Applied Statistical Programming - Spring 2022

Alma Velazquez

Problem Set 5

Due Wednesday, March 30, 10:00 AM (Before Class)

Instructions

1. The following questions should each be answered within an Rmarkdown file. Be sure to provide many comments in your code blocks to facilitate grading. Undocumented code will not be graded.
2. Work on git. Continue to work in the repository you forked from https://github.com/johnsontr/AppliedStatisticalProgramming2022 and add your code for Problem Set 5. Commit and push frequently. Use meaningful commit messages because these will affect your grade.
3. You may work in teams, but each student should develop their own Rmarkdown file. To be clear, there should be no copy and paste. Each keystroke in the assignment should be your own.

Numerical Integration

Approximating an integral is a necessary part of Bayesian statistics. Numerical integration is inherent to Monte Carlo integration, rejection and importance sampling, Markov chain Monte Carlo, and many other methods. Two common approximations for integration are based on the trapezoidal rule and Simpson's rule. In this exercise, you will create an `S4` package that performs numerical integration using these two rules. Read the following two subsections about these approximations, then follow the instructions to create the package.

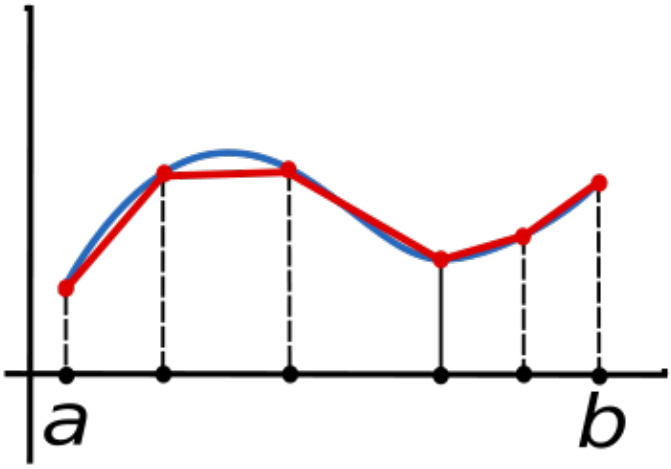
Trapezoidal Rule

Trapezoid rule is a technique for approximating the definite integral, and it follows the mathematical expression below:

$$\int_a^b f(x)dx \approx T$$

$$T = \frac{h}{2}(f(x_0) + 2f(x_1) + ... + 2(f_{n-1}) + f(x_n))$$

$$\text{where } h = \frac{b-a}{n}$$



Simpsons Rule

Simpson's rule is another technique for approximating a definite integrals, numerically. It follows the approximation below:

$$\int_a^b f(x)dx \approx S$$

$$S = \frac{h}{3}(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + ... + 4(f_{n-1}) + f(x_n))$$

$$\text{where } h = \frac{b-a}{h}$$

Image that you are drawing a parabola from the points $(a, f(a))$ to $(b, f(b))$ that also goes through the $(m, f(m))$. We can approximate the area under the parabola as being equal to $\int_a^b f(x)dx$.

For example, the parabola between any two points, $(u, f(u))$ and $(w, f(w))$ is drawn according to the formula (where $v = \frac{w-u}{2}$):

$$p(x) = f(u)\frac{(x-v)(x-w)}{(u-v)(u-w)} + f(v)\frac{(x-u)(x-w)}{(v-u)(v-w)} + f(w)\frac{(x-u)(x-v)}{(w-u)(w-v)}$$

Then, the integral under that parabola is:

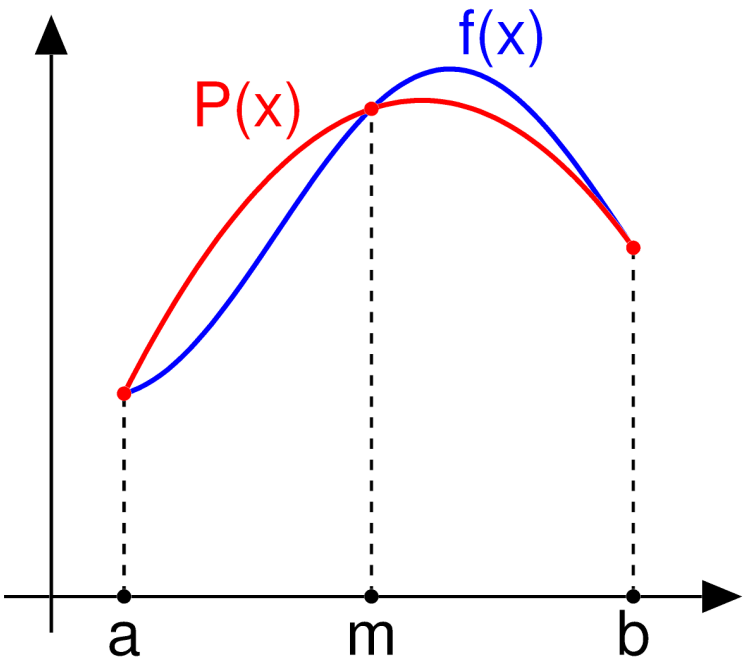
$$\int_u^w p(x)dx = \frac{h}{3}(f(u) + 4(f(v)) + f(w))$$

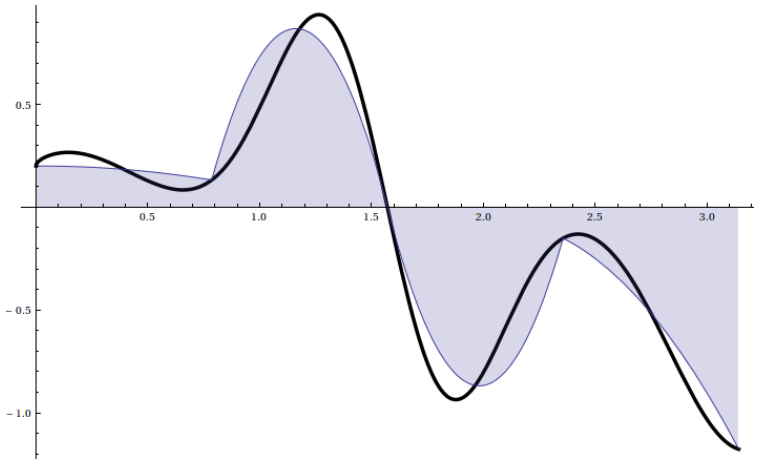
If we imagine carrying on that calculation many times between different points along the curve, we get

$$\int_a^b f(x)dx \approx S$$

$$S = \frac{h}{3}(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3)... + 4f(x_{n-1}) + f(x_n))$$

$$\text{where } h = \frac{b-a}{n}$$





Create an S4 Package

1. To prepare for your midterm, you will use `devtools` to create an S4 R package named `integrateIt`.
2. The package should include appropriate functions and appropriate documentation. For example, your package documentation needs to provide example usage that verifies basic functionality.
3. The package should have two classes: `Trapezoid` and `Simpson`
4. The package should have one generic: `integrateIt`
5. The package should contain at least two methods: `integrateIt`, `print`, and an optional extra credit method `tolTest`
   (a) `integrateIt` method
      i. Takes four arguments: a vector of values (x), a vector of evaluated values ($f(x) = y$), starting/ending values (a, b), and a *Rule* argument that can be either "Trapezoid" or "Simpson".

ii. Have three outputs: an object of class `Trapezoid` or class `Simpson`, the values of x and y, and the result

iii. Both classes should have validation methods that include a few appropriate tests

iv. you will need to create an `initialize` function for each class, which will be used internally by `integrateIt`

(b) `print` method

i. A very simple print method for each class, which prints out just the integrated value (rather than all of the results)

(a) Extra Credit: `tolTest` method

i. A `fun` (function)

ii. A `tolerance` argument

iii. A `rule` argument that indicates whether the Trapezoidal or Simpson's

iv. A `start` argument for the number of intervals it should start with

v. A `correct` argument that provides the correct answer for the integral

`tolTest` should take in a function and increase the number of intervals n until the answer it provides using the specified approximation is within `tolerance` of the correct answer. Use `integrate()` to do this. `tolTest` output should be

i. The inputs

ii. The final n

iii. The absolute error of the estimate

The following installs the completed `integrateIt` package from GitHub and demonstrates its use on 3 functions.

```
## Load libraries and set working directory
library(devtools)
```

```
## Loading required package: usethis
```

```
library(roxygen2)
install_github("a-velazquez/AppliedStatisticalProgramming2022/PS5/integrateIt")
```

```
## Skipping install of 'integrateIt' from a github remote, the SHA1 (66e87cd9) has not changed since las
##   Use 'force = TRUE' to force installation
```

```
library(integrateIt)
```

$$\int_1^3 x^2\,dx$$

```
# Define function
xSq <- function(x){x^2}

# Integrate with each rule using different n values.
xSq_Trap <- integrateIt(xSq, 200, c(1,3), rule="Trapezoid")
xSq_Simp <- integrateIt(xSq, 3, c(1,3), rule="Simpson")

# Print both objects.
print(xSq_Simp)
```

```
## [1] 7.111111
```

```
print(xSq_Trap)
```

```
## [1] 8.6667
```

```
# Use tolTest to find a more accurate n.
tolTest(xSq, 3, 0.001, c(1,3), "Trapezoid")
```

```
## $function_input
## function(x){x^2}
## <bytecode: 0x7fc9ae5abdc0>
##
## $start_n
## [1] 3
##
## $desired_tol
## [1] 0.001
##
## $integration_method
## [1] "Trapezoid"
##
## $interval
## [1] 1 3
##
## $final_n
## [1] 37
##
## $absolute_error
## [1] 0.0009739469
```

$$\int_1^{10} e^x \, dx$$

```
# Create Trapezoid object for the exponential function.
expTrap <- integrateIt(exp, 200, c(1,10), rule="Trapezoid")

# Print the result.
print(expTrap)
```

```
## [1] 22027.46
```

```
# Use tolTest to find an accurate n for the Simpson rule.
tolTest(exp, 3, 0.001, c(1,3), "Simpson")
```

```
## $function_input
## function (x)  .Primitive("exp")
##
## $start_n
## [1] 3
##
## $desired_tol
## [1] 0.001
##
## $integration_method
## [1] "Simpson"
```

```
##
## $interval
## [1] 1 3
##
## $final_n
## [1] 8
##
## $absolute_error
## [1] 0.0003741076
```

```
# Create Simpson object for the exponential function using the n output.
expSimp <- integrateIt(exp, 10, c(1,10), rule="Simpson")

# Print the result.
print(expSimp)
```

```
## [1] 22096.89
```

$$\int_0^{\frac{\pi}{4}} \sin(x)\, dx$$

```
# Create Trapezoid object for the sine function.
sinTrap <- integrateIt(sin, 200, c(0,(pi/4)), rule="Trapezoid")

# Print the result.
print(sinTrap)
```

```
## [1] 0.2928928
```

```
# Use tolTest to find an accurate n for the Simpson rule.
tolTest(sin, 3, 0.00001, c(0,(pi/4)), "Simpson")
```

```
## $function_input
## function (x)  .Primitive("sin")
##
## $start_n
## [1] 3
##
## $desired_tol
## [1] 1e-05
##
## $integration_method
## [1] "Simpson"
##
## $interval
## [1] 0.0000000 0.7853982
##
## $final_n
## [1] 4
##
## $absolute_error
## [1] 2.429703e-06
```

```
# Create Simpson object for the sine function using the n output.
sinSimp <- integrateIt(sin, 4, c(0,(pi/4)), rule="Simpson")
```

```r
# Print the result.
print(sinSimp)
```

```
## [1] 0.2928956
```