Random Variable Generation

Sujit Sandipan Chaugule[1*], Dr. Amiya Ranjan Bhowmick[2]

[1*]Department of Pharmaceutical Sciences and Technology, Institute of Chemical Technology, Mumbai

[2]Department of Mathematics, Institute of Chemical Technology, Mumbai

1. Introduction

In this chapter, we shall learn about basic principles and implementation of some algorithms for generating random numbers following some specific probability distributions. Suppose that a random variable X has the cumulative distribution function (CDF) $F(x)$. $F(x)$ will be a step function if the range of X is finite or countable. If X is continuous, then $F(x)$ is continuous function. Several possibilities for X can appear. For example, X may be a mixture of two other random variables. Another possibility may be X is a linear combination of discrete and continuous random variables. In this document, we shall discuss the simulation of random numbers for various possibilities of the CDF $F(x)$ and then also discuss the simulation algorithms for multivariate distributions. We start our discussion with the most fundamental result, called the probability integral transform. Unless otherwise stated, F will be used for cumulative distribution function and f will be used for probability density function or mass function.

2. Probability Integral Transform

If $X \sim F(x)$, then $Y = F(X) \sim \text{uniform}(0,1)$. The result is true for any random variable X whether it is discrete or continuous type.

The above statement is not merely a simple observation, but, it has remarkable implications. We have learned about several types of random variables and certainly simulation of these random numbers is not at all a trivial task. However, by virtue of the above statement, any random variable can be simulated by simulating from the $\text{uniform}(0,1)$ distribution by providing the inverse transformation of the cumulative distribution. The idea is that if we simulate $U \sim \text{uniform}(0,1)$, then $X = F^{-1}(U)$ is a realization from the desired CDF $F(x)$. We demonstrate this using examples below.

2.1. Simulation of continuous random variables

When X is continuous, then $F(x)$ is strictly monotone and we can uniquely compute the inverse function $F^{-1}(y)$ for each $y \in (0,1)$. If $X \sim \exp(\beta)$, then

$$F(x) = \begin{cases} 0, & x < 0 \\ 1 - e^{-\frac{x}{\beta}}, & 0 \le x < \infty \end{cases}.$$

Then $F^{-1}(y) = -\beta \ln(1-y), \quad y \in (0,1)$. Now simulate $y_1, y_2, \ldots, y_n$ from the $\text{uniform}(0,1)$, and compute $x_i = F^{-1}(y_i) = -\beta \ln(1-y_i)$. Then $x_1, x_2, \ldots, x_n$ will be realization from the $\exp(\beta)$ distribution function. The simulation scheme is implemented in the following code.

In [1]:
```
using Plots, Statistics, StatsBase, LaTeXStrings, Distributions
```

In [2]:
```
n = 1000 # sample size
beta = 2 # true parameter
f(x) = (1/beta).*exp.(-x/beta)*(x>0) # population pdf f(x)

F(x) =  (1- exp.(-x/beta))*(x>0) # population cdf F(x)
x = -1:0.1:6 # mesh for interval [-1,6]
y = zeros(length(x))

for i in 1:length(x)
    y[i] = F(x[i])
end

p1 = plot(x,y, color = "red", xlabel = L"x", ylabel = L"F_x {X}",
    lw = 2, label = "")

function inv_F(y)    # inverse of F(x)
    if y <= 0 || y >= 1
        return "The function is defined only on (0,1)"
    end
    return -beta* log(1 - y)
end

n = 1000
y =  rand(Uniform(0,1),n)  # simulate from U(0,1)

p2 = histogram(y, bins=30, normed=true, xlims=(-0.2, 1.2),
    title="", label="", xlabel = "y", ylabel = "density")

dist = Uniform(0, 1)
x_vals = range(-0.2, 1.2, length=100)
y_vals = pdf(dist, x_vals)

plot!(x_vals, y_vals, color=:red, lw=2, label="")

x = zeros(n)
for i in 1:length(x)
    x[i] = inv_F(y[i])
end

println(first(x, 6))
```

```
[2.205055852917289, 0.8125845445789754, 0.8178073523416582, 1.4092215751975237, 0.9113
912365233457, 6.944621974838722]
```

In [3]:
```
p3 = histogram(x, normalize=true, title="", label = "")
z = 0:0.1:maximum(x)
f_val = zeros(length(z))
for i in 1:length(z)
    f_val[i] = f(z[i])
end
plot!(z, f_val, color=:red, linewidth=2, label = L"exp(\beta)")

plot(p1,p2,p3, layout = (2,2),  size=(800, 600))
```

Out[3]:

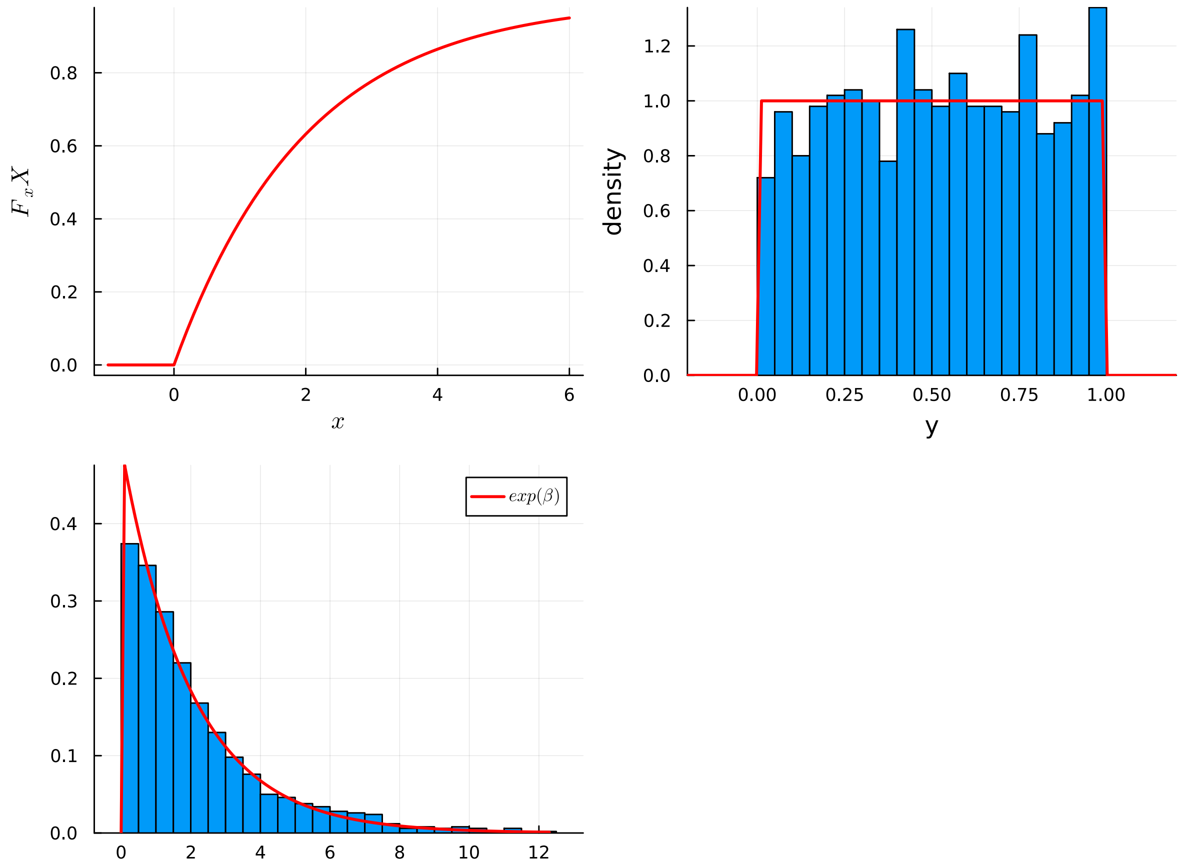
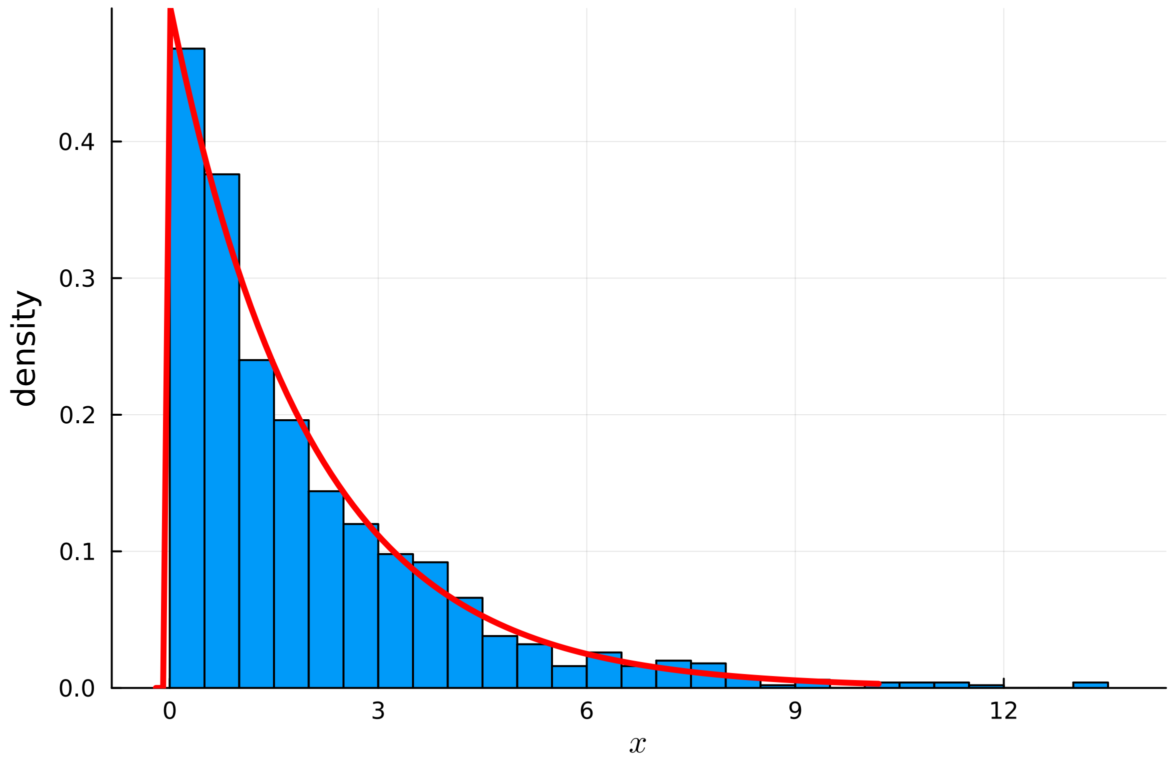


Figure 1: Cumulative distribution function of the exponential density with rate 1/2. Histgoram of the simulated values from the uniform(0,1) distribution. The values have undergone the inverse transformation F^{-1} to obtain the samples from the desired exponential density function

In [4]:

```
n = 1000
x = -2 * log.(1 .- rand(Uniform(0, 1), n))
histogram(x, normalize=true, label="", xlabel=L"x", ylabel="density")
beta = 2
dist = Exponential(beta)
x_vals = range(-0.2, 10.2, length=100)
y_vals = pdf(dist, x_vals)
plot!(x_vals, y_vals, color = "red",lw = 3, label = "")
```

Out[4]:



Although a few lines of code is fine, but every time we will not be lucky to have inbuilt functions for probability distributions in Julia. For example, in the following, we simulate random numbers from the following probability density function

$$f(x) = \begin{cases} 3x^2, & 0 < x < 1 \\ 0, & \text{otherwise} \end{cases}$$

In [5]:

```
using Plots, Statistics, StatsBase, LaTeXStrings, Distributions
```

In [6]:

```
f(x) = 3*x.^2*(x>0)*(x<1) # define the function # target pdf
x = -0.2:0.001:1.2
f_val = zeros(length(x))

for i in 1:length(x)
    f_val[i] = f(x[i])
end
p1 = plot(x,f_val, color = "red", lw = 2, ylabel =L"f_X(x)",
    title = "density function", label = "")

# CDF of x
function F(x)
    if x < 0
        return 0
    end
    if x >= 1
        return 1
    end
    return x.^3
end

x = -0.2:0.001:1.2
F_val = zeros(length(x))
for i in 1:length(x)
    F_val[i] = F(x[i])
end
```

```
p2 = plot(x, F_val, color = "red", lw = 2, ylabel =L"F_X(x)",
    title = "CDF", label = "") # plot the CDF

# Compute the inverse pdf
function inv_F(y)
    if y <= 0
        return "The function is defined only on (0,1)"
    end
    if y>=1
        return "The function is defined only on (0,1)"
    end
    return y.^(1/3)
end

n = 1000 # no of simulations
y = rand(Uniform(0,1), n) # simulate from Uniform (0,1)
x = zeros(n)

for i in 1:n
    x[i] = inv_F(y[i]) # inverse transform
end

println(first(x,6))   # print first 6 values
```

```
[0.9167695681759569, 0.9755325700215416, 0.16893724804131488, 0.7331668294050351, 0.78
67095844346131, 0.17025443546972308]
```

In [7]:
```
p3 = histogram(x, normalize = :pdf, ylims = (0,4), bins = 30,
    title = "Histogram of x", label = "") # histogram of x
z = -0.1:0.001:1.1
f_val = zeros(length(z))

for i in 1:length(z)
    f_val[i] = f(z[i])
end

plot!(z, f_val, color = "red", lw = 2, label = L"f_X(x) = 3x^2", legend=:topleft)

plot(p1,p2,p3, layout = (2,2),  size=(800, 600))
```

Out[7]:

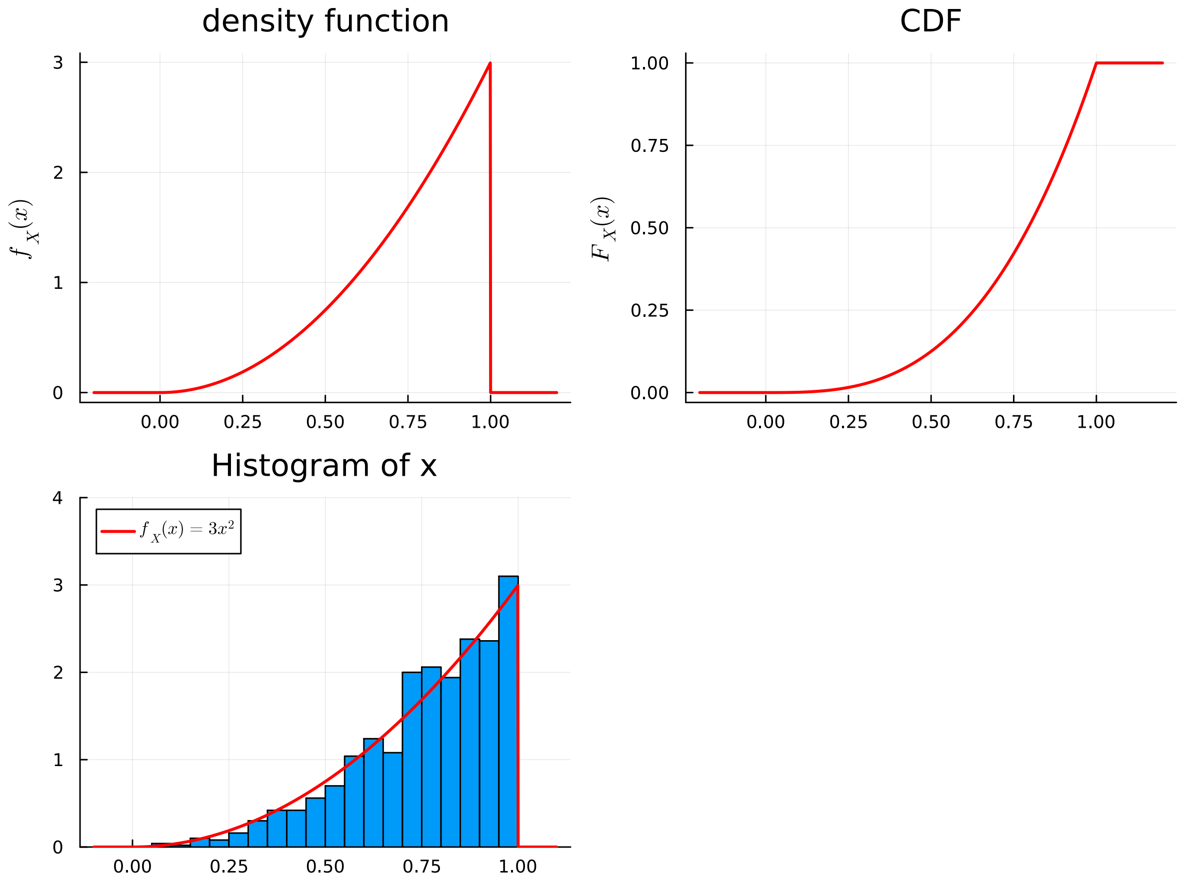


Figure 2: The steps for simulation of random numbers following some probability density function by probability integral transform.

2.2. Simulation of discrete random variables

In the previous section, we can easily compute the inverse function. However, for a discrete random variable, CDF $F(x)$ is a step function. So, there are many points in (0,1) for which we do not have a unique pre-image. In such a scenario, the pre-image, that is the inverse, is calculated as follows:

$$y \in (0,1), \quad F^{-1}(y) = \inf\{x : F(x) \geq y\}.$$

For example, if we consider $X \sim \text{binomial}(1, 0.4)$, then the CDF of X is given as

$$F(x) = \begin{cases} 0, & -\infty < x < 0 \\ 0.6, & 0 \leq x < 1 \\ 1, & 1 \leq x < \infty \end{cases}$$

Therefore, from the definition above, for all $y \in (0, 0.6]$, $F^{-1}(y) = 0$ and for all $y \in (0.6, 1)$, $F^{-1}(y) = 1$. The idea can be generalized for all discrete probability distributions. If the range of X is $\{\cdots < x_{i-1} < x_i < x_{i+1} < \ldots\}$, then the CDF $F(x)$ has discontinuities at these points. For $F^{-1}(y) = x_i$ if $F(x_{i-1}) < y \leq F(x_i)$. Therefore, the algorithm works as follows:

1. Simulate $y \sim \text{uniform}(0, 1)$.
2. If $F(x_{i-1}) < y \leq F(x_i)$, deliver x_i.

Let us verify this with the $binomial(2, 0.4)$ distribution using the following codes.

In [8]:
```
n = 2
p = 0.4
x = 0:n
p_x = pdf.(Binomial(n,p),x)

# CDF function
function F(x)
    if x < 0
        return 0
    elseif 0 <= x && x < 1
        return p_x[1]
    elseif 1 <= x && x < 2
        return sum(p_x[1:2])
    elseif 2 <= x
        return sum(p_x[1:3])
    end
end

m = 1000  # number of samples to be generated
out = zeros(Int, m)  # allocate array
y = rand(Uniform(0,1),m)  # simulate uniform(0,1)

for i in 1:m
    if y[i] < F(0)  # 0 is the pre-image
        out[i] = 0
    elseif F(0) < y[i] && y[i] <= F(1)  # 1 is the pre-image
        out[i] = 1
    elseif F(1) < y[i] && y[i] <= F(2)  # 2 is the pre-image
        out[i] = 2
    end
end
```

3. Simulation by using transformation

We can consider different transformations of the $uniform(0,1)$ random variable to generate other random variables. In the probability theory courses, we have come across the problems related to finding the CDF of some transformation of a given random variable. For example, we have shown that if $X \sim \mathcal{N}(0,1)$, then $X^2 \sim \chi_1^2$. In the following, we shall see how

$$\text{uniform} \rightarrow \text{exponential} \rightarrow \text{gamma} \rightarrow \text{beta}$$

random variables can be generated. We recollect that if $X \sim \text{gamma}(\alpha, \beta)$, then the probability density function of X is given by

$$f(x) = \begin{cases} \frac{e^{-\frac{x}{\beta}} x^{\alpha-1}}{\beta^\alpha \Gamma(\alpha)}, & 0 < x < \infty \\ 0, & \text{otherwise} \end{cases}$$

Let us suppose $X_1, X_2, \ldots, X_n$ are iid $\text{gamma}(1,1)$ (that is, $\exp(1)$ random variables, then the following three standard distributions can be derived directly from the exponential distribution.

$$Y_1 = 2\sum_{j=1}^{m} X_j \sim \chi^2_{2m}, \quad m \in \mathbb{N},$$

$$Y_2 = \sum_{j=1}^{a} X_j \sim \text{gamma}(a, 1), \quad a \in \mathbb{N},$$

$$Y_3 = \frac{\sum_{j=1}^{a} X_j}{\sum_{j=1}^{a+b} X_j} \sim \text{beta}(a, b), \quad a, b \in \mathbb{N}.$$

Using the moment generating function we can prove the above claim. Reader is encouraged to do this calculation on their own. Also, try to apply the techniques for finding distributions for the functions of random variables (Jacobian formula). In the following code, we have simulated from the χ^2_{30} using the above transformation. The process also gives the $\text{gamma}(a, \beta)$ distribution where the transformation (addition) of $\exp(3)$ random variables has been considered.

In [9]:
```
using Random, Distributions

n = 1000 # number of simulations
m = 10   # degrees of freedom for chi-squared distribution (χ²(20) = 2m)
y1 = zeros(n)

for i in 1:n
    u = rand(Uniform(0,1), m)
    x = -log.(1 .- u)
    y1[i] = 2 * sum(x)
end

p1 = histogram(y1, bins = 30, normalize = true, label = "",
     xlabel = L"Y_1", title = "")
x_vals = range(0, maximum(y1), length=100)
pdf_vals = pdf.(Chisq(2m), x_vals)
plot!(x_vals, pdf_vals, lw = 2, color = "red", label = L"\chi^2(20)")

# simulation of gamma(a,beta)
beta = 2.5
y2 = zeros(n)
a = 4

for i in 1:n
    u = rand(Uniform(0,1),a)
    x = -log.(1 .- u)
    y2[i] = 2 * sum(x)
end

p2 = histogram(y2, bins = 30, normalize = true, label = "",
     xlabel = L"Y_2", title = "")
x_vals = range(0, maximum(y2), length=1000)
pdf_vals = pdf.(Gamma(a,beta),x_vals)
plot!(x_vals, pdf_vals, lw = 2, color = "red", label = L"\Gamma(4,2.5)")


using Plots, Distributions, Random

n = 1000  # Number of simulations
a = 3     # Shape parameter 1 (integer)
b = 2     # Shape parameter 2 (integer)

y3 = zeros(n)

for i in 1:n
```

```
    u = rand(Uniform(0,1), a + b)
    x = -log.(1 .- u)
    y3[i] = sum(x[1:a]) / sum(x)
end

p3 = histogram(y3, bins = 30, normalize = true, label = "",
     xlabel = L"Y_3")
x_vals = range(0, 1, length=1000)
pdf_vals = pdf.(Beta(a,b), x_vals)
plot!(x_vals, pdf_vals, lw = 2, color = "red", label = L"\beta(3,2)")

plot(p1,p2,p3, layout = (2,2),  size=(800, 600))
```

Out[9]:

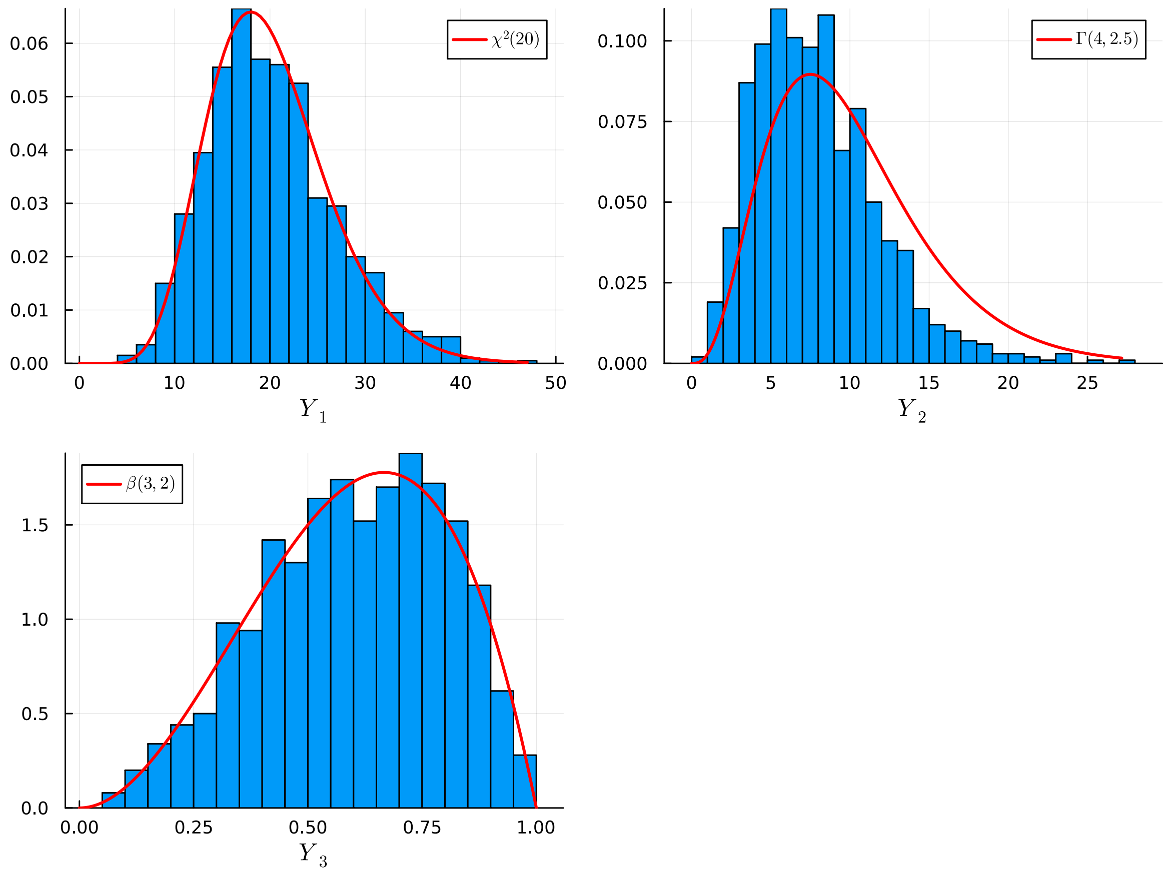


Figure 3: From the uniform(0,1) random variables, we can generate random numbers from exponential, gamma and beta density functions as verified by the agreements of histogram and the density functions (from left to right in order).

4. Accept/Reject Algorithm

Imagine that you are interested to simulate random numbers uniformly from the interval (0,0.5). However, your computer only gives numbers from the interval (0,1). A natural way to obtain random numbers from uniform(0,0.5) is to simulate 100 random numbers, say, from (0,1) and accept only those values which are falling below 0.5. Intuitively it is clear that approximately 50% of the values are going to be accepted as your desired values and the rest (which are falling in (0.5,1)) will be rejected. The basic idea of the accept-reject algorithm is that you simulate from some distribution, which you know how to simulate from, and some of those simulated values will be accepted as realizations from your desired distribution.
Intuitively, it is easy to understand that the above process, simulating from uniform(0,1) will not work if you desire to simulate uniform(0,2) distribution. Let us now formalize the algorithm below along with sample Julia codes.

The acceptance-rejection method is an algorithm for generating random samples from an arbitrary probability distribution, given as active ingredients random samples from a related distribution and the uniform distribution. The acceptance-rejection method is an advantage over the inverse CDF method of generating random numbers as it requires neither the cumulative distribution function nor its inverse to be computed. So in many cases, it can run faster. When no easily found direct transformation is available to generate the desired random variable, an extremely powerful indirect method, the Accept/Reject algorithm can often provide a solution.
Now we describe the setup for generating the random numbers. Let Y be a random variable with probability distribution $g(t)$. Suppose that we know how to draw random samples from the distribution of Y. Let X be another random variable with probability distribution $f(t)$ and we are interested in generating random samples from it. Using the random sample from the distribution of Y, random samples can be generated from the distribution of X. In such a case, the density functions $g(t)$ and $f(t)$ are denoted by candidate density and target density, respectively. Both distributions X and Y have the same support.

The Accept/Reject Algorithm

Suppose that there is a positive number (c) such that

$$\frac{f(t)}{g(t)} \leq c \quad \text{for all } t$$

1. **Simulate** $Y \sim g$ and $U \sim \text{uniform}(0, 1)$. U and Y are independent.
2. **Acceptance Criterion:**
   - If

$$U < \frac{1}{c} \frac{f(Y)}{g(Y)}$$

   then accept $x = y$, otherwise repeat step (1).

Observations:

- **Observation - I:** The quantity c must be greater than 1. Try to argue why c cannot be less than 1.
- **Observation - II:** In the process, we are actually simulating from the **candidate density** (g) and some of those simulated values are accepted as a sample from f.

Two Natural Questions:

- If you generate N values from the distribution of Y, approximately how many values will be accepted as realizations from X?
- Can we give a formal proof that the accepted values are actually from the distribution of X ?

Before answering these questions, let us understand the different components of the algorithm step by step. I have considered these steps, which will be helpful to understand how the algorithm works.

Step - I

Without loss of generality, we consider that f and g are probability mass functions, and X and Y are discrete random variables having the same support. We compute the probability that if $Y = y$ is simulated, then with what probability it will be accepted.

$$P(\text{Accept}|Y = y) = P\left(U < \frac{f(Y)}{c \cdot g(Y)}\middle|Y = y\right) = P\left(U < \frac{f(y)}{c \cdot g(y)}\right) = \frac{1}{c} \cdot \frac{f(y)}{g(y)}$$

Step - II

We compute the unconditional probability of acceptance, that is,

$$P(\text{Accept}) = \sum_y P(\text{Accept}|Y = y)P(Y = y) = \sum_y \frac{1}{c} \cdot \frac{f(y)}{g(y)} \cdot g(y) = \frac{1}{c}\sum_y f(y) = \frac{1}{c}$$

Therefore, if N is the number of simulations required to get one acceptance event, then

$$N \sim \text{geometric}\left(\frac{1}{c}\right)$$

$$E(N) = c$$

Therefore, the above algorithm will be most efficient if the value of c is minimum. So, the best choice of c is M, where

$$M = \sup_i \frac{f(i)}{g(i)} < \infty$$

Step - III

To check whether the accepted values are actually realizations from f only, we evaluate the following conditional probability:

$$P(Y = k|\text{Accept}) = \frac{P(\text{Accept}|Y = k)P(Y = k)}{P(\text{Accept})} \qquad \text{(Bayes theorem)}$$

$$= c \cdot \frac{1}{c} \cdot \frac{f(k)}{g(k)} g(k) = f(k) = P(X = k)$$

Therefore, the accepted values are indeed from the distribution of X. Given g, we get the best choice of c. However, the candidate density g should cover all possible examples from where we are mainly selecting realizations for f only.

$Remark$: The importance of the requirement that $M < \infty$ should be stressed. This can be interpreted as requiring the density of Y ($candidate\ density$) to have heavier tails than the density of X ($target\ density$). The requirement tends to ensure that we will obtain a good representation of the values of X, even those values that are in the tails. For example, if $Y \sim \text{Cauchy}(1, 0)$ and $X \sim \mathcal{N}(0, 1)$, then we can expect the range of Y samples to be wider than that of X samples, and we should expect good performance from the Accept/Reject algorithm based on these densities. However, it is much more difficult to change $\mathcal{N}(0, 1)$ random variables into a Cauchy random variable because the extremes will be underrepresented.

4.1. Example 1

Suppose the goal is to generate $X \sim \exp(\text{rate} = 2)$, the target density. First, we generate $U \sim \text{uniform}(0, 1)$ and also $Y \sim \exp(1)$, the candidate density. Then calculate M as,

$$M = \sup_{t \geq 0} \frac{f(t)}{g(t)} = \sup_{t \geq 0} \frac{2e^{-2t}}{e^{-t}} = \sup_{t \geq 0} 2e^{-t} = 2.$$

In [10]:
```
using Plots, Statistics, StatsBase, LaTeXStrings, Distributions
```

In [11]:
```
M = 2                          # Best choice of c (scaling factor)
f(x) = 2 * exp.(-2 * x)         # Target density: Exponential with rate 2
g(x) = exp.(-x)                 # Candidate density: Exponential with rate 1

N = 100000                     # Number of simulations
u = rand(Uniform(0, 1), N) # Generate N uniform random numbers
y = rand(Exponential(1), N) # Generate N from candidate density (Exp(1))
ind = u .< (1 / M) .* f.(y) ./ g.(y)
x = y[ind]

histogram(x, normalize = true, bins = 30,color = "grey", ylims = (0,2),
    label = "", xlabel = "x", ylabel = "density")
plot!(f,color = "red", lw = 2, label = "Target (exp(2))")
plot!(g,color = "blue", lw = 2, label = "candidate (exp(1))")
```

Out[11]:

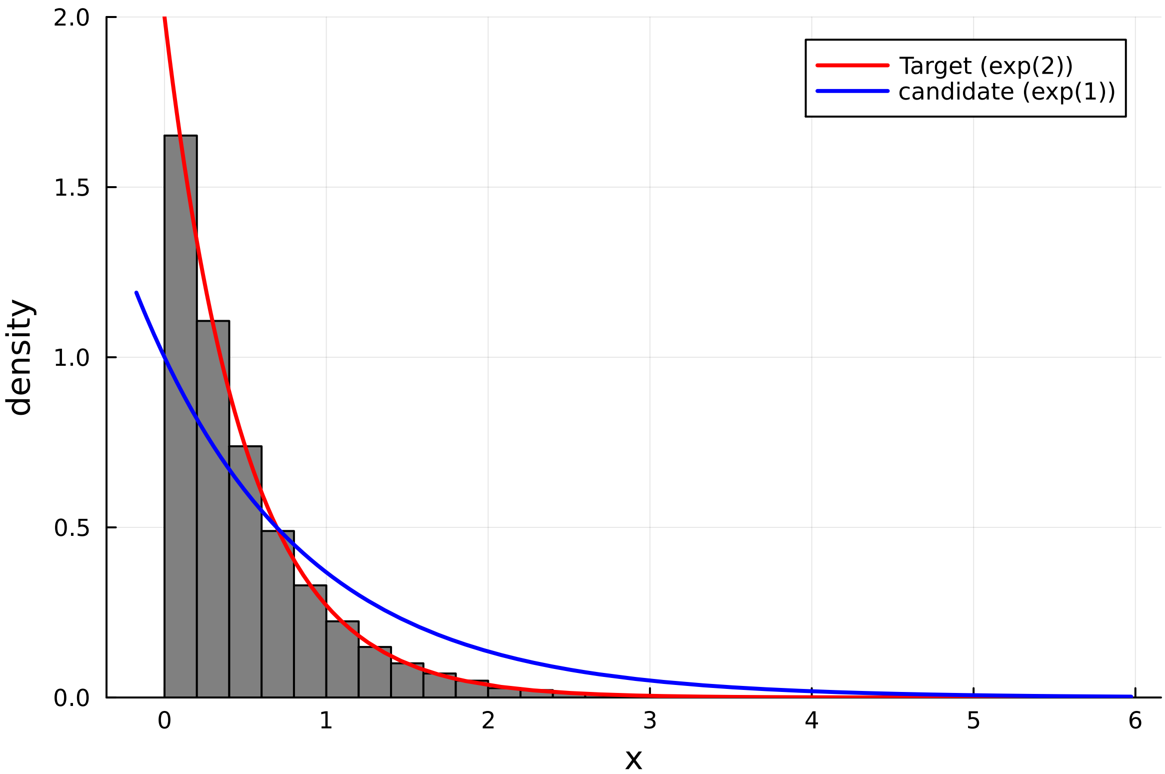


Figure 4: The blue line indicates target density which is exponential(2) and red line indicate candidate density exponential(1)

4.2. Example 2

Now suppose if we want to generate random numbers from the $\mathcal{N}(0, 1)$ distribution. First, we generate $U \sim \text{uniform}(0, 1)$ and also $Y \sim \text{Cauchy}(0, 1)$. Then calculate M as,

$$M = \sup_{t \in \mathbb{R}} \frac{f(t)}{g(t)}$$

$$= \sup_{t \in \mathbb{R}} \frac{\frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}}}{\frac{1}{\pi} \frac{1}{1+t^2}}$$

$$= \sup_{t \in \mathbb{R}} \frac{\pi}{\sqrt{2\pi}} (1+t^2) e^{-\frac{t^2}{2}}.$$

In [12]:
```
using Plots, Statistics, StatsBase, LaTeXStrings, Distributions
using Roots,Symbolics
```

In [13]:
```
g(x) = (1/pi)*(1/(1+x.^2))
f(x) = (1/sqrt(2*pi))*exp.(-x.^2/2)
plot(x->(pi/sqrt(2*pi))*(1+x.^2)*exp(-x.^2/2),-6,6, lw = 2,
    color = "red", label = L"\frac{f(x)}{g(x)}")
# symbolic computation can be done using Julia
# this can be done using Symbolics package or SymPy Package
@variables x
F_sym = (pi/sqrt(2*pi)) * (1 + x^2) * exp(-x^2/2)
dF_sym = Symbolics.derivative(F_sym, x)
println("dF(x): ", dF_sym)
```

```
dF(x): 2.5066282746310007x*exp((-1//2)*(x^2)) - 1.2533141373155003x*(1 + x^2)*exp((-
1//2)*(x^2))
```

In [14]:
```
# now define the functionloc
function diffe_F(x)
    h = (2.5066282746310007x*exp((-1//2)*(x^2))) -
         (1.2533141373155003x*(1 + x^2)*exp((-1//2)*(x^2)))
    return h
end
```

Out[14]: `diffe_F (generic function with 1 method)`

In [15]:
```
# we can find the root of the equation by using Roots package
roots = find_zeros(diffe_F, -4, 4)
println("Roots = ",roots)
```

```
Roots = [-1.0, 0.0, 1.0]
```

In [16]:
```
F(x) = (pi/sqrt(2*pi)) * (1 + x.^2) * exp.(-x.^2/2)
p1 = plot(x->(pi/sqrt(2*pi))*(1+x.^2)*exp(-x.^2/2),-6,6, lw = 2,
    color = "red", label = L"\frac{f(x)}{g(x)}")
scatter!([-1,1],[F(-1),F(1)], color = "blue", markersize = 5,label = "" )
M = F(-1)
N = 100000
u = rand(Uniform(0,1), N)
y = rand(Cauchy(0,1), N)
ind = u .< (1 / M) .* f.(y) ./ g.(y)
x = y[ind]
println("The number of accepted samples is :" , length(x))
```

```
The number of accepted samples is :65767
```

In [17]:
```
p2 = histogram(x, normalize = true, color = "lightgrey", label = "",
        xlabel = "x", ylabel = "density")
x_vals = range(minimum(x), maximum(x), length=1000)
pdf_norm = pdf.(Normal(0,1),x_vals)
pdf_cauchy = pdf.(Cauchy(0,1),x_vals)
plot!(x_vals,pdf_norm, color = "red", lw = 2, label = "N(0,1)")
plot!(x_vals,pdf_cauchy, color = "blue", lw = 2, label = "C(0,1)")
```

```
plot(p1,p2, layout = (1,2),  size=(800, 300))
```

Out[17]:

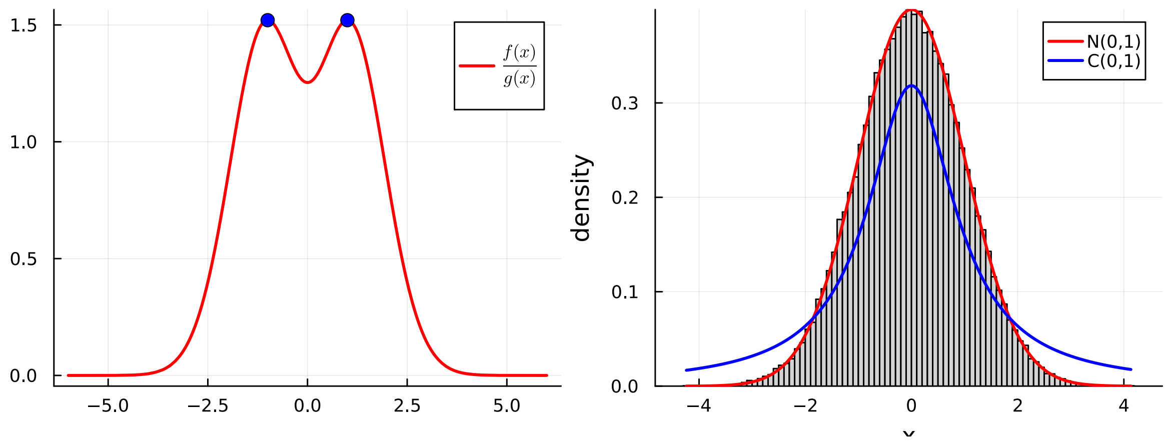


Figure 5: The left panel describes the shape of the function $\frac{f(t)}{g(t)}$ and it is clearly visible that the maximum is attained at two points. The best choice of c, that is M, is shown using blue dots. In the right panel, candidate and target densities are shown. The histogram of the simulated values is a very good approximation to the target density $\mathcal{N}(0,1)$.

4.3. Example 3

Suppose it is desired to generate $X \sim \text{beta}(a,b)$ where a and b are not integers. If the parameters are integers, we can simulate them by using transformation starting with uniform random numbers from (0,1). Therefore, we consider beta density with integer a and b as the candidate density. In the following code, we perform an experiment to simulate $\text{beta}(a,b)$ random variables with uniform(0,1) as the candidate density. Suppose that we consider $c = 4$. The following simulation shows the proportion of acceptance when $c = 4$ and candidate density is uniform(0,1).

In [18]:

```
using Plots, Statistics, StatsBase, LaTeXStrings, Distributions
using Roots,Symbolics, SpecialFunctions
```

```
WARNING: using SpecialFunctions.beta in module Main conflicts with an existing identif
ier.
```

In [19]:

```
using Distributions, Plots, SpecialFunctions

# Define Beta function manually
BetaFunction(a, b) = gamma(a) * gamma(b) / gamma(a + b)
a = 2.7  # Shape 1 parameter
b = 6.3  # Shape 2 parameter
target_density(x) = (x^(a-1) * (1-x)^(b-1)) / BetaFunction(a, b)

c = 4  # Bigger than optimal c
p1 = plot(target_density, 0, 1, color="red", lw=2,
          ylabel=L"f(x)", label="", ylims=(0,5))
hline!([c], color="blue", lw=3, linestyle=:dash, label="")
annotate!(0.1, 4.4, "c = 4")

# Accept-Reject Algorithm
m = 1000  # Number of simulations
u = rand(Uniform(0,1), m)
y = rand(Uniform(0,1), m)
ind = u .< (1 / c) .* target_density.(y) ./ pdf(Uniform(0,1), y)  # Proposal PDF is 1
```

```
x = y[ind]

p2 = histogram(x, normalize=:pdf, color="grey", bins=30,
               xlabel=L"x", ylabel="Density", label="")
plot!(target_density, 0, 1, color="red", lw=2, label="")


plot(p1, p2, layout=(1,2), size=(800, 300))
```

Out[19]:

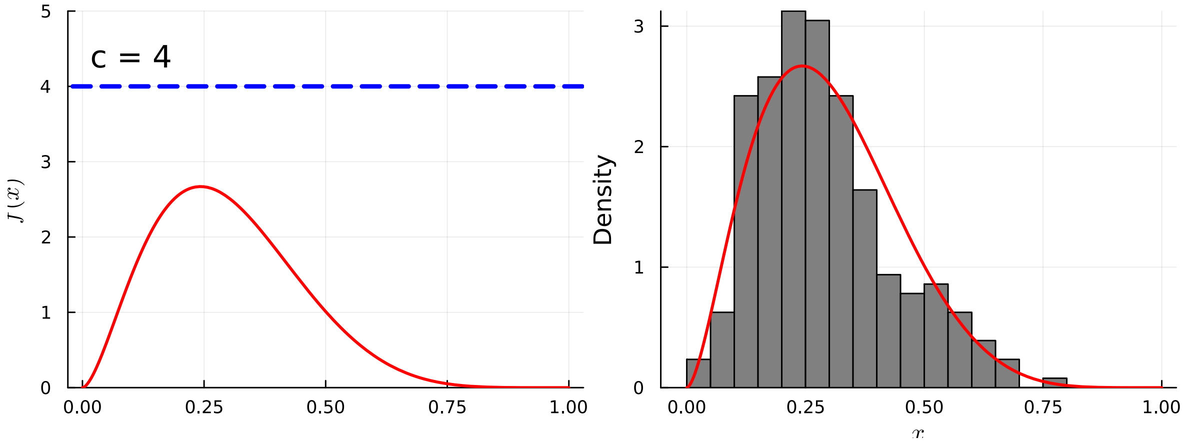


Figure 6: Target density in $uniform(0,1)$ and the choice of $c = 4$. In fact any value of c which is bigger than the maximum value of $f(x)$ will work

In [20]:
```
println("proportion of acceptance when we use uniform(0,1) as candidate density: ",
        sum(ind)/m)
```

```
proportion of acceptance when we use uniform(0,1) as candidate density: 0.256
```

In the following, let us now compute the optimal choice of c keeping $uniform(0,1)$ as the candidate density. This is equivalent to finding the maximum value of the target density $f(x)$.

In [21]:
```
@variables x
f_s = x^(a-1)*(1-x)^(b-1) # f(x)
df = Symbolics.derivative(f_s,x) # symbolic derivative
```

Out[21]:

$$-5.3(1-x)^{4.3}x^{1.7} + 1.7(1-x)^{5.3}x^{0.7} \tag{1}$$

In [22]:
```
println("Derivative of f(x) = ", df)
```

```
Derivative of f(x) = -5.3(x^1.7000000000000002)*((1 - x)^4.3) + 1.7000000000000002(x^
0.7000000000000002)*((1 - x)^5.3)
```

In [23]:
```
function diff_f(x)
    h = - 5.3*(x^1.7)*((1 - x)^4.3) + 1.7*(x^0.7)*((1 - x)^5.3)
    return h
end

p1 = plot(diff_f,color="blue", lw=2, label="", xlabel = L"x",
      ylabel=L"\nabla f(x)")
hline!([0], lw = 2, label = "")
root = find_zeros(diff_f, 0.2,0.6)
scatter!([root],[0], color = "red", markersize = 5, label = "")
println("The roots of f'(x) = 0 are"," ", root )
```

```
The roots of f'(x) = 0 are [0.24285714285714285]
```

In [24]:
```
optimal_c = target_density.(root)
println("The optimal choice of c is ", optimal_c)
```

```
The optimal choice of c is [2.6697440111492075]
```

In [25]:
```
println(" Let us perform the sampling with the best choice of c")
```

```
 Let us perform the sampling with the best choice of c
```

In [26]:
```
p2 = plot(target_density,0,1, ylims = (0,4), lw = 2,
     color = "red", ylabel = L"f(x)", label = "")
hline!([optimal_c], lw = 3, color = "blue",
        linestyle = :dash, label = "")
annotate!([0.4], 3, "c = 2.669744")

m = 1000 # no of simulations
u = rand(Uniform(0,1), m)
y = rand(Uniform(0,1), m)
ind =  u .< (1 / optimal_c) .* target_density.(y)
x = y[ind]
p3  = histogram(x, normalize = true, color = "grey", bins = 30,
         xlabel = L"x", ylabel = "density", label = "")
plot!(target_density, 0,1, color = "red", lw = 2, label = L"\beta(2.7,6.3)")
plot(p1,p2,p3, layout = (2,2),  size=(800, 600))
```

Out[26]:

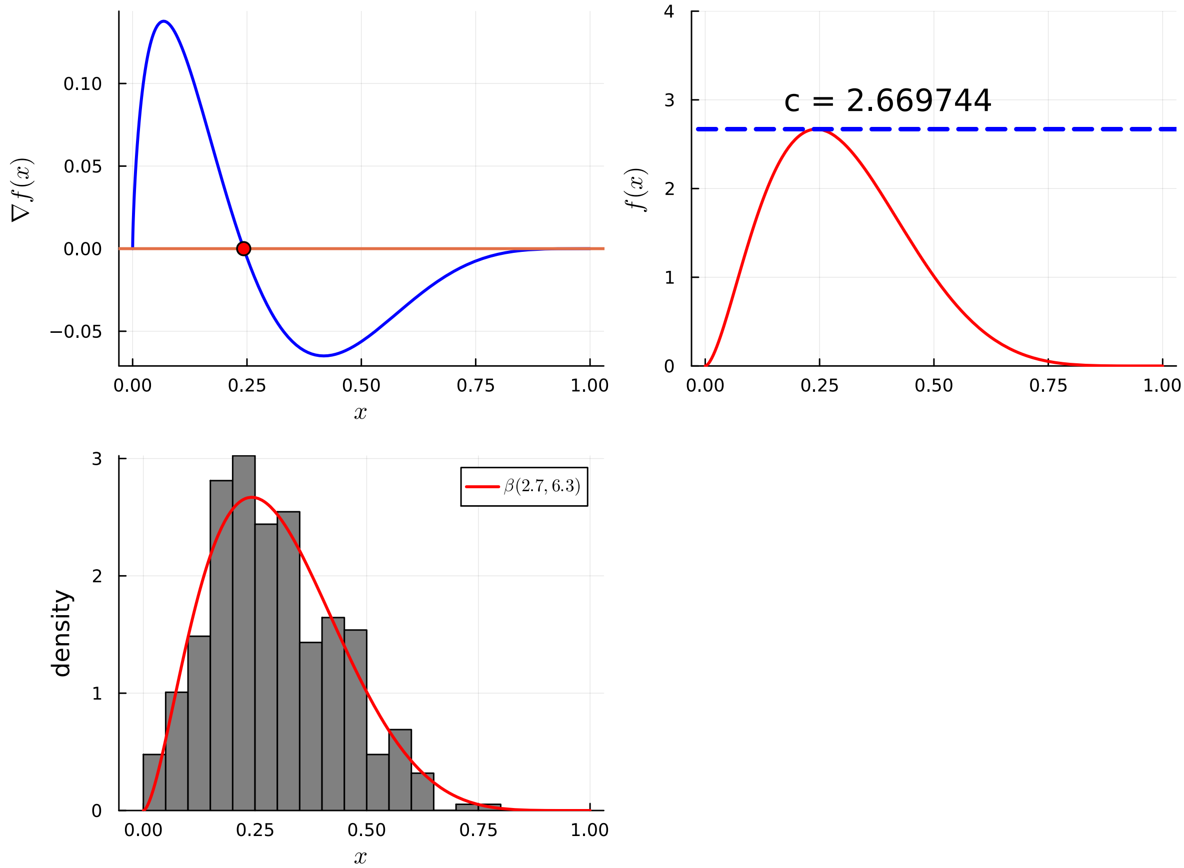


Figure 7: (a) Plot of the function f'. There is a unique maximum in $(0, 1)$, which can be obtained by differentiating and solving $f'(x) = 0$. (c) Histogram of the accepted values simulated from the candidate density. Candidate density and the target density, $\mathrm{Beta}(2, 6)$ and $\mathrm{Beta}(2.7, 6.3)$, respectively, are overlaid on the same plot.

In [27]:
```
println("proportion of acceptance when we use uniform(0,1) as candidate density ", su
```

```
proportion of acceptance when we use uniform(0,1) as candidate density 0.377
```

You are encouraged to run the above program with the best choice of c and check the average number of acceptance based on several replications. Even with the optimal choice of c, the probability of acceptance remains pretty small. Note that in both the above simulation experiments, we used $\text{uniform}(0,1)$ as the candidate density function. In the following, we consider beta density as the candidate density. Recall that by using transformation we can simulate $\text{Beta}(a,b)$ if a and b are positive integers. Therefore, it is natural to ask for non-integer values of a and b. What would a good choice of a candidate density which belongs to the beta family only but with integer shape parameters? The following facts will help us to choose a nice candidate beta density.

- (a) First verify that $Y \sim \text{Beta}([a],[b])$ will result in a finite value of $M = \sup_t \frac{f(t)}{g(t)}$
- (b) Also, it can be shown that using $Y \sim \text{Gamma}([a], b)$ will result in a finite value of $M = \sup_t \frac{f(t)}{g(t)}$
- (c) However, in each of parts (a) and (b), if Y had parameter $[a]+1$, then M would be infinite.
- (d) In each of parts (a) and (b) find optimal values of the parameters of Y in the sense of minimizing $E(N)$,where N is the number of (U,Y) pairs required for one X

In [28]:
```
using Plots, Statistics, StatsBase, LaTeXStrings, Distributions
using Roots,Symbolics, SpecialFunctions
```

In [29]:
```
using SpecialFunctions, Plots  # Ensure SpecialFunctions for beta()

a = 2.7  # shape 1
b = 6.3  # shape 2
BetaFunction(a, b) = gamma(a) * gamma(b) / gamma(a + b)

# Target density function
function f(x)
    (x^(a-1) * (1-x)^(b-1) / BetaFunction(a, b)) * (x > 0) * (x < 1)
end

# Candidate density function
function g(x)
    (x^(floor(a)-1) * (1-x)^(floor(b)-1) /
        BetaFunction(floor(a), floor(b))) * (x > 0) * (x < 1)
end

# Ratio function
function psi(x)
    return f(x) / g(x)
end


p1 = plot(psi, 0, 1, color = "red", lw = 2, ylabel = L"\psi(x)", label = "")
box_a = floor(Int, a)
box_b = floor(Int, b)

@variables x
expression_psi = x^(a-1) * (1-x)^(b-1) / (x^(box_a-1) * (1-x)^(box_b-1))
```

Out[29]:

$$x^{0.7}(1-x)^{0.3} \tag{2}$$

In [30]:
```
d_psi = Symbolics.derivative(expression_psi, x)
```

Out[30]:

$$\frac{-0.3x^{0.7}}{(1-x)^{0.7}} + \frac{0.7(1-x)^{0.3}}{x^{0.3}} \tag{3}$$

In [31]:
```
println("derivative: ", d_psi)
```

```
derivative: (-0.29999999999999998(x^0.7000000000000002)) / ((1 - x)^0.7000000000000002)
+ (0.7000000000000002((1 - x)^0.29999999999999998)) / (x^0.29999999999999998)
```

In [32]:
```
function diff_psi(x)
    (-0.3*(x^0.7)) / ((1 - x)^0.7) +
    (0.7*((1 - x)^0.3)) / (x^0.23)
end

p2 = plot(diff_psi, color = "blue", ylims = (-8,3), lw = 2,
     ylabel = L"\nabla{f(x)}", label = "")
hline!([0], color = "green", lw = 2, label = "")
root = find_zeros(diff_psi, 0.4, 0.8)
scatter!([root],[0], color = "red", label = "root of diff_psi")
println("The root of derivative of f(x)/g(x) is: ", root)
```

```
The root of derivative of f(x)/g(x) is: [0.6946162079404271]
```

In [33]:
```
M = psi.(root) # best choice of c
println("Probability of acceptance for the best choice of c is: ",(1/M))
```

```
Probability of acceptance for the best choice of c is: [0.5981958978922645;;]
```

In [34]:
```
m = 10000
y = zeros(m)
for i in 1:m
    v = rand(Uniform(0,1), Int(box_a + box_b))
    y[i] = sum(-log.(1 .- v[1:Int(box_a)])) / sum(-log.(1 .- v))
end

u = rand(Uniform(0,1), m)
ind =  u .< (1 /M) .* f.(y)./g.(y)
x = y[ind]
p3 = histogram(x , normalize = true, bins = 30, ylims = (0,4),
        xlabel = "x", ylabel = "density", label = "")
plot!(g, color = "red", lw = 2, label = L"β(2,6)")
plot!(f, color = "blue", lw = 2, label = L"β(2.7,6.3)")
println("Proportion of acceptance when we use beta(2,6) as candidate density: ",
         sum(ind)/m)
```

```
Proportion of acceptance when we use beta(2,6) as candidate density: 0.5991
```

In [35]:
```
plot(p1,p2,p3, layout = (2,2),  size=(800, 600))
```

Out[35]:

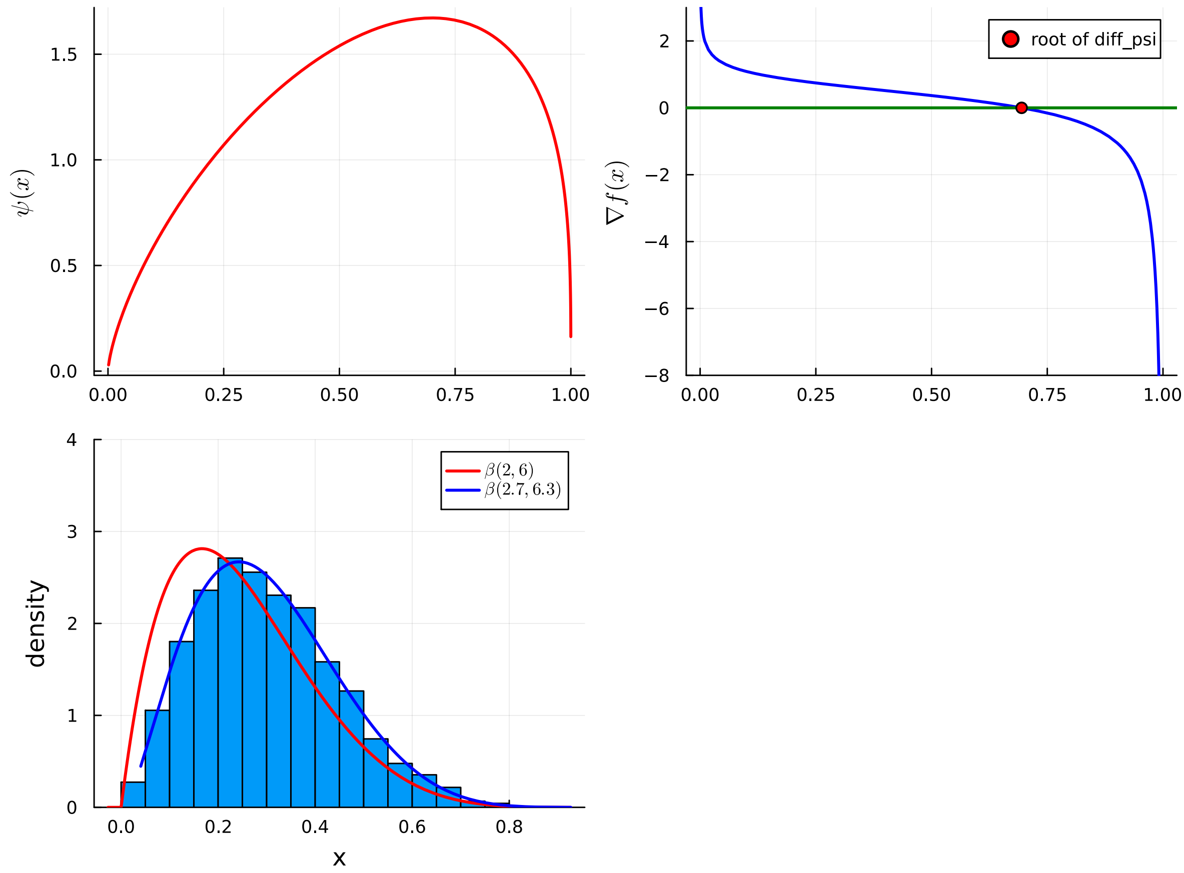


Figure 8: (a) Plot of the function $\psi(x) = \frac{f(x)}{g(x)}$. There is a unique maximum in (0,1), which can be obtained by differentiating and solving $\psi'(x) = 0$. (c) Histogram of the accepted values simulated from the candidate density. Candidate density and the target density, beta(2,6) and beta(2.7, 6.3), respectively, are overlaid on the same plot.

5. Simulation using slicse sampling

Suppose that we are interested to generate random number from the probability density function $f(x)$. In slice sampling first we identify an interval where $f(x) > 0$. Let $[a, b]$ be the interval, we randomly generate a number from $[a, b]$ uniformly, say x_1, then generate y_1 uniformly from $[0, f(x_1)]$. Basically, y_1 gives a horizontal line that passes through the density $f(x)$ cutting into the pieces $\{y > f(x)\}$ and $\{y < f(x)\}$. We consider the pre-image $f^{-1}[y, \infty]$ and generate x_2 uniformly from this pre-image. Again we simulate $y_2 \sim U[0, f(x_2)]$ and the process continues. In the following, we consider two examples. For the first example, the explicit mathematical form of $f^{-1}[y, \infty]$ is available. In the second example, we consider a situation where the inverse image can be obtained explicitly. We shall see how \textbf{do - while} loop can be effectively used to perform the simulation efficiently.

5.1. Example 1

To demonstrate slice sampling let the random variable $X \sim \mathcal{N}(3, 1)$, that is, the density function of X is given by

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-3)^2}{2\cdot 1}}, \quad -\infty < x < \infty.$$

Let $a = 2$ and $b = 8$. $f^{-1}[y, \infty] = \left(3 \pm \sqrt{-\log(\sqrt{2\pi y})}\right)$. We simulate 1000 random number using this algorithm and histogram shown in the following figure. The histogram accurately approximate the $\mathcal{N}(3,1)$ density function. The R code is given below.

In [36]:
```
using Plots, Statistics, StatsBase, LaTeXStrings, Distributions
```

In [37]:
```
using Random, Distributions, Plots

mu = 3  # population mean
sigma = 1  # population standard deviation

function f(x)
    (1/(sqrt(2*pi*sigma^2)))*exp.(-(x.-mu)^2/(2*sigma^2))
end

a, b = 2, 8
n = 1000
x = Vector{Float64}(undef, n)
for i in 1:n
    if i == 1
        x[i] = rand(Uniform(a, b))  # First step initialization
    else
        y = rand(Uniform(0, f(x[i-1])))
        x[i] = rand(Uniform(mu - sqrt(-2 * log(sqrt(2 * pi) * y)),
                            mu + sqrt(-2 * log(sqrt(2 * pi) * y))))
    end
end

histogram(x, normalize = true, color=:grey, bins=40, label="")
plot!(f,color = "red", lw = 2, label = "N(3,1)")
```

Out[37]:

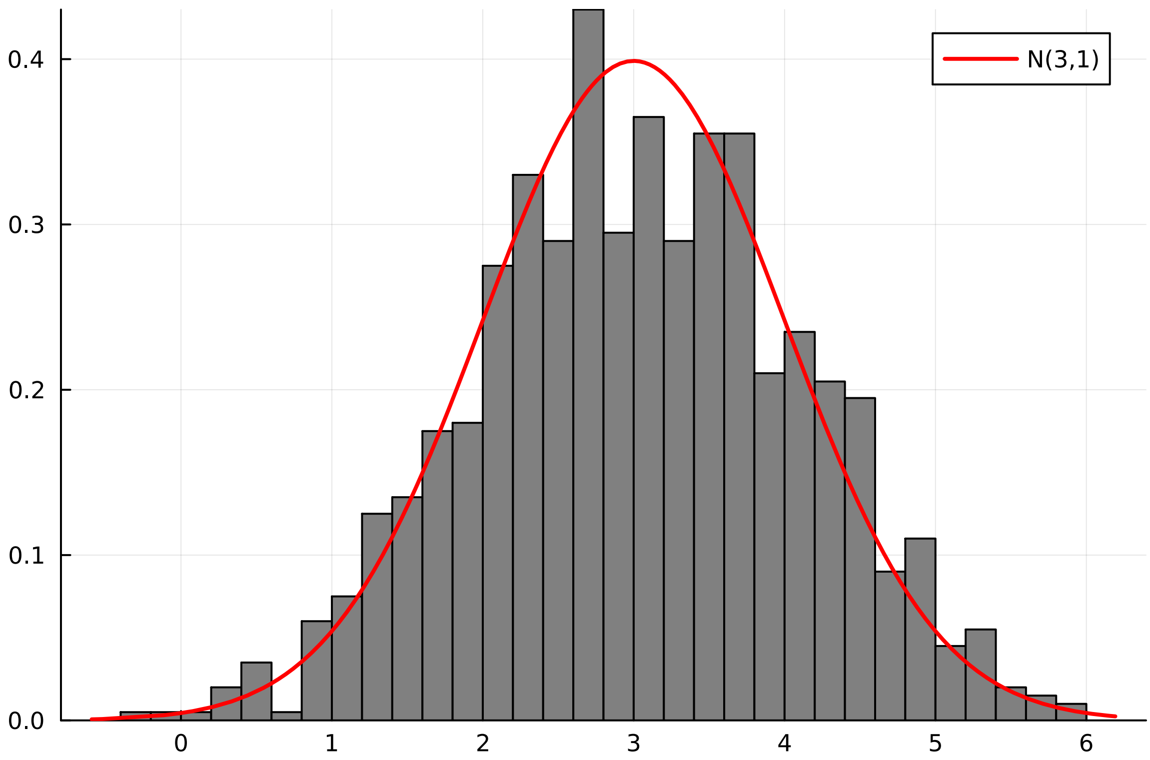


Figure 9: Simulation of the normal random variables using the slice sampling

5.2. Example 2

Now suppose that we want to simulate random numbers from a density function for which the set $f^{-1}(y, \infty)$ cannot be obtained analytically. In that case, we shall generate z_i from (a, b) uniformly, but it will be considered as a realization from $f(x)$ only if $y < f(z)$ is valid. For demonstration, we consider a mixture of two normal density functions as

$$f(x) = 0.4\mathcal{N}(0, 1) + 0.6\mathcal{N}(4, 1).$$

The histogram of the simulated realization clearly approximates the density (f(x)) closely.

In [38]:

```
function f(x)
    0.4 * pdf.(Normal(0, 1), x) + 0.6 * pdf.(Normal(4, 1), x)
end

# Parameters
n = 10000  # Number of samples
a, b = -6, 12  # Sample space of X

# Allocate space
x = Vector{Float64}(undef, n)

# Simulate first value
x[1] = rand(Uniform(a, b))

# Generate samples
for i in 2:n
    y = rand(Uniform(0, f(x[i-1])))
    z = rand(Uniform(a, b))
    if y <= f(z)
        x[i] = z
    else
        while y > f(z)
            z = rand(Uniform(a, b))
            if y <= f(z)
                x[i] = z
                break
            end
        end
    end
end


histogram(x, normalize = true, bins=30, label="")
plot!(f, a, b, linewidth=2, color=:red, label="")
```

Out[38]:

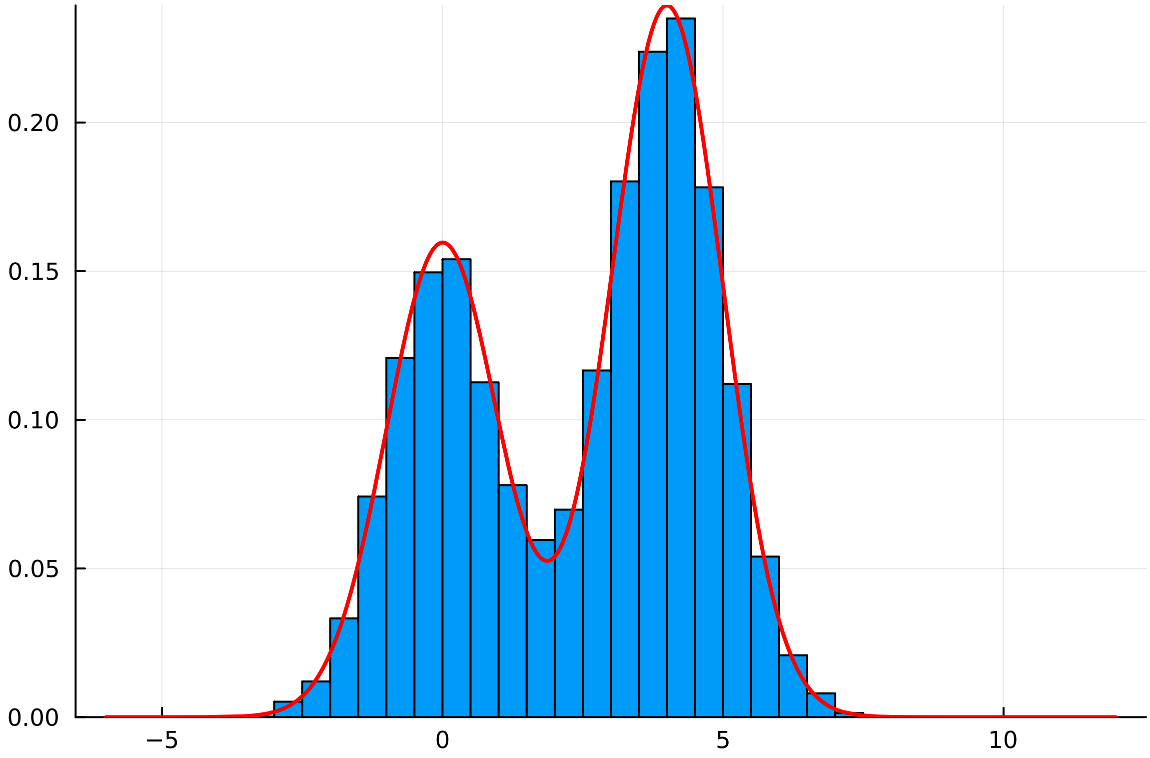


Figure 10 : Demonstration of the slice sampling where the explicit computation of the inverse image is not possible. The simulation is carried out from the mixture of two normal distributions. 10000 samples have been generated

From the second example, it is clear that if we know the density function $f(x)$ (it may be very complicated), we can use slice sampling to simulate from $f(x)$. Another interesting fact about the slice sampling is that it requires only simulation from the uniform distribution, irrespective of the mathematical form of $f(x)$

6. Grid Approximation to Continuous density function

In [39]:

```
using Plots, Statistics, StatsBase, LaTeXStrings, Distributions, Random
```

In [40]:

```
f(x) = (1 / sqrt(2 * pi)) * exp(-x^2 / 2)
g(x) = exp(-x^2 / 2)  # Use g for simulation

grid_vals = collect(-4:0.1:4)
grid_sum = sum(g(x) for x in grid_vals)
grid_prob = [g(x) / grid_sum for x in grid_vals]


p1 = plot(grid_vals, grid_prob, seriestype=:stem, color=:red,
    ylim=(0, 0.4), label="")
plot!(grid_vals, f.(grid_vals), color=:magenta, linewidth=2,
    label="")

x_samples = sample(grid_vals, Weights(grid_prob), 1000)
p2 = histogram(x_samples, normalize = true, bins=20, alpha=0.5,
    label="")
plot!(f, color = "magenta", lw = 2, label = "")

plot(p1,p2, layout = (1,2),  size=(800, 400))
```

Out[40]:

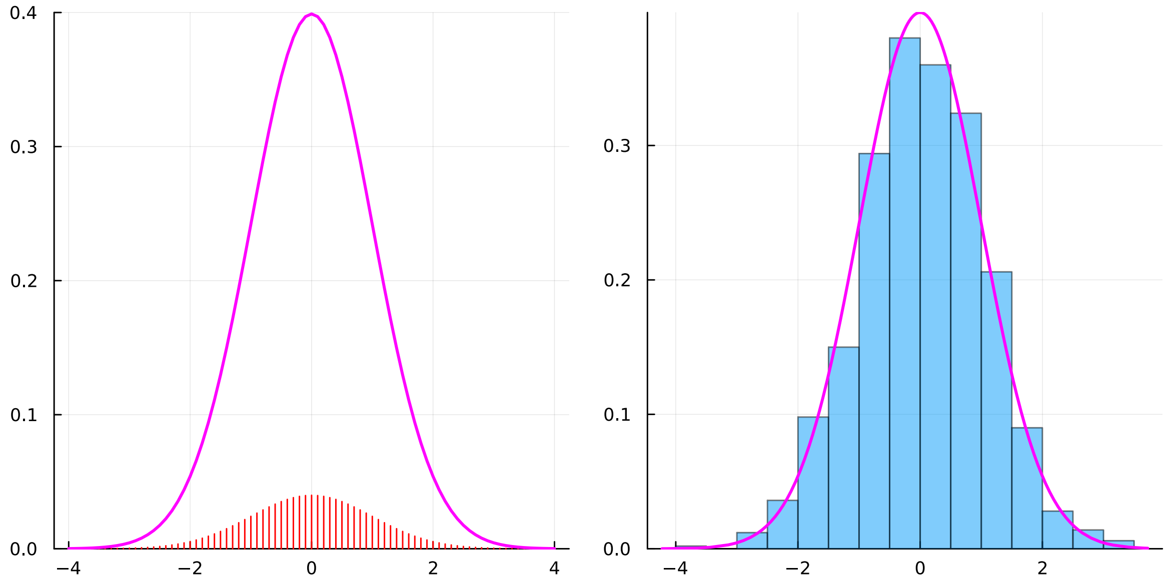


Figure 11: Simulation from the standard normal density function using the grid approximation. The function $g(x)$ is the kernel of the normal density $f(x)$. Basically f is known only to a proportionality constant to g

7. References

- George Casella, Roger L. Berger, Statistical Inference, Second Edition, Duxbury Advanced Learning, India Edition (2011)