Generating Generalized Exponentially Distributed Random Variates with Transformed Density Rejection and Ratio-of-Uniform Methods

Yik Yang

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical Engineering

William H. Tranter, Chair
R. Michael Buehrer
Wayne A. Scales

December 21, 2004
Blacksburg, Virginia

Keywords: Transformed Density Rejection, Generalized Exponential pdf, Random Variates Generation, Ratio-of-Uniform

Generating Generalized Exponentially Distributed Random Variates with Transformed Density Rejection and Ratio-of-Uniform Methods

Yik Yang

(ABSTRACT)

To analyze a communication system without the aid of simulation, the channel noise for the simulation must be assumed to be normal. The assumption is often valid, but the normal distribution may not be able to model the channel noise adequately in some environments. This thesis will explore the generalized exponential distribution for better noise modeling and robustness testing in communication system.

When using the generalized exponential distribution for the channel noise, the analysis will become analytically intractable, and simulation becomes mandatory. To generate the noise with the distribution, the rejection method can be used. However, since the distribution can take on different shapes, finding the appropriate Upper Bounding Function (UBF) for the method is very difficult. Thus, two modified versions of the rejection method will be examined. They are the Transformed Density Rejection (TDR) and Ratio-of-Uniform (RoU) method; their quality, efficient, trade-offs, etc will be discussed.

Choosing TDR, a simulation of a BPSK communication system will be performed. With the simulation, it can further ascertain that the random variates generated by TDR can be used to model the channel noise and to test the robustness of a communication system.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In order to analyze a communication system without the aid of simulation, many assumptions must be made to make the analysis analytically tractable [10]. The assumption of the channel noise being normally distributed is one of the most common assumptions. The normal (Gaussian) assumption is usually valid, because the noise is the combination of different random factors, and according to the central limit theorem, if the random variable X is the sum of an infinite number of random variables for any kinds of distributions, X is normally distributed [7]. However, for many practical channels, the noise may not be adequately modeled by the normal distribution. Nevertheless, the distribution of the noise may still resemble the normal distribution and can be represented by the generalized exponential distribution

$$f_\nu(x) = \frac{\nu}{\sqrt{8}\sigma\Gamma(\frac{1}{\nu})}\ exp\left(-\left|\frac{x-m}{\sqrt{2}\sigma}\right|^\nu\right) \tag{1.1}$$

where ν is the mode, and $\Gamma(.)$ is the gamma function. Note that for $\nu = 2$, $f_\nu(x)$ is normal

and σ is the standard deviation. The generalized exponential distribution could potentially be a better model for channel noise since the mode can be adjusted to fit an observed pdf. By building a noise generator that generates random variates with the generalized exponential distribution, we are able to model the noise and other channel disturbances of a channel more accurately. However, the analysis will become analytically intractable, and we will have to turn our attention to simulation. In addition to being a more flexible model for the channel, the generalized exponential random variates could also test the robustness of a system under the Gaussian assumption. By changing the mode ν of the generalized exponential pdf, we can generate noise that deviates from the normal distribution by a desirable amount to see the change in system performance under a slight deviation from the Gaussian noise assumption. As an example, one could execute a simulation of a given system over a range of ν and thereby test the departure of the performance measure (BER for example) from the $\nu = 2$ (Gaussian) result. In order to perform simulations for the mode parameter in the neighborhood of $\nu = 2$ we need an algorithm for generating random numbers from a generalized normal distribution. The development and testing of such an algorithm is the subject of this thesis.

To generate generalized exponentially distributed random variates, we must examine some techniques for random variates generation. When a random sequence with a specific pdf, $f_x(x)$, is desired, the inverse transform method would usually be considered first. The method requires the inverse CDF, $F^{-1}(U)$ of $f_x(X)$. If $F_x(X)$ or $F^{-1}(X)$ proves to be difficult or impossible to evaluate, the rejection method would then be considered due to its ability to generate random sequences for any pdf. To generate a random sequence with the rejection method, we need to find a Upper Bounding Function (UBF, $h(x)$. The UBF $h(x)$ is a quasi-density, a pdf that is multiplied by an constant in our context, $Cf_h(X)$, that is greater or equal to the target pdf, $f_x(x)$ at any given point, $h(x) \geq f_x(x)$. The $f_h(X)$ for the UBF must

have an inverse CDF that can be obtain with moderate effort, so that we can use the inverse transform method to generate random sequences with the pdf $f_h(X)$. After we generate a sequence with the quasi-density $h(x)$, we can implement a rejection process which rejects the random variates that fall outside the boundary between $f_x(x)$ and the x-axis. After the rejection process is completed, we obtain a random sequence with the distribution of $f_x(x)$, hence the name rejection method.

For the rejection method, we would like to minimize the difference between $h(x)$ and $f_x(x)$ in order to decrease the number of random variates rejected, i.e., increase the efficiency of the method. If we use the rejection method to generate random variates with the pdf given in (1.1), we must consider the mode, ν, of the pdf. By changing the value for ν, the pdf can change into different types of pdfs within the generalized exponential class as shown in Figure 1.1. For example, if we set $\nu = 1$, we will have the Laplacian distribution. If we set $\nu = 2$, we will obtain the Gaussian or normal distribution. In the limit as $\nu \to \infty$, we will obtain the uniform distribution. Thus, the generalized exponential pdf encompasses an infinite number of different distributions with each distribution having a different mode, ν.

To generate random variates with the generalized exponential pdf, the inverse transform method is not suitable since the cumulative distribution function is not obtainable in closed form. Thus, we resort to the rejection method. For the rejection method, finding a tight UBF for the pdf proves to be very difficult. Thus, we will examine two modified versions of the rejection method, the Transform Density Rejection (TDR) method and Ratio-of-Uniform (RoU) method. The TDR can provide us a way to a find a tight UBF for the rejection method that causes the efficiency of the implementation to approach 100% (the efficiency is the ratio of the accepted random variables to the total number generated by $f_h(X)$ in the UBF), but it has an expensive setup (code requiring execution before the generation of random numbers), and a complicated UBF. On the other hand, the RoU is not very efficient, but it is very

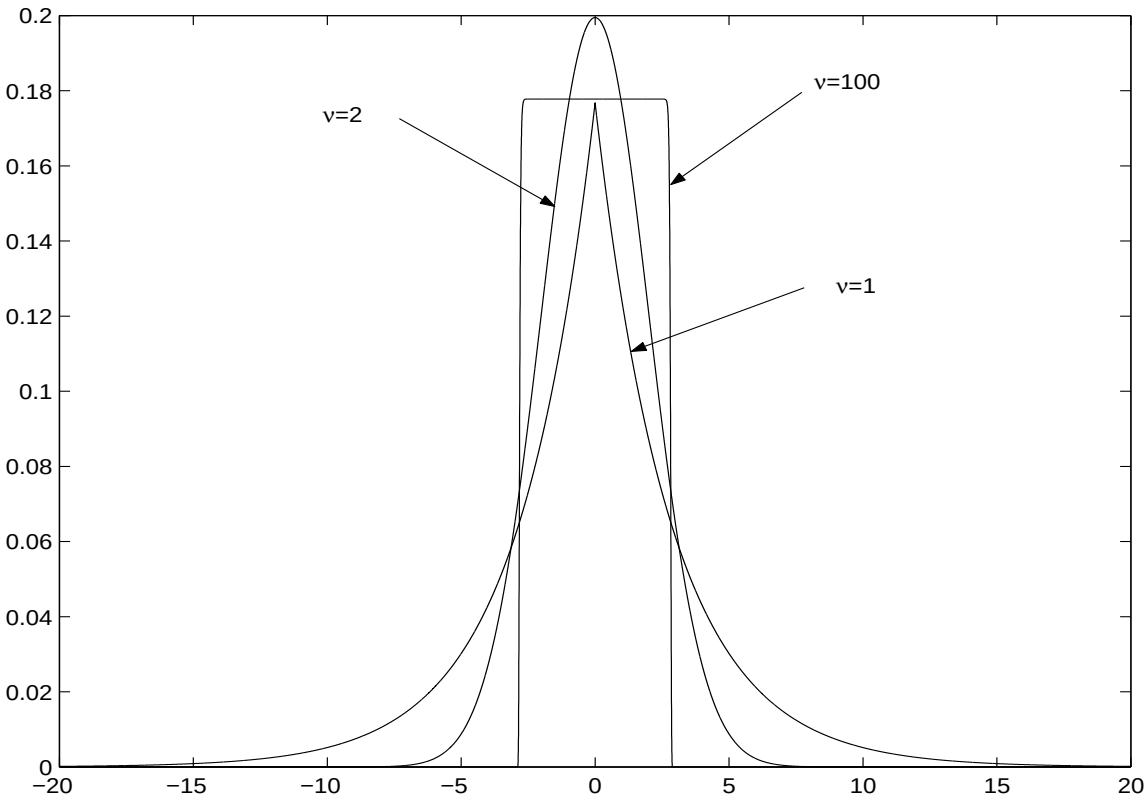


Figure 1.1: *The pdf* $f_\nu(X)$ *for* $\nu = 1, 2, 100$

fast and simple, virtually no setup is required. After examining the two methods, we will use one of the method to generate random variates with the generalized exponential pdf as the channel noise for a BPSK communication system. Then, we will analyze the impact of varying ν (mode) on the performance of a communication system.

Chapter 2

Background

Before we examine the Transformed Density Rejection (TDR) and the Ratio-of-Uniform (RoU) methods, we will go over several topics that are essential in understanding the methods. These topics are inverse transform, rejection, composition-rejection, and index-search methods. These methods are used for generating continuous and discrete random sequences with a given pdf, and the methods are critical components of TDR and RoU.

2.1 The Inverse Transform Method

The inverse transform method allows us to transform an uncorrelated uniformly distributed sequence U with domain $[0, 1]$ into an uncorrelated random sequence X with the cumulative density function (CDF) $F_X(x)$ of the pdf $f_X(x)$ [10]. In order to derive the transformation, we set the CDF $F_X(x)$ equal to U, i.e.

$$U = F_X(X) \tag{2.1}$$

The next step is to express X in terms of U, i.e., we are taking the inverse of $F_X(X)$ to obtain $F_X^{-1}(U)$ and setting the inverse equal to X, so that

$$X = F_X^{-1}(U) \tag{2.2}$$

The algorithm for the inverse transform method is given in Algorithm 1. The method is very easy and straightforward. However, computing $F_X(X)$ and $F_X^{-1}(U)$ for the desired pdf in closed form is not possible for many cases. The Gaussian pdf is an example in which the pdf can be expressed in closed form but the CDF cannot be expressed in closed form. Thus, we must find an alternative way to generate random sequences in such cases.

Algorithm 1 Inverse transform method [3]

Require: The inverse CDF $F_X^{-1}(U)$ of the desired $f_X(x)$ and number of random variates wanted, N.

Ensure: The random sequence $X = [X_1, X_2, \ldots, X_N]$ with the given pdf, $f_X(x)$

```
i = 1
while i ≠ N do
    U ~ U(0,1) (U is equal to a random number in a random sequence that is uniformly
    distributed in [0,1])
    X_i = F_x^{-1}(U)
    Return X_i
    i = i + 1
end while
```

2.2 The Rejection Method

In many situations, we may not be able to use the inverse transform method directly, since we cannot obtain the inverse CDF in closed form. However, we can always use the rejection method, although it may not be very efficient.

The rejection method is a universal method for generating random variates. It is a universal

method, because it is able to generate random numbers for any target pdf. However, when the method is implemented, the simulation time could be a concern (too slow) if the efficiency of the algorithm is too low.

With the rejection method, an UBF $h(x)$, a quasi-density, a distribution multiplied by a constant in our context, must first be selected. The UBF must be equal to or greater than $f_X(x)$ of the desired random sequence. In order to maximize the efficiency of the method, we must choose $h(x)$ that minimizes the difference between $f_X(x)$ and $h(x)$, so that the bound is as tight as possible. Also, the inverse CDF, $H^{-1}(U)$ of $h(x)$ must be easy to obtain, so that we can use $h(x)$ to efficiently generate random variates with the inverse transform method and reject the variates that fall outside of the area between $f_X(x)$ and the x-axis to obtain the desired random sequence X.

Figure 2.1 shows $h(x)$, a quasi-density proportional to an exponential distribution, and $f_X(x)$, the desired pdf for our illustration. Figure 2.1 is constructed for a gamma distribution. In the figure, the x-coordinates of the circles and dots represent the random variates that are generated with $h(x)$ using the inverse transform method. The y-coordinates of the circles and dots represent random variates that are uniformly distributed between 0 and $h(X_i)$ with X_i being the x-coordinates. The dots fall outside above the gamma distribution, so they are rejected; and the circles are accepted, because they are within the area between the gamma distribution and the x-axis as shown in the figure. Algorithm 2 provides the details in implementing the rejection method.

2.3 The Rejection Method with LBF

In line 7 of Algorithm 2, we can see that $f_X(x)$, the pdf of the random sequence that we desire to generate, is evaluated N times when the rejection method is implemented.

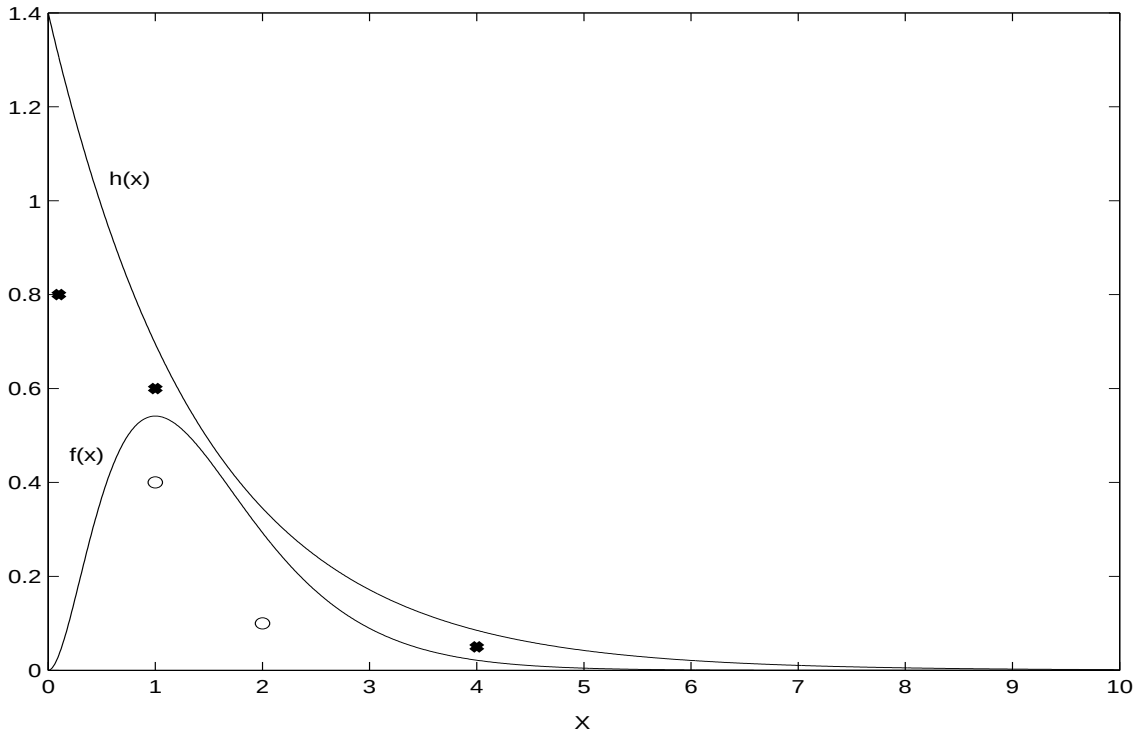


Figure 2.1: *An illustration of the rejection method*

Algorithm 2 The Rejection Method [3]

Require: The pdf of the random sequence desired, $f_X(x)$, the UBF $h(x)$, and the inverse CDF of $h(x)$, $H^{-1}(U)$.

Ensure: The random sequence $X = [X1, X2, \ldots, X_i]$ with the given pdf, $f_X(x)$

```
i = 0 (to count number of accepted random numbers)
r = 0 (to count number of rejected random numbers)
while i ≠ N do
    U ~ U(0,1) (U is equal to a random number in a random sequence that is uniformly
    distributed in [0,1])
    X_{i+1} = H^{-1}(U)
    V ~ U(0,1) (Y is equal to a random number in a random sequence that is uniformly
    distributed in [0,1])
    Y = V h(X_i)
    if Y ≤ f(X_i) then
        return X_i, i = i + 1
    else
        reject X_i, r = r + 1
    end if
end while
```

Although computing $f_X(x)$ is easy to do with a computer, the computation time can become burdensome if the efficiency is low. In order to decrease the computation time when we implement the rejection method, we will introduce the Lower Bounding Function (LBF).

The LBF $s(x)$ is equal to or less than the target pdf $f_X(x)$, in contrast to $h(x)$, which is equal to or greater than the target pdf $f_X(x)$. Figure 2.2 shows $f_X(x)$ with a uniform UBF and triangular under-bound. Besides being equal to or less than $f_X(x)$, the LBF also needs to be much easier to evaluate than $f_X(x)$. While we are implementing the rejection process, with the LBF, we can evaluate $s(x)$ and perhaps not be required to evaluate $f_X(x)$. Since $s(x)$ is equal to or less than $f_X(x)$, if a random variate falls under $s(x)$, it must also fall under $f_X(x)$, and we can accept that random variate. However, if the random variate falls above $s(x)$, we must then evaluate $f_X(x)$ to decide whether or not the random variate fall outside of $f_X(x)$. If $Y < s(x_i)$, we can accept X_i immediately without evaluating $f_X(x)$. However, if $Y > s(x_i)$, we will have to evaluate $f_X(x)$ to decide whether to reject or to accept x_i. If we make $s(x)$ as close to $f_X(x)$ as possible, in many incidences at which acceptance occur, we will not have to evaluate $f_X(x)$, and thus, we can reduce computation time. Algorithm 3 illustrates how we can incorporate the LBF into the rejection method.

2.4 The Composition-Rejection Method

For the rejection method, we must find a UBF that bounds $f_X(x)$ tightly to achieve high efficiency. However, finding a single UBF that bounds $f_X(x)$ tightly can be very difficult. The Composition-Rejection Method is a modified version of the rejection method, and it allows us to use several UBFs for different subintervals of $f_X(x)$. The UBF of the method assigns different UBFs $h_i(x)$ for different subintervals in the domain of $f_X(x)$. Figure 2.3 shows $f_X(x)$ and its UBF which is composed of three UBFs on three subintervals of $f_X(x)$. Recalling the

Algorithm 3 The Rejection Method with LBF [3]

Require: The pdf of the random sequence desired, $f_X(x)$, the UBF $h(x)$, the LBF $s(x)$, and the inverse CDF of $h(x)$, $H^{-1}(x)$.

Ensure: The random sequence $X = [X_1, X_2, \ldots, X_i]$ with the given pdf, $f_x(X)$

```
i = 0 (to count number of accepted random numbers)
r = 0 (to count number of rejected random numbers)
while i ≠ N do
    U ~ U(0,1) (U is equal to a random number in a random sequence that is uniformly
    distributed in [0,1])
    X_{i+1} = H^{-1}(U)
    V ~ U(0,1) (Y is equal to a random number in a random sequence that is uniformly
    distributed in [0,1])
    Y = V h(X_i)
    if Y ≤ s(X_i) then
        return X_i, i = i + 1
    else if Y ≤ f(X_i) then
        return X_i, i = i + 1
    else
        reject X_i, r = r + 1
    end if
end while
```

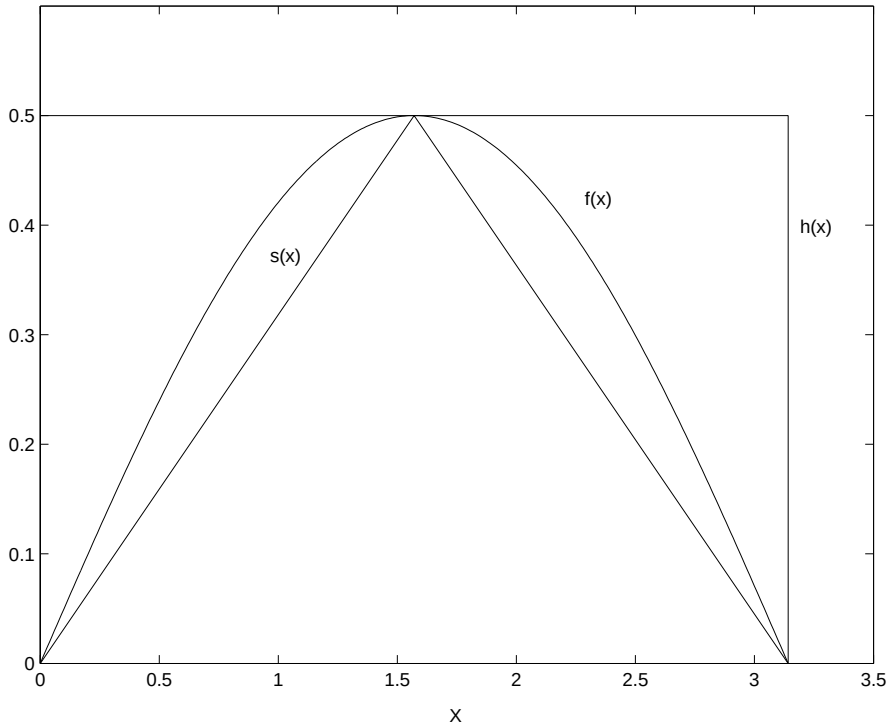


Figure 2.2: *Quasi-density f(x) with a uniform UBF, $h(x)$, and a triangular LBF, $s(x)$*

rejection method, one important step was to generate a random number with the inverse CDF of the UBF, $H^{-1}(U)$. In the Composition-Rejection Method, we have a different $H_i^{-1}(U)$ for the N different subintervals of the domain. To ensure that we are using the appropriate $H_i^{-1}(U)$ for a given sample, a probability p_i is assigned to the areas A_i under $h_i(x)$ corresponding to the UBFs on different subintervals. The probability vector p_i is obtained by normalizing A_i with the total area A under the UBF, A_i/A. With these probabilities, we can generate N discrete numbers in the range of $[0, N-1]$ and use the discrete numbers to represent each A_i accordingly. With the random discrete variates, we can go back to the rejection method and use the discrete numbers to select which $H_i^{-1}(U)$ we need to use at different points of the rejection process. Algorithm 4 details the composition-rejection method. Moreover, when the LBF principle is also incorporated with the composition-rejection method, the method is called Rejection-Immediate-Acceptance method.

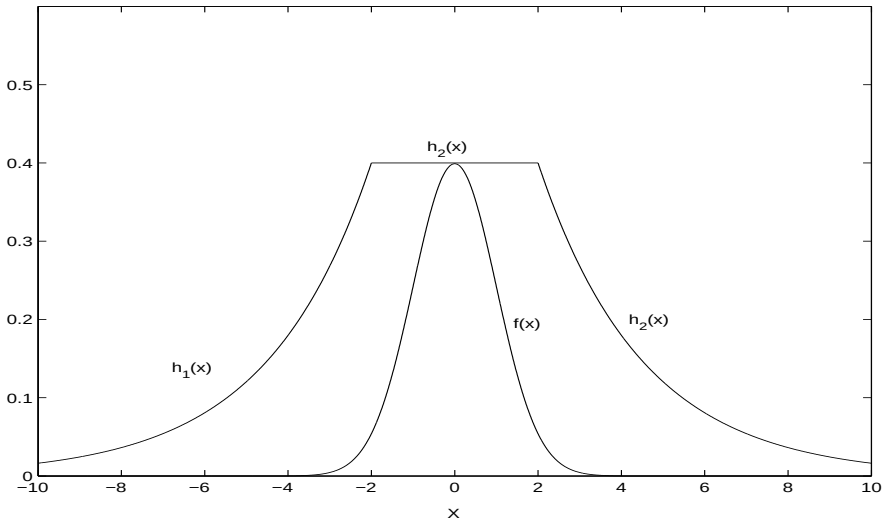


Figure 2.3: *Quasi-density $f_X(x)$ with UBF composed of $k_i h_i(x)$ with $i = 1, 2, 3$*

Algorithm 4 The Composition-Rejection Method [3]

Require: The pdf of the random sequence desired, $f_x(X)$, the scalar k times the UBF $h_i(x)$ for all the subintervals, probabilities associate with the subintervals, and the inverse CDF of $h(x)$, $H_i^{-1}(x)$.

Ensure: The random sequence $X = [X1, X2, \ldots, X_i]$ with the given pdf, $f_x(X)$

```
i = 0 (to count number of accepted random numbers)
r = 0 (to count number of rejected random numbers)
while i ≠ N do
    Generate a discrete random number J with the pdf that corresponds to the areas
    under the subintervals.
    U U(0,1) (U is equal to a random number in a random sequence that is uniformly
    distributed in [0,1])
    X_{i+1} = H_J^{-1}(U)
    V ~ U(0,1) (Y is equal to a random number in a random sequence that is uniformly
    distributed in [0,1])
    Y = V h(X_i)
    if Y ≤ f(X_i) then
        return X_i, i = i + 1
    else
        reject X_i, r = r + 1
    end if
end while
```

2.5 The Index Search Method

For the composition-rejection method, we need to have discrete random variates to select which $H_i^{-1}(U)$ to use in generating the random numbers under $h(x)$. For our application, we will use the index search method to generate the discrete random numbers.

The index search method allows us to generate N integers, $[0, N-1]$, for a given probability vector. The method is like a discrete version of the inverse transform method. In the continuous case, we can substitute a uniformly distributed random number into the inverse CDF of the desired distribution to obtain the desired random number. However, in the discrete case, we do not have a mathematical expression for the CDF. Thus, we also will not have a mathematical expression for the inverse CDF. Hence, we must search for the desired random numbers in the inverse CDF of the desired $f_X(x)$.

The method is set up like a dictionary. For example, if we look for a word, let use the word "zoo", we will not start from page one of the dictionary and look at each word of the dictionary until we find our word. Instead, we will go to the "z" section of the dictionary and start looking our word from there.

First, we must set up a table called the guide table that contains the indexes for the search, and the indexes are analogous to the indexes A-Z in an English dictionary. Figure 2.4 shows how the method provides the starting index for a search.

Figure 2.4 shows a discrete CDF for the events $\{0, 1, 2, 3\}$ with a guide table for $C = 5$. The lower bounds of each box in the guide table are $u_i = i/C, i = 0, \ldots, C-1$. The elements of the guide table, g_j, are obtained by finding the least k that satisfies $F(k) \geq u_j = j/C$. After we have created the guide table, we can start generating discrete random variates. For example, in Figure 2.4, if we generate a uniformly distributed random variate that takes on the value 0.3, we will multiply 0.3 and C, and we will round down the result and obtain 1.

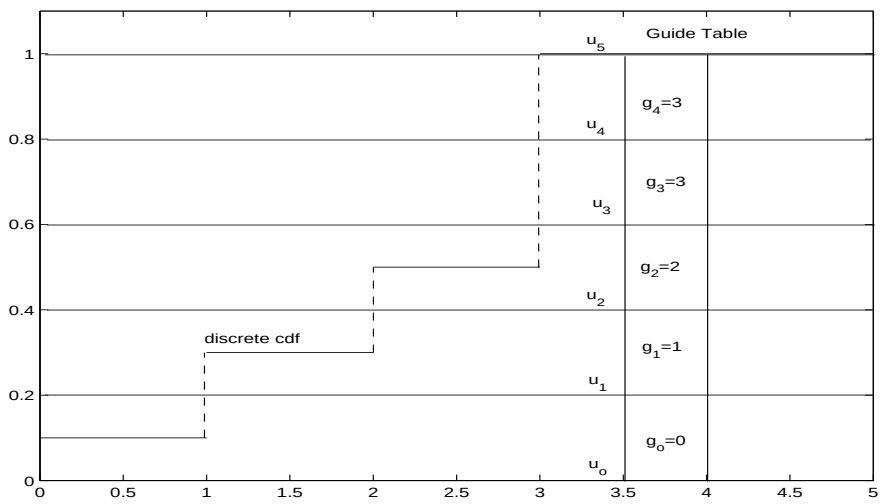


Figure 2.4: *A discrete CDF with a guide table for* $C = 5$

Thus, we will start our search at g_1=1. The detail Algorithm of the method is in Algorithm 5.

With the index search method, we can generate a set of discrete random numbers to identify each subinterval under the UBF and use different subinterval according to the distribution of the discrete random numbers.

Algorithm 5 The Index Search Method [3]

Require: The probability vector $(p_o, p_1, \ldots, p_{L-1})$ of the random sequence desired and the size C of the guide table

Ensure: The random sequence $X = [X_1, X_2, \ldots, X_N]$ with the given probability vector

```
Compute the cumulative probability P_i ← Σ_{j=0}^{i} p_j
{*Setup: Computes the guide table elements g_i for i = 0, 1, ..., C − 1.*}
Set first element of the guide table g_o ← 0 and set i ← 0.
for all j = 1 to C − 1 do
    while j/C > P_i do
        Set i ← i + 1
    end while
    Set g_j ← i
end for
{*Generator*}
Count = 1
while Count ≤ N do
    U ∼ U(0, 1) (U is equal to a random number in a random sequence that is uniformly
    distributed in [0, 1])
    Set X_Count ← g_floor(UC) (floor(UC) means round down U times C)
    while U > P_x do
        Set X_Count ← X_Count + 1
    end while
    Return X_Count
    Count = Count + 1
end while
```

Chapter 3

Transformed Density Rejection (TDR)

The Transformed Density Rejection method is a modified version of the composition-rejection method with an LBF. TDR enables us to build a very tight UBF and LBF, so that we can achieve an efficiency approaching 100%. In using TDR, we must first transform the target pdf into another domain, so that the transformed density is a concave function. With the transformed pdf $\widetilde{f}_X(x)$, we can construct the UBF with tangents on some predefined construction points. For the LBF, it is just the line segments that connect the construction points together. The LBF is built to increase the execution speed of the TDR method. After we have obtained the UBF and the LBF in the transformed domain, we can then inverse transform them to the original domain for the rejection process. In this chapter, we will go over in detail how the TDR works. The materials are based on the work of Hörmann[3].

3.1 Transformation

In order to use the TDR method to generate random sequences with the pdf $f_\nu(x)$ given in (1.1), we must transform $f_\nu(x)$ into another domain, so that $f_\nu(x)$ is a concave function in the transformed domain. After this transformation, the function is called a T-concave function. If we were to build a UBF with tangents for a function that is not concave, the UBF will not be a bounding function. In our case, we will use the natural logarithm to transform $f_\nu(x)$ into a T-concave function, $\widetilde{f}(x) = \ln(f(x))$. In order for a transformation to be useful in constructing the UBF and the LBF, the transformation must satisfy the three following conditions [2]

1. $T : R^+ \rightarrow R$ is differentiable and $T^{'}(x) > 0$
2. $lim_{x \rightarrow 0} T(x) = -\infty$
3. The inverse transform $T^{-1}(x)$ has an integrable left tail, i.e., $\int_{-\infty}^{x} T^{-1}(t)dt$ is bounded for all fixed x in the image of T

However, if the domain of the pdf is finite, the transformation will only need to satisfy Condition 1. For our case, all the conditions must be fulfilled, since $f_\nu(x)$ for the generalized exponential pdf has a domain of $[-\infty, \infty]$.

Many transformations fulfill the conditions above, but not all of them are optimized, not efficient and easy enough to implement. In order for us to select the optimal transformations within the class of transformations that fulfill the above conditions, we must impose four more criteria on the transformations in additional to the conditions above. Below are the criteria [2]:

1. T, F_T and their inverse functions must be easy to compute.

	$c > 0$	$c = 0$	$c < 0, c \neq -1$	$c = -1/2$
$T(x)$	$R^+ \to R^+ : x^c$	$R^+ \to R : \ln(x)$	$R^+ \to R^- : -x^c$	$-x^{-1/2}$
$T^{-1}(x)$	$R^+ \to R^+ : x^{1/c}$	$R \to R^+ : e^x$	$R^- \to R^+ : (-x)^{1/c}$	x^{-2}
$F_T(x)$	$R^+ \to R^+ : \frac{x^{\frac{c+1}{c}}}{\frac{c+1}{c}}$	$R \to R^+ : e^x$	$R^- \to R^+ : \frac{-(-x)^{\frac{c+1}{c}}}{\frac{c+1}{c}}$	$-1/x$
$F_T^{-1}(x)$	$R^+ \to R^+ : (x\frac{c+1}{c})^{\frac{c}{c+1}}$	$R^+ \to R : \ln(x)$	$R^+ \to R^- : -(-x\frac{c+1}{c})^{\frac{c}{c+1}}$	$-1/x$

Table 3.1: *The T_c class of tranformations*

2. The transformation should lead to a scale invariant UBF.

3. The transformation should be able to force a large number of pdf into a T-concave function.

4. The transformation should lead to a good fitting UBF that yields high efficiency.

After the additional criteria are imposed, we arrived at a class of transformation called the class of T_c transformations. Only the T_c transformations fulfill all the conditions and criteria mentioned above. Table 3.1 from [2] summarizes the T_c class. Also, it is important to note that for a pdf with an unbounded domain, the T_c transformations are only applicable for $c \in (-1, 0]$.

As c gets larger, the UBF would get tighter for a given number of construction points due to the nature of the transformation. However, we must choose a value for c that allows the UBF to intersect only with the pdf at the construction points, since the UBF must always be equal to or greater than the target pdf. For our pdf, $f_\nu(x)$ in (1.1), we choose $c = 0$ arbitrary. Then, we need to test the transformation T_0 to see is it suitable for our application. To complete the test, we need to show that the inequality

$$f''(x) - \frac{f'(x)^2}{f(x)} \leq 0 \tag{3.1}$$

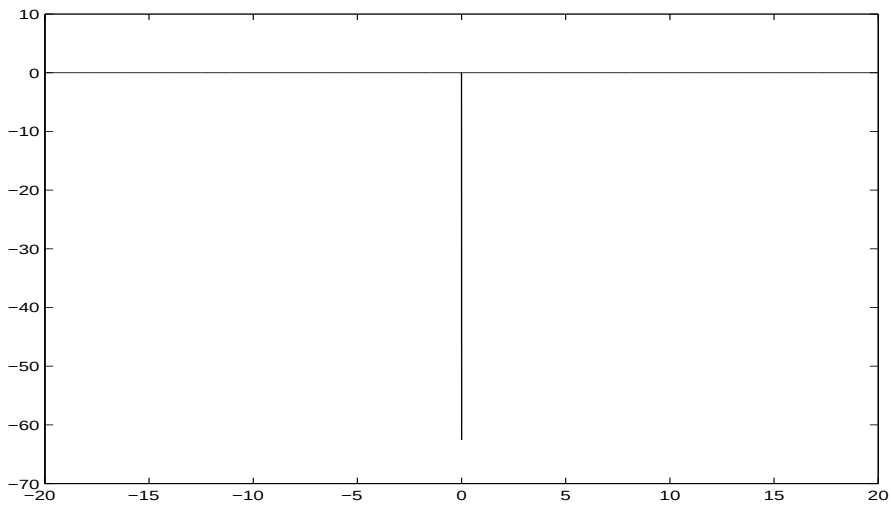


Figure 3.1: *Left side of (3.1) for* $\nu = 1$

is satisfied. If the inequality is satisfied, we can be certain that T_0 is a suitable transformation for $f_\nu(x)$.

For any pdfs that satisfy the inequality , we can use T_0 as the transformations to construct the corresponding UBFs. After a pdf is transformed by the T_0 transformation, the transformed pdf is called a log-concave pdf. Figure 3.1 and 3.2 show the plot of the left side of (3.1) for $\nu = 1$ and $\nu = 4$. It turns out that the pdf given in (1.1) is a log-concave function for $\nu \geq 1$. Two limiting cases for ν are checked, and they are shown in Figures 3.1 and 3.2. By examining both figures, it is obvious that they are both equal to or less than zero. Thus, the inequality is satisfied for $\nu \geq 1$. However, for $\nu < 1$, the inequality would not be satisfied.

3.2 Construction Points

After examining the properties of the transformations, we arrived at the T_0 transformation for the generalized exponential pdf. Since T_0 can be used for our pdf, the pdf is a log-concave function. With the log-concave function, we can choose N suitable design points

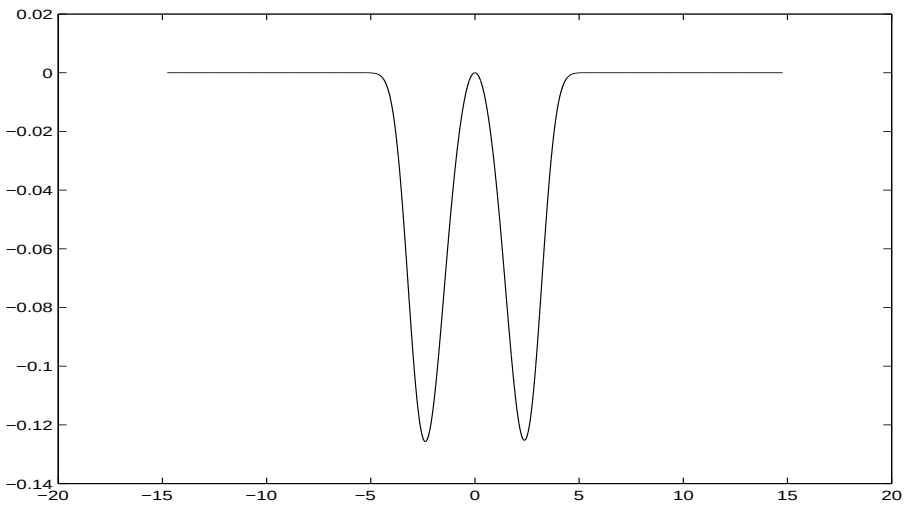


Figure 3.2: *Left side of (3.1) for* $\nu = 4$

$b_l < p_1 < p_2 < \ldots < p_N < b_r$ to construct the tangents of the UBF. The variables b_l and b_r are the lower and upper endpoints for the domain of the transformed density $\widetilde{f}(x)$, and p_i are the construction points where the tangents are constructed. Since the domain of the generalized exponential pdf is infinite, we must truncate the pdf to a finite domain of about $[-10\sigma, 10\sigma]$ Also, we will make b_i for $i = 1, 2, \ldots, N-1$ as the intersection points among the tangents. In our implementation, we use an odd number of equiangular points given by (3.2) to select our construction points, because equiangular points are easy to generate and only relatively few of them are needed to construct an efficient UBF. By using an odd number of construction points, we are able to increase the efficiency for the rejection method and to prevent the tangents from crossing the x-axis in the transformed domain. We want to prevent the tangents from crossing the x-axis because, depending on the transformation, we may not be able to perform an inverse transform correctly if any of the tangents cross the x-axis.

$$p_i = m + \zeta \tan(-\pi/2 + i\pi/(N+1)),\ i = 1, \ldots, N \tag{3.2}$$

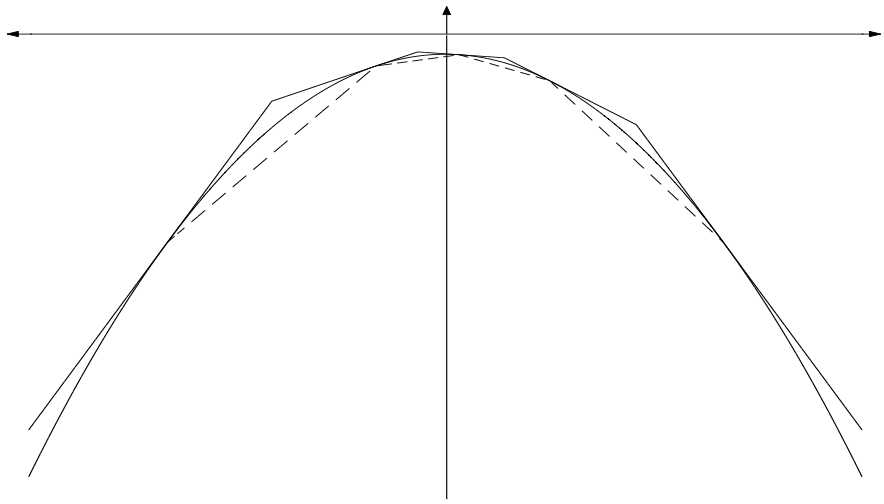


Figure 3.3: *The transformed pdf, the UBF, and the LBF (with dash line)*

Equation (3.2) shows the method for obtaining the equiangular points for construction. In (3.2), p_i are the equiangular points, m is peak location of the pdf, and ζ is chosen to be the standard deviation, σ of the pdf. The equiangular points will cluster densely around the peak location of the pdf, and the points will become less dense as they deviate farther away from the peak location of the pdf. Figure 3.3 and Figure 3.4 illustrate the UBF and LBF constructions of the TDR.

3.3 UBF in the Transformed Domain

The UBF is the function that we use to generate random numbers prior to the rejection process when we are implementing the rejection method. The UBF is a quasi-density, which is proportional to a pdf and can be used to generate random sequences with the distribution proportional to itself easily with the inverse transform method.

First, we calculate an adequate number of equiangular points and use them as our construction points (design points) to build tangents for the transformed density. The tangents are

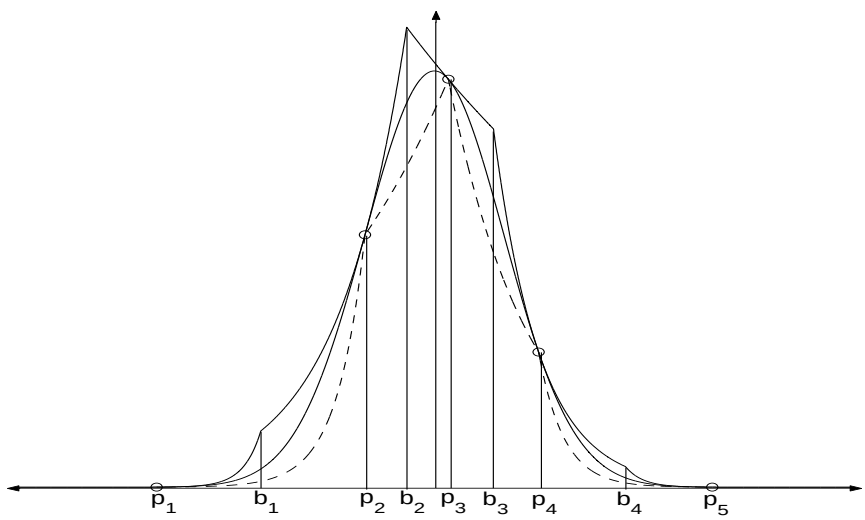


Figure 3.4: *The pdf, the UBF, the LBF (with dash line), obtained from inverse transforming the functions from previous Figure 3.3*

generated with the point-slope formula $\widetilde{f}(p_i) + \widetilde{f}'(p_i)(x - p_i)$ on all the design points. Then, we compute the intersection points, b_i among the tangents

$$\widetilde{h}_i(x) = \widetilde{f}(p_i) + \widetilde{f}'(x - p_i), for\ b_{i-1} < x \leq b_i \tag{3.3}$$

and set b_o as the left side limit of the pdf's domain and b_N as right side limit of the pdf's domain. After we have obtained the construction points, tangents, and intersections, we are ready to construct the transformed UBF, $\widetilde{h}(x)$, which is a summation of all the tangents and is given by

$$\widetilde{h}(x) = \sum_{i=1}^{N} \widetilde{h}_i(x) \tag{3.4}$$

3.4 LBF in the Transformed Domain

The LBF is used to speed up the rejection method during simulation. The LBF is a function that can be computed much quicker than the target pdf, and the LBF is less than or equal to the target pdf. After generating a random variate from the UBF, instead of evaluating the target pdf to see if the random number falls under the pdf, we evaluate the LBF first. If the random variate falls under the LBF, we will accept the random variate immediately; since the LBF is always less than or equal to the pdf, and if the number fall under the LBF, it must fall under the pdf also. However, if the number fall above the LBF, we must evaluate the target pdf to see if the number fall under the pdf. Given that the LBF approximates the pdf, we are able to save a significant amount of computation time, since we are evaluating a much simpler function, the LBF, for our rejection process.

To construct the LBF of the transformed density, we simply connect all the construction points, $(p_i, \widetilde{f}(p_i))$, with secants. The LBF at each subinterval will become

$$\widetilde{s}_i(x) = \widetilde{f}(p_i) + \frac{\widetilde{f}(p_{i+1}) - \widetilde{f}(p_i)}{p_{i+1} - p_i}(x - p_i), \;\; for p_i \leq x < p_{i+1}, \tag{3.5}$$

and the LBF for the whole domain is just the summation of all the secants given by

$$\widetilde{s}(x) = \sum_{i=1}^{N-1} \widetilde{x}_i(x) \tag{3.6}$$

3.5 The Inverse Transformation

After we have obtained the UBF and LBF for the transformed pdf, we can inverse transform them. Since we used ln(.) for our transformation, we can use exp(.) for our inverse

transformation, (see Table 3.1). The inverse transformations are given by

$$h(x) = T^{-1}(\widetilde{h}(x)) = \exp(\widetilde{h}(x)) \tag{3.7}$$

and

$$s(x) = T^{-1}(\widetilde{s}(x)) = \exp(\widetilde{s}(x)) \tag{3.8}$$

3.6 Composition-Rejection Method in TDR

Since each tangent of the UBF has a different CDF, we need to be able to decide which $H_i^{-1}(U)$ to use in generating random variates for the rejection process. First, we need to find the areas A_i under each of these tangents in the subintervals (b_{i-1}, b_i) for $i = 1, \ldots, N$. Then, we can sum them to find the total area under the UBF, $A \leftarrow \sum_{i=1}^{N} A_i$. After the calculations of the areas, we can use the values of the areas for the composition-rejection method by normalizing each area of the subintervals with the total area under the UBF to find the probabilities associate with each of the subintervals. With these probabilities, we can choose the appropriate $H_i^{-1}(U)$ to generate random variates for the rejection process at different time instances in the simulation. The inverse CDF for all the subintervals is

$$H_i^{-1}(u) = p_i + \frac{1}{\widetilde{f}'(p_i)} \left(F_T^{-1} \left(\widetilde{f}'(p_i)u + F_T(\widetilde{f}(p_i) + \widetilde{f}'(p_i)(b_{i-1} - p_i)) \right) - \widetilde{f}(p_i) \right), \; for\; 0 \leq u \leq A_i, \tag{3.9}$$

and we will use that inverse CDF to generate random variates under the UBF for TDR. For reference, the CDF is given

$$H_i(u) = \frac{1}{\widetilde{f}'(p_i)} \Big(F_T(\widetilde{f}(p_i) + \widetilde{f}'(p_i)(x - p_i)) - F_T(\widetilde{f}'(p_i) + \widetilde{f}'(p_i)(b_{i-1} - p_i) \Big) \tag{3.10}$$

The inverse CDF in (3.9) works for all subintervals. However, when $\widetilde{f}'(p_i) = 0$, i.e., when the derivative of the transformed pdf is equal to zero at p_i,

$$H_i^{-1}(u) = b_{i-1} + \frac{u}{f(p_i)} \tag{3.11}$$

Algorithm 6 gives the detailed steps for using the TDR.

3.7 Potential Improvements for TDR

The construction points that we used for TDR are equiangular points given by (3.2). They cluster around the mode and get less dense as they deviate from the mode. For our MATLAB implementation in Appendix A, we used seven equiangular points for our construction. Seven points were selected by trial and error; a different number of points were used in the simulation, and the efficiency for each set of construction points were examined. Seven points cause the efficiency of TDR to stay about 95% for $\nu \geq 1$. Thus, seven points are recommended. However, seven points is redundant for some modes of our application. For example, for $\nu = 1$, the efficiency is approximately 100%. The example shows that to achieve better computational speed, we do not want to use a fixed number of equiangular points for our application. Instead, we use a different number of points for different modes of our application to ensure a fixed efficiency in the random variates generation. The methods are referred to as the Adaptive Rejection Sampling (ARS) and Derandomized Adaptive Rejection Sampling (DARS) can help us find an appropriate number of construction points and

Algorithm 6 Transformed Density Rejection (TDR) [3]

Require: The pdf of the random sequence desired with bounds (b_l, b_r), transformation $T(x)$, constructions points $p_1 < \ldots < p_N$.

Ensure: The random sequence $X = [X_1, X_2, \ldots, X_i]$ with the given pdf, $f_x(X)$

{*Setup: construct UBF $h(x)$ and LBF $s(x)$*}

Compute $\widetilde{f}(p_i) = T(f(p_i))$ and $\widetilde{f}'(p_i),\ for\ i = 1, \ldots, N$.

Compute $\widetilde{s}'(p_i) = (\widetilde{f}(p_i))/(p_{i+1} - p_i), i = 1, \ldots, N$.

Compute the intersection points b_i of the tangents of $\widetilde{f}(x)$ in the points p_i and p_{i+1}, for $i = 1, \ldots, N-1$. Set $b_o \leftarrow b_l$ and $b_N \leftarrow b_r$.

Compute areas A_i below the UBF $h(x)$ for each subinterval (b_{i-1}, b_i), for $i = 1, \ldots, N$.

Set $A \leftarrow \sum_{i=1}^{N} A_i$

{*Generator*}

loop

  Generate $V \sim U(0, A)$

  Generate a discrete random variate J with probability vector proportional to $(A_1, \ldots, A_N)$.

  Compute $X \leftarrow H_J^{-1}(V)$ using (3.9) with $H_J^{-1}(V)$ corresponding to subintervals chosen by J.

  Compute $h(X) \leftarrow T^{-1}(\widetilde{f}(p_J) + \widetilde{f}'(p_J)(X - p_J))$

  if $J > 1\ and\ X < p_J$ **then**

    Compute $s(X) \leftarrow T^{-1}(\widetilde{f}(p_J) + \widetilde{s}'(p_J)(X - p_J))$

  else if $J < N\ and\ X > p_J$ **then**

    Compute $s(x) \leftarrow T^{-1}(\widetilde{f}(p_J) + \widetilde{s}'(p_J)(X - p_J))$

  else

    Set $s(X) \leftarrow 0$

  end if

  Generate $U \sim U(0, 1)$.

  if $Uh(X) \leq s(X)$ **then**

    return X

  end if

  if $Uh(X) \leq f(X)$ **then**

    return X

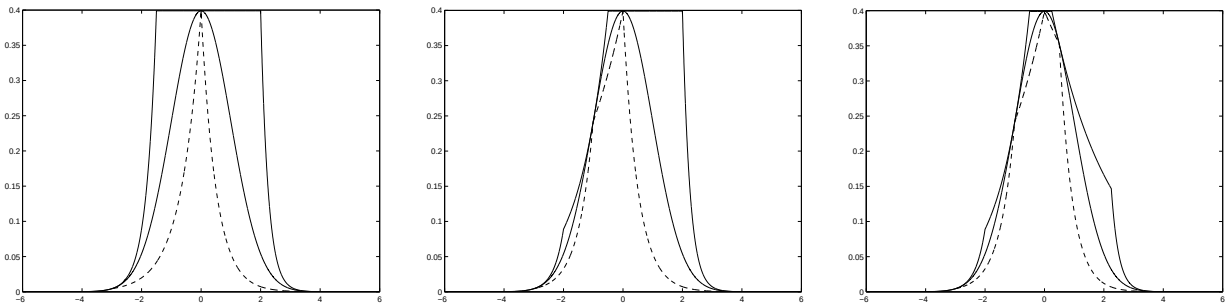
  end if

end loop

the appropriate placement of the points to ensure a desired and consistent efficiency.

3.7.1 Adaptive Rejection Sampling (ARS)

The concept Adaptive Rejection Sampling (ARS) was introduced by Gilks and Wild [1]. When using ARS, we first need to pick three arbitrary points to construct a temporary UBF. We can then use the temporary UBF for the rejection process. When a random variate generated by the temporary UBF is rejected, the x-coordinate of the variate will be used as a new construction point in addition to the three construction points. With the new construction point, we can build a new UBF for the rejection process. We continue to update the UBF until the ratio between the area under the pdf and the area under the UBF is acceptable. Figure 3.5 shows the process of ARS in selecting construction points to build the UBF for the rejection process.



(a) (b) (c)

Figure 3.5: *(a)The initial phase of ARS (with three construction points), (b)The process of ARS (an additional point is added), (c)The process of ARS (two additional points is added to the initial setup)*

3.7.2 Derandomized Adaptive Rejection Sampling (DARS)

The ARS method works very well in selecting construction points and for building a suitable UBF for the rejection method. However, when using ARS, we can never be certain of how many construction points the method will yield since the production is stochastic in nature. Also, in TDR, many steps require knowledge of the UBF, such as the composition part where we need the area under the UBF. Thus, when we are using ARS, we need to recalculate many parameters in TDR every time the UBF is reconstructed. Moreover, ARS obscures the separation between the setup part and the generation part of TDR, and it makes the implementation of TDR more complex.

To counter the disadvantages of ARS, we can use the expected value of the quasi-density $h(x) - s(x)$ to choose construction points, and the method is called Derandomized Adaptive Rejection Sampling (DARS). The idea results in the method below [4]:

1. Start with three arbitrary points like ARS
2. The intervals of the tangents (p_{i-1}, p_i) are split by new construction points if the area under $h_i(x) - s_i(x)$ for any of them is greater than a fraction κ of the average area of $h(x) - s(x)$,

$$A_{i,h} - A_{i,s} \geq \kappa(A_{h-s})/(N+1)$$

   κ is chosen to be less than one, so that the majority of the intervals will at least split once.
3. Check the ratio between the area under the pdf and the area under the UBF. If the ratio is not acceptable (small efficiency), repeat step 2.

	storage: small	storage: large
speed:very fast	Eq_2 and ARS	Eq_2 and ARS
speed: fast	Opt_3 and ARS	Eq_5 and ARS
speed: slow	Opt_n	Eq_{10} and DARS or Opt_n

Table 3.2: *Performance for different options in TDR*

By using DARS, we can optimize the UBF without making the selection of the construction points stochastic. Also, the setup and generation part of TDR will be separated, and many parameters will not have to be regenerated when the UBF is being updated. Table 3.2 is from [3] and shows the performance of TDR when different options for construction points selection are used. Eq_n refers to using n equiangular points for construction; Opt_n is some other types of construction points that we have not mentioned, for more information on Opt_n, refer to [3].

Chapter 4

The Ratio-of-Uniforms Method (RoU)

In the previous chapter, we examined the Transformed Density Rejection (TDR). It has a very high efficiency, and the efficiency is adjustable, but it requires a tedious and expensive (in terms of computation time) setup. The code for implementing the method is shown in Appendix A. The simulation time could become burdensome in some applications. For simulations that require one time generation of random variates, TDR works fine, since we only have to setup TDR once. However, for applications that require sets of random variates having different means and modes to be generated numerous times in a loop, TDR will not work well, because setting up TDR is very expensive, and every time it is called for different mean or mode, it needs to setup itself again for generation. The setup times will accumulate and reduce the speed of the application. Thus, we will look at an alternative, the Ratio-of-Uniform (RoU) Method.

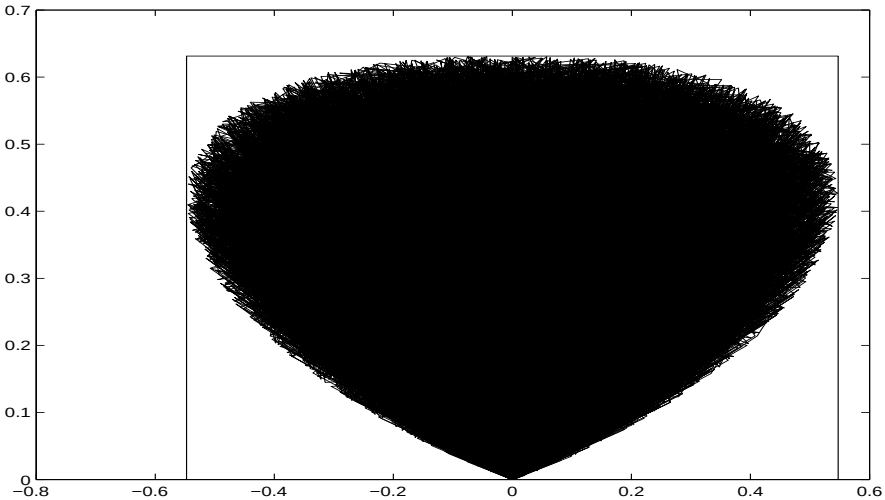


Figure 4.1: *The UBF and the area A_f (shaded) of the transformed pdf*

4.1 Implementation of RoU

Similar to TDR, the RoU method is also a modified version of the rejection method. The RoU uses a uniform UBF for the rejection process after transforming the pdf, $f_X(x)$ from (X, Y) into (U, V), see Figure 4.1. The transformation enables the uniform UBF and the x-axis to be a reasonable tight envelop for $f_X(x)$ in the transformed domain. Thus, the transformation increases efficiency. Figure 4.2 shows the corresponding UBF and pdf in the original (X, Y) domain.

Compared to TDR, the RoU method is not as numerically efficient. The average efficiency of the ROU method is approximately 60%, and the efficiency is not adjustable. However, RoU requires virtually no setup as shown in Appendix B. The coding of the method is also simple, and the program is short. In addition, it does not require the generation of discrete random variates (a very computation time consuming part in TDR) nor a complex UBF, which can introduce round-off error.

The concept of RoU is based on Theorem 1 from [5]. The theorem states that if we do

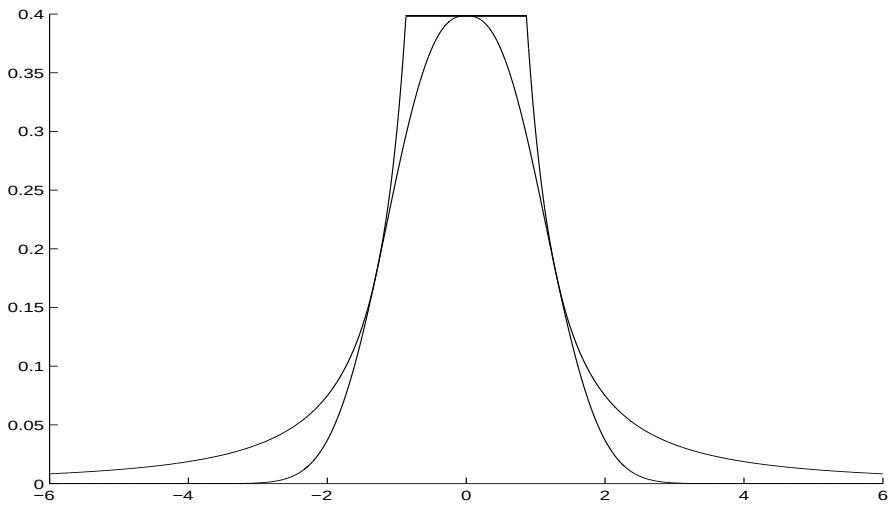


Figure 4.2: *The UBF $h(x)$ and the pdf $f_X(x)$ of the original (x, y) domain*

the transformation $(X, Y) \to (U, V)$ with $(X, Y) = (U/V + \mu, V^2)$ for $f_X(x)$, $f_X(x)$ will be mapped to (U, V) one-to-one as a closed region with an area, A_f as shown in Figure 4.1 (an upside-down teardrop shape), defined by (4.1). Furthermore, for random variates (U, V) that lie inside the closed region, after transforming them back to (X, Y) by $X = U/V + \mu$, X will take on the pdf, $f_X(x)$. The parameter μ is usually set to the mode (location of maximum) of the pdf.

Theorem 1. *Let X be a random variable with quasi-density function $f_X(x)$ with domain (b_l, b_r) (finite or infinite in duration). Let μ be a constant. If (U, V) is uniformly distributed in*

$$A_f = (u, v) : 0 \leq \sqrt{f(u/v + \mu)},\ b_l < u/v + \mu < b_r, \tag{4.1}$$

then $X = U/V + \mu$ takes on the generalized exponential probability density function $f_X(x)$ defined by (1.1) [3].

By transforming $f_X(x)$ to the (U, V) domain, we are able to use the uniform distribution as the UBF, $h(U)$ for the rejection process (see Figure 4.1). The uniform UBF is constructed with the domain and range given by

$$u^- \leq u \leq u^+, \quad 0 < v < v^+ \tag{4.2}$$

for

$$u^+ = max((x-\mu)\sqrt{f_x X}) \tag{4.3}$$

$$u^- = min((x-\mu)\sqrt{f_x(X)}) \tag{4.4}$$

$$v^+ = max(\sqrt{f_x(X)}) \tag{4.5}$$

Similar to the TDR, the RoU method can incorporate the LBF concept for increased computation speed. For the RoU, the LBF can be easily constructed by connecting the points $(0,0)$, $(u^-, v(u^-))$, $(0, v^+)$, $(u^+, v(u^+))$, and $(0,0)$ in the (U,V) domain within the teardrop region. The detail of the RoU's implementation is given in Algorithm 7.

Algorithm 7 RoU (Ratio-of-Uniform) [3]

Require: Quasi-density $f_X(x)$ with domain (b_l, b_r), constant μ, (minimal) bounding rectangle $(u^-, u^+) \times (0, v^+)$

Ensure: The random sequence $X = [X_1, X_2, \ldots, X_i]$ with the given pdf, $f_X(x)$

```
for i = 0 to N do
    Generate U ~ U(u⁻, u⁺) and V ~ U(0, v⁺).
    X ← U/V + μ
    if V² ≤ f_X(x) then
        Return X
    end if
end for
```

4.2 Potential Improvements on RoU

Using RoU, we transform $f_\nu(x)$ in (1.1) from the (X,Y) domain to the (U,V) domain. During the transformation the area under $f_\nu(x)$ is mapped one-to-one onto the (U,V) domain as a teardrop shape (see Figure 4.1). The new shape allows a uniform UBF and sustain a

efficiency of about 60%. The efficiency that the uniform UBF achieves is usually acceptable, but we can do much better by using a polygonal envelope as the UBF of the teardrop, and the idea was introduced by Loydold [8].

The polygonal envelope concept is the same as TDR. The envelope is constructed using the same method that the TDR uses, by choosing construction points on the teardrop and building tangents on these points. The LBF can be constructed by connecting the construction points together. The bounding expression

$$p_v = \sqrt{f_X(x)},\ p_u = xp_v, \tag{4.6}$$

refers to (p_u, p_v), which are the outer boundary of the teardrop and the possible coordinates for the construction points. The tangent expression

$$a_u u + a_v = a_c = a_u p_u + a_v p_v \tag{4.7}$$

$$a_v = 2p_v + f_X^{'}(x)x/p_v \ and\ a_u = -f_X^{'}(x)/p_v, \tag{4.8}$$

shows the tangents that are used for the envelop construction. The equations are taken from [3]. It is important to take note that with the new polygonal envelope, we will have to divide the envelop into triangles and use composition-rejection to generate random variate with the envelop to prepare for the rejection process.

In the last section, we examined the concept of polygonal envelops that can increase the efficiency of RoU. The envelop concept is equivalent to TDR. There are many similarities between RoU and TDR indeed. As a matter of fact, with some modifications, we will be

able to use equiangular points, ARS, DARS, etc to select construction points for RoU.

Chapter 5

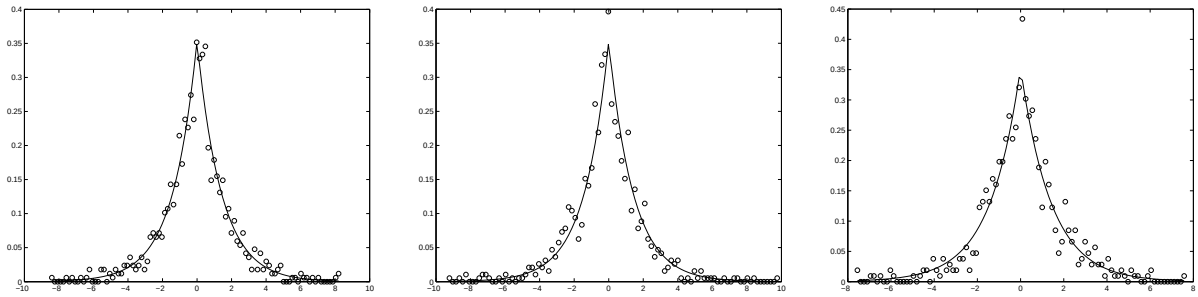
Simulation Results and Application

5.1 Simulation Results

After implementing Algorithm 6 for TDR and Algorithm 7 for RoU in MATLAB, it is appropriate to examine the results generated from the algorithms. In this section, we will compare the results generated from TDR, RoU, and other popular methods for some common distributions. Then, we will examine the results generated for different modes for the generalized exponential distribution given by (1.1).

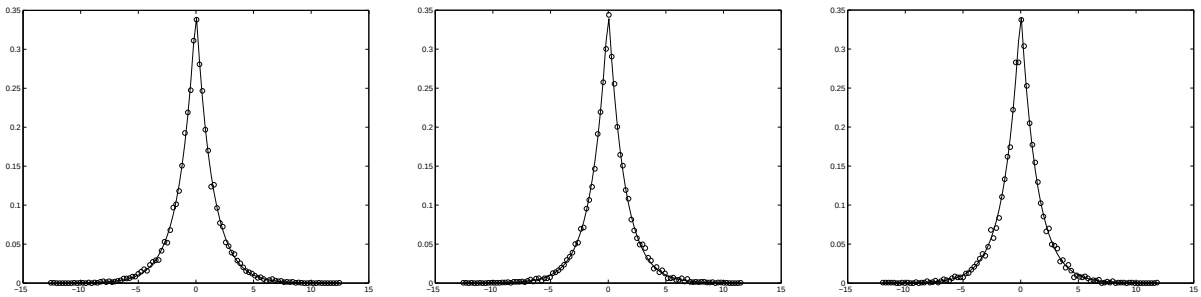
First, we generate two sets $n = 1000$ and $n = 10000$ random variates with the Laplacian pdf using the inverse transformation, TDR, and RoU. The results are shown in Figures 5.1 and 5.2. For the implementation of TDR, the efficiency is approximately 100% with 11 construction points, and the quality of the random variates compare to the quality of the random variates generated by the inverse transform method, see Figure 5.2a and 5.2b. For RoU, the efficiency is about 68%, and the results are also comparable to the results of the inverse transform method. The random numbers from TDR and RoU also converge to the

Laplacian distribution as n approaches infinity as shown in Figures 5.1 and 5.2.



(a) (b) (c)

Figure 5.1: *Laplacian random variates generated (a) by the inverse transformation, (b) by TDR with $\eta = 1$, chisq=26.08, (c) by RoU with $\eta = 0.69$, chisq=26.88 with n=1000*

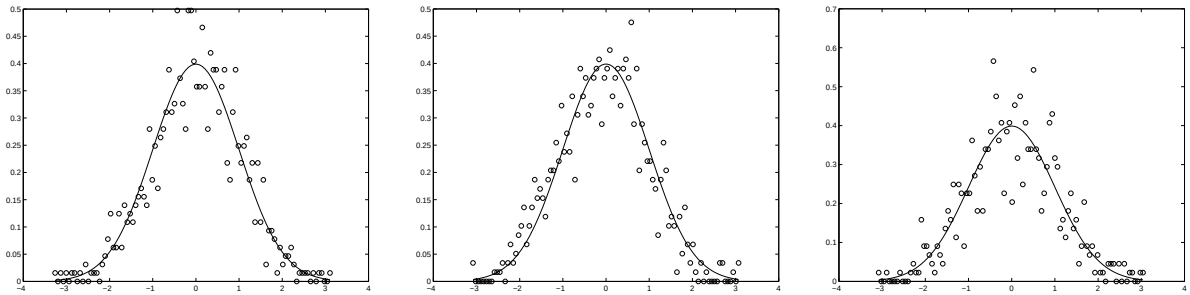


(a) (b) (c)

Figure 5.2: *Laplacian random variates generated (a) by the inverse transformation, (b) by TDR with $\eta = 1$, (c) by RoU with $\eta = 0.68$ with n=10000*

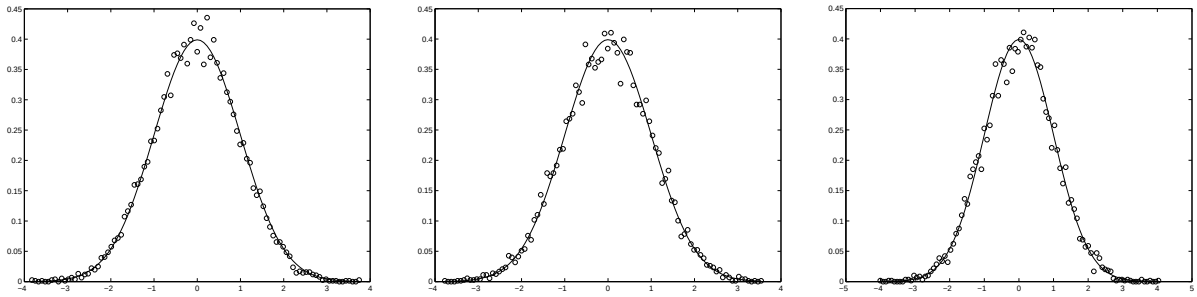
After we have examined the Laplacian case, we will examine the Gaussian case. For the Gaussian case, we generate two set $n = 1000$ and $n = 10000$ random variates with the Gaussian pdf using the Box-Muller method, TDR, and RoU. Figures 5.3 and 5.3 show the results. The result of the Box-Muller method is illustrated using only one of its two Gaussian outputs. For the implementation of TDR, the efficiency is approximately 100% for 11 construction points, and the quality of the random variates compare to the quality of the random variates generated by the inverse transform method, see Figure 5.2a and 5.2b. For RoU, the efficiency is about 73%, and the results are also comparable to the results

of the Box-Muller method. From these results, it is obvious that the random variates are converging to the desired pdfs. The random numbers from TDR and RoU also converge to the Gaussian distribution as n approaches infinity (see Figures 5.3 and 5.4).



(a) (b) (c)

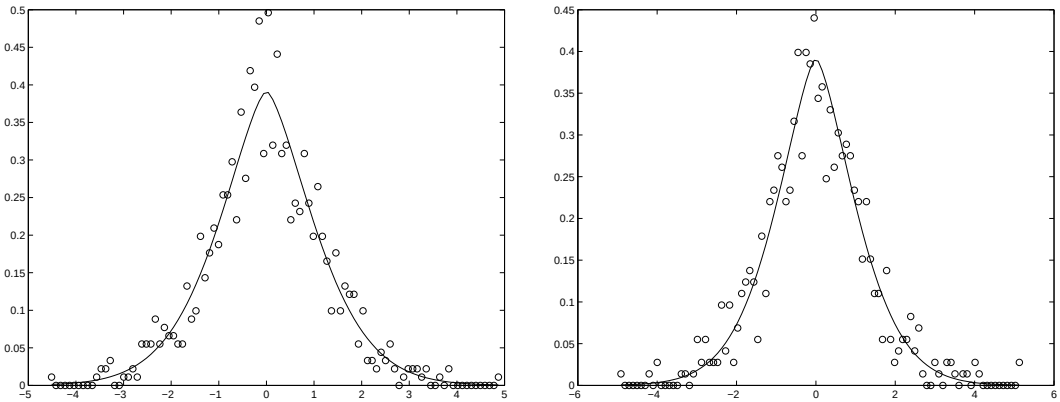
Figure 5.3: *Gaussian random variates generated (a) by box-muller, (b) by TDR with* $\eta = 0.95$*, (c) by RoU with* $\eta = 0.71$ *for* $n = 1000$



(a) (b) (c)

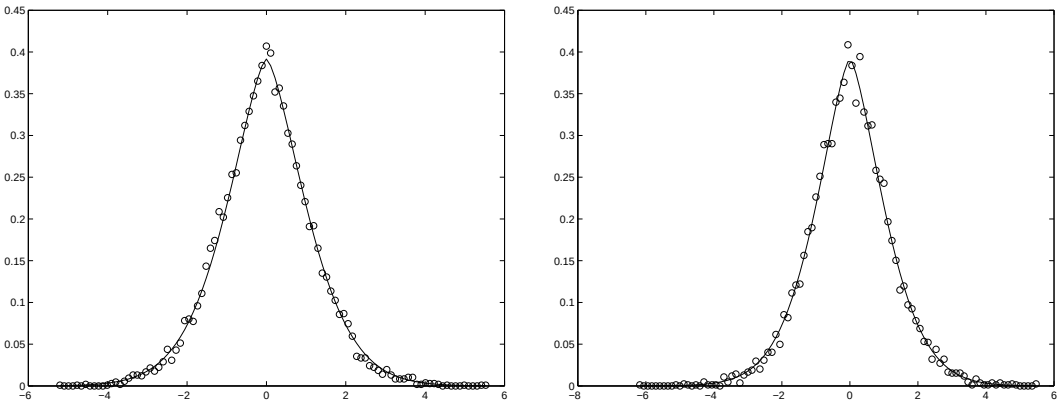
Figure 5.4: *Gaussian random variates generated (a) by box-muller, (b) by TDR with* $\eta = 1$*, (c) by RoU with* $\eta = 0.73$ *for* $n = 10000$

After comparing the random variates generated by TDR and RoU with the random variates generated by popular methods for the Laplacian and Gaussian distributions, we will generate random variates with other modes in the pdf given in (1.1) that cannot be generated with either the inverse transformation or the Box-Muller method. In Figure 5.6 and 5.8, the random variates with $\nu = 1.5$ and 1.6 respectively with TDR and RoU for $n = 10000$. From the figures, it is clear that the random variates are converging to the desired pdfs.



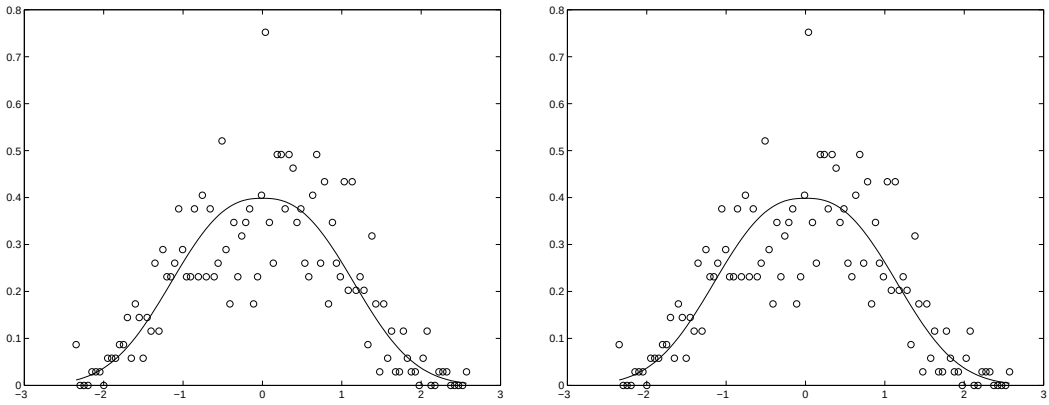
(a) (b)

Figure 5.5: *Random variates generated for* $\nu = 1.5$ *(a) by TDR with* $\eta = 0.96$*, (b) by RoU for* $\eta = 0.74$ *for* $n = 1000$



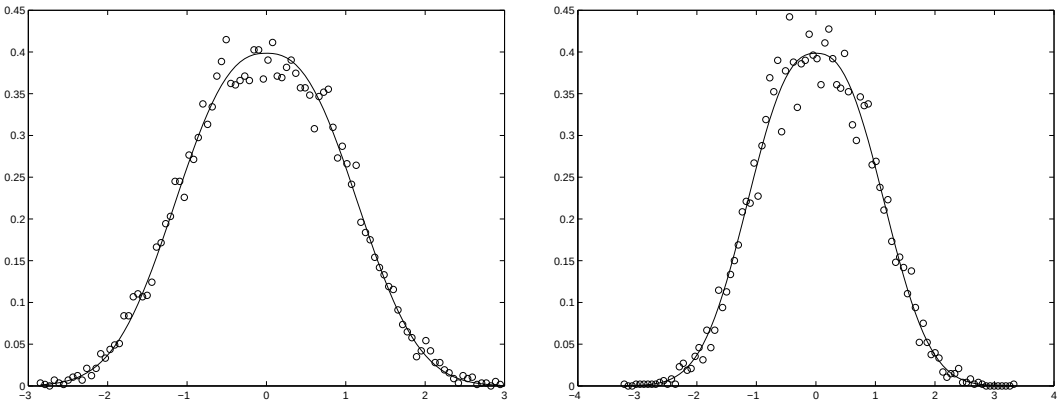
(a) (b)

Figure 5.6: *Random variates generated for* $\nu = 1.5$ *(a) by TDR with* $\eta = 0.99$*, (b) by RoU for* $\eta = 0.73$ *for* $n = 10000$



(a) (b)

Figure 5.7: *Random variates generated for* $\nu = 2.5$ *(a) by TDR with* $\eta = 0.98$*, (b) by RoU for* $\eta = 0.69$ *for* $n = 1000$



(a) (b)

Figure 5.8: *Random variates generated for* $\nu = 2.5$ *(a) by TDR with* $\eta = 0.98$*, (b) by RoU for* $\eta = 0.73$ *for* $n = 10000$

After examining how the random variates generated from TDR and RoU coverage to the desired pdfs, we will approximate the mean, variance, skewness, and kurtosis for random variables with given pdfs. The approximations are accomplished by using the random variates generated by TDR and RoU, and we will compare the estimations with the exact results. The theoretical even moments for the Gaussian distribution is given in (5.1), and the odd moments are equal to zero.

$$E(x^k) = 1 \cdot 3 \cdots (k-1)\sigma_x^n, \ k \ even \tag{5.1}$$

The moment generating function is [7]

$$\Phi_x(\omega) = \frac{\alpha^2}{\omega^2 + \alpha^2} \tag{5.2}$$

Before we approximate moments, we will explain briefly what they mean. The mean is located at the point that divide the area under the pdf into two equal portions. To estimate the mean from a random variates sequence, we can use (5.5). Given that the mean is equal to zero, the variance would equal to the second moment of a random variable; it measures how wide or narrow does a pdf spread.

$$Var(x) = E(x^2) - E(x)^2 \tag{5.3}$$

$$\sigma = \sqrt{Var(x)} \tag{5.4}$$

The approximation for the mean (first moment) is the sample mean,

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{5.5}$$

and the approximation for the variance is the sample variance

$$Var(x_1, \ldots, x_N) = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - E(x))^2 \tag{5.6}$$

Skewness is the third moment of a random variable, and it measures the symmetry of a pdf. For example, if a pdf is symmetric relative to its mean, its skewness would be approximately equal to zero. For a pdf with a larger area on its right side relative to its mean, its skewness would be positive, and if the pdf has a larger area on its left, its skewness would be negative. The kurtosis measures how sharp or flat the peak of a pdf is. For the Gaussian distribution, the kurtosis is equal to three, which makes it a leptokurtic distribution. The kurtosis for other pdfs can be examined relative to the class of leptokurtic distributions. For any distributions having a kurtosis greater than three, it is called a platykurtic distribution, and its peak will look sharper than a leptokurtic distribution. For distributions with a kurtosis less than three, it is called a mesokurtic distribution, and it will look more flat than a leptokurtic distribution [9]. To approximate the skewness and kurtosis of a random variable, the expressions

$$Skew(x_1, \ldots, x_N) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{x_i - E(x)}{\sigma}\right)^3 \tag{5.7}$$

and

$$Kurt(x_1, \ldots, x_N) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{x_i - E(x)}{\sigma}\right)^4 \tag{5.8}$$

can be used [9]. In Table 5.1, 5.2, 5.3, and 5.4, they show the results of the approximations

	$E(x)$	$E(x^2)$	$E(x^3)$	$E(x^4)$
MATLAB(randn)	0.0018	1.0002	-2.1891e-004	3.0022
TDR	-0.0028	0.9964	-0.0087	3.0032
RoU	-0.0023	0.9946	0.0056	3.0377
In Theory	0	1	0	3

Table 5.1: *Approximate moments for Gaussian samples using different methods with* $n = 5000$

	$E(x)$	$E(x^2)$	$E(x^3)$	$E(x^4)$
MATLAB(randn)	0.0018	1.0002	-2.1891e-004	3.0022
TDR	-0.0014	1.0023	-0.0030	3.0249
RoU	3.3912e-004	1.0027	0.0088	3.0090
In Theory	0	1	0	3

Table 5.2: *Approximate moments for Gaussian samples using different methods with* $n = 10000$

for the means, variances, skewness, and kurtsis approximated with different random variates sequences and different methods of random variates' generations. The results in the tables show that the approximations closely approximate the exact results as n increases.

To further investigate the quality of the random variables generated by the TDR and the RoU methods, we will use the Kolmogorov-Smirnov (K-S) Test to examine how well the random variates fit the target pdf. With the K-S test, we can declare that a sequence random variates do not fit the target pdf well, and the sample is to be rejected if

$$(\sqrt{n} + 0.12 + \frac{0.11}{\sqrt{n}})D_n > c_{1-\alpha} \tag{5.9}$$

is true [6]. $c_{1-\alpha} = 1.138$ and D_n is defined as

	$E(x)$	$E(x^2)$	$E(x^3)$	$E(x^4)$
Inv. Trans.	-6.5251e-004	3.9930	0.0022	5.9289
TDR	0.0249	4.0286	0.0427	5.7115
RoU	0.0013	3.9993	-0.0240	5.9547
In Theory	0	4	0	96

Table 5.3: *Approximate moments for Laplacian samples using different methods with* $n = 5000$

	$E(x)$	$E(x^2)$	$E(x^3)$	$E(x^4)$
Inv. Trans.	-6.5251e-004	3.9930	0.0022	5.9289
TDR	0.0019	3.9897	-0.0037	5.9088
RoU	0.0051	4.0048	0.0042	6.0309
In Theory	0	4	0	96

Table 5.4: *Approximate moments for Laplacian samples using different methods with* $n = 10000$

	$\nu = 1$	$\nu = 2$	$\nu = 1.5$	$\nu = 2.5$
TDR	0.7188	0.7475	0.4538	0.8364
RoU	0.7234	0.7297	0.9008	1.0140

Table 5.5: *Kolmogorov-Smirnov Test for random variates generation (values in the tables are the left side of (5.9))*

$$D_n = max(D_n^+, D_n^-) \tag{5.10}$$

for

$$D_n^+ = max_{1 \leq i \leq n}(\frac{i}{n} - \hat{F}(X_i)), \tag{5.11}$$

$$D_n^- = max_{1 \leq i \leq n}(\hat{F}(X_i) - \frac{i-1}{n}) \tag{5.12}$$

With $n = 1000$, we perform the K-S test on the generated random variates for modes $\nu = 1$, $\nu = 2$, $\nu = 1.5$, and $\nu = 2.5$. We will generate some samples with TDR and RoU and perform the K-S test to see how well do the samples fit the target distribution. The results are shown in Table 5.5

Table 5.5 shows that all the test values are less than $c_{1-\alpha} = 1.138$, so the samples generated by TDR and RoU pass the K-S test, and the samples are accepted.

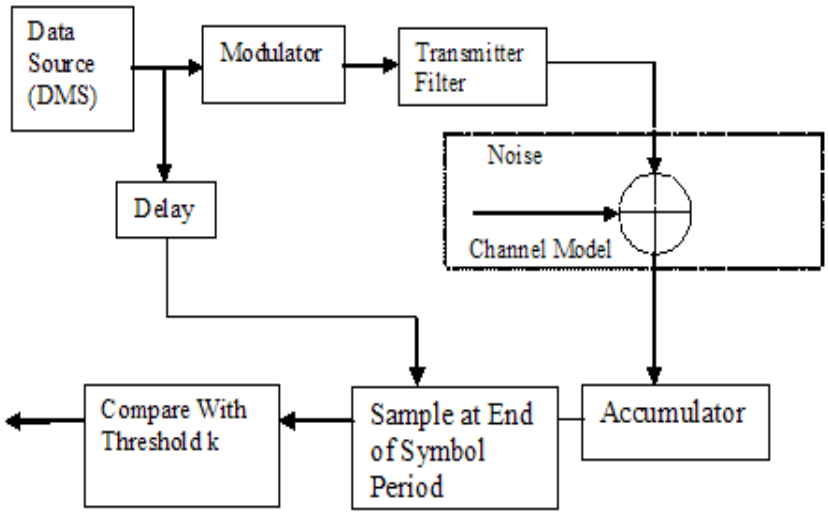


Figure 5.9: *Basic BPSK Communication System [10]*

5.2 Application: BPSK System

After we examine the quality of the random variates generated by TDR and RoU, we will use TDR as the channel noise for a BPSK communication system to illustrate further that generalized exponentially distributed random variates have good qualities and can be used to test the robustness of a communication system. As an example, we will generate random numbers with the generalized exponential pdf using TDR. We choose TDR over RoU for our application, because TDR is faster and more efficient for our application. The example is taken out from [10] with some modifications. The block diagram system for our example is shown in Figure 5.9.

In the example, we will consider a BPSK system with both signal points lying only in the in-phase channel of the signal constellation. We will examine several cases of the BPSK system.

The first case is to use the randn command in MATLAB to generate the channel noise with a transmitter filter, which is a third-order Butterworth filter with a bandwidth equal to the bit rate, r_b, so that the bandwidth causes intersymbol interference (ISI) to occur. For the second case, we will also used the randn command in MATLAB to generate the channel noise for the BPSK system, but we will do the simulation without the transmitter filter; the simulation from the second case should approach the theoretical BER curve. Then, we will simulate the two cases again with the TDR method for the Gaussian noise's generation.

After the simulations, we compare the results. The comparison among the simulations shows that for the first case with the MATLAB randn generator, BER increases and deviates from the theoretical BER curve as expected since the transmitter filter introduces ISI, see Figure 5.10a. As for the second case with the MATLAB randn generator, the simulation results lie closely to the theoretical BER curve as expected, see Figure 5.10b. Using TDR, we simulate the first and second cases again; the results are shown in Figure 5.10c and 5.10d. The comparison shows that the simulations with TDR exhibit similar results compare to the simulations generated by the MATLAB command randn.

Since TDR gives good performance for the Gaussian case, it is safe to assume that it will work for other cases. To be certain, we must use TDR to generate noise with ν not equal to two and do a sanity check to make sure that TDR is generating noise with the correct distribution. First, we will assume that the BPSK system is a robust system, which means that when ν deviates slightly from two, the BER simulations for different ν should only deviate from the simulations in Figure 5.10 slightly. For our sanity check, we will only look at the case with no transmitter filter. The results for the case without transmitter filter with generalized exponentially distributed noise generated by TDR are given in Figure 5.11. The figures show that BER deviate only slightly as ν changes. The Figure 5.11 shows that as ν decreases, BER gets worse; when ν increases, BER gets better. Regardless of what the

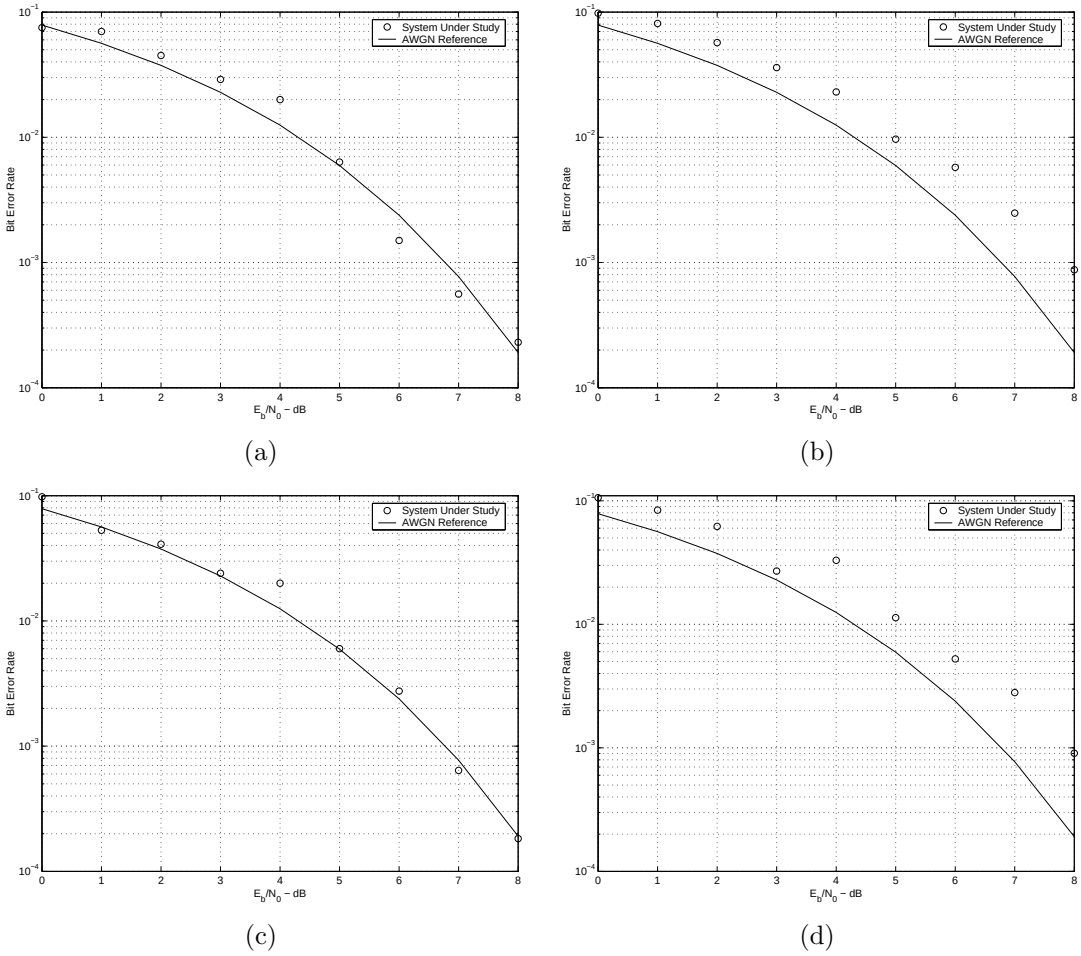


Figure 5.10: *BPSK's simulation with Gaussian channel noise (a) generated by MATLAB randn for a system with transmitter filter (b) generated by Matlab randn for a system without transmitter filter (c) generated by TDR for a system with transmitter filter (d) generated by TDR for a system without transmitter filter*

changes are, the BER for the BPSK system only deviates slightly when ν is changed slightly, and the results show that if the BPSK is a robust system, the random variates generated by TDR must model the generalized exponential distribution. Thus, the random variates generated by TDR are able to model the noise of a communication system and to test its robustness.

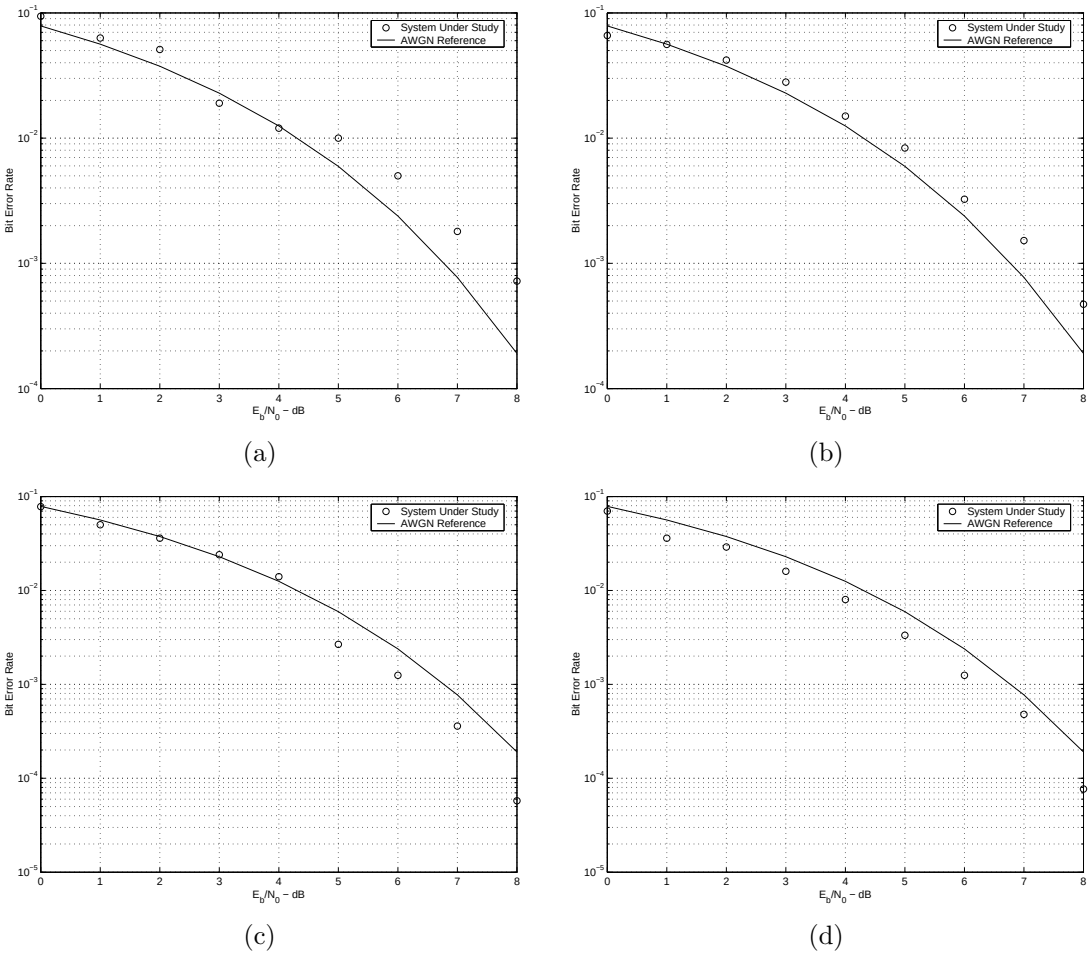


Figure 5.11: *Ideal BPSK's simulation with generalized exponential channel noise (a) for nu* $= 1.8$ *(b) for nu* $= 1.9$ *(c) for nu* $= 2.1$ *(d) for nu* $= 2.2$

Chapter 6

Conclusion

In this thesis, we examined two methods for non-uniform random variates generation. The methods are Transformed Density Rejection (TDR) and Ratio-of-Uniform (RoU). With the two methods, we generated random variates with the generalized exponential distribution. For TDR, we learned that it transforms the generalized exponential distribution into a concave function. With the transformed pdf, we then can build a UBF with tangents for the rejection process. The UBF will allows the efficiency of the rejection process to be high and adjustable. However, the UBF setup will cause the implementation to be long and complex.

For RoU, we learned that it transforms the generalized exponential distribution into a teardrop shape that is finite in domain and range. With the teardrop shape, we can use a very simple UBF, a uniform UBF, for the rejection process. The implementation is short and simple, but the efficiency of the method is low.

In order to understand TDR and RoU more clearly, we studied the components that TDR and RoU need. The components are inverse transform, rejection, composition-rejection, and index-search for continuous and discrete random variates generation. With the background

information, we were able to learn TDR and RoU more effectively.

After we learned the concepts of TDR and RoU, we examined their qualities. In order to examine their qualities, we did several tests. First, we generated some Laplacian and Gaussian random variates with TDR, RoU, and some other popular methods. We examined how close the random variates lie to the desired pdf. It turned out that the random variates generated by TDR and RoU have the same qualities as the other popular methods since the random variates generated were all lying very close to the desired pdfs. Second, we estimated the first four moments from random variates generated with TDR and RoU and compare them with the theoretical moments of the distributions. It turned out that the moments estimations were very close the the exact values for the moments. Third, we simulated a BPSK system with Gaussian noise generated by the MATLAB command randn, and we simulated the same system again with Gaussian noise generated by TDR. The simulation results show that the noise generated by the two methods are equivalent. Finally, we assumed that the BPSK system was a robust system and varied the mode of the noise. If the assumption was true, the BER of the simulations for different modes that deviated from the Gaussian mode slightly should only deviate from the theoretical BER for the Gaussian noise by a small margin. The simulations of the BPSK system with different modes of channel noise showed that the BER only deviate slightly from the theoretical BER for the Gaussian noise, which showed that the non-uniform random variates generated by TDR with the generalized exponential distribution is able to better model the channel noise and to test the robustness of a communication system.

Bibliography

[1] W.R. Gilks and P. Wild, *Adaptive Rejection Sampling for Gibbs Sampling*, Applied Statistics **41(2)** (1992).

[2] Wolfgang Hörmann, *A Rejection Technique for Sampling from T-concave Distributions*, ACM Transactions on Mathematical Software **21(2)** (1995).

[3] Wolfgang Hörmann and G. Derflinger, *Statistics and Computing Automatic Nonuniform Random Variate Generation*, Springer–Verlag, New York, 2004.

[4] E. Janka J. Leydold and W. Hörmann, *Variants of Transformed Density Rejection and Correlation Induction. In K.-T. Fang, F. J. Hickernell, and H. Niederreiter (Eds.), Monte Carlo and Quasi-Monte Carlo Method 2000*, Springer–Verlag, Heidelberg, 2002.

[5] A. J. Kinderman and J. F. Monahan, *Computer Generation of Random Variables Using the Ratio of Uniform Deviates*, ACM Transactions on Mathematical Software **3(3)** (1977).

[6] Averill M. Law and W. David Kelton, *Simulation Modeling and Analysis*, McGraw-Hill Companies, Inc., New York, 2000.

[7] Alberto Leon-Garcia, *Probability and Random Processes for Electrical Engineering, Second Ed.*, Addison-Wesley Publishing Company, Inc., Massachusetts, 1994.

[8] J. Leydold, *Automatic Sampling with the Ratio-of-Uniforms Method*, ACM Trans. Math. Software **26(1)** (2000).

[9] William H. Press, *Numerical Recipes in C the Art of Scientific Computing*, Cambridge University Press, New York, 2002.

[10] William H. Tranter, K. Sam Shanmugan, Theodore S. Rappaport, and K. L. Kosbar, *Principles of Communication System Simulation with Wireless Application*, Pearson Education, Inc, New Jersey, 2004.

Appendix A

Appendix

A.1 TDR in MATLAB

```
%Description: This program generate n generalized exponentially distributed
%             random variates with the Transformed Density Rejection (TDR)
%             method.  For the method, 7 construction points are used for
%             UBF construction.  Also, the method is only valid for nu
%             greater or equal to 1.
%Input:       n=number of random variates desired, m=mean of the pdf,
%             sig=standard deviation of the pdf, nu=mode of the pdf
%Output:      xx=n random variates with the generalized exponential
%             distribution


%function [xx]=TDR(n,m,sig,nu)

NumConstructPt=7;            %Default number of construction points
nb=ceil(n*1.15);             %Increase n to compensate for rejection

lend=-sig*6;                 %Left bound for the x coordinate of pdf
rend=sig*6;                  %Right bound for the x coordinate of pdf
step=10000;                  %Resolution of x
stepsize=(rend-lend)/step;

x=lend+m:stepsize:rend+m;    %the x coordinates for the pdf

a=nu/(sqrt(8)*sig*gamma(1/nu));             %y coordinates for the pdf
f=(a*exp(-abs((x-m)/(sqrt(2)*sig)).^nu));
```

```
%Derivative of pdf
fdiff=a.*(-nu.*(sign(x-m).*(x-m)./(sqrt(2)*sig)).^(nu-1).* ...
(sign(x-m)./(sqrt(2)*sig)).*exp(-(sign(x-m).*(x-m)./(sqrt(2).*sig)).^nu));
%Transformed pdf
logf=log(f);
%Derivative of transformed pdf
logfdiff=1./f.*fdiff;

%making construction points
xs=[1:NumConstructPt];
ConstructPt=m+sig*tan(-pi/2+xs.*pi./(NumConstructPt+1));

%Find the y coordinates for the derivative of the transformed pdf
%at the construction points
fc=(a*exp(-abs((ConstructPt-m)/(sqrt(2)*sig)).^nu));
TransformPdfCp=log(fc);
fdiffc=a.*(-nu.*(sign(ConstructPt-m).*(ConstructPt-m)./(sqrt(2)*sig)).^ ...
(nu-1).*(sign(ConstructPt-m)./(sqrt(2)*sig)).* ...
exp(-(sign(ConstructPt-m).*(ConstructPt-m)./(sqrt(2).*sig)).^nu));
TransformDiffPdfCp=1./fc.*fdiffc;

%Setting the derivate of the mode equal to zero
if floor(NumConstructPt/2)~=NumConstructPt/2
    TransformDiffPdfCp(ceil(NumConstructPt/2))=0;
end

%Calculate the slopes of the squeeze at all subintervals
SqueezeDiff=[];
for i=1:(NumConstructPt-1)
    SqueezeDiff(i)=(TransformPdfCp(i+1)-TransformPdfCp(i))./ ...
    (ConstructPt(i+1)-ConstructPt(i));
end

%Find the intersection of the tangents
[Intersections]=TangentsIntersections(NumConstructPt,ConstructPt, ...
TransformPdfCp,TransformDiffPdfCp,SqueezeDiff);
%Find the xy coordinates of the UBF and squeeze
[XHat,Hat,XSqueeze,Squeeze]=HatAndSqueeze(NumConstructPt, ...
ConstructPt,Intersections,TransformPdfCp,TransformDiffPdfCp,SqueezeDiff,x);
%Inverse transform the UBF and LBF back to the original domain
Hat=exp(Hat);
Squeeze=exp(Squeeze);

%Find the areas under each tangents and the total area under the UBF
```

```
[Areas,TotalAreas]=FindAreas(Intersections,XHat,TransformDiffPdfCp, ...
Hat,NumConstructPt);

%generator
%Generate discrete random variates for composition-rejection
c=floor(NumConstructPt/3);
p=Areas./TotalAreas;
[z]=IndexSearch(c,p,nb);

%Generate a vector xrvc of random variates with the distribution of the UBF
%hcdfin=[];
binsends=[-inf Intersections];
uai=[];
for i=1:nb
    %Choose a tangent UBF according to the distribution of the discrete
    %random variate and use that UBF to generate a random variate
    uai=rand(1)*Areas(z(i)+1);
    ag=TransformDiffPdfCp(z(i)+1)*uai+exp(TransformPdfCp(z(i)+1)+ ...
    TransformDiffPdfCp(z(i)+1)*(binsends(z(i)+1)-ConstructPt(z(i)+1)));
    if TransformDiffPdfCp(z(i)+1)==0
        xrvc(i)=binsends(z(i)+1)+uai./fc(z(i)+1);
    else
        xrvc(i)=ConstructPt(z(i)+1)+1./TransformDiffPdfCp(z(i)+1)* ...
        (log(ag)-TransformPdfCp(z(i)+1));
    end
end

he=[];
acce=0;
reje=0;

xx=[];

he=exp(TransformPdfCp(z+1)+(TransformDiffPdfCp(z+1).* ...
(xrvc-ConstructPt(z+1))));

for i=1:nb

    %if (z(i)+1)>1 & xrvc(i)<ConstructPt(z(i)+1)
    %se=exp(TransformPdfCp(z(i))+SqueezeDiff(z(i)).* ...
    %(xrvc(i)-ConstructPt(z(i))));
    %elseif (z(i)+1)<NumConstructPt & xrvc(i)> ConstructPt(z(i)+1)
    %   se=exp(TransformPdfCp(z(i)+1)+SqueezeDiff(z(i)+1).* ...
    %(xrvc(i)-ConstructPt(z(i)+1)));
    %else
```

```
    %    se=0;
    %end

    usc=rand(1);
    if usc*he(i)<=se
        xx(acce+1)=xrvc(i);
        acce=acce+1;

    elseif usc.*he(i)<=(a*exp(-abs((xrvc(i)-m)/(sqrt(2)*sig)).^nu))
        xx(acce+1)=xrvc(i);
        acce=acce+1;
    else
        reje=reje+1;
    end
end

%select only the request number of random variates n
xx=xx(1:n);
%End of function file

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Below are some extra options for generator performance%
%They are commented out                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%The two lines below that are commented out are for
%efficiency display
%eff=acce/nb;
%fprintf(['\nThe efficiency is ' num2str(eff) '%.'])

%The four lines below that are commented out are for
%a random variates fitting curve
%[am,where]=hist(xx,100);
%y=a*exp(-abs((where-m)./(sqrt(2)*sig)).^nu);
%prop=am/n/(where(3)-where(2));
%plot(where,y,where,prop,'o')

%The three lines that are commented out are for
%a plot with the pdf, UBF, and LBF in
%original domain
%plot(x,f),hold on
%plot(XHat,Hat,'-g'),hold on
%plot(XSqueeze,Squeeze,'--m'),hold on
```

A.2 Function files for TDR

A.2.1 Intersections Finding

```
%Description: This program finds the intersections among the tangents for
%             the UBF.
%Input:       NumConstructPt=# of Construction points used, ConstructPt=the
%             x-coordinates of the construction points, TransformPdfCp=the
%             transformed density evaluated at the construction points,
%             TransformDiffPdfCp=the derivative of the transformed density
%             evaluated at the construction points,
%             SqueezeDiff=the derivative of the LBF at different
%             subintervals
%Output:      Intersections=the intersection among the tangents

function [Intersections]=TangentsIntersections(NumConstructPt, ...
ConstructPt,TransformPdfCp,TransformDiffPdfCp,SqueezeDiff)

%Default value of the resolution of the tangents
StepTangents=1000;

%Initialize variables
TangentInterval=[];
Intersections=[];
TangentLeft=0;
TangentRight=0;


for i=1:NumConstructPt-1
%Calculate the intervals for the tangents where they overlap
    TangentOverInterval=ConstructPt(i):(ConstructPt(i+1)- ...
    ConstructPt(i))/StepTangents:ConstructPt(i+1);
%Calculate the y-coordinates of the tangent on the left for the overlap
%domain
    TangentLeft=TransformPdfCp(i)+(TransformDiffPdfCp(i).* ...
    (TangentOverInterval-ConstructPt(i)));
%Calculate the y-coordinates of the tangent on the right for the overlap
%domain
    TangentRight=TransformPdfCp(i+1)+(TransformDiffPdfCp(i+1).* ...
    (TangentOverInterval-ConstructPt(i+1)));
%Subtract the two overlapped tangents and find their intersection
    [tc,LocationOfIntersection]=min(abs(TangentLeft-TangentRight));
    Intersections(i)=TangentOverInterval(LocationOfIntersection);
end
```

```
%End of function file
```

A.2.2 UBF and LBF

```
%Description: This program finds xy coordinates of the UBF and LBF in
%             the transformed domain.
%Input:       NumConstructPt=# of Construction points used, ConstructPt=the
%             x-coordinates of the construction points,
%             Intersections=intersections of the tangents
%             TransformPdfCp=the transformed density evaluated at the
%             construction points, TransformDiffPdfCp=the derivative of
%             the transformed density evaluated at the
%             construction points, SqueezeDiff=the derivative of the LBF
%             at different, subintervals, XOfPdf=x coordinates of the pdf
%Output:      XHat=x coordinates of the UBF, Hat=y coordinates of the UBF
%             XSqueeze=x coordinates of the LBF, Squeeze=y coordinates
%             of the LBF

function [XHat,Hat,XSqueeze,Squeeze]=HatAndSqueeze(NumConstructPt, ...
ConstructPt,Intersections,TransformPdfCp,TransformDiffPdfCp,SqueezeDiff, ...
XOfPdf)

StepTangents=1000;

%Calculate the first xy coordinate of the UBF
XHat=min(XOfPdf):(Intersections(1)-ConstructPt(1))/ ...
StepTangents:Intersections(1);
Hat=TransformPdfCp(1)+(TransformDiffPdfCp(1).* ...
(XHat-ConstructPt(1)));

%Calculate the first xy coordinate of the LBF
XSqueeze=ConstructPt(1):(ConstructPt(2)-ConstructPt(1))/ ...
StepTangents:ConstructPt(2);
Squeeze=TransformPdfCp(1)+SqueezeDiff(1).*(XSqueeze- ...
ConstructPt(1));

%Initialize variables
xsm=[];
htsm=[];
xsmm=[];
stmm=[];
```

```
for i=2:NumConstructPt-1

%Continue to calculate x coordinate of the UBF
    xsm=Intersections(i-1):(Intersections(i)- ...
    Intersections(i-1))/StepTangents:Intersections(i);
    XHat=[XHat xsm];

%Continue to calculate y coordinate of the UBF
    htsm=TransformPdfCp(i)+(TransformDiffPdfCp(i).* ...
    (xsm-ConstructPt(i)));
    Hat=[Hat htsm];

%Continue to calculate x coordinate of the LBF
    xsmm=ConstructPt(i):(ConstructPt(i+1)-ConstructPt(i))/ ...
    StepTangents:ConstructPt(i+1);
    XSqueeze=[XSqueeze xsmm];

%Continue to calculate y coordinate of the LBF
    stmm=TransformPdfCp(i)+SqueezeDiff(i).*(xsmm-ConstructPt(i));
    Squeeze=[Squeeze stmm];
end

%Calculate the last xy coordinate of the UBF
xhte=Intersections(NumConstructPt-1):(ConstructPt(NumConstructPt ...
-Intersections(NumConstructPt-1))/StepTangents:max(XOfPdf);
hte=TransformPdfCp(NumConstructPt+(TransformDiffPdfCp ...
(NumConstructPt.*(xhte-ConstructPt(NumConstructPt)));

%Final xy coordinates of the UBF
XHat=[XHat xhte];
Hat=[Hat hte];

%End of function file
```

A.2.3 Finding areas under the UBF

```
%Description: This program calculates the areas under the UBF
%             for all individual subinterval and the total
%             area under the UBF.
%Input:       c=number of entry of the guide table, p=probability
%             vector, n=number of random variates desired
%Output:      z=n random variates with the probability vector p
```

```
function [Areas, TotalAreas]=FindAreas(Intersections, ...
XHat,TransformDiffPdfCp,Hat,NumConstructPt)

%Finding the area of the left tail of the pdf
Areas=[];
[j]=find(Intersections(1)-XHat==0);
Areas(1)=1/TransformDiffPdfCp(1)*(Hat(j(1)));

%check to see is the area of the left tail is defined or not
%if it is undefined, it is too small, set it equal to zero
if isnan(Areas(1))==1
    Areas(1)=0;
end


for i=2:NumConstructPt-1
%Find the location of the intersection in the XUBF vector
    [j]=find(Intersections(i)-XHat==0);
    [k]=find(Intersections(i-1)-XHat==0);

%Check to see is the area under the horizontal tangent
%if it is, calculate it as a rectangle
%else, calculate with the result of integration
    if TransformDiffPdfCp(i)==0
        Areas(i)=(Intersections(i)-Intersections(i-1))*max(Hat);
    else
        Areas(i)=1/TransformDiffPdfCp(i)*(Hat(j(1))-Hat(k(1)));
    end
%Make sure that the area calculated is defined.
%if it is not defined, it is too small, set to zero
    if isnan(Areas(i))==1
    Areas(i)=0;
end

end

%Finding the area of the right tail of the pdf
[j]=find(Intersections(NumConstructPt-1)-XHat==0);
Areas(NumConstructPt)=1/TransformDiffPdfCp(NumConstructPt)* ...
(-Hat(j(1)));

%check to see is the area of the right tail is defined or not
%if it is undefined, it is too small, set it equal to zero
```

```
if isnan(Areas(NumConstructPt))==1
    Areas(NumConstructPt)=0;
end

%Calculate the total area under the UBF
TotalAreas=sum(Areas);

%End of function file
```

A.2.4 Index Search Method

```
%Description: This program generate n discrete random variates
%             with probability vector p=[p1,p2,...,pN] in the
%             range of [0,N] using the index search method.
%Input:       c=number of entry of the guide table, p=probability
%             vector, n=number of random variates desired
%Output:      z=n random variates with the probability vector p

function [x]=IndexSearch(c,p,n)

%Computing the CDF with the given probability vector p
j=length(p);
cp=[];
temp=0;
for i=1:j
    cp(i)=temp+p(i);
    temp=cp(i);
end

%Building the guide table g
g=[];
g(1)=0;
i=0;
for j=1:c-1
    while j/c>cp(i+1)
        i=i+1;
    end
    g(j+1)=i;
end

%Generate random variates
x=[];
u=rand(1,n);
x=g(floor(1+u.*c));
```

```
for i=1:n
    while u(i)>cp(x(i)+1)
        x(i)=x(i)+1;
    end
end
%End of function file
```

Appendix B

Appendix

B.1 RoU in MATLAB

```
%Description: This program generate n generalized exponentially
%             distributed random variates with the Ratio-of-Uniform
%             (RoU) method.
%Input:       n=number of random variates desired, m=mean of the pdf,
%             sig=standard deviation of the pdf, nu=mode of the pdf
%Output:      xx=n random variates with the generalized exponential
%             distribution


%function [xx]=RoU(n,m,sig,nu)

nb=n;         %save the desire number of random variates wanted in nb
n=n*2;        %increase n to compensate for the rejection

%generate the domain for the pdf
lend=-sig*6;
rend=sig*6;
step=10000;
stepsize=(rend-lend)/step;
x=lend+m:stepsize:rend+m;

%generate pdf vector
a=nu/(sqrt(8)*sig*gamma(1/nu));
gef=a*exp(-abs((x-m)/(sqrt(2)*sig)).^nu);

%set the mode of the pdf equal to the mean
```

```
mode=m;

%Calculate the bounds for the domain and range
%in the (U,V) domain
vp=sqrt(max(gef));
un=min((x-mode).*sqrt(gef));
up=max((x-mode).*sqrt(gef));

%prepare for the rejection process
length=(abs(up)+abs(un));
u=(rand(1,n)-.5+(abs(up)-abs(un))).*length;
v=rand(1,n).*vp;

xn=u./v+mode;
vsq=v.^2;
fxn=a*exp(-abs((xn-m)/(sqrt(2)*sig)).^nu);

xx=[];
caccepted=0;
jj=1;

%Rejection process
for i=1:n
    if vsq(i)<=fxn(i)
        xx(caccepted+1)=xn(i);
        caccepted=caccepted+1;
        uacc(jj)=u(i);      %For last option, can be omitted
        vacc(jj)=v(i);      %For last option, can be omitted
    end
    jj=jj+1;                %For last option, can be omitted
end

%Deliver the desired number of random variates
check=xx;
xx=xx(1:nb);

%End of function file


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Below are some extra options for generator performance%
%They are commented out                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%The two lines below that are commented out are for
```

```
%efficiency display
eff=caccepted/n;
fprintf(['\nThe efficiency is ' num2str(eff) '%.'])

%The four lines below that are commented out are for
%a random variates fitting curve

%[am,where]=hist(check,100);
%y=a*exp(-abs((where-m)./(sqrt(2)*sig)).^nu);
%prop=am/caccepted/(where(3)-where(2));
%plot(where,y,where,prop,'ok')

%The three lines that are commented out are for
%a plot with the pdf, UBF, and LBF in
%original domain
%plot(uacc,vacc)
%hold on
%plot([un,un,up,up],[0,vp,vp,0])
```

Appendix C

Appendix

C.1 File: c10_MCBPSKber.m

```
% File: c10_MCBPSKber.m
% Software given here is to accompany the textbook: W.H. Tranter,
% K.S. Shanmugan, T.S. Rappaport, and K.S. Kosbar, Principles of
% Communication Systems Simulation with Wireless Applications,
% Prentice Hall PTR, 2004.
%
EbNodB = 0:8; % vector of Eb/No (dB) values
z = 10.^(EbNodB/10); % convert to linear scale
delay = 5; % enter delay value (samples)
BER = zeros(1,length(z)); % initialize BER vector
Errors = zeros(1,length(z)); % initialize Errors vector
BER_T = q(sqrt(2*z)); % theoretical (AWGN) BER vector
N = round(20./BER_T); % 20 errors for ideal (zero ISI) system
FilterSwitch = 1; % Tx filter out (0) or in (1)
for k=1:length(z)
   N(k) = max(1000,N(k)); % ensure at least one block processed
   [BER(k),Errors(k)] = c10_MCBPSKrun(N(k),z(k),delay,FilterSwitch)
end
semilogy(EbNodB,BER,'o',EbNodB,BER_T)
xlabel('E_b/N_0 - dB'); ylabel('Bit Error Rate'); grid
legend('System Under Study','AWGN Reference',0)
% End of script file.
```

C.2 File: c10_MCBPSKdelay.m

```
% File: c10_MCBPSKdelay.m
% Software given here is to accompany the textbook: W.H. Tranter,
% K.S. Shanmugan, T.S. Rappaport, and K.S. Kosbar, Principles of
% Communication Systems Simulation with Wireless Applications,
% Prentice Hall PTR, 2004.
%
EbNodB = 6; % Eb/No (dB) value
z = 10.^(EbNodB/10); % convert to linear scale
delay = 0:8; % delay vector
BER = zeros(1,length(delay)); % initialize BER vector
Errors = zeros(1,length(delay)); % initialize Errors vector
BER_T = q(sqrt(2*z))*ones(1,length(delay));     % theoretical BER vector
N = round(100./BER_T); % 100 errors for ideal (zero ISI) system
FilterSwitch = 1; % set filter switch (in=1 or out=0)
for k=1:length(delay)
   [BER(k),Errors(k)] = c10_MCBPSKrun(N(k),z,delay(k),FilterSwitch)
end
semilogy(delay,BER,'o',delay,BER_T,'-'); grid;
xlabel('Delay'); ylabel('Bit Error Rate');
% End of script file.
```

C.3 File: c10_MCBPSKdelay.m

```
% File: c10_MCBPSKrun.m
% Software given here is to accompany the textbook: W.H. Tranter,
% K.S. Shanmugan, T.S. Rappaport, and K.S. Kosbar, Principles of
% Communication Systems Simulation with Wireless Applications,
% Prentice Hall PTR, 2004.
%
function [BER,Errors]=MCBPSKrun(N,EbNo,delay,FilterSwitch)
SamplesPerSymbol = 10;    % samples per symbol
BlockSize = 1000; % block size
NoiseSigma = sqrt(SamplesPerSymbol/(2*EbNo));   % scale noise level
DetectedSymbols = zeros(1,BlockSize); % initialize vector
NumberOfBlocks = floor(N/BlockSize); % number of blocks processed
[BTx,ATx] = butter(5,2/SamplesPerSymbol);   % compute filter parameters
[TxOutput,TxFilterState] = filter(BTx,ATx,0); % initialize state vector
BRx = ones(1,SamplesPerSymbol); ARx=1;          % matched filter parameters
Errors = 0; % initialize error counter
%
% Simulation loop begine here.
```

```
%
for Block=1:NumberOfBlocks
     %
     % Generate transmitted symbols.
     %
     [SymbolSamples,TxSymbols] = random_binary(BlockSize,SamplesPerSymbol);
     %
     % Transmitter filter if desired.
     %
     if FilterSwitch==0
        TxOutput = SymbolSamples;
     else
        [TxOutput,TxFilterState] = filter(BTx,ATx,SymbolSamples,TxFilterState);
     end
     %
     % Generate channel noise.
     %
     NoiseSamples = NoiseSigma*randn(size(TxOutput));
     %
     % Add signal and noise.
     %
     RxInput = TxOutput + NoiseSamples;
     %
     % Pass Received signal through matched filter.
     %
     IntegratorOutput = filter(BRx,ARx,RxInput);
     %
     % Sample matched filter output every SamplesPerSymbol samples,
     % compare to transmitted bit, and count errors.
     %
     for k=1:BlockSize,
        m = k*SamplesPerSymbol+delay;
        if (m < length(IntegratorOutput))
           DetectedSymbols(k) = (1-sign(IntegratorOutput(m)))/2;
           if (DetectedSymbols(k) ~= TxSymbols(k))
              Errors = Errors + 1;
           end
        end
     end
end
BER = Errors/(BlockSize*NumberOfBlocks);  % calculate BER
% End of function file.
```

C.4 File: q.m

```
% File: q.m
% Software given here is to accompany the textbook: W.H. Tranter,
% K.S. Shanmugan, T.S. Rappaport, and K.S. Kosbar, Principles of
% Communication Systems Simulation with Wireless Applications,
% Prentice Hall PTR, 2004.
%
function y=q(x)
y = 0.5*erfc(x/sqrt(2));
% End function file.
```

Appendix D

Vita

Yik Yang was born in Hong Kong, China on September 4, 1981. He received his Bachelor of Science degree in Electrical Engineering (BSEE) from Virginia Polytechnic Institute and State University (Virginia Tech) in the summer of 2003. After he obtained his BSEE degree, he continued his study at Virginia Tech as he was pursuing his Master of Science degree in Electrical Engineering (MSEE). During the beginning of his graduate studies, he focused on electromagnetics and RF communication. Later in his studies, he decided to work for Dr. W. Tranter in noise simulation. Besides his interest in simulation, he is also interested in electronic, electromagnetics, RF communication, and mechatronics.