STAT 447C: Bayesian Statistics **Fall 2024**

Exercise 3

Junsong Tang *junsong.tang@stat.ubc.ca*

1 functions on the unit interval

1.

```r
  suppressPackageStartupMessages(library(extraDistr))
  suppressPackageStartupMessages(library(distr))
  set.seed(2024)

  # (1)
  mc_estimate = function(f) {
    sum = 0
    for (i in 1:10000) {
      x = runif(1, 0, 1)
      sum = sum + f(x)
    }
    return (sum / 10000)
  }

```

2.

```r
# (2)
my_fun = function(x) exp(-x^2)
print(mc_estimate(my_fun)) # 0.7495085
```

3.

```r
# (3)
fun = function(x) sin(cos(sin(x)))
print(mc_estimate(fun))  # 0.7590194
```

2 implementing SNIS for simPPLe

1.

```r
  weight = 1.0
  # .GlobalEnv$weight = 1.0
  coin_flips = rep(0, 4)


  ## Utilities to make the distr library a bit nicer to use

  p <- function(distribution, realization) {
    d(distribution)(realization) # return the PMF or density
  }

  Bern = function(probability_to_get_one) {
    DiscreteDistribution(supp = 0:1, prob = c(1-probability_to_get_one, probability_to_get_one))
  }

  ## Key functions called by simPPLe programs

  # Use simulate(distribution) for unobserved random variables
  simulate <- function(distribution) {
    r(distribution)(1) # sample once from the given distribution
  }

  observe = function(realization, distribution) {
    # `<<-` lets us modify variables that live in the global scope from inside a function
    weight <<- weight * p(distribution, realization)
```

```
}

```

2.

```
# (4)
posterior = function(ppl_function, number_of_iterations) {
  numerator = 0.0
  denominator = 0.0
  for (i in 1:number_of_iterations) {
    weight <<- 1.0
    g_i = ppl_function()
    # update numerator and denominator
    numerator = numerator + g_i * weight
    denominator = denominator + weight
  }
  return(numerator/denominator)
}

```

3.

```
# (5)
my_ppl = 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))
}

posterior(my_ppl, 10000) - 1/17 # -0.00101723975274511

```