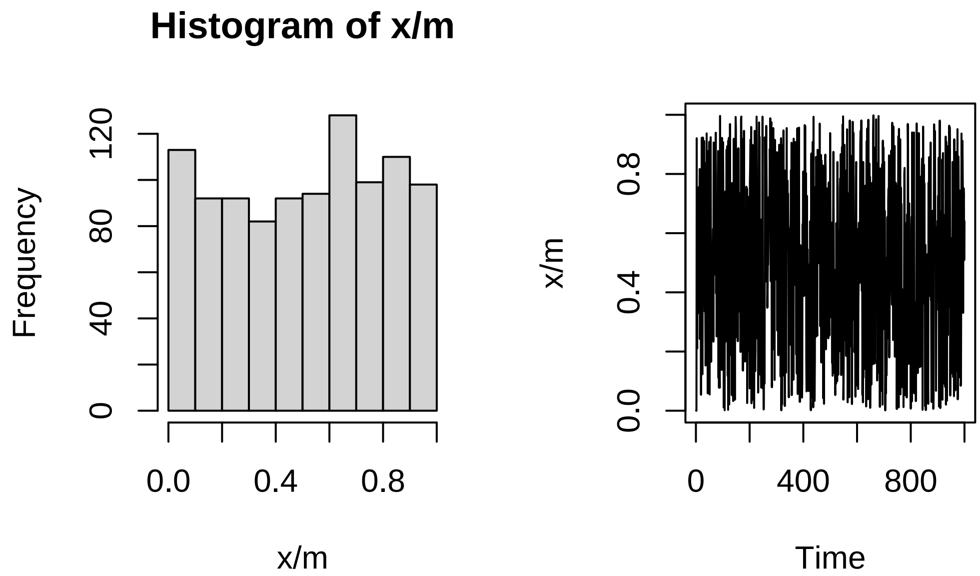
MTH210: Lab 1 Solutions

1. **Read through the code in `PRNG.R` that attempts to implement the multiplicative congruentrial method to generate pseudo-random numbers. (Some values are missing in those.) Try various values of m, a, and the seed x_0, and assess the performance of the generator.**

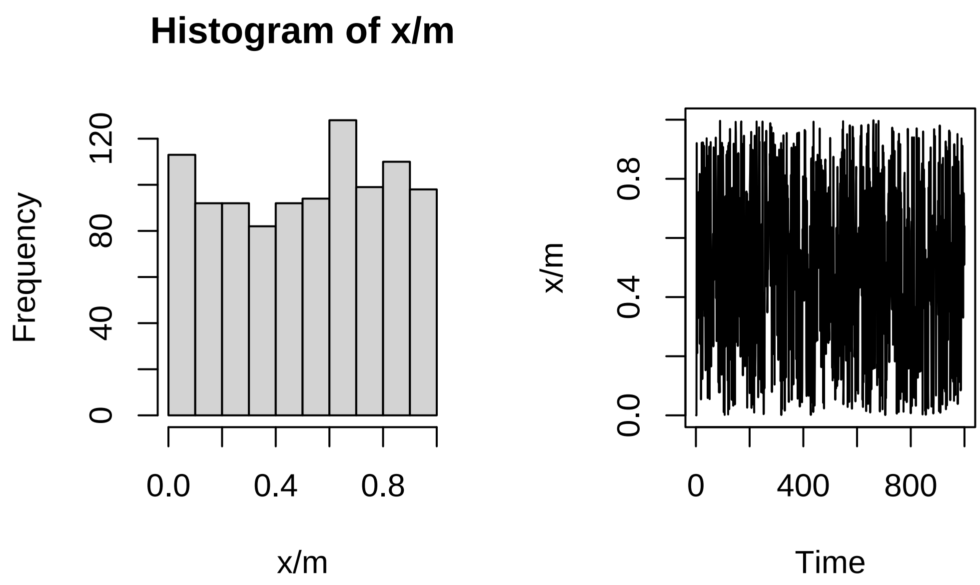
   We use the recommend choices of a and m and I make a choice of x_0.

   ```
   ############################################
   ## Pseudo-random number generation
   ## using multiple congruential method
   ############################################
   m <- 2^31 - 1  # choose a value you want
   a <-  7^5 # choose a value you want
   x <- numeric(length = 1e3)
   x[1] <- 7  #x0 -- choose
   for(i in 2:1e3)
   {
     x[i] <- (a * x[i-1]) %% m
   }

   # We will visualize using histograms (for testing uniformity)
   # and trace plots (for checking independence)
   par(mfrow = c(1,2))
   hist(x/m) # looks close to uniformly distributed
   plot.ts(x/m) # look like it's jumping around too
   ```



2. **Change the code in `PRNG.R` to implement the mixed congruential method. Is there much visible difference between these methods?**

```
############################################
## Pseudo-random number generation
## using mixed congruential method
############################################
m <- 2^31 - 1  # choose a value you want
a <-  7^5 # choose a value you want
c <- 13
x <- numeric(length = 1e3)
x[1] <- 7  #x0 -- choose
for(i in 2:1e3)
{
  x[i] <- (a * x[i-1]) %% m
}

# We will visualize using histograms (for testing uniformity)
# and trace plots (for checking independence)
par(mfrow = c(1,2))
hist(x/m) # looks close to uniformly distributed
plot.ts(x/m) # look like it's jumping around too
```



3. **Write a function to generate a Bern(p) random draw using Inverse Transform method. Call this function `myBern`.**

```
myBern <- function(p)
{
  # Uniform draw
  U <- runif(1)

  if(U < 1-p) return(0)
  return(1)
}
```

4. **For $p = .25$, call the `myBern` function 1000 times and come up with a method to check whether `myBern` is coded correctly.**

```
# I will use replicate function, instead of loop.
# although, for loop will be faster
out <- replicate(1000, myBern(.25))

# in order to check, I will check proportion of successes
mean(out)  # close to .25
```

```
[1] 0.249
```

5. **Using only `runif()` function, write your version of the `sample` function, called `mySample`. It should have all the features of the `sample` function.**

The `sample()` function in `R` has the following arguments:

```
sample(x, size, replace = FALSE, prob = NULL)
```

where `x` are the things we want to sample from, `size` are the number of draws, and `prob` is the pmf over `x`. For convenience, I will assume `replace = TRUE` only, since it is not clear to me how to do `replace = FALSE`!

```
mySample <- function(x, size, prob)
{
  out <- numeric(size)

  # number of things to choose from
  n_opts <- length(prob)

  # draw multiple times
  for(i in 1:size)
  {
    # Draw uniform
    U <- runif(1)

    # check where U lies in the probabilities
    for(k in 1:n_opts)
    {
      if(U < sum(prob[1:k]))
      {
        out[i] <- x[k]
        break   # if yes, then break out of the "k" loop
      }
    }
  }

  # return the full output vector
  return(out)
}
```

Now that the above function is ready, I will run it once and check. In the code below, I want to sample numbers (3, 6, 9) with probability (.3, .4, .3).

```
mySample(x = c(3,6,9), size = 1, prob = c(.3, .4, .3))
```

```
[1] 9
```

Ok, it returns a number I would expect, but we don't know for sure if the function works! To know for sure (sort of), we will replicate this (say) 1e4 times, and calculate the proportion of times each number is sampled:

```r
samples <- mySample(x = c(3,6,9), size = 1e4, prob = c(.3, .4, .3))
table(samples)/1e4
```

```
samples
     3      6      9 
0.2985 0.3981 0.3034 
```

I will need to think a bit more about how to write this function when `replace = FALSE`. For now, we move on.