STAT 447 Assignment 3

Caden Hewlett

2024-01-24

Question 1 : Functions on the Unit Interval

For this question, use Simple Monte Carlo. The main twist compared to week one is that you will use a continuous random variable.

Part 1

Write a function called `mc_estimate` that takes a function $f : [0, 1] \to \mathbb{R}$ and outputs the Monte Carlo estimator of $\int_0^1 f(x)dx$ using $n = 100,000$ independent samples from unif$(0, 1)$.

NOTE: Because my computer could handle it, I used $M = 100,000$ rather than $M = 10,000$ just for fun. The results are similar (but obviously more precise for larger M.)

The function is created using the code below:

```
mc_estimate <- function(f){
  # declare the number of iterations
  M <- 100000
  # randomly generate M total observations in [0, 1]
  m_vals <- runif(M)
  # then, for all m in random generations, evaluate f(m)
  G_m <- f(m_vals)
  # then compute the average
  G_hat_m <- (1/M)*sum(G_m)
  # and return
  return(G_hat_m)
}
```

Part 2

Consider the function $f : [0, 1] \to [0, \infty)$ given by:

$$f(x) = \frac{1}{\sqrt[3]{x^2(1-x)}}$$

Note, importantly, that

$$\mathcal{I}_1 = \int_0^1 f(x)\mathrm{d}x = \frac{\pi}{\sin\left(\frac{\pi}{3}\right)}$$

Test your implementation of `mc_estimate` by checking that it produces an answer close to the value above.

We'll start by computing the actual result:

```
expected = pi / (sin(pi/3))
expected
```

```
## [1] 3.627599
```

Now, we implement `mc_estimate`.

```
# Apollo 11 moon landing, as an integer
set.seed(19690720)

f <- function(x){
  1 / ( ((x^2)*(1 - x))^(1/3) )
}

observed <- mc_estimate(f)
observed
```

```
## [1] 3.629373
```

It seems we got pretty close! Let's calculate the percent difference.

```
(abs(observed - expected) / expected)*100
```

```
## [1] 0.04892034
```

It looks like there's about a 0.05% difference between $\hat{G}_M$ for $M = 100,000$.

For completeness, I also ran the code with $M = 10,000$ and observed approximately a 2.52% difference.

Part 3

The following integral, known as the sine integral, does not admit a closed-form expression.

$$\mathcal{I}_2 = \int_0^1 \frac{\sin(t)}{t} \mathrm{d}t$$

It does not admit a closed-form expression. Estimate its value using `mc_estimate(f)`.

To test our Monte Carlo Approximation, we will evaluate Si(1) using the `pracma` package.

```
Si(1)
```

```
## [1] 0.9460831
```

Now, we can see how close our `mc_estimate` gets. We let $g(x) = \frac{\sin(x)}{x}$.

```r
set.seed(19690720)

g <- function(x){
  sin(x) / x
}
observed = mc_estimate(g)
observed
```

```
## [1] 0.9464033
```

It looks like we got really close! Let's find the percent difference:

```r
expected = Si(1)
# we overwrite our old variables for convenience (and memory)
(abs(observed - expected) / expected)*100
```

```
## [1] 0.03384555
```

It looks like there's about a 0.03% difference between $\hat{G}_M$ for $M = 100,000$.

For completeness, I also ran the code with $M = 10,000$ and observed approximately a 0.09% difference.

Question 2 : Implementing SNIS for simPPLe

Part 1

First, install the package `distr`.

Since this is a `.Rmd` file, all packages are loaded at the beginning in a hidden cell with suppressed warnings.

```r
"distr" %in% installed.packages()
```

```
## [1] TRUE
```

Part 2

Read this short tutorial on `distr`. Nothing to submit for this item.

Part 3

Read the "scaffold code", and use `distr` and two of the functions in the example to create a fair coin, flip it, and to compute the probability of that flip:

The scaffold code is given below:

```r
source("scaffold.R")
```

The documented code to complete this task is shown below:

```r
# create the coin (Bernoulli distribution)
coin <- Bern(1/2)
# flip the coin once
flip <- simulate(coin)
# compute the probability of the flip
prob <- p(coin, flip)
# report the findings
print(paste("The coin flipped a ", flip,
      ". The probability of this flip is ",
       prob, sep = ""))
```

```
## [1] "The coin flipped a 1. The probability of this flip is 0.5"
```

Part 4

Complete the implementation of the function `posterior`:

To do this, let's consider what inputs we are given, relative to the Importance Sampling procedure we discussed in lecture.

Notably, we aren't given a test function $g(x)$ so we will assume it is the indicator function.

We will use:

$$g(x) = \mathbb{1}(X = 1)$$

So we can approximate $P(X = 1 \mid Y = y)$.

Following the Algorithm, the first thing we have to do is find $(X^{(m)})$ for iteration $m \in [1, \ldots, M] \subseteq \mathbb{N}$.

We'll do this by adding the line defining the `m` variable.

Further, we'll define the function $g(x)$ prior to the `for` loop, explicitly as a function. This means we can return to this code later and improve it (perhaps allowing $g(x)$ to be passed as a parameter.) This allows us to compute $G^{(m)} = g(X^{(m)})$.

```r
posterior = function(ppl_function, number_of_iterations) {
  numerator = 0.0
  denominator = 0.0
  g = function(x){x == 1}
  for (i in 1:number_of_iterations) {
    weight <<- 1.0
    m <- ppl_function()
    G <- g(m)
    # update numerator and denominator
    numerator <- numerator + (weight*G)
    denominator <- denominator + weight
  }
  return(numerator/denominator)
}
```

Part 5

Test your program by checking that you can approximate the posterior probability of the fair coin obtained in exercise 1, Q.2.

We'll first recite the Coin Flip example in the context of simPPLe, and load the scaffold.

```r
coin_flips = rep(0, 4) # "dataset" of four identical coin flips = (0, 0, 0, 0)
# simPPLe's description of our "bag of coins" example
my_first_probabilistic_program = function() {
  # Similar to forward sampling, but use 'observe' when the variable is observed
  coin_index = simulate(DiscreteDistribution(supp = 0:2))
  for (i in seq_along(coin_flips)) {
    prob_heads = coin_index/2
    observe(coin_flips[i], Bern(1 - prob_heads))
}

  # return the test function g(x, y)
  return(ifelse(coin_index == 1, 1, 0))
}
```

Recalling my answer from 1.2 we know that we should be convergent in probability to 1/17.

In other words, since $g(X) = \mathbb{1}(X = 1)$,

$$\lim_{M \to \infty} \left(\hat{G}_m\right) = \mathbb{E}[g(X)] = P(X = 1 \mid Y_{1:4} = \vec{0}) = \frac{1}{17}$$

Let's try some different values of M.

```r
set.seed(08041961)
posterior(my_first_probabilistic_program, 100)
```

```
## [1] 0.05226481
```

```r
set.seed(08041961)
posterior(my_first_probabilistic_program, 1000)
```

```
## [1] 0.0553609
```

```r
set.seed(08041961)
posterior(my_first_probabilistic_program, 5000)
```

```
## [1] 0.05809581
```

We seem to be getting closer to the truth as we increase M.

```r
1/17
```

```
## [1] 0.05882353
```