Collaborator 1: Bowie Chuang (NetID: bchuang)
Collaborator 2: Isabella Sri (NetID: isabellasri)
Collaborator 3: Sunny Zhong (NetID: chenyangzhong)

PSTAT 194CS Midterm Project Report

The article that we selected, "Generating Random Variables that follow the Beta Distribution Using the Neutrosophic Acceptance-Rejection Method", introduces the process of generating random variables that follow the beta distribution utilizing the Neutrosophic Acceptance-Rejection Method, a technique that combines neutrosophic set theory with the acceptance-rejection principle. A neutrosophic probability distribution is a probability distribution that not only includes the true likelihood value of a random variable within the range from 0 to 1, but also expresses the indeterminacy and uncertainty associated with each likelihood. Therefore, neutrosophic random variables embody the idea of imprecisely determined quantities, which captures the uncertainty of real-world situations. Because the beta distribution is widely used to model random variables that are constrained to the range from 0 to 1, using neutrosophic set theory in conjunction with the acceptance-rejection principle to generate random variables that follow the beta distribution proves to be useful for random number generation and simulating situations with uncertain outcomes across a wide range of disciplines, such as the outcome of a baseball game or the state of a financial market.

Description of the Random Number Generator:

1. Find a four digit decimal number to set a seed R_0. Generate two random numbers R_1 and through the method of squaring the average of R_0:

   For example: Let's say our R_0 *is* 0.1273*, then the method of squaring the average will give us* 0.1273^2*, which is* 0.01620592*. We then extract the middle four digits after the decimal to randomly generate our* R_1*, which is* 0.6205*. We proceed to find* R_2 *by squaring* R_1 *and extract the middle four digits.*

2. Convert R_1 and R_2 to neutrosophic random numbers NR_1 and $NR_2$by taking into account uncertainty in both lower and upper limits. $NR_1 = R_1 - \varepsilon$ and $NR_2 = R_2 - \varepsilon$ where $NR_1 = R_1 - \varepsilon \in [R_1, R_1 - \varepsilon]$ and $NR_1 = R_1 - \varepsilon \in [R_1, R_1 - \varepsilon]$.

   a. Find ε : ***Example 1*** in the article used the fixed non-deterministic bounds of ε ∈ [0, 0.02].

b. Find NR_1 : Since $R_1 = 0.6205, NR_1 = [0.6205,\ 0.6005]$

c. Find NR_2 : Since $R_2 = 0.5020, NR_2 = [0.5020,\ 0.482]$

3. Add indeterminacy to the beta distribution by adding indeterminacy to the parameters α and β and generate neutrosophic values: $\beta_N = \beta + \delta$ and $\alpha_N = \alpha + \varepsilon$

   a. Both δ and ε represent indeterminacy for our α and β parameters

4. To generate neutrosophic random variables that follow the beta distribution, we use the following probability density function:

$$f(x) = \frac{\Gamma(\alpha_N + \beta_N)}{\Gamma(\alpha_N)\Gamma(\beta_N)} x^{\alpha_N - 1}(1 - x)^{\beta_N - 1}\ ;\ 0 \le x \le 1$$

5. We want to find the maximum value of this distribution, which entails taking the derivative of the beta function we established and setting it equal to 0. By finding the max value (M_N), we can test the inequality of $NR_2 \le \frac{f(NR_1)}{M_N}$. If the equality is satisfied, then we accept that $Nx_1 = NR_1$ follows the beta distribution by this equation:

$$f(x) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha - 1}(1 - x)^{\beta - 1};\ 0 \le x \le 1 \quad ; \quad M_N = \frac{\alpha_N - 1}{\alpha_N + \beta_N - 2}$$

6. If NR_2 does not satisfy the inequality in Step 5, we reject both NR_1 and NR_2, returning to Step 1 to generate a new pair of random numbers until we accept it. This step embodies the accept-reject principle by examining the ratio of the two samples to ensure it is greater than or equal to NR_2. Iteratively repeating this process ensures the generated samples meet the specified criteria.

Application of the Methods Used to Generate the Selected Random Variables

While implementing the RNG method for the beta distribution as described, we aim to address all three scenarios: $\alpha > \beta$, $\alpha = \beta$, and $\alpha < \beta$. We are also interested in exploring both scenarios: one incorporating neutrosophic random variables for the parameters α and β, and the other without. In total, we will implement six cases and discuss their respective results and findings. We initiated the application process by determining the seed number, denoted as R_0. To achieve this, we generated ten random numbers between 0 and 1 using the runif() function ten times consecutively. Subsequently, we calculated the mean of these ten numbers to obtain a more randomized seed number. Afterwards, we developed functions to extract the middle four digits following the implementation of squaring methods to determine R_1 and R_2, as discussed in the previous section. We imposed conditions on our error terms ε and δ, setting them to fixed values, to generate our parameters α and β. We then executed the inequality function to verify whether our generated NR_1 and NR_2 meet acceptance criteria. If not, we rerun our code with a new R_0 determined, and restart the process to generate new R_1 and R_2, repeating the accept-reject method.

Discussion of Challenges Faced: Solutions and Insights

Throughout our project, we encountered several challenges. The first obstacle we faced was the initial comprehension of the research article of the RNG method, as the concepts and formulas presented in the academic journal required substantial prior knowledge. However, after individually reading the article and engaging in group discussions, we were able to exchange ideas and gain a clearer understanding of the content of the article. We also encountered challenges when we began implementing the method, as the article did not provide any guidance or aspects on the implementation of the RNG method in R. We brainstormed different approaches for developing the algorithms, yet we faced uncertainty regarding the successful selection of essential random variables to initiate the neutrosophic rejection and acceptance method, such as R_0, δ, and ε. We tackled this challenge by establishing a structured framework for the project. We then engaged in discussions with Professor Ravat and TA Isaiah to brainstorm the most effective approach for this process. Through collaboration and consultation, we successfully overcame our challenges.

For the results we generated on the R Markdown file, we noticed that the algorithms that we implemented are really sensitive to the R_0 seed value that we randomly generated. We encountered numerous rejections when trying different set.seed() numbers as well as different ε and δ values, but none of the adjustments we made to the error terms actually influenced the results of our accept and reject algorithm. With the set.seed(111) selected in our code, we were able to achieve acceptance for all three different scenarios ($\alpha > \beta$, $\alpha = \beta$, and $\alpha < \beta$) in both the RNG method with and without the neutrosophic random variables. This concluded that the RNG method we implemented requires a certain R_0 seed number while the error terms for our parameter values appear to have less pronounced effects on our results.

Appendix

<u>Reference:</u>

Jdid, Maissam and Nada A. Nabeeh. "Generating Random Variables that follow the Beta Distribution Using the Neutrosophic Acceptance-Rejection Method." Neutrosophic Sets and Systems 58, 1 (2023). https://digitalrepository.unm.edu/nss_journal/vol58/iss1/9

** Code is on the next page*

PSTAT 194CS Midterm Project Appendix: Code

Sunny Zhong, Isabella Sri, Bowie Chuang

2024-05-13

Generating Seed Number: R_0

```r
# Generate 10 random numbers and take the mean of that 10 numbers to generate R_0
set.seed(111)
R_0 = round(runif(10, max = 1, min = 0), 4)
R_0 = mean(R_0)
```

Finding R_1 and R_2: Developing the function that picks the middle four digits

```r
extract_middle_four = function(input_avg_number) {
  sqrd_x = input_avg_number * input_avg_number
  sqrd_x_str = as.character(sqrd_x)
  middle_start = ceiling((nchar(sqrd_x_str) - 4) / 2) + 2
  middle_digits = substr(sqrd_x_str, start = middle_start, stop = middle_start + 3)
  middle_digits = as.integer(middle_digits)
  x = format(middle_digits / 10^4, nsmall=4)
  x = as.numeric(x)
  x = round(x, digits=4)
  return(x)
}

R_1 = extract_middle_four(R_0)
R_2 = extract_middle_four(R_1)
```

Beta Function

```r
beta_func = function(x, a, b) {
  beta_result = beta(a, b) * (x ^ (a - 1)) * ((1 - x) ^ (b - 1))
  return(beta_result)
}
```

Three Scenarios of Parameters (Using Examples from the Article):

$\alpha > \beta; \alpha = 7, \beta = 3$

$\alpha = \beta; \alpha = 5, \beta = 5$

$\alpha < \beta; \alpha = 3, \beta = 7$

$\epsilon \in [0, 0.02]$ *is constant for all three scenarios*

We will use the same seed number R_0 and the random numbers R_1 and R_2 for all three scenarios.

Implementing RNG Method for Beta Distribution Parameters Excluding Neutrosophic Random Variables

```
epsilon = c(0, 0.02)
NR_1 = R_1 - epsilon
NR_2 = R_2 - epsilon
```

$\alpha > \beta; \alpha = 7, \beta = 3$

```
# a = 7, b = 3
a = 7
b = 3
M = (a - 1)/(a + b - 2)

acceptance <- TRUE
for (i in 1:length(NR_2)) {
  NR1_function = beta_func(NR_1[i], a, b)/ M
  if (NR_2[i] > NR1_function){
    acceptance <- FALSE
    break
  }
}

if (acceptance) {
  print("We accept that N(R1) follows the beta distribution")
} else {
  print("We reject that N(R1) follows the beta distribution. Try Again with a different Random Number")
}
```

```
## [1] "We accept that N(R1) follows the beta distribution"
```

$\alpha = \beta; \alpha = 5, \beta = 5$

```
# a = 5, b = 5
a2 = 5
b2 = 5
M = (a2 - 1)/(a2 + b2 - 2)

acceptance <- TRUE
for (i in 1:length(NR_2)) {
  NR1_function = beta_func(NR_1[i], a2, b2)/ M
  if (NR_2[i] > NR1_function){
    acceptance <- FALSE
    break
  }
```

```r
}

if (acceptance) {
  print("We accept that N(R1) follows the beta distribution")
} else {
  print("We reject that N(R1) follows the beta distribution. Try Again with a different Random Number")
}
```

```
## [1] "We accept that N(R1) follows the beta distribution"
```

$\alpha < \beta; \alpha = 3, \beta = 7$

```r
# a = 3, b = 7
a3 = 3
b3 = 7
M = (a3 - 1)/(a3 + b3 - 2)

acceptance <- TRUE
for (i in 1:length(NR_2)) {
  NR1_function = beta_func(NR_1[i], a3, b3)/ M
  if (NR_2[i] > NR1_function){
    acceptance <- FALSE
    break
  }
}

if (acceptance) {
  print("We accept that N(R1) follows the beta distribution")
} else {
  print("We reject that N(R1) follows the beta distribution. Try Again with a different Random Number")
}
```

```
## [1] "We accept that N(R1) follows the beta distribution"
```

Implementing the RNG Method to Neutrosophic Random Variables with $\delta = 0.1$ and $\epsilon = 0.2$

```r
NR_1 = R_1 - epsilon
NR_2 = R_2 - epsilon

epsilon_1 = 0.2
delta = 0.1
```

$\alpha > \beta; \alpha = 7, \beta = 3$

```r
a_n = a + epsilon_1
b_n = b + delta
M = (a_n - 1)/(a_n + b_n - 2)
NR1_function = beta_func(NR_1, a, b)/ M

acceptance <- TRUE
```

```r
for (i in 1:length(NR_2)) {
  NR1_function = beta_func(NR_1[i], a_n, b_n)/ M
  if (NR_2[i] > NR1_function){
    acceptance <- FALSE
    break
  }
}

if (acceptance) {
  print("We accept that N(R1) follows the beta distribution")
} else {
  print("We reject that N(R1) follows the beta distribution. Try Again with a different Random Number")
}
```

```
## [1] "We accept that N(R1) follows the beta distribution"
```

$\alpha = \beta; \alpha = 5, \beta = 5$

```r
a_n2 = a2 + epsilon_1
b_n2 = b2 + delta
M = (a_n2 - 1)/(a_n2 + b_n2 - 2)
NR1_function = beta_func(NR_1, a2, b2)/ M

acceptance <- TRUE
for (i in 1:length(NR_2)) {
  NR1_function = beta_func(NR_1[i], a_n2, b_n2)/ M
  if (NR_2[i] > NR1_function){
    acceptance <- FALSE
    break
  }
}

if (acceptance) {
  print("We accept that N(R1) follows the beta distribution")
} else {
  print("We reject that N(R1) follows the beta distribution. Try Again with a different Random Number")
}
```

```
## [1] "We accept that N(R1) follows the beta distribution"
```

$\alpha < \beta; \alpha = 3, \beta = 7$

```r
a_n3 = a3 + epsilon_1
b_n3 = b3 + delta
M = (a_n3 - 1)/(a_n3 + b_n3 - 2)
NR1_function = beta_func(NR_1, a2, b2)/ M

acceptance <- TRUE
for (i in 1:length(NR_2)) {
  NR1_function = beta_func(NR_1[i], a_n3, b_n3)/ M
  if (NR_2[i] > NR1_function){
    acceptance <- FALSE
    break
  }
}

if (acceptance) {
  print("We accept that N(R1) follows the beta distribution")
} else {
  print("We reject that N(R1) follows the beta distribution. Try Again with a different Random Number")
}
```

```
## [1] "We accept that N(R1) follows the beta distribution"
```