distributed random number. In reality, there exist many inbuilt functions in R (or in packages that can be imported to R) that can automate this task with similar or better results. However such methods may be useful when using programming languages that do not have these inbuilt functions.

# Chapter 2

# Worksheet 2

## 2.1 Introduction

There exist many numerical methods for estimating integrals which can't be solved analytically, one such method being Monte Carlo Integration. Crude Monte Carlo Integration evaluates integrals of the type $\theta = E_f[\gamma(x)] = \int_{x\in\chi} \gamma(x)f(x)dx$ where $X$ is a continuous random variable (or random vector) with probability distribution function $f$, and $\chi$ is the range of values of $X$ using the following estimator: $\hat{\theta} = \bar{\gamma}_N(X) = \frac{1}{N}\sum_{i=1}^{N}\gamma(X_i)$.

In Section 2.2 (Task A), $\int_2^8 \frac{\sin x}{\ln x}dx$ shall be integrated using Crude Monte Carlo Integration. $N$ will be taken to be 100,1000,10000,100000,1000000 and at each $N$, the variance of these estimates shall be calculated using a numerically obtained value. Finally Monte Carlo Estimates of the variance of these estimates shall be obtained.

In Section 2.3 (Task B), the following questions shall be answered:
"Suppose we would like to compute $P(X > 20)$ for a standard normally distributed random variable $X$.

1. Write down the Crude Monte Carlo Integral equation, and explain why using crude Monte Carlo in this case does not make practical sense.

2. Explain why a sampling distribution defined on $[20, \infty)$ of the type $g^{(k)}(x) = (k-1)20^{k-1}x^{-k}; k > 2$ could make an appropriate variance-reducing sample distribution to estimate $P(X > 20)$.

3. Using $g^{(k)}$ as a sampling distribution, obtain estimates for $P(X > 20)$ for $k = 2, 3, ..., 10$ and $N = 100, 1000, 10000, 100000, 1000000$. Also obtain Monte Carlo Estimates of the variance of these estimates at each $N$. Present your results in an appropriate table.

4. Which value of $k$ appears to be the most appropriate for estimating more accurately this probability?"

## 2.2 Task A

Note that when integrating numerically $\int_2^8 \frac{sinx}{lnx}dx$ using Mathematica, the answer is equal to -0.053792. For the sake of this exercise I shall assume a mistake in the question and compare the results of the Monte Carlo integral to this value.

$$\begin{aligned}\int_2^8 \frac{sinx}{lnx}dx &= 6\int_2^8 \frac{1}{6}\frac{sinx}{lnx}dx \\ &= 6\mathbb{E}[\frac{sinx}{lnx}]; x \sim Unif(2,8)\end{aligned}$$

Therefore our Monte Carlo integral can be calculated as follows:

$$\int_2^8 \frac{sinx}{lnx}dx \simeq \frac{6}{N}\sum_{i=1}^{N}\frac{sinx_i}{lnx_i}; x_i \sim Unif(2,8) \tag{2.1}$$

The code in listing 6.7 was used to compute an approximation of the integral. The results can be seen in Table 2.1

| N | 100 | 1000 | 10000 | 100000 | 1000000 |
|---|---|---|---|---|---|
| thetahat | -0.390582 | -0.1624749 | -7.96254e-02 | -5.103975e-02 | -5.627143e-02 |

Table 2.1: Crude Monte Carlo Integral Results

In order to calculate the variance, the following equation was used (where $\bar{\gamma}(x) = -0.053792$):

$$S_\theta^2 = \frac{1}{N(N-1)}\sum_{i=1}^{N}(\frac{sinx_i}{lnx_i} - \bar{\gamma}(x))^2 \tag{2.2}$$

The Monte Carlo estimates for the variance were then calculated using the following result:

$$\begin{aligned} \sigma_\theta^2 &= \frac{1}{6N}\int_2^8 (\frac{6sinx}{lnx} - \bar{\gamma}(x))^2 dx \\ \implies S_\theta^2 &= \frac{1}{6N(N-1)}\sum_{i=1}^{N}(\frac{6sinx_i}{lnx_i} - \hat{\theta})^2 \end{aligned} \tag{2.3}$$

The final code used to obtain the results seen in Table 2.2 can be seen in listing 6.8.

| N | 100 | 1000 | 10000 | 100000 | 1000000 |
|---|---|---|---|---|---|
| thetahat | 3.332358e-02 | -1.161485e-01 | -3.501639e-02 | -4.791467e-02 | -5.265917e-02 |
| variance | 3.076055e-03 | 2.378905e-04 | 2.438599e-05 | 2.487159e-06 | 2.476489e-07 |
| varhatthetahat | 3.047775e-03 | 2.392530e-04 | 2.443356e-05 | 2.500751e-06 | 2.491877e-07 |

Table 2.2: Task 2A Results

## 2.3 Task B

**1.** Crude Monte Carlo Integral:

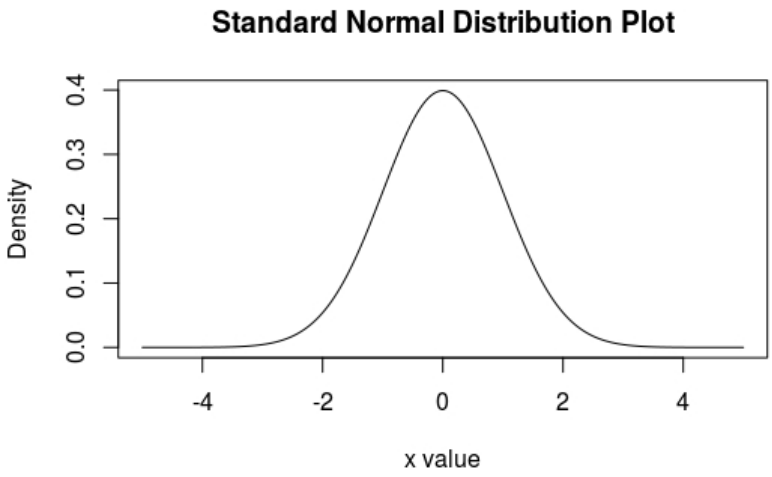
$$\begin{aligned} \theta &= \int_\Omega \mathbf{1}_{\{x:x>20\}}(x)\Phi(x)dx \\ \implies \hat{\theta} &= \frac{1}{N}\sum_{i=1}^{N}\mathbf{1}_{\{x:x>20\}}(x_i); x_i \sim N(0,1) \end{aligned}$$

It doesn't make sense to use the crude Monte Carlo Integral in this case because even for generating 1,000,000 points, probably none of these points would be $> 20$, thus the resultant value would always be 0.

**2.** From the plot of the standard normal distribution (see Figure 2.1), it can be noted that it is large in some places where $\mathbf{1}_{\{x:x>20\}}(x)$ is zero and

small where $\mathbf{1}_{\{x:x>20\}}(x)$ is one. Evaluating Crude Monte Carlo Integrals of this kind will lead to an estimator with an extremely large variance and thus better accuracy could be achieved using importance sampling.

Figure 2.1: Normal Distribution




Thus, taking as importance sampler the function $g^{(k)}(x) = (k-1)20^{k-1}x^{-k}; k \geq 2$ which is defined on the range $[20, \infty)$ should lead to a better estimate of the required integral. As can be seen in Figures 2.2 and 2.3, where $k = 5$, $g(x)$ has a similar shape to $\mathbf{1}_{\{x:x>20\}}(x)\Phi(x)$ and hence samples points from regions where these are most required.

The new integral to be evaluated using importance sampling:

$$\theta = \int_{x\in\chi} \gamma(x)w(x)dx; w(x) = \frac{f(x)}{g(x)}$$
$$\implies \hat{\theta} = \frac{1}{N}\sum_{i=1}^{N} \gamma(X_i)w(X_i)$$

Where $X_1, ..., X_N$ are $g$-distributed.

Figure 2.2: Importance Sampler g

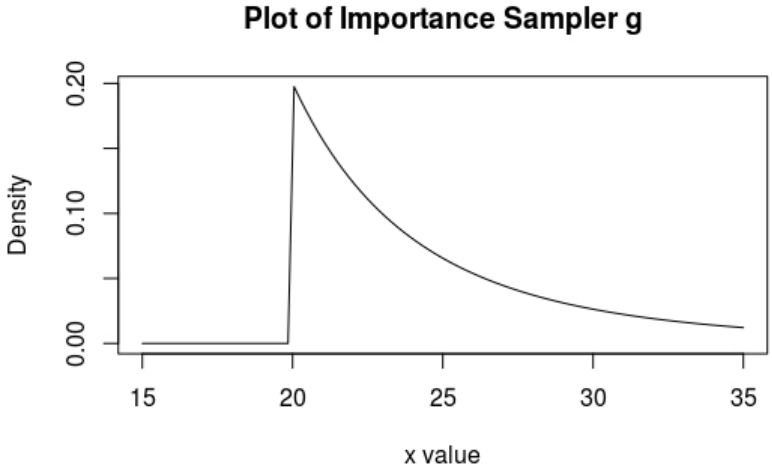
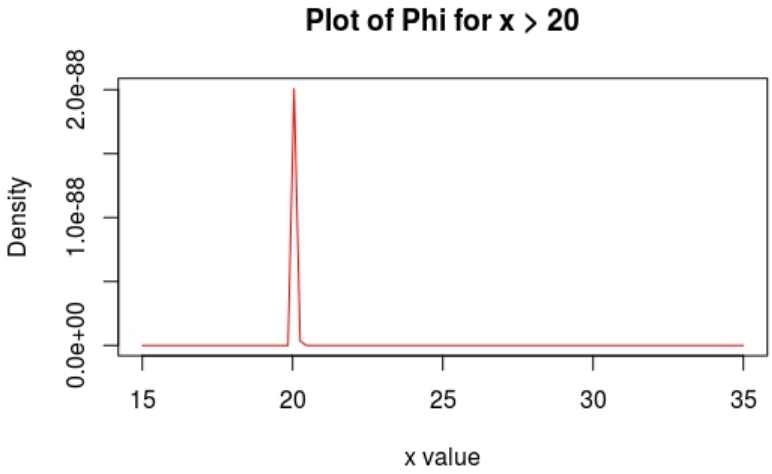



Figure 2.3: Phi




**3.** In order to use $g^{(k)}$ as a sampling distribution, $g$ shall be simulated using the inversion method.(Note that $\int_{20}^{\infty} g^{(k)}(x) = 1$ for $k = 2, 3, ..., 10$ and also always positive, thus $g$ is in fact a distribution function). By working out the integral $G_{(k)}(x) = \int_{20}^{x} g^{(k)}(x)$, we have that $G_{(k)}(x) = 1 - 20^{(k-1)}x^{-(k-1)}$, thus $G_{(k)}^{-1}(x) = (\frac{20^{(k-1)}}{1-x})^{\frac{1}{(k-1)}}$ for $k = 3, 4, ..., 10$.

The code used to evaluate the results seen in Tables 2.3 and 2.4 is listed in Listing 6.9.

| | **100** | **1000** | **10000** | **100000** | **1000000** |
|---|---|---|---|---|---|
| **2** | 1.431494e-89 | 2.556542e-89 | 2.046599e-89 | 2.816939e-89 | 2.741404e-89 |
| **3** | 3.230082e-90 | 2.536751e-89 | 3.018749e-89 | 2.559027e-89 | 2.766811e-89 |
| **4** | 2.802467e-89 | 1.144899e-89 | 2.483318e-89 | 2.666589e-89 | 2.765502e-89 |
| **5** | 4.989252e-89 | 3.185126e-89 | 2.686524e-89 | 2.816248e-89 | 2.766248e-89 |
| **6** | 2.477491e-91 | 2.868881e-89 | 2.527291e-89 | 2.801112e-89 | 2.778682e-89 |
| **7** | 1.169344e-89 | 2.587175e-89 | 2.739460e-89 | 2.808193e-89 | 2.752334e-89 |
| **8** | 5.204709e-89 | 2.564028e-89 | 2.888337e-89 | 2.797936e-89 | 2.774699e-89 |
| **9** | 1.420079e-89 | 2.680081e-89 | 2.933646e-89 | 2.652205e-89 | 2.743138e-89 |
| **10** | 2.311774e-89 | 3.203120e-89 | 2.765658e-89 | 2.768061e-89 | 2.769803e-89 |

Table 2.3: $\hat{\theta}$

| | **100** | **1000** | **10000** | **100000** | **1000000** |
|---|---|---|---|---|---|
| **2** | 2.667979e-213 | 1.081140e-213 | 1.340489e-215 | 8.489719e-216 | 3.227463e-218 |
| **3** | 6.337118e-214 | 2.564832e-213 | 1.231320e-215 | 2.143839e-215 | 4.155809e-217 |
| **4** | 9.409443e-212 | 8.366001e-216 | 2.243981e-215 | 1.144027e-215 | 4.381525e-216 |
| **5** | 2.607832e-212 | 2.621750e-213 | 7.627930e-215 | 5.616732e-218 | 2.532833e-216 |
| **6** | 5.842405e-217 | 5.802036e-214 | 3.537457e-215 | 5.746608e-216 | 6.147907e-216 |
| **7** | 8.300163e-213 | 2.806684e-214 | 5.420857e-216 | 5.852233e-215 | 3.808483e-216 |
| **8** | 6.629427e-212 | 3.044791e-213 | 3.826458e-214 | 2.117801e-215 | 3.367116e-221 |
| **9** | 6.549805e-214 | 5.199809e-214 | 6.917800e-216 | 2.892819e-216 | 2.404266e-216 |
| **10** | 4.161482e-213 | 7.501559e-213 | 4.035333e-215 | 6.585696e-217 | 2.543427e-216 |

Table 2.4: Monte Carlo Estimate of Variance of $\hat{\theta}$

**4.** As can be seen in Table 2.4 it would seem that for $k = 2$ the estimator has the smallest variance, thus taking that value for $k$ would be the most appropriate for estimating most accurately this probability.

## 2.4 Conclusion

The exercises of this worksheet show examples of methods on estimating integrals using Monte-Carlo methods and how to reduce the variance of these estimators using Importance Sampling. These techniques come in handy when in situations where solving integrals analytically is impossible. Some examples of where Monte Carlo Methods are used are in Bayesian Statistics and Computer Rendering and Shading [2].

# Chapter 3

# Worksheet 3

## 3.1 Introduction

Resampling methods are computational methods to estimate properties of estimators when information of the population is unknown. This is achieved by using subsets of available data, or by drawing randomly with replacement from a set of data points. Two of the most popular data resampling methods are the Jackknife and Bootstrap methods.

In this worksheet, the Jackknife (Section 3.2) and Bootstrap (Section 3.3) estimators for the median, lower quartile and upper quartile of a provided dataset will be found, along with their 95% confidence interval using the normal distribution. In the case of the Jackknife estimator, the data will be sectioned into 5 partitions of size 20, while in the case of the Bootstrap estimator, 1000 samples of size 100 will be generated with replacement. The empirical confidence interval for the Bootstrap estimator will also be derived.

## 3.2 Jackknife Estimator

In this section the theory and methods used to come up with the Jackknife estimated values for the median, lower quartile and upper quartile of the provided dataset shall be covered. Note that for this section and section 3.3, the data provided was saved as a '.csv' file in order to simplify importing into R.

The idea behind Jackknife estimation is that of sampling without replacement, where disjoint subsets of the original sample shall be omitted from the sample and the effect this has on the estimator will be studied. In particular for the provided sample, the data shall be split into 5 random partitions, each of size 20. This was done in R by randomly ordering the provided sample using the 'sample()' function and then grouping the data into the 5 partitions depending on their new order.

In order to obtain the Jackknife estimator, each partition is omitted once and the statistic of interest is calculated on the new sample, denoted by $\hat{\theta}_{n-m,a}(\omega, \omega')$. The Jackknife estimator is then obtained as follows:

$$\hat{\theta}_{JK}(\omega, \omega') = \frac{1}{A} \sum_{a=1}^{A} \hat{\theta}_{m,a}(\omega, \omega')$$

where $\hat{\theta}_{m,a}(\omega, \omega') = A\hat{\theta}_n(\omega) - (A-1)\hat{\theta}_{n-m,a}(\omega, \omega')$ and $A$ refers to the number of partitions,$m$ the size of the partition and $n$ the original sample size.

A 95% confidence interval of the estimator can then be found using the normal distribution since it is known that for large $n$, the Jackknife estimate is approximately normally distributed. Thus the 95% confidence interval for the true parameter $\theta$ was calculated as follows:

$$\hat{\theta}_{JK} + z_{0.025}\hat{\sigma}_{\hat{\theta}_JK} < \theta < \hat{\theta}_{JK} + z_{0.975}\hat{\sigma}_{\hat{\theta}_JK}$$

where $\hat{\sigma}_{\hat{\theta}_JK}$ is an estimator for the variance of the Jackknife estimator calculated using the equation:

$$\hat{\sigma}_{\hat{\theta}_JK} = (\frac{1}{A(A-1)} \sum_{a-1}^{A} (\hat{\theta}_{m,a} - \hat{\theta}_{JK})^2)^{\frac{1}{2}}$$

The code in Listing 6.11 was used to calculate the results listed in Table 3.1 using the methods as described in this section.

| | $\hat{\theta}_{JK}$ | $\hat{\sigma}^2_{\hat{\theta}_{JK}}$ | **Lower C.I.** | **Upper C.I** |
|---|---|---|---|---|
| **median** | 25.31392 | 2.0049388 | 22.53869 | 28.08915 |
| **lower quartile** | 16.68231 | 0.2630999 | 15.67698 | 17.68764 |
| **upper quartile** | 31.27063 | 3.8049424 | 27.44747 | 35.09379 |

Table 3.1: Results of Jackknife Estimation

## 3.3 Bootstrap Estimator

In this section the theory and methods used to come up with the Bootstrap estimated values for the median, lower quartile and upper quartile of the provided dataset shall be covered.

Similarly to the Jackknife estimator, the Bootstrap estimator resamples the sample data in order to study the effect of the change on the estimator. However, with Bootstrap estimation, resampling is done with replacement. In fact, Bootstrap estimation replaces the unknown population distribution of the data by an estimate of it. This is achieved by using the empirical distribution defined below:

**Definition:** For a sample $(Y_1, ..., Y_n)$of independent real-valued random variables with distribution $F$, we define the empirical distribution $\hat{F}$ by:

$$\hat{F}(A) = \frac{1}{n}\sum_{i=1}^{n} 1_A(Y_i)$$

for a set $A \in \mathbb{R}$ such that $A$ has a non-zero Lebesgue measure.

To sample $m$ values from vector $X$ from the empirical distribution in R, the function 'sample(X,m, replace = TRUE)' can be used. From each Bootstrap sample, an estimate of the parameter of interest $\theta$ is then obtained (denoted by $\hat{\theta}_1^{(m)}, ..., \hat{\theta}_B^{(m)}$). The Bootstrap estimator is then formed by taking

a simple average of the separate estimators, hence

$$\hat{\theta}_{BS} = \frac{1}{B}\sum_{b-1}^{B}\hat{\theta}_b^{(m)}$$

In order to obtain the normal confidence interval of this estimator, the standard deviation of $\hat{\theta}_{BS}$ can be obtained using the formula

$$\hat{\sigma}_{\hat{\theta}_{BS}} = [\,\frac{1}{B-1}\sum_{b=1}^{B}(\hat{\theta}_b^{(m)} - \hat{\theta}_{BS})^2]^{\frac{1}{2}}$$

Hence the normal confidence interval is calculated in a similar way to that of the Jackknife estimator.

Since the empirical distributions was used to estimate the CDF of the underlying distribution, it is also possible to obtain empirical Bootstrap confidence intervals by ordering $\hat{\theta}_1^{(m)}, ..., \hat{\theta}_B^{(m)}$ and selecting the $\frac{B\alpha}{2}$-th and $\frac{B(1-\alpha)}{2}$-th confidence intervals.

The code in Listing 6.12 was used to calculate the results listed in Table 3.2 using the methods as described in this section.

| | $\hat{\theta}_{JK}$ | $\hat{\sigma}^2_{\hat{\theta}_JK}$ | **Norm. C.I.** | **Norm. C.I** | **Emp. C.I** | **Emp. C.I** |
|---|---|---|---|---|---|---|
| **median** | 24.48435 | 3.001454 | 21.08877 | 27.87993 | 21.44247 | 27.72782 |
| **lq** | 16.12052 | 1.030456 | 14.13094 | 18.11011 | 14.47880 | 18.35909 |
| **uq** | 32.41878 | 3.961275 | 28.51787 | 36.31969 | 29.28571 | 36.16872 |

Table 3.2: Results of Bootstrap Estimation

## 3.4 Conclusion

From the results obtained in sections 3.2 and 3.3 it seems that for this particular sample the Jackknife estimator has a lower variance than the Bootstrap estimator. The values for the estimators are similar as well as their confidence intervals.

Resampling methods have wide applications in the world of statistics today as computations are less expensive nowadays than they were a few years ago. Some applications of such methods are in robust statistics and hypothesis testing.

# Chapter 4

# Worksheet 4

## 4.1 Introduction

Stochastic optimisation is a technique used when when trying to find an optimal value of a problem or function using random or Monte Carlo methods. Recently, these algorithms have been growing rapidly in popularity and application and are capable of solving challenging optimisation problems [3]. Methods in stochastic optimisation provide methods of coping with nonlinear, multidimensional problems where classical optimisation would be either impossible to use or very hard to compute. Two examples of stochastic optimisation algorithms are the 'Blind Random Search' and the 'Localised Random Search'.

In this worksheet, a maximization problem shall be solved in Section 4.2 where the optimal shipment with the maximum value based on some provided data and a weight restriction will be estimated using the blind search method. A plot of the iteration number versus the maximum value reached up to that point will also be presented.

In Section 4.3, the global minimum of a function with 4 local minima will try to be obtained using the localised random search.

## 4.2 Task A

The Blind Random Search algorithm is as follows (*Taken from [3]*):

1. (*Initialization*) Choose an initial value of $\theta$, say $\hat{\theta}_0 \in \Theta$, either randomly or deterministically. (If random, usually a uniform distribution on $\Theta$ is used.) Calculate $L(\hat{\theta}_0)$. Set $k = 0$.

2. Generate a new independent value $\theta_{new}(k+1) \in \Theta$, according to the chosen probability distribution. If $L(\theta_{new}(k+1)) < L(\hat{\theta}_k)$, set $\hat{\theta}_{k+1} = \theta_{new}(k+1)$. Else, take $\hat{\theta}_{k+1} = \hat{\theta}_k$.

3. Stop if the maximum number of $L$ evaluations has been reached or the user is otherwise satisfied with the current estimate for $\theta$ via appropriate stopping criteria; else, return to Step 2 with the new $k$ set to the former $k+1$.

In the case of this particular optimisation problem, $\Theta$ is the set of all possible combinations of items to ship that have their weight $\leq 100$. The problem states that the value of each item needs to be maximised, thus the function $L(\boldsymbol{\theta}) = -\sum_{i=1}^{n_{\theta_i}} value(\theta_i)$, where $n_\theta$ is the total number of items being shipped for that particular $\theta_i$.

The above algorithm was run in R (see Listing 6.13) and the optimal solution, the composition and value of the shipment considered in the $100000^{th}$ iteration and the plot of iteration number versus the maximum value reached up to that point can be seen in Tables 4.1, 4.2 and Figure 4.1 respectively.

| **Item** | **1** | **2** | **3** | **4** |
|---|---|---|---|---|
| **Count** | 8 | 4 | 17 | 22 |

Table 4.1: Optimal Solution

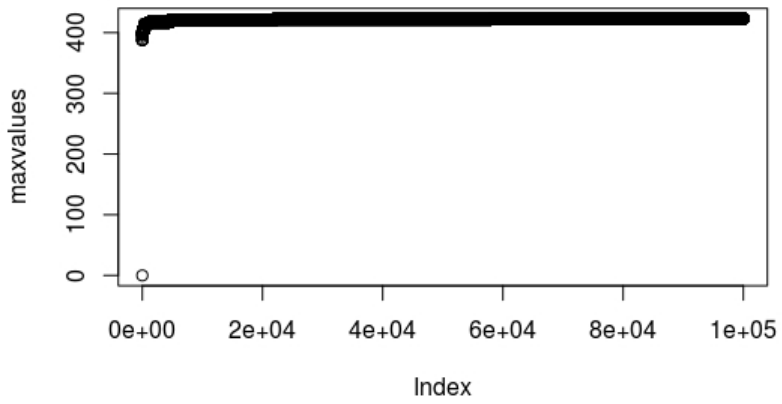
Weight: 11, Value: 423

| **Item** | **1** | **2** | **3** | **4** |
|---|---|---|---|---|
| **Count** | 10 | 8 | 12 | 10 |

Table 4.2: Shipment Considered in the $100000^{th}$ iteration

Weight: 98, Value: 388

Figure 4.1: Plot of Optimal Value vs. Iteration




## 4.3 Task B

Blind search is the simplest random search algorithm as the generation of the new $\theta$'s does not depend on the location of the previous estimates of $\theta$. The Localized Random Search algorithm however is slightly more sophisticated since the random sampling is a function of the position of the current best estimate for $\theta$.

The Localized Random Search algorithm is worked out as follows: (*Taken from [3]*)

1. (*Initialization*) Pick an initial guess $\hat{\boldsymbol{\theta}}_0 \in \Theta$, either randomly or with prior information. Set $k = 0$.

2. Generate an independent random vector $\mathbf{d}_k \in \mathbb{R}^p$ and add it to the current $\boldsymbol{\theta}$ value, $\hat{\boldsymbol{\theta}}_k$. Check if $\hat{\boldsymbol{\theta}}_k + \mathbf{d}_k \in \Theta$. If $\hat{\boldsymbol{\theta}}_k + \mathbf{d}_k \notin \Theta$, generate a new $\mathbf{d}_k$ and repeat or, alternatively, move $\hat{\boldsymbol{\theta}}_k + \mathbf{d}_k$ to the nearest valid point within $\Theta$ or the aforementioned nearest valid point in $\Theta$.

3. If $L(\boldsymbol{\theta}_{new}(k+1)) < L(\hat{\boldsymbol{\theta}}_k)$, set $\hat{\boldsymbol{\theta}}_{k+1} = \boldsymbol{\theta}_{new}(k+1)$; else, set $\hat{\boldsymbol{\theta}}_{k+1} = \hat{\boldsymbol{\theta}}_k$.

4. Stop if the maximum number of $L$ evaluations has been reached or the user is otherwise satisfied with the current estimate for $\boldsymbol{\theta}$ via

appropriate stopping criteria; else, return to Step 2 with the new $k$ set to the former $k + 1$.

The task in this case is to find the minimum of the function:

$$f(x, y) = 0.5[(x^4 - 16x^2 + 5x) + (y^4 - 16y^2 + 5y)]$$

which has 4 local minima. This was achieved using the code in Listing 6.14, where $\hat{\boldsymbol{\theta}}_0$ was initially set to $(4, 6.4)'$. Each direction vector $\mathbf{d}_k$ was randomly generated from a normal distribution with mean $\mathbf{0}$ and variance-covariance matrix $k\mathbf{I}$, where $k$ varied between 1 and 10 in order to find the best value of k and $\mathbf{I}$ is the 2x2 identity matrix. The test was then run 100 times for each k, with 500 iterations for each test.

As can be seen in Table 4.3, the global minimum found by this algorithm seems to be at around $-78.2$. Table 4.3 also shows the variance of the minimum values found in each iteration and when running the algorithm multiple times, taking $k \geq 6$ seems to give a small variance, with the smaller variances being at $k = 7, 8$ or 9. This could mean that taking such values for $k$ would be more optimal in finding the global minimum. This result is further shown in Figures 4.2-**??**, these figures also show the values of the local minima that the algorithm occasionally converges to. As can be seen in these plots, there are three values that the algorithm converges to, however we know that the function has 4 local minima, thus it must be the case that 2 of the local minima have the same value.

| k | Mean | Variance | Minimum |
|---|---|---|---|
| **1** | -50.98794 | 13.1710450 | -64.19301 |
| **2** | -55.55735 | 62.7123604 | -78.24301 |
| **3** | -66.94815 | 103.8027358 | -78.33123 |
| **4** | -72.85456 | 53.7120605 | -78.32145 |
| **5** | -76.69233 | 15.8632589 | -78.32419 |
| **6** | -77.34126 | 4.7580555 | -78.31965 |
| **7** | -77.68830 | 0.7183572 | -78.32616 |
| **8** | -77.36123 | 3.4399416 | -78.32612 |
| **9** | -77.41022 | 1.8912071 | -78.32527 |
| **10** | -77.38476 | 2.3759828 | -78.31206 |

Table 4.3: Results of 1 Localized Random Search run

Figure 4.2: Plot of Optimal Values for k = 2

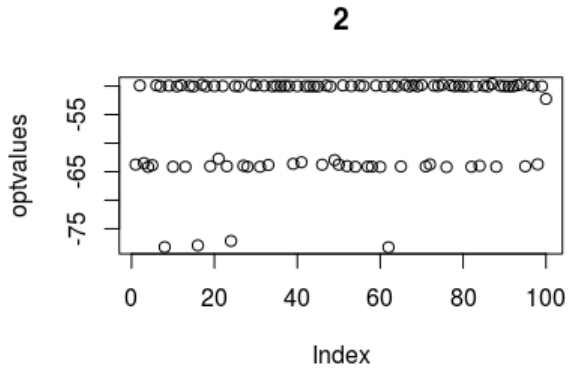


Figure 4.3: Plot of Optimal Values for k = 4

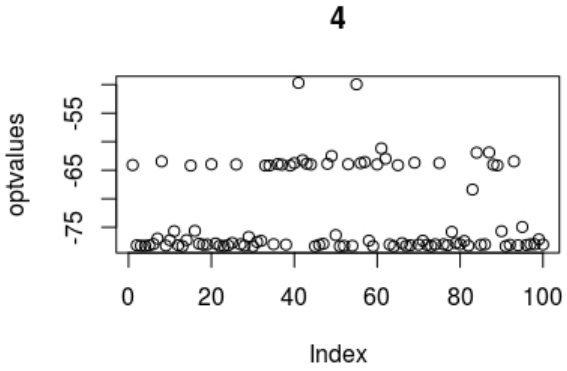



Figure 4.4: Plot of Optimal Values for k = 6

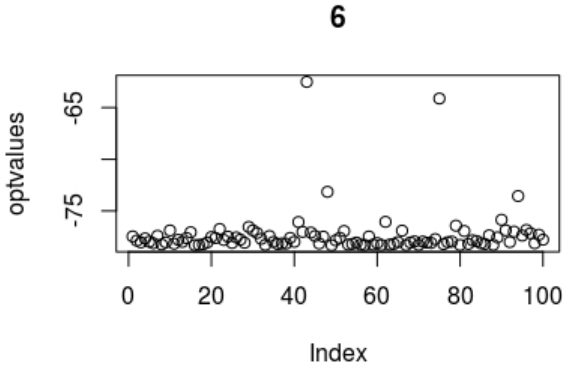
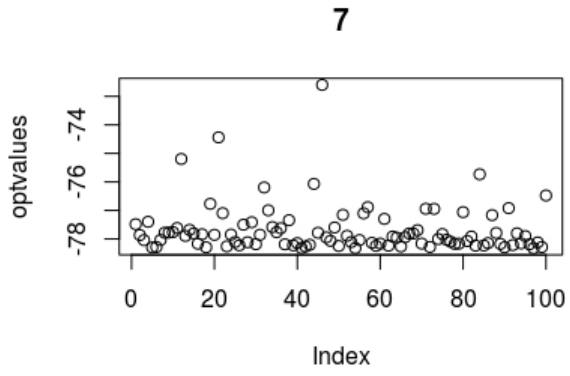


Figure 4.5: Plot of Optimal Values for k = 7




## 4.4 Conclusion

Stochastic optimisation algorithms have been growing in popularity in recent decades and have many applications in many different fields. Some applications include the analysis, design, and operation of modern systems, design of experiments, response surface, engineering, and business [3].

However, these algorithms are computationally expensive, in fact the complexity of the localised blind search algorithm implemented in Section 4.3 is $O(n * b)$, where $n$ is the number of experiments and $b$ is the number of iterations per experiment.

# Chapter 5

# Worksheet 5

## 5.1 Introduction

In Section 2.3, the idea of an Importance Sampler was used in order to estimate the value of a parameter with good accuracy. This technique can also be used to reduce the variance of the Crude Monte Carlo estimator. However, other methods also exist to accomplish this task, such as making use of Control Variates and Antithetic Variables.

In Section 5.2, the concept of a control variate shall be explored. The expectation $\mathbb{E}[exp(-U^2)]$ for $U \sim Unif(0,2)$ will initially be worked out using the Crude Monte Carlo Integral, then using $2exp(-2U)$ as the control variate. This will be done for different sample sizes and the results will then be compared.

In Section **??**, the expectation $\mathbb{E}[ln(1+X^2)]$ where $X \sim Exp(1)$ will be evaluated using the Crude Monte Carlo Integral and using the antithetic variates method for multiple sample sizes and their results also compared.

## 5.2 Task A

For this example, the desired simulation quantity $\theta = \mathbb{E}[exp(-U^2)]$, where $U \sim Unif(0,2)$, thus let $X = exp(-U^2)$. By including the control variate

$Y = 2exp(-2U)$, $\mu_Y = \mathbb{E}[Y]$ is known (See Equation 5.1)

$$\begin{aligned}
\mu_Y &= \mathbb{E}[2exp(-2U)] \\
&= 2\mathbb{E}[exp(-2U)] \\
&= 2\int_0^2 e^{(-2u)}.\frac{1}{2}du \\
&= \int_0^2 e^{(-2u)}du \\
&= \frac{1}{2} - \frac{1}{2e^4} \approx 0.490842
\end{aligned} \tag{5.1}$$

Thus for any $c$, Random Variable $Z = X + c(Y - \mu_Y)$, is an unbiased estimator of $\theta$, because $\mathbb{E}[Z] = \mathbb{E}[X] + c(\mathbb{E}[Y] - \mu_Y) = \theta$ [4].

The control variate method has an improved variance to the Crude Monte Carlo method when $Y$ is correlated with $X$ because $Var(X + c^*Y) = Var(X) - Cov(X,Y)^2/Var(Y)$, where $c^* = -Cov(X,Y)^2/Var(Y)$. These values can all be estimated from the data as follows:

$$\begin{aligned}
Cov(X,Y) \approx \hat{Cov}(X,Y) &= \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y}) \\
Var(Y) \approx \hat{Var}(Y) &= \frac{1}{n-1}\sum_{i=1}^{n}(Y_i - \bar{Y})^2 \\
c^* \approx \hat{c^*} &= -\frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}
\end{aligned}$$

The above information was then used to write the code in Listing 6.15 to compare Crude Monte Carlo integration with the control variate method. The results can be seen in Table 5.1.

| N | $\hat{\theta}_{crude}$ | $\hat{\theta}_{cv}$ | $S^2_{crude}$ | $S^2_{CV}$ |
|---|---|---|---|---|
| 100 | 0.4101733 | 0.3837011 | 1.208492e-03 | 1.931236e-04 |
| 1000 | 0.4297848 | 0.4212441 | 1.123531e-04 | 1.728845e-05 |
| 10000 | 0.4401700 | 0.4386240 | 1.186225e-05 | 1.684340e-06 |
| 100000 | 0.4425477 | 0.4437668 | 1.190590e-06 | 1.689741e-07 |

Table 5.1: Results of Control Variate Method

As can be seen in Table 5.1, this method significantly reduces the variance

in the estimator, thus giving a more accurate reading for the desired result.

## 5.3 Task B

The idea behind antithetic variables is as follows; if $X_1$ and $X_2$ are identically distributed Random Variables with mean $\theta$ then $Var(\frac{X_1+X_2}{2}) = \frac{1}{4}(Var(X_1)+Var(X_2)+2Cov(X_1,X_2))$, so the variance is reduced if $X_1$ and $X_2$ have $Cov(X_1,X_2) \leq 0$. In the case of this worksheet, $\theta = \mathbb{E}[Y]$ where $Y = ln(1+X^2)$ and $X \sim Exp(1)$.

The antithetic variables method has a smaller variance to that of the Crude Monte Carlo Method when $X_1$ and $X_2$ have a negative correlation. Thus by letting $Y = X_1$ in the given example, and letting $X_2 = ln(1+(1-X)^2)$, the new estimator $\hat{\theta}_{av} = \mathbb{E}[\frac{1}{2}(X_1+X_2)]$ will have a smaller variance to the Crude Monte Carlo Estimate (as seen in Table 5.2).

| N | $\hat{\theta}_{crude}$ | $\hat{\theta}_{av}$ | $S^2_{crude}$ | $S^2_{av}$ |
|---|---|---|---|---|
| 100 | 0.7565874 | 0.6385673 | 3.517973e-03 | 4.449314e-03 |
| 1000 | 0.6776703 | 0.5630815 | 2.931409e-04 | 3.596706e-04 |
| 10000 | 0.6931628 | 0.5755416 | 2.953411e-05 | 3.555261e-05 |
| 100000 | 0.6887140 | 0.5744264 | 2.954970e-06 | 3.540121e-06 |

Table 5.2: Results of Antithetic Variables Method

Note that the variance for the antithetic variables method is smaller that that of the Crude Monte Carlo method for all values of $N$ in the above example. The code used to obtain the results in Table 5.2 can be seen in Listing 6.16.

## 5.4 Conclusion

Variance reduction is a very important tool to improve the accuracy of Monte Carlo estimators. As can be seen in Tables 5.1 and 5.2, the variance of these estimators were significantly and consistently reduced. The above methods have vast applications including Bayesian Statistics and Machine Rendering.

# Chapter 6

# Appendix

## 6.1 Worksheet 1

```
1  rm(list=ls())
2  #set mu to be the mean of the distribution
3  mu = c(1,0,-0.5)
4  #Set A to the variance covariance matrix
5  A = matrix(c(3,1,0.3,1,1.5,0.5,0.3,0.5,2),nrow=3,
     ncol=3)
6  #Cholesky Factorisation of matrix A
7  C = t(chol(A))
```

Listing 6.1: Cholesky Factorisation of variance covariance matrix

```
1  p = 3 #size of row of A
2  #Assign memory for generated random vectors
3  X = mat.or.vec(10000,p)
4  #simulate 10000 readings:
5  for (reading in 1:10000) {
6    #generate std. normal random number
7    Y = rnorm(p)
```

```
8     #Work out the value of the random number following
      our distribution
9     temp = C%*%Y+mu
10    #Save the values in matrix X
11    for (i in 1:p) {
12      X[reading,i] = temp[i,1]
13    }
14  }
15
16  par(mfrow=c(3,1))
17  hist(X[,1])
18  hist(X[,2])
19  hist(X[,3])
20
```

Listing 6.2: Generating the 10000 samples from the trivariate normal distribution

```
1  #Code used to generate the variance-covariance
    matrix of X
2  Ones = matrix(rep(1,len=p*10000),nrow=10000,ncol=p)
3  a = X-Ones*X*(1/10000)
4  V = t(a)%*%a/10000
5
```

Listing 6.3: Working out the generated Variance Covariance Matrix

```
1  rm(list=ls())
2
3  #create a sequence of numbers beween 0 and 5
    incrememnting by 0.01
4  x = seq(0,5,by = .01)
5  #create a std normal distribution
6  y = dnorm(x,0,1)
```

```
7
8   #create multiple exponential distributions
9   e1 = dexp(x,1)
10  e8 = dexp(x,0.8)
11  e65 = dexp(x,0.65)
12  e5 = dexp(x,0.5)
13
14  #plot generated distributions and add a legend to
     the plot
15  plot(c(0,5),c(0,1),type='n',
16     ylab='f(x)')
17  lines(x,y)
18  lines(x,e1,col='red')
19  lines(x,e8,col='cyan')
20  lines(x,e65,col='magenta')
21  lines(x,e5,col='green')
22  legend(2,1,c('Std. Normal','exp. with a = 1'
23          ,'exp. with a = 0.8',
24          'exp. with a = 0.65',
25          'exp. with a = 0.5'),
26        lty=c(1,1,1,1,1), lwd=c(.5,.5,.5,.5,.5),
27        col=c('black','red','cyan','magenta','green'))
28
```

Listing 6.4: Generating Exponential Distributions with varying Majorising Constant a

```
1   rm(list=ls())
2   M = 0.8 #majorising constant
3   noToGenerate = 10000 #num of points required to
     generate
4   noGenerated = 0 #num. of points accepted, to stop
     when this reaches 10000
5   generatedRandomNumbers=mat.or.vec(1,noToGenerate) #
     vector to store numbers generated
```

```
6   ptr = 1 #pointer for vector
7   loopCount = 0 #loop counter to calculate efficiency
8   #main loop:
9   while (noGenerated < noToGenerate) {
10    x = rexp(1) #x~exp(1)
11    #generate y on (0,ae^-x)
12    y = runif(1,0,M*exp(-x))
13    #value of normal distribution at x
14    normconst = (1/sqrt(2*pi))*exp(-(x^2)/2)
15    if (y <= normconst) {
16      #accept the value
17      #give it a sign with probability 0.5:
18      sign = runif(1)
19      if (sign <= 0.5) {
20        generatedRandomNumbers[1,ptr] = -y
21      } else {
22        generatedRandomNumbers[1,ptr] = y
23      }
24    ptr = ptr + 1
25    #increment no Generated
26    noGenerated = noGenerated+1
27    }
28    loopCount =loopCount + 1
29  }
30
31  hist(generatedRandomNumbers[1,])
32
```

Listing 6.5: Generating Random Numbers using the Accept Reject Method

```
1   rm(list=ls())
2   M = c(0.9,0.8,0.7,0.68) #majorising constant
3   counts = mat.or.vec(length(M),1)
4   #loop counter to calculate efficiency
5   for (i in (1:length(M))) {
```

```
6     noToGenerate = 10000 #num of points required to
      generate
7     noGenerated = 0 #num. of points accepted, to stop
      when this reaches 10000
8     generatedRandomNumbers=mat.or.vec(1,noToGenerate)
      #vector to store numbers generated
9     ptr = 1 #pointer for vector
10    loopCount = 0
11    #main loop:
12    while (noGenerated < noToGenerate) {
13      x = rexp(1) #x~exp(1)
14      #generate y on (0,ae^-x)
15      y = runif(1,0,M[i]*exp(-x))
16      #value of normal distribution at x
17      normconst = (1/sqrt(2*pi))*exp(-(x^2)/2)
18      if (y <= normconst) {
19        #accept the value
20        #give it a sign with probability 0.5:
21        sign = runif(1)
22        if (sign <= 0.5) {
23          generatedRandomNumbers[1,ptr] = -y
24        } else {
25          generatedRandomNumbers[1,ptr] = y
26        }
27        ptr = ptr + 1
28        #increment no Generated
29        noGenerated = noGenerated+1
30      }
31      loopCount =loopCount + 1
32    }
33    counts[i] = loopCount
34  }
```

Listing 6.6: Final Code for Task B

## 6.2 Worksheet 2

```
1  rm(list=ls())
2
3  N <-c(100,1000,10000,100000,1000000)
4  thetahat <- mat.or.vec(5,1)
5  for (i in 1:5) {
6    x <- runif(N[i],2,8)
7    thetahat[i] <- 6*mean(sin(x)/log(x))
8  }
9  Table <- rbind(N, thetahat)
10 Table
```

Listing 6.7: Crude Monte Carlo Integration

```
1  rm(list=ls())
2
3  N <-c(100,1000,10000,100000,1000000)
4  thetahat <- mat.or.vec(5,1)
5  variance <- mat.or.vec(5,1)
6  varhatthetahat <- mat.or.vec(5,1)
7  for (i in 1:5) {
8    x <- runif(N[i],2,8)
9    #Monte Carlo Estimate
10   thetahat[i] <- 6*mean(sin(x)/log(x))
11   #Actual Variance
12   variance[i] <- ((N[i]-1)^-1)*mean(((sin(x)/log(x))
     +0.053792)^2)
13   #variance of the Monte Carlo Integral
14   varhatthetahat[i]<-((N[i]-1)^-1)*mean(((sin(x)/log
     (x))-thetahat)^2)
15 }
16 Table <- rbind(N, thetahat, variance, varhatthetahat)
17 Table
```

Listing 6.8: Code Used to answer Task 2A

```
1  rm(list=ls())
2
3  K<-c(2,3,4,5,6,7,8,9,10)
4  N<-c(100, 1000, 10000, 100000, 1000000)
5  thetahat<-mat.or.vec(9,5)
6  varhatthetahat<-mat.or.vec(9,5)
7  for (i in 1:5) {
8    for (j in 1:9) {
9      u<-runif(N[i],0,1)
10     x<-((20^(K[j]-1))/(1-u))^(1/(K[j]-1))
11     thetahat[j,i]<-mean(dnorm(x)/((K[j]-1)*20^(K[j
       ]-1)*x^(-K[j])))
12     varhatthetahat[j,i]<-(N[i]-1)^(-1)*mean(dnorm(x)
       /((K[j]-1)*20^(K[j]-1)*x^(-K[j]))-thetahat[j,i])^2
13   }
14 }
```

Listing 6.9: Code Used to answer Task 2B

```
1  rm(list=ls())
2
3  x <- seq(-5,5,length=1000);
4  y = dnorm(x);
5  plot(x,y,type="l",xlab="x value", ylab="Density",
     main="Standard Normal Distribution Plot");
6
7  rm(list=ls())
8  x <- seq(15,35,length = 100);
9  Phi <- mat.or.vec(100,1)
10 g <- mat.or.vec(100,1)
11 for (i in 1:length(x)) {
12   if(x[i]<=20){
13     Phi[i] = 0;
```

```
14        g[i] = 0;
15      } else {
16        Phi[i] = dnorm(x[i]);
17        g[i] = 4*20^4*(x[i])^(-5);
18      }
19    }
20    #plot(x,Phi,type="l",xlab="x value", ylab="Density",
        main="Plot of Phi for x > 20",col="red");
21    #lines(x,Phi,col="red")
22    plot(x,g,type="l",xlab="x value", ylab="Density",
        main="Plot of Importance Sampler g");
```

Listing 6.10: Code Used to Plot Certain Figures in Task 2

## 6.3 Worksheet 3

```
1   rm(list=ls())
2
3   n = 100 #size of sample
4   A = 5 #no. of partitions
5   m = 20 #partition size
6   medianhat = mat.or.vec(A,1)
7   lqhat = mat.or.vec(A,1)
8   uqhat = mat.or.vec(A,1)
9
10  #Choose sample.csv
11  Data = read.csv(file.choose(),header=FALSE)
12  X = Data[,1]
13  rm(Data)
14  medianX = median(X)
15  lqX = quantile(X,0.25)
16  uqX = quantile(X,0.75)
17
```

```
18  #randomly change order of sample
19  Y = mat.or.vec(n,1)
20  Y = sample(X,n,replace=F)
21
22  temp = mat.or.vec(n-m,1) #to store each sample with
     a removed partition
23
24  #loop over every partition
25  for (i in 1:A) {
26    lb = 1+(i-1)*m #first index of partition
27    ub = m*i #last index of partition
28    remove = Y[lb:ub]
29    temp = setdiff(Y,remove) #temp is Y without the
     partition
30    #calculate jackknife values:
31    medianhat[i]=A*medianX-(A-1)*median(temp)
32    lqhat[i]=A*lqX-(A-1)*quantile(temp,0.25)
33    uqhat[i]=A*uqX-(A-1)*quantile(temp,0.75)
34  }
35
36  #calculate jackknife estimates:
37  medianJackknife = mean(medianhat)
38  lqJackknife = mean(lqhat)
39  uqJackknife = mean(uqhat)
40
41  #calculate the variance of the jackknife estimates:
42  varMedianJK = (1/A)*var(medianhat)
43  varlqJK = (1/A)*var(lqhat)
44  varuqJK = (1/A)*var(uqhat)
45
46  #calculate the confidence intervals
47  CImedian = c(medianJackknife+qnorm(0.025)*sqrt(
     varMedianJK),medianJackknife+qnorm(0.975)*sqrt(
     varMedianJK))
48  CIlq = c(lqJackknife+qnorm(0.025)*sqrt(varlqJK),
     lqJackknife+qnorm(0.975)*sqrt(varlqJK))
```

```
49  CIuq = c(uqJackknife+qnorm(0.025)*sqrt(varuqJK),
      uqJackknife+qnorm(0.975)*sqrt(varuqJK))
50
51  #code for diplaying data nicely
52  medinfo = c(medianJackknife, varMedianJK, CImedian)
53  lqinfo = c(lqJackknife, varlqJK, CIlq)
54  uqinfo = c(uqJackknife, varuqJK, CIuq)
55
56  table = rbind(medinfo, lqinfo, uqinfo)
57  table
```

Listing 6.11: Jackknife Estimation

```
1   rm(list=ls())
2
3   n = 100 #size of sample
4   b = 1000 #no. of bootstrap samples to generate
5   medians = mat.or.vec(b,1)
6   lqs = mat.or.vec(b,1)
7   uqs = mat.or.vec(b,1)
8
9   #Choose sample.csv
10  Data = read.csv(file.choose(),header=FALSE)
11  X = Data[,1]
12  rm(Data)
13
14
15  #generate 1000 bootstrap readings
16  for (i in 1:b) {
17    sampleboot = sample(X,n,replace = TRUE)
18    medians[i]= median(sampleboot)
19    lqs[i] = quantile(sampleboot,0.25)
20    uqs[i] = quantile(sampleboot,0.75)
21  }
22
```

```
23  #calculate bootstrap estimates:
24  medianBootstrap = mean(medians)
25  lqBootstrap = mean(lqs)
26  uqBootstrap = mean(uqs)
27
28  #calculate the variance of the bootstrap estimates:
29  varMedianBS = var(medians)
30  varlqBS = var(lqs)
31  varuqBS = var(uqs)
32
33  #calculate the normal confidence intervals
34  CImedian = c(medianBootstrap+qnorm(0.025)*sqrt(
     varMedianBS),medianBootstrap+qnorm(0.975)*sqrt(
     varMedianBS))
35  CIlq = c(lqBootstrap+qnorm(0.025)*sqrt(varlqBS),
     lqBootstrap+qnorm(0.975)*sqrt(varlqBS))
36  CIuq = c(uqBootstrap+qnorm(0.025)*sqrt(varuqBS),
     uqBootstrap+qnorm(0.975)*sqrt(varuqBS))
37
38  #calculate the empirical confidence interval
39  sortedMed = sort(medians)
40  sortedLq = sort(lqs)
41  sortedUq = sort(uqs)
42  ECIMedian = c(sortedMed[25],sortedMed[975])
43  ECILq = c(sortedLq[25],sortedLq[975])
44  ECIUq = c(sortedUq[25],sortedUq[975])
45
46  #code for diplaying data nicely
47  medinfo = c(medianBootstrap, varMedianBS, CImedian,
     ECIMedian)
48  lqinfo = c(lqBootstrap, varlqBS,CIlq,ECILq)
49  uqinfo = c(uqBootstrap, varuqBS,CIuq, ECIUq)
50
51  table = rbind(medinfo,lqinfo,uqinfo)
52  table
```

Listing 6.12: Bootstrap Estimation

## 6.4 Worksheet 4

```
1   rm(list=ls())
2
3   #declare variables
4   b = 100000 #no of iterations
5   weights = c(4,3,2,1)
6   values = c(15,10,9,5)
7   listvalues = c(b,1)
8   maxvalues = c(b+1,1)
9   maxvalues[1] = 0
10
11  for (j in 1:b) {
12    weight = 0
13    value = 0
14    i = 1
15    item = c()
16    #loop to generate thetas
17    while (weight < 100) {
18      item[i] = sample(1:4,1) #generate a uniform
      random number between 1 and 4
19      weight = weight+weights[item[i]]
20      value = value + values[item[i]]
21      i=i+1
22    }
23    if (weight <= 100) {
24      listvalues[j] = value
25    } else {
26      #if weight is > 100, remove last element
27      value = value-values[item[i-1]]
28      listvalues[j] = value
```

```
29      }
30      if (listvalues[j]>maxvalues[j]){
31        maxvalues[j+1] = listvalues[j]
32        thetahat = item
33      } else {
34        maxvalues[j+1] = maxvalues[j]
35      }
36    }
37    plot(maxvalues, title="Optimal Value at each
       Iteration")
38    print("Optimal Solution:")
39    table(thetahat)
40    print("Last Iteration:")
41    table(item)
```

Listing 6.13: Blind Search Optimisation Algorithm

```
1   rm(list=ls())
2   require(MASS)
3
4   f <- function(x) 0.5*(x[1]^4-16*x[1]^2+5*x[1]+x
     [2]^4-16*x[2]^2+5*x[2])
5   n = 100 #no of experiments
6   b = 500 #no of iterations
7   optvalues = c()
8   optpoints = c()
9
10
11  for (k in 1:10) {
12    #for each experiment
13    for (i in 1:n) {
14      theta=c(4,6.4) # set initial theta
15      Lprev = f(theta)
16      #iterations of each experiment
17      for (j in 1:b){
```

```
18          d = mvrnorm(n=1,c(0,0),k*diag(2))
19          newTheta = theta+d
20          L = f(newTheta)
21          if(L<Lprev) {
22            theta = newTheta
23            Lprev = L
24          }
25        }
26        optvalues[i] = Lprev
27      }
28      plot(optvalues, main = k)
29      if (k==1) {
30        results = rbind(c(mean(optvalues), var(optvalues
      ),min(optvalues)))
31      } else {
32        results = rbind(results,c(mean(optvalues), var(
      optvalues),min(optvalues)))
33      }
34    }
35    results
```

Listing 6.14: Localised Random Search Algorithm

## 6.5 Worksheet 5

```
1   rm(list=ls())
2
3   N = c(100,1000,10000,100000) #sample sizes
4   muy = 0.490842 #mu of Y
5   #vector declarations:
6   crudetheta = mat.or.vec(4,1)
7   crudevariance = mat.or.vec(4,1)
8   cstar = mat.or.vec(4,1)
9   cvtheta = mat.or.vec(4,1)
```

```
10  cvvariance = mat.or.vec(4,1)
11  sigmax = mat.or.vec(4,1)
12  sigmay = mat.or.vec(4,1)
13  sigmaxy = mat.or.vec(4,1)
14
15  #main loop over all sample sizes
16  for (i in 1:4) {
17    U = runif(N[i],0,2) #generate N[i] uniformly (0,2)
      distributed RV
18    X = exp(-U^2)
19    Y = 2*exp(-2*U)
20    #Crude Monte Carlo Integral:
21    crudetheta[i] = mean(X)
22    crudevariance[i] = ((N[i]-1)^-1)*mean((X-
     crudetheta[i])^2)
23    #Calculating sigmas:
24    sigmax[i] = crudevariance[i]*N[i]
25    sigmay[i] = mean((Y-muy)^2)
26    sigmaxy[i] = mean((X-crudetheta[i])*(Y-muy))
27    #Using the control variate
28    cstar[i] = -sigmaxy[i]/sigmay[i]
29    cvtheta[i] = mean(X-cstar*(Y-muy))
30    cvvariance[i] = (sigmax[i]-(sigmaxy[i]^2/sigmay[i
     ]))/N[i]
31  }
32  #display results:
33  Table = rbind(crudetheta,cvtheta,crudevariance,
     cvvariance)
34  t(Table)
```

Listing 6.15: Control Variate Code

```
1  rm(list=ls())
2
3  #variable initialization
```

```
4   N = c(100,1000,10000,100000)
5   thetamc = mat.or.vec(4,1)
6   mcVar = mat.or.vec(4,1)
7   thetaav = mat.or.vec(4,1)
8   avVar = mat.or.vec(4,1)
9
10  #loop over different sample sizes
11  for (i in 1:length(N)) {
12    u = rexp(2*N[i])
13    X = log(1+u^2)
14    thetamc[i] = mean(X)
15    mcVar[i] = ((2*N[i]-1)^-1)*mean((X-thetamc[i])^2)
16    u2 = rexp(N[i],rate=1)
17    Y = (log(1+u^2)+log(1+(1-u)^2))/2
18    thetaav[i] = mean(Y)
19    avVar[i] = ((N[i]-1)^-1)*mean((Y-thetaav[i])^2)
20  }
21  Table = rbind(thetamc,thetaav,mcVar,avVar)
22  t(Table)
```

Listing 6.16: Antithetic Variables Code

# Bibliography

[1] StatTrek. *Variance-Covariance Matrix*. `http://stattrek.com/matrix-algebra/covariance-matrix.aspx`. [6th November 2017].

[2] scratchapixel. *Monte Carlo Methods in Practice*. `https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/monte-carlo-methods-in-practice/monte-carlo-integration`. [10th February 2018].

[3] J. C. Spall. *Handbook of Computational Statistics*. Chapter: Stochastic Optimization. Springer Heidelberg,J, Gentle, W. Hardle and Y. Mori, 2004

[4] Unknown. *Antithetic Variables, Control Variates*. `http://www.math.wsu.edu/faculty/genz/416/lect/l08-12.pdf`. [14th February 2018].