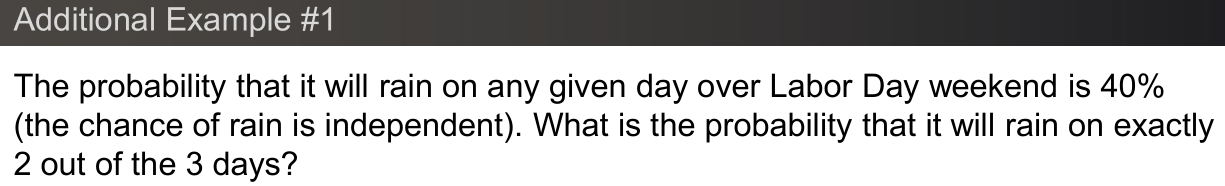
# Additional Example #1 M2L5 M2L12 6:20

Topics: Binomial Probability



## [Video](https://gtvault.sharepoint.com/:v:/s/ISyE6644OMSATAs/EWCikD5ay3dOiEiln66g4CoBRz4Z0bDUPw0a9Mq_pLbyWA?e=ZmA3dz)

## [Slides](https://gtvault.sharepoint.com/:b:/s/ISyE6644OMSATAs/EbIxNu-a9dVFpchNdFHNokUBmXYkKxaDw7GT59BT7YCmeg?e=wDKSgB)

## R Code

This code is performing a simulation in R to estimate the probability of it raining over a three-day weekend. The random variable of interest is the number of days it rains, which can take on values 0, 1, 2, or 3.

The set.seed(6644) function is used to set the seed of R’s random number generator, which is useful for creating simulations or random objects that can be reproduced.

The function generate\_weekend() simulates one three-day weekend. It uses the sample() function to randomly select "Rain" or "No Rain" for each of the three days, with a probability of 0.4 for "Rain" and 0.6 for "No Rain". The function then returns the number of days it rained.

The replicate() function is then used to repeat this process n times, where n is 1e5 (or 100,000). The results are stored in the variable days.

A barplot is generated to visualize the distribution of days.

The function prop.table(table(days)) generates a proportion table of the number of days it rained. This is essentially the empirical probability distribution of the number of rainy days over a three-day weekend, as estimated from the simulation.

Finally, the actual probabilities are calculated using the dbinom() function, which gives the probability mass function of a binomial random variable. Here, the size argument is 3 (the number of trials or days), and the prob argument is 0.4 (the probability of it raining on any given day).

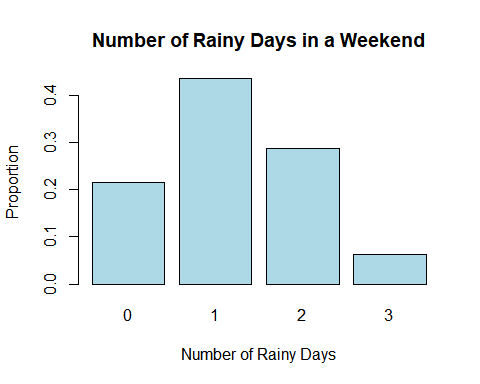
*# Additional Example #1*  
set.seed(6644)  
n <- 1e5  
  
generate\_weekend <- **function**(verbose = FALSE) {  
 *# Create a vector with 3 days of outcomes*  
 *# P(Rain) = 0.4, P(No Rain) = 0.6*  
 weekend <- sample(c("Rain", "No Rain"), size = 3, replace = TRUE,  
 prob = c(0.4, 0.6))  
 **if** (verbose) print(weekend)  
 *# Return how many days were Rain*  
 sum(weekend == "Rain")  
}  
  
generate\_weekend(verbose = TRUE)

## [1] "No Rain" "No Rain" "Rain"

## [1] 1

set.seed(6644)  
  
*# Generate n trials/samples*  
days <- replicate(n, generate\_weekend())

barplot(prop.table(table(days)), col = c("lightblue"),   
 main = "Number of Rainy Days in a Weekend",  
 xlab = "Number of Rainy Days", ylab = "Proportion")



*# Simulation Solution*  
prop.table(table(days))

## days  
## 0 1 2 3   
## 0.21450 0.43523 0.28705 0.06322

*# Actual Solution*  
dbinom(0:3, size = 3, prob = 0.4)

## [1] 0.216 0.432 0.288 0.064

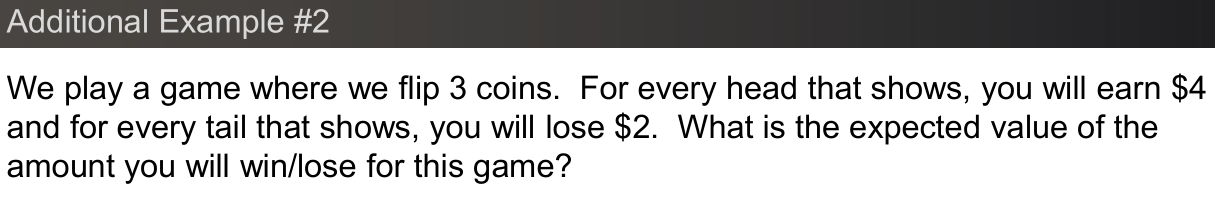
## [1] 0.216 0.432 0.288 0.064

## Python Code

[Python Notebook](https://colab.research.google.com/drive/1dz_Yy7u9xt1IN2Uwb3YCYvL7I7mc6Ml8#scrollTo=EPYDU5vWR4I6)

# Additional Example #2 M2L7 M2L8 22:12

Topics: Functions of a Random Variable, Expected Value, Variance, Moment Generating Functions



## [Video](https://gtvault.sharepoint.com/:v:/s/ISyE6644OMSATAs/EYLS6NtkYEFMnYeRm6yv3w0BOITx2ZJWRqfchi0tGWagjQ?e=09L0zP)

## [Slides](https://gtvault.sharepoint.com/:b:/s/ISyE6644OMSATAs/EShjNcsIOOJGhOaxV4aCRGcBfqrkSbgUI93erPAGgPA_jg?e=Qp2X8U)

## R Code

This R script is performing a simulation to estimate the probability distribution of the number of heads in three tosses of a fair coin. It also calculates the mean and variance of the number of heads, and transforms the number of heads by a linear transformation, then calculates the mean and variance of the transformed values.

First, it generates all possible sequences of heads (H) and tails (T) in three coin tosses, and calculates the number of heads in each sequence.

Then it defines a function generate\_tosses() that simulates n\_tosses tosses of a fair coin, and tests this function.

After setting the seed for reproducibility, it simulates a large number (n) of trials of three coin tosses each, and calculates the number of heads in each trial.

It then computes the empirical probability distribution of the number of heads, and compares this with the theoretical distribution, given by the binomial distribution with parameters n\_tosses (number of trials) and prob = 0.5 (probability of getting heads in one toss).

It defines a new random variable Y which is a linear transformation of the number of heads X, and computes the empirical distribution of Y.

Finally, it calculates the mean and variance of X and Y. According to the properties of the mean and variance, the mean of Y should be 6\*mean(X) - 6, and the variance of Y should be 6^2 \* var(X) (since the variance of a random variable is multiplied by the square of the scaling factor when the variable is scaled).

The code is demonstrating and verifying these properties of the mean and variance through simulation. By comparing the empirical results with the theoretical results, we can see how close the simulation comes to the true probabilities and statistics.

*# Additional Example #2*  
set.seed(6644)  
n <- 1e5  
n\_tosses <- 3  
  
*# Create all combinations of heads and tails in n\_tosses*  
coin <- c("H", "T")  
coins <- expand.grid(rep(list(coin), n\_tosses))  
sequences <- apply(coins, 1, **function**(x) paste(as.character(x), collapse = ""))  
num\_heads <- sapply(sequences, **function**(x) sum(charToRaw(x) == charToRaw("H")))  
num\_heads

## HHH THH HTH TTH HHT THT HTT TTT   
## 3 2 2 1 2 1 1 0

generate\_tosses <- **function**(n\_tosses) {  
 paste(sample(c("H", "T"), size = n\_tosses, replace = TRUE), collapse = "")  
}  
  
generate\_tosses(n\_tosses)

## [1] "HHH"

generate\_tosses(n\_tosses)

## [1] "THT"

set.seed(6644)  
trials <- replicate(n, generate\_tosses(n\_tosses))  
head(trials)

## [1] "HHH" "THT" "HTT" "TTT" "HTT" "TTT"

X <- num\_heads[trials]  
head(X)

## HHH THT HTT TTT HTT TTT   
## 3 1 1 0 1 0

prop.table(table(X))

## X  
## 0 1 2 3   
## 0.12549 0.37317 0.37628 0.12506

dbinom(0:n\_tosses, size = n\_tosses, prob = 0.5)

## [1] 0.125 0.375 0.375 0.125

Y <- 6\*X - 6  
prop.table(table(Y))

## Y  
## -6 0 6 12   
## 0.12549 0.37317 0.37628 0.12506

mean(X)

## [1] 1.50091

6 \* mean(X) - 6

## [1] 3.00546

mean(Y)

## [1] 3.00546

var(X)

## [1] 0.7511067

6^2 \* var(X)

## [1] 27.03984

var(Y)

## [1] 27.03984

## Python Code

[Python Notebook](https://colab.research.google.com/drive/1dz_Yy7u9xt1IN2Uwb3YCYvL7I7mc6Ml8#scrollTo=7EbF9OCVR-EN)

# Additional Example #3 M2L6 M2L8 10:01

Topics: Functions of a Random Variable, Inverse Transform



## [Video](https://gtvault.sharepoint.com/:v:/s/ISyE6644OMSATAs/EVnYFTmU6-VOlz7QvFP3UqwBwTFKh7QxNcEDwX0KkNSIBA?e=V5UzA1)

## [Slides](https://gtvault.sharepoint.com/:b:/s/ISyE6644OMSATAs/Ed8phe2iaQVDk-DgQ2SJanoB1EHkIjeS3A9pJDeZTKdFlg?e=b38l7e)

## R Code

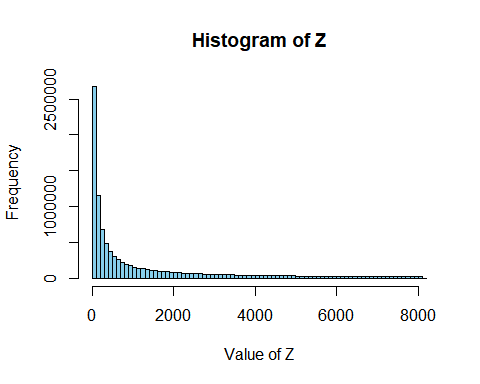
This R script is simulating and analyzing the distributions of two random variables, Z and Z\_IT.

Z is derived from a uniform random variable X that ranges from 1 to 3 by taking exp(3\*X). This is generating Z as a function of X.

Z\_IT is derived from another uniform random variable U (which ranges from 0 to 1 by default) by taking exp(6\*U+3). This is using inverse transform to generate Z directly from its CDF.

The purpose of the code is to compare these two derived variables. The code then creates histograms to visualize the distributions of Z and Z\_IT, and provides summary statistics for each variable.

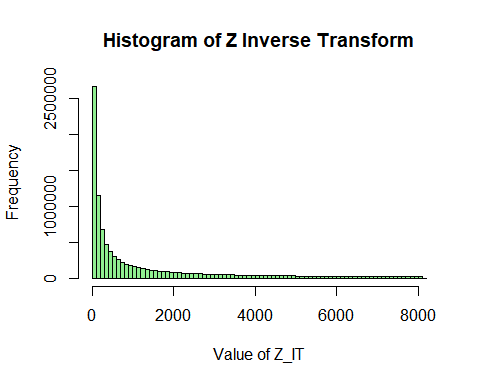
*# Additional Example #3*  
set.seed(6644)  
n <- 1e7  
  
*# Generate X*  
X <- runif(n, min = 1, max = 3)  
  
*# Z = e^(3X)*  
Z <- exp(3\*X)  
hist(Z, 100, col = "skyblue", main = "Histogram of Z",   
 xlab = "Value of Z", ylab = "Frequency")



summary(Z)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 20.09 89.93 402.96 1346.68 1806.96 8103.08

*# Generate Z Inverse Transform*  
U <- runif(n)  
  
*# Z = e^(6U+3)*  
Z\_IT <- exp(6\*U+3)  
hist(Z\_IT, 100, col = "lightgreen", main = "Histogram of Z Inverse Transform",   
 xlab = "Value of Z\_IT", ylab = "Frequency")



summary(Z\_IT)

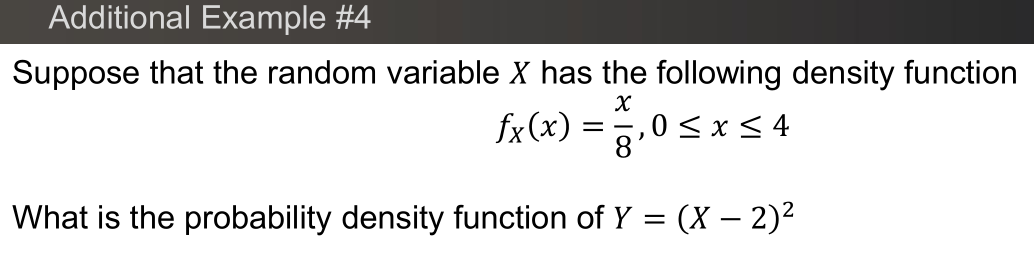
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 20.09 90.05 403.07 1346.36 1805.74 8103.08

## Python Code

[Python Notebook](https://colab.research.google.com/drive/1dz_Yy7u9xt1IN2Uwb3YCYvL7I7mc6Ml8#scrollTo=_DZuTAHXSAaS)

# Additional Example #4 M2L6 M2L8 9:30

Topics: Functions of a Random Variable, Inverse Transform



## [Video](https://gtvault.sharepoint.com/:v:/s/ISyE6644OMSATAs/EXFVsW29XlxFhg4ge0abvhgBB7gvLRH47YbzI-W-3FhNkg?e=a1nPXW)

## [Slides](https://gtvault.sharepoint.com/:b:/s/ISyE6644OMSATAs/EcV1Mf9Ydt1PlqONmv0tRxwBmc3ES9eeiPdNPhqp-wOMsQ?e=E7vLju)

## R Code

This R script is simulating and analyzing the distributions of two random variables, X and Y, and another random variable Y generated using an inverse transform method.

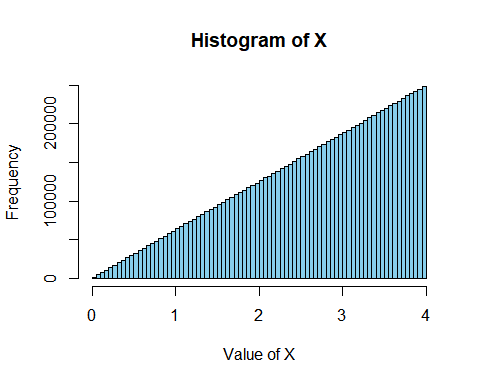
X is generated by taking the square root of U (which is uniformly distributed between 0 and 1) and multiplying the result by 4.

Y is then generated from X by subtracting 2 from X and squaring the result. The purpose of the code is likely to examine the distribution of Y, given the transformation (X-2)^2.

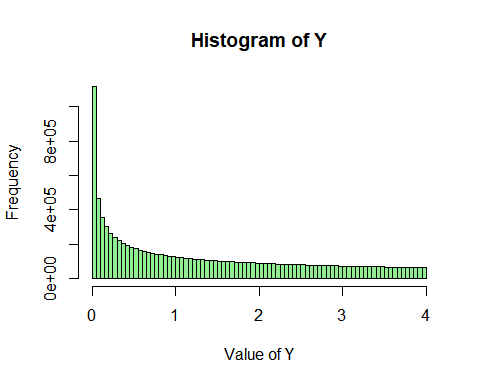
The script also creates another version of Y using the inverse transform method, where U is squared and then multiplied by 4.

The script then creates histograms to visualize the distributions of X, Y, and Y from the inverse transform method, and provides summary statistics for each of Y and Y from the inverse transform method.

*# Additional Example #4*  
set.seed(6644)  
n <- 1e7  
  
*# Generate X*  
U <- runif(n)  
X <- 4\*sqrt(U)  
hist(X, 100, col = "skyblue", main = "Histogram of X",   
 xlab = "Value of X", ylab = "Frequency")



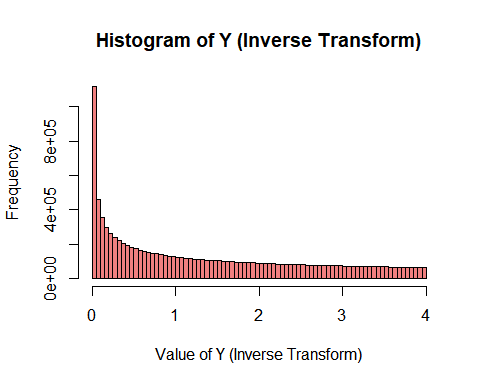
*# Y = (X-2)^2*  
Y <- (X-2)^2  
hist(Y, 100, col = "lightgreen", main = "Histogram of Y",   
 xlab = "Value of Y", ylab = "Frequency")



summary(Y)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.0000 0.2497 0.9989 1.3327 2.2490 4.0000

*# Generate Y Inverse Transform*  
U <- runif(n)  
Y <- 4\*U^2  
hist(Y, 100, col = "lightcoral", main = "Histogram of Y (Inverse Transform)",   
 xlab = "Value of Y (Inverse Transform)", ylab = "Frequency")



summary(Y)

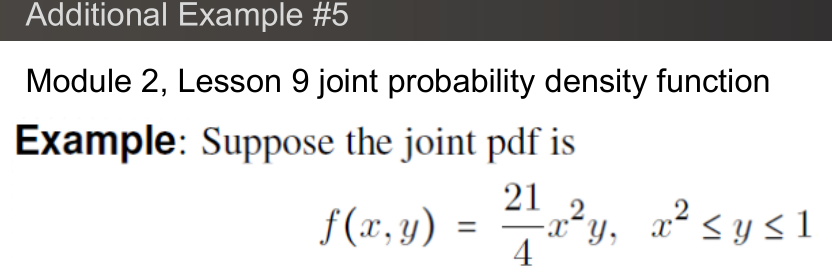
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.0000 0.2501 0.9994 1.3329 2.2487 4.0000

## Python Code

[Python Notebook](https://colab.research.google.com/drive/1dz_Yy7u9xt1IN2Uwb3YCYvL7I7mc6Ml8#scrollTo=v2ewqoecSDNj)

# Additional Example #5 M2L9 4:26

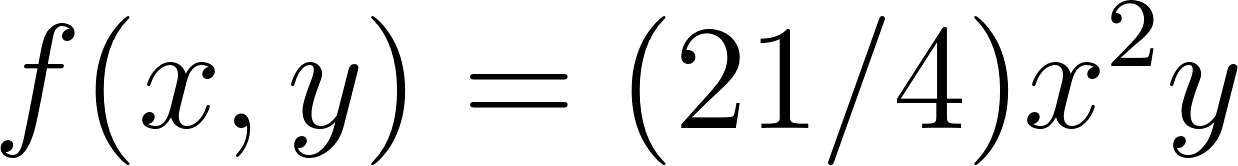
Topics: Jointly Distributed Random Variables, Double Integration



## [Video](https://gtvault.sharepoint.com/:v:/s/ISyE6644OMSATAs/EVny0SusWVxDjdM7rOuc_YABCT4fByuz784WnsCNzmok_A?e=UKjHuh)

## [Slides](https://gtvault.sharepoint.com/:b:/s/ISyE6644OMSATAs/EQtjCGgPk-dOidz_PdPMEEoBu2SoI4n3TwRiFkHq9NPFPQ?e=AH2eYZ)

## R Code

The code defines the joint probability density function (PDF) of two continuous random variables x and y as [](https://www.codecogs.com/eqnedit.php?latex=f(x%2C%20y)%20%3D%20(21%2F4)%20x%5E2%20y#0).

It then defines a sequence of x and y values and creates a grid of x and y values. The grid is used to compute the PDF values at each point using the with() function and the ifelse() statement.

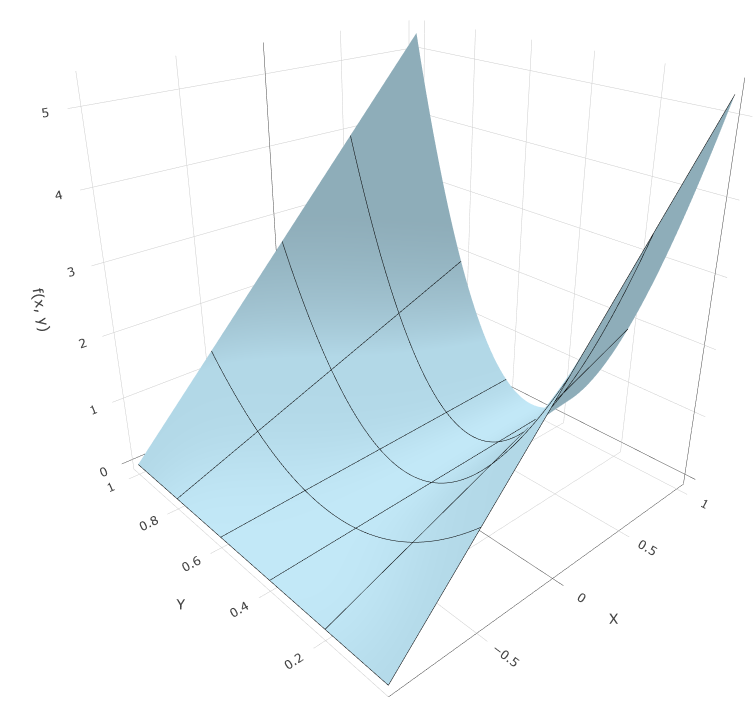
The grid is converted to a matrix format suitable for plotting using the matrix() function. The plot\_ly() function from the plotly package is used to create a 3D surface plot of the PDF, with a single color, no colorbar, and grid.

Next, the code creates a 3D perspective plot of the PDF using the persp() function in R. The plot is similar to the surface plot created with plot\_ly(), but is a static plot rather than an interactive plot.

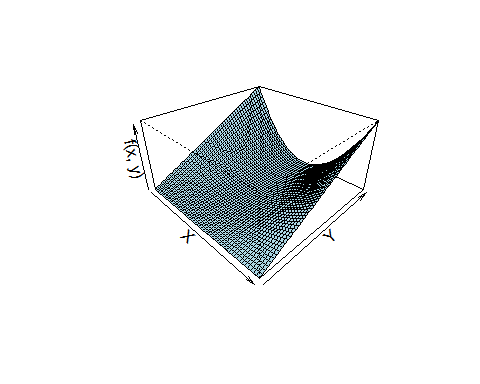
Finally, the code performs a double integration of the PDF over y for each x value using the integrate() function. The result of the integration is shown as output.

*# Additional Example #5*  
  
library(plotly)

*# Define the joint pdf*  
f <- **function**(x, y) {  
 21/4 \* x^2 \* y  
}  
  
*# Define a sequence of x and y values*  
x <- seq(-1, 1, length.out = 100)  
y <- seq(0, 1, length.out = 100)  
  
*# Create a grid of x and y values*  
grid <- expand.grid(X = x, Y = y)  
  
*# Compute the z values (pdf) at each point in the grid*  
grid$Z <- with(grid, ifelse(X >= -1 & X <= 1 & Y >= 0 & Y <= 1, f(X,Y), NA))  
  
*# Convert the data frame to a 3D array for plotting*  
z\_matrix <- with(grid, matrix(Z, length(y), length(x)))  
  
*# Create a 3D surface plot with single color, no colorbar, and grid*  
plot\_ly(x = ~x, y = ~y, z = ~z\_matrix) %>%  
 add\_surface(  
 colorscale = list(c(0, 1), c("lightblue", "lightblue")),  
 showscale = FALSE,  
 contours = list(  
 x = list(show = TRUE, color = "black"),  
 y = list(show = TRUE, color = "black")  
 )  
 ) %>%  
 layout(scene = list(xaxis = list(title = "X"),  
 yaxis = list(title = "Y"),  
 zaxis = list(title = "f(x, y)")))



*# Define a sequence of x and y values with less resolution for faster plotting*  
x <- seq(-1, 1, length.out = 50)  
y <- seq(0, 1, length.out = 50)  
  
*# Create the z-matrix using outer*  
z\_matrix <- outer(x, y, **function**(x, y) {  
 ifelse(x >= -1 & x <= 1 & y >= 0 & y <= 1, f(x,y), NA)   
 })  
  
*# Create a 3D perspective plot*  
persp(x, y, z\_matrix, theta = 45, phi = 30, expand = 0.5, col = "lightblue",   
 xlab = "X", ylab = "Y", zlab = "f(x, y)")



*# Perform the double integration dydx*  
result <- integrate(**function**(x) {  
 sapply(x, **function**(x) {  
 integrate(**function**(y) f(x, y), x^2, 1)$value  
 })  
}, -1, 1)  
  
result

## 1 with absolute error < 1.1e-14

*# Perform the double integration dxdy*  
result <- integrate(**function**(y) {  
 sapply(y, **function**(y) {  
 integrate(**function**(x) f(x, y), -sqrt(y), sqrt(y))$value  
 })  
}, 0, 1)  
  
result

## 1 with absolute error < 2.6e-08

## Python Code

[Python Notebook](https://colab.research.google.com/drive/1dz_Yy7u9xt1IN2Uwb3YCYvL7I7mc6Ml8#scrollTo=7C7eJHbwSFx9)