# Integration Computer Exercises

Review Section 1.2.2 in Goldsman, D. & Goldsman, P. (2020). A First Course in Probability and Statistics. Lulu. [Download link](https://www.lulu.com/search?page=1&q=goldsman&pageSize=10&adult_audience_rating=00)

Other resources that are helpful when reviewing calculus are:

[Paul’s Online Math Notes](https://tutorial.math.lamar.edu/)

[Khan Academy](https://www.khanacademy.org/)

[YouTube](https://www.youtube.com/results?search_query=Calculus)

<https://math.libretexts.org/Bookshelves/Calculus>

## Riemann Sums

[Riemann sums](https://en.wikipedia.org/wiki/Riemann_sum) are a mathematical technique used to approximate the integral or the area under a curve (function) over a given interval. Named after the German mathematician [Bernhard Riemann](https://en.wikipedia.org/wiki/Bernhard_Riemann), Riemann sums serve as a fundamental concept in calculus, particularly in the area of [numerical integration](https://en.wikipedia.org/wiki/Numerical_integration).

A Riemann sum divides the area under a curve into a series of smaller, simpler geometric shapes (typically rectangles) to estimate the total area. The more rectangles used, the more accurate the approximation. In the context of integration, Riemann sums are a precursor to the more general concept of [definite integrals](https://en.wikipedia.org/wiki/Integral), which represent the exact area under a curve.

As n approaches infinity (i.e., the number of rectangles becomes very large), the width of each rectangle approaches zero, and the Riemann sum converges to the exact value of the definite integral of f(x) over the interval [a, b]:

Riemann sums provide a foundation for understanding the concept of integration and the area under a function. By dividing the region into smaller and smaller rectangles, Riemann sums help us transition from a discrete approximation to the continuous concept of integration, which is a central idea in calculus.

## Trapezoidal Rule

The [Trapezoidal Rule](https://en.wikipedia.org/wiki/Trapezoidal_rule) is a numerical integration technique used to approximate the definite integral or the area under a curve (function) over a given interval. It is a more accurate method than Riemann sums and is based on approximating the area under the curve using trapezoids instead of rectangles. The Trapezoidal Rule is particularly useful when dealing with functions that are difficult or impossible to integrate analytically.

It provides an approximation method for calculating definite integrals when analytical integration is not possible or inconvenient. The use of trapezoids improves the accuracy of the approximation compared to Riemann sums, making the Trapezoidal Rule a valuable tool for numerical integration. In general, the Trapezoidal Rule performs better for functions that have less curvature and for those with relatively small intervals [a, b]. For functions with significant curvature or larger intervals, other numerical integration techniques, such as [Simpson's Rule](https://en.wikipedia.org/wiki/Simpson%27s_rule), may provide better approximations.

## Monte Carlo Integration

[Monte Carlo Integration](https://en.wikipedia.org/wiki/Monte_Carlo_integration) is a numerical integration technique used to approximate the definite integral or the area under a curve (function) over a given interval. It is based on the principles of probability and statistical sampling, leveraging random numbers to estimate the integral. Monte Carlo Integration is particularly useful for [high-dimensional integrals](https://en.wikipedia.org/wiki/Multiple_integral), where traditional numerical integration methods become computationally expensive or impractical.

It provides an alternative method for approximating definite integrals when traditional numerical integration techniques, such as the Trapezoidal Rule or Simpson's Rule, become inefficient or infeasible. This is especially relevant for high-dimensional integrals, where the number of evaluations required by traditional methods grows exponentially with the dimensionality of the problem.

The accuracy of Monte Carlo Integration depends on the number of random points (N) used in the approximation. As N increases, the approximation converges to the true value of the integral. The error in Monte Carlo Integration typically decreases at a rate proportional to 1/√N, which is slower than traditional methods for low-dimensional problems but becomes advantageous as the dimensionality increases.

Monte Carlo Integration is a versatile technique with applications in many areas of science and engineering, such as physics, finance, and computer graphics. Its primary strength lies in its ability to tackle complex, high-dimensional integrals that are otherwise challenging or impossible to solve using traditional numerical methods.

## Integration Exercises Example R Code

This R script calculates the Left Riemann sum, Right Riemann sum, Midpoint Riemann sum, Trapezoidal Rule, [Simpson’s Rule](https://en.wikipedia.org/wiki/Simpson%27s_rule), and Monte Carlo integration for the given function and interval using 1000 partitions or samples. You can adjust the values of 'n' and 'N' to experiment with the accuracy of the different methods.

*# Define the function*  
f <- **function**(x) {  
 sin(pi \* x / 2)  
}  
  
*# Define the interval*  
a <- 0  
b <- 1  
  
*# Left Riemann Sum*  
left\_riemann\_sum <- **function**(f, a, b, n) {  
 dx <- (b - a) / n  
 sum <- 0  
 **for** (i **in** 0:(n-1)) {  
 xi <- a + i \* dx  
 sum <- sum + f(xi)  
 }  
 return(sum \* dx)  
}  
  
*# Right Riemann Sum*  
right\_riemann\_sum <- **function**(f, a, b, n) {  
 dx <- (b - a) / n  
 sum <- 0  
 **for** (i **in** 1:n) {  
 xi <- a + i \* dx  
 sum <- sum + f(xi)  
 }  
 return(sum \* dx)  
}  
  
*# Midpoint Riemann Sum*  
midpoint\_riemann\_sum <- **function**(f, a, b, n) {  
 dx <- (b - a) / n  
 sum <- 0  
 **for** (i **in** 1:n) {  
 xi <- a + (i - 0.5) \* dx  
 sum <- sum + f(xi) \* dx  
 }  
 return(sum)  
}  
  
*# Trapezoidal Rule*  
trapezoidal\_rule <- **function**(f, a, b, n) {  
 dx <- (b - a) / n  
 sum <- 0.5 \* (f(a) + f(b))  
 **for** (i **in** 1:(n - 1)) {  
 xi <- a + i \* dx  
 sum <- sum + f(xi)  
 }  
 return(sum \* dx)  
}  
  
*# Simpson's Rule*  
simpsons\_rule <- **function**(f, a, b, n) {  
 dx <- (b - a) / n  
 sum <- f(a) + f(b)  
 **for** (i **in** 1:(n-1)) {  
 xi <- a + i \* dx  
 **if** (i %% 2 == 0) {  
 sum <- sum + 2 \* f(xi)  
 } **else** {  
 sum <- sum + 4 \* f(xi)  
 }  
 }  
 return((dx / 3) \* sum)  
}  
  
*# Monte Carlo Integration*  
monte\_carlo\_integration <- **function**(f, a, b, N, seed = 1) {  
 set.seed(seed)  
   
 return(((b - a) / N) \* sum(f(a + (b-a)\*runif(N))))  
}  
  
*# Set the number of partitions (for Riemann Sum and Trapezoidal Rule) and*   
*# samples (for Monte Carlo)*  
n <- 100  
N <- 1000  
  
*# Calculate the approximations*  
left\_sum <- left\_riemann\_sum(f, a, b, n)  
right\_sum <- right\_riemann\_sum(f, a, b, n)  
midpoint\_sum <- midpoint\_riemann\_sum(f, a, b, n)  
trap\_rule <- trapezoidal\_rule(f, a, b, n)  
simpsons <- simpsons\_rule(f, a, b, n)  
monte\_carlo <- monte\_carlo\_integration(f, a, b, N)  
exact <- integrate(f, a, b)$value  
  
*# Print results*  
cat("Left Riemann Sum:", left\_sum, "\n")

## Left Riemann Sum: 0.6316067

cat("Right Riemann Sum:", right\_sum, "\n")

## Right Riemann Sum: 0.6416067

cat("Midpoint Riemann Sum:", midpoint\_sum, "\n")

## Midpoint Riemann Sum: 0.6366263

cat("Trapezoidal Rule:", trap\_rule, "\n")

## Trapezoidal Rule: 0.6366067

cat("Simpson's Rule:", simpsons, "\n")

## Simpson's Rule: 0.6366198

cat("Monte Carlo Integration:", monte\_carlo, "\n")

## Monte Carlo Integration: 0.6361259

cat("Exact Answer:", exact, "\n")

## Exact Answer: 0.6366198

## Integration Exercises Example Python Code

[Python Notebook](https://colab.research.google.com/drive/1wPFWsz_wHGYmkrXOko6tRWBYOnlvHAPt)