Normally Distributed Sample Generation

N Toma, University of Bucharest I Radu, University of Bucharest

Contents

Introduction

Problem statement

We were assigned the task of developing an application illustrating the procedure of generating a normally distributed random variable using the acceptance-rejection algorithm and the Box-Muller method.

Necessary libraries and dependencies

In order for the following code snippets to run correctly, both 'Shiny' (web app package) and 'bslib' (theme package) must be installed and included in the *app.py* file.

```
library("shiny")
library("bslib")
```

```
##
## Attaching package: 'bslib'
```

```
## The following object is masked from 'package:utils':
##
##     page
```

The User Interface

An appropriate user interface was developed using the built-in shiny components so that the user can make use of all the available features.

```
ui <- fluidPage(
  # bootstrap theme
  theme = bs_theme(bg = "#232324", fg = "white", primary = "#F9E153"),

  h4(strong("Normally Distributed Sample Generation Engine"), align = "center"),

  br(),
  tabsetPanel(
             tabPanel(title = "Box-Muller",
                      sidebarLayout(
                        sidebarPanel(
                          sliderInput(inputId = "numberOfIterationsBoxMuller",
                                      label = "Number of Iterations", min = 10000
                                      , max = 500000, value = 100000, step =  1),
                          sliderInput(inputId = "sampleRangeBoxMuller",
                                      label = "Sample Range", min = -500,
                                      max = 500, value = c(-100, 100), step = 1),
                          sliderInput(inputId = "numberOfBarsBoxMuller",
                                      label = "Number of Histogram Bars",
                                      min = 50, max = 1000, value = 500,
                                      step =  1),
                          numericInput(inputId = "desiredMeanBoxMuller",
                                      label = "Desired Mean",
                                      value = 0, step = 0.1),
                          numericInput(inputId = "desiredVarianceBoxMuller",
                                       label = "Desired Variance", min = 0,
                                       value = 1, step = 0.1),
                          checkboxInput(inputId = "showPDFBoxMuller",
                                        label = "Show PDF", value = FALSE),
                          submitButton(text = "Generate"),
                          br(),
                          p("The Box-Muller method makes use of two randomly
                            generated uniformly distributed numbers to generate
                            a pair of numbers with a standard normal
                            distribution (mean = 0, variance = 1)."),
                          p("The generated sample can then be transformed into
                            another normally distributed sample with the
                            specified mean and variance.")
                        ),
                        mainPanel(
                          br(),
                          plotOutput(outputId = "BoxMuller")
                        )
                      )),

             tabPanel(title = "Rejection",
                      sidebarLayout(
                        sidebarPanel(
                          sliderInput(inputId = "numberOfIterationsRejection",
                                      label = "Number of Iterations", min = 10000,
                                      max = 500000, value = 100000, step =  1),
                          sliderInput(inputId = "sampleRangeRejection",
                                      label = "Sample Range", min = -500,
```

```
                                max = 500, value = c(-5, 5), step = 1),
                    sliderInput(inputId = "numberOfBarsRejection",
                                label = "Number of Histogram Bars",
                                min = 50, max = 1000,
                                value = 500, step =  1),
                    numericInput(inputId = "desiredMeanRejection",
                                 label = "Desired Mean",
                                 value = 0, step = 0.1),
                    numericInput(inputId = "desiredVarianceRejection",
                                 label = "Desired Variance", min = 0,
                                 value = 1, step = 0.1),
                    radioButtons(inputId = "typeOfGraph",
                                 label = "Type of Graph",
                                 choices = c("Histogram" = 0,
                                      "Accepted Point Visualization" = 1)),
                    checkboxInput(inputId = "showPDFRejection",
                                  label = "Show PDF", value = FALSE),
                    submitButton(text = "Generate"),
                    br(),
                    p("Rejection Sampling makes use of two independent
                      uniformly distributed random variables to generate
                      an element in the specified range and a number in
                      the range of possible values taken by the desired
                      normal distribution's PDF."),
                    p("When the (y-axis) generated value of the latter
                      uniform variable is greater than the value of the
                      PDF at the former uniform number, said number is
                      rejected. Else, it is accepted and admitted into
                      the sample."),
                    p("Intuitively, at the points at which the PDF is
                      greater, more uniform numbers will be accepted,
                      since their pair y-axis uniform values have a
                      larger 'acceptance' range.")
                  ),
                  mainPanel(
                    br(),
                    plotOutput(outputId = "Rejection")
                  )
                )),
            ),
  br(),

  tags$div(h6("This WebApp was developed as a project for the ",
           strong("Probabilities and Statistics"), " course of the ",
           strong("University of Bucharest.")),
  h6("Project Contributors: ", strong("playback0022 (Toma), IRadu15 (Radu)")),
           style = "background-color:#303030; padding: 0.5rem; margins: 0;
                    text-align: center;")
)
```

The Acceptance-Rejection algorithm

The rejection sampling method is a technique for generating samples from a target probability density function (PDF), by generating samples from an auxiliary pdf and accepting or rejecting them based on the ratio of the target pdf to the auxiliary pdf. This method is particularly useful for generating samples from distributions that have no closed-form solutions for generating random samples.

Steps

1. Choose an auxiliary PDF, g(x), that is easy to sample from (uniform density function in our case), and that envelops the target PDF, f(x) (a normal density function)
2. Generate a sample, x, from the auxiliary PDF, g(x)
3. Generate a sample, y, from a uniform distribution between 0 and the maximum value of g(x)
4. Compare y/f(x) to 1. If the ratio is less than or equal to 1, accept x as a sample from the target PDF, otherwise reject x
5. Repeat steps 2-4 a desired number of times

Intuitively, at the points at which the PDF is greater, more uniform numbers will be accepted, since their pair y-axis uniform values have a larger 'acceptance' range.

Our application allows the users to select a number of iterations for which the algorithm is run, so that a larger sample can be generated. A range of numbers can also be specified and the desired mean and variance are modifiable.

Storing the pairs of coordinates, we were able to illustrate how the algorithm works, by plotting the points and coloring them differently. The resulting plot shows that points falling under f(x) were accepted and that they closely follow its curve.

A histogram of the accepted numbers on the x-axis was also employed to prove that the generated sample follows the description given by the desired PDF.

The Box-Muller method

The Box-Muller method is a technique for generating pairs of independent, standard, normally distributed (i.e. Gaussian) random numbers, given a pair of independent, uniformly distributed random numbers. This method is particularly useful for generating samples from normal distributions in simulations were performance is critical, as it is much more efficient in practice than the rejection sampling algorithm.

Steps

1. Generate two independent samples, u1 and u2, from the uniform distribution U(0, 1)
2. Compute the standard normal random variables, z1 and z2, using the following equations:

- z1 = sqrt(-2 * log(u1)) * cos(2 * pi * u2)
- z2 = sqrt(-2 * log(u1)) * sin(2 * pi * u2)

3. The pair (z1, z2) is a sample from the standard normal distribution

The end-user can choose from the same provided parameters described previously. A histogram of the generated numbers was employed, just as for the rejection sampling method.

Server code and algorithm implementations

```
server = function(input, output) {
  output$BoxMuller <- renderPlot ({
    # set the plot background color
    par(bg = "#f2f2f2")

    numberOfIterations <- input$numberOfIterationsBoxMuller
    mean <- input$desiredMeanBoxMuller
    variance <- input$desiredVarianceBoxMuller
    startRange <- input$sampleRangeBoxMuller[1]
    endRange <- input$sampleRangeBoxMuller[2]
    numberOfBreaks <- input$numberOfBarsBoxMuller

    generateNormallyDistributedPair <- function () {
      # generating the two independent uniformly distributed random variables
      uniformFirst <- runif(1)
      uniformSecond <- runif(1)

      # computing various component parts of the formula
      k <- sqrt(-2 * log(uniformFirst))
      t <- 2 * pi * uniformSecond

      # generating the normally distributed random variables
      normalPair <- c(k * cos(t), k * sin(t))
      return((normalPair))
    }

    # vector storing the generated samples;
    # at most 'numberOfIterations' pairs of samples can be generated;
    # allocating a large size to the vector from the very
    # beginning is much faster then appending elements
    generatedSample <- numeric(2 * numberOfIterations)
    count <- 0

    for (i in seq(1, numberOfIterations, 1)) {
      sample <- generateNormallyDistributedPair()
      # transform generated sample (normal distribution with mean = 0,
      # variance = 1) to the normal distribution with the specified parameters;
      # multiplying by the standard deviation and adding mean of the
      # desired distribution
      sample <- sqrt(variance) * sample + mean
      # filter out the elements outside the specified range of values
      sample <- sample[sample >= startRange & sample <= endRange]

      # there is no certainty about the accepted sample size
      # (might be 0, 1, 2), so we iterate through the sample
      for (number in sample) {
        count <- count + 1
        generatedSample[count] = number
      }
    }
```

```
  # the histogram and curve are shown only when a
  # non-null number of samples has been generated
  if (count) {
    # plotting the histogram of the generated sample
    hist(generatedSample[1:count], main = "Box-Muller Sampling",
         col = "#b5b5b5", border = "#b5b5b5", xlab = "Generated Sample",
         probability = TRUE, breaks = numberOfBreaks)
    # generating the sequence of values in the specified
    # range based on which to generate and plot the PDF
    x <- seq(startRange, endRange, 0.01)
    # plotting PDF over the histogram
    if (input$showPDFBoxMuller)
      curve(dnorm(x, mean = mean, sd = sqrt(variance)), lwd = 3,
            col = "#F9E153", add = TRUE)
  }
})


  output$Rejection <- renderPlot({
  numberOfIterations <- input$numberOfIterationsRejection
  mean <- input$desiredMeanRejection
  variance <- input$desiredVarianceRejection
  startRange <- input$sampleRangeRejection[1]
  endRange <- input$sampleRangeRejection[2]
  numberOfBreaks <- input$numberOfBarsRejection

  # plotting normal PDF with the desired parameters
  x <- seq(startRange, endRange, length.out = 500)
  y <- dnorm(x, mean = mean, sd = sqrt(variance))
  plot(x, y, type = "l", xlab = "X", ylab = "Y", main = "Rejection Sampling")
  maxValuePDf <- dnorm(mean, mean = mean, sd = sqrt(variance))

  # generating normally-distributed sample;
  # both the accepted and rejected points will be stored,
  # in order to plot the acceptance and rejection regions
  # and illustrate that the accepted samples fall within
  # the PDF boundaries;
  # as previously stated, allocating a large size to the vector
  # from the very beginning is much faster then appending elements
  accepted_samples_x <- numeric(numberOfIterations)
  accepted_samples_y <- numeric(numberOfIterations)
  rejected_samples_x <- numeric(numberOfIterations)
  rejected_samples_y <- numeric(numberOfIterations)

  accepted_count <- 0
  rejected_count <- 0

  for (i in seq(1, numberOfIterations, 1)) {
    # elements in the sample range
    x <- runif(1, min = startRange, max = endRange)
    # numbers describing x's probability
    y <- runif(1, min = 0, max = maxValuePDf)
```

```
      # as specified, only the points falling under the PDF are accepted
      if (y <= dnorm(x, mean = mean, sd = sqrt(variance))) {
        accepted_count <- accepted_count + 1
        accepted_samples_x[accepted_count] <- x
        accepted_samples_y[accepted_count] <- y
      }
      else{
        rejected_count <- rejected_count + 1
        rejected_samples_x[rejected_count] <- x
        rejected_samples_y[rejected_count] <- y
      }
    }

    # accepted rate visualization
    if (input$typeOfGraph == 1) {
      points(accepted_samples_x, accepted_samples_y, pch = 19, col = "#F9E153")
      points(rejected_samples_x, rejected_samples_y, pch = 3, col = "#b5b5b5")
    }
    # histogram
    else {
      # plotting the histogram of the generated sample
      par(bg = "#f2f2f2")
      hist(accepted_samples_x[1:accepted_count], main = "Rejection Sampling",
           col = "#b5b5b5", border = "#b5b5b5", xlab = "Generated Sample",
           probability = TRUE, breaks = numberOfBreaks)
      # generating the sequence of values in the specified
      # range based on which to generate and plot the PDF
      x <- seq(startRange, endRange, 0.01)
      # plotting PDF over the histogram
      if (input$showPDFRejection)
        curve(dnorm(x, mean = mean, sd = sqrt(variance)), lwd = 3,
              col = "#F9E153", add = TRUE)
    }
  })
}



shinyApp(ui = ui, server = server)
```

Demo

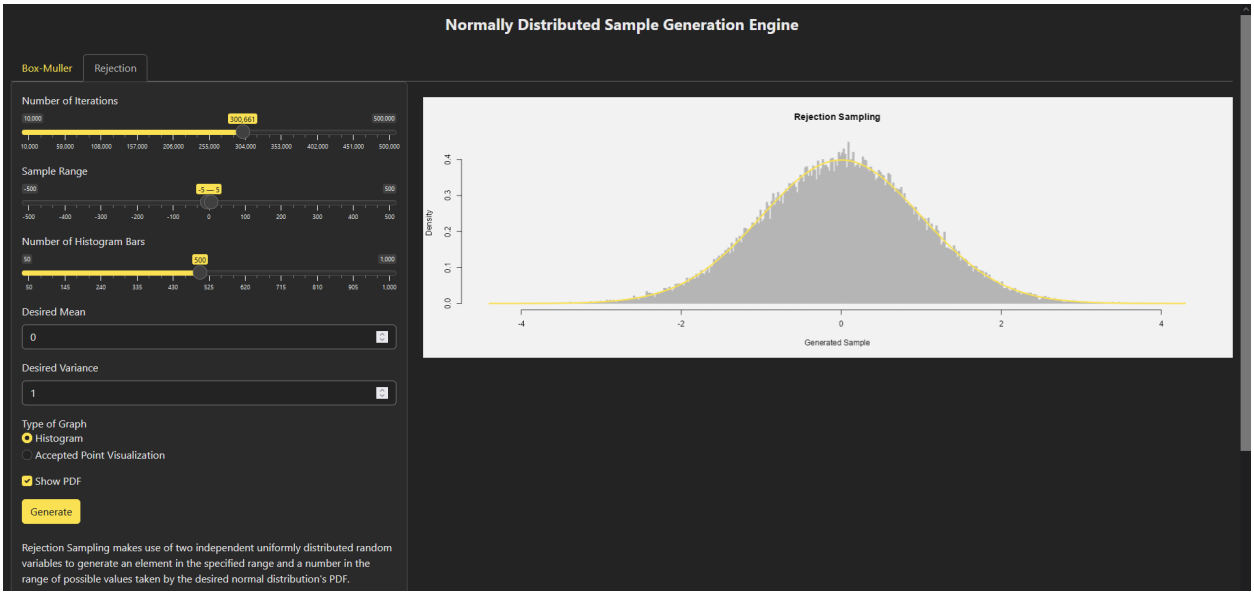

Figure 1: Acceptance-Rejection Histogram Preview

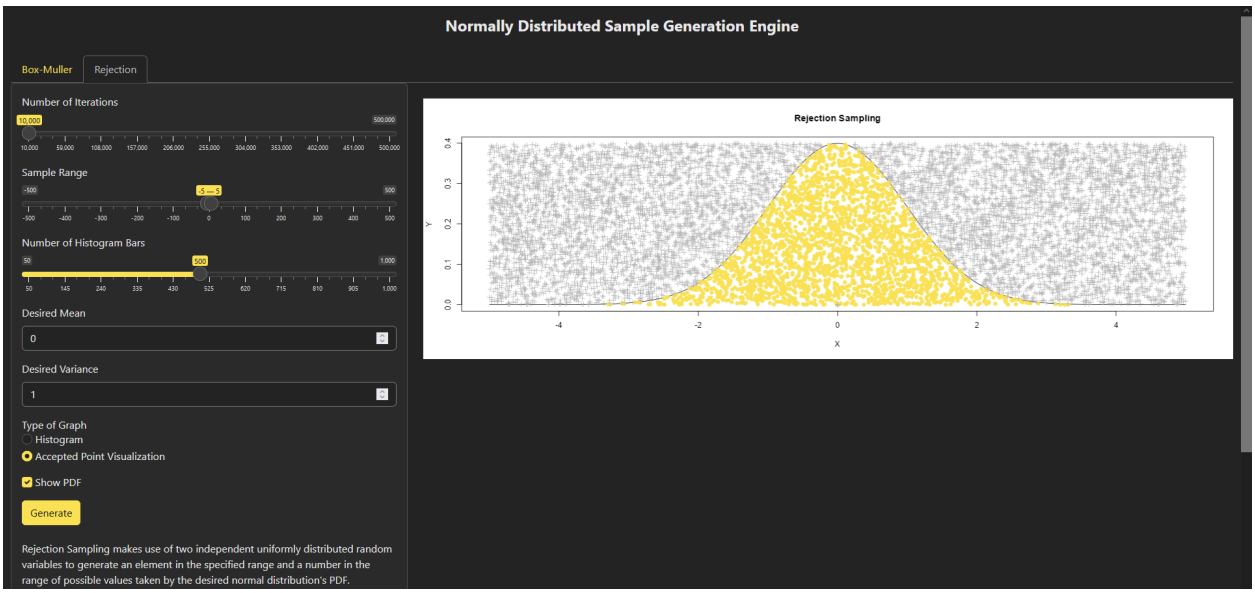

Figure 2: Acceptance-Rejection Plot Preview

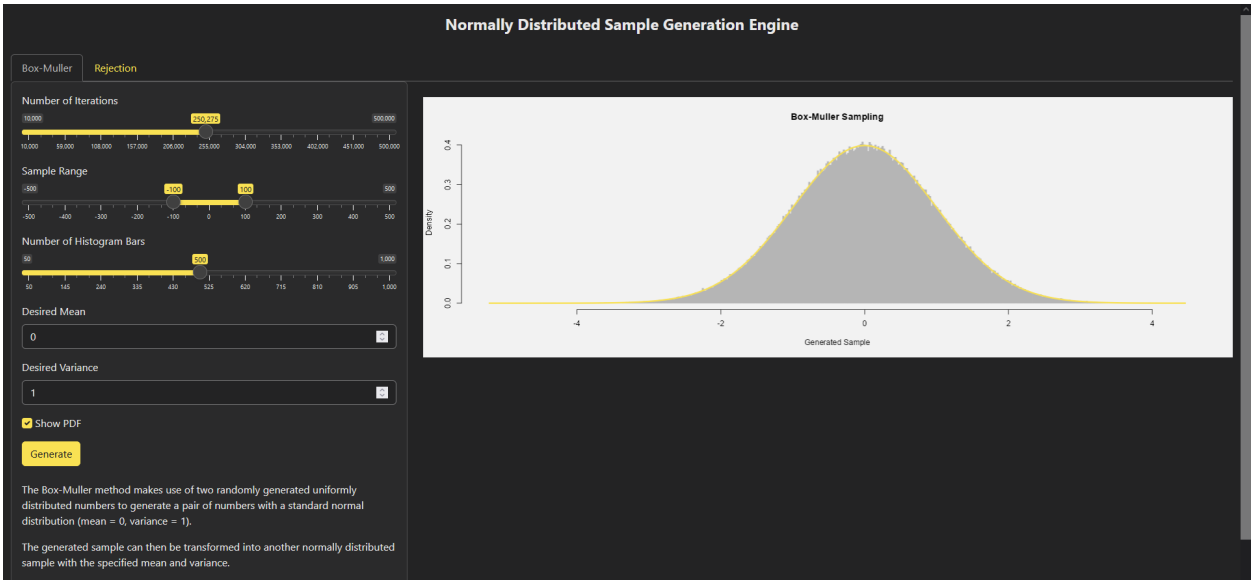

Figure 3: Box-Muller Histogram Preview