Practical 2 – Simulation and Statistical Programming

Q1. Write a function that uses a for loop to sum a geometric series with n terms, first term a and common ratio r. The function should take three arguments : a, r and n and should return the sum. Check that it works using the formula for the sum of a geometric series.

```r
geomsum = function(a, r, n) {
    x = 0
    for(i in 1:n) x = x + a * r^(i-1)
    return(x)
}
```

Q2. Sieve of Eratosthenes. Here is a plan I wrote for an R function to find all primes between 2 and n inclusive:
   a) Create a vector s of numbers from 2 to n.
   b) The first entry 2 is a prime. Find all multiples of 2 and remove them from the vector s.
   c) The vector is shorter now because we removed 2 and all its multiples. It has a new first entry (i.e. 3). This first entry hasn't been eliminated so it can't be divisible by any smaller number. It must be a prime.
   d) Repeat this until the vector s is empty, saving the primes as we go.

The following code implements this.

```r
Eratosthenes = function(n) {
    # return all prime numbers up to n
    if(n < 2) stop("Input value must be >= 2")
    s = 2:n
    primes=c() #start with no primes
    while (length(s)>0) {
        p=s[1]
        primes=c(primes,p)
        i=which(s%%p==0)
        s=s[-i]
    }
    return(primes)
}
```

Q3. (optional) Write an R function that returns the real roots of the quadratic ax^2 + bx + c. The function should take a, b and c as arguments and return appropriate messages if the values entered don't specify a quadratic or if there are no real roots. Use the function to determine the roots of 2x^2 - x - 3.

```r
f = function(a,b,c) {
    if(!identical(a, 0)) {
        x = b^2 - 4*a*c
        if(identical(x, 0)) return(-b/(2*a))
        if(x < 0) return("No real roots")
        if(x > 0) return(c((-b + c(-sqrt(x)),sqrt(x))) / (2*a))
    } else {
        stop("a = 0 so not a quadratic")
    }
}
```

> f(2,-1,-3)
Q4. Last week we saw we could sample a discrete distribution \(p=(p_{1},p_{2},\ldots,p_{m})\) by simulating \(u \sim U[0,1]\) (\(u=runif(1)\)) and looking for the smallest \(x\) in \(\{1,2,\ldots,m\}\) such that

\[ u < p_{1} + p_{2} + \ldots + p_{x} \]

For example

```r
> p <- c(0.1, 0.2, 0.3, 0.4)
> cp <- cumsum(p)
> min(which(runif(1) < cp))
```

Make sure you understand how the above code works. Write a function which takes as input a pmf \(p=c(p[1],p[2],\ldots,p[m])\) and a number \(n\) and returns \(n\) samples \(X[1],X[2],\ldots,X[n]\) distributed according to \(p\). Comment on how to test your function.

```r
discrete <- function(p, n=1) {
    #sample X~p, p a pmf satisfying p[i]>=0, sum(p)=1
    X <- numeric(n)
    cp <- cumsum(p)
    for (i in 1:n) {X[i] <- min(which(runif(1) < cp))}
    return(X)
}
```

```r
> X <- discrete(c(0.1, 0.2, 0.3, 0.4), 10000)
> mean(X==1)
[1] 0.1014
> mean(X==4)
[1] 0.4069
```

Q5. Write an R function to simulate \(X \sim N(0,1)\) using rejection with proposal \(Y \sim \exp(-|x|)\).

i. Write a function to simulate \(n\) iid values of \(Y\). Make the default \(n\)-value \(n=1\).

Here are a couple of possible solutions. You could also use a for-loop, though that is less efficient and harder to read.

```r
dbexp <- function(n=1) {
    X <- log(runif(n))
    Y <- sample(c(-1, 1), n, replace=T)
    return(X*Y)
}
```

Alternative to “sample()” would be “\(Y<-2*\text{round}(\text{runif}(n))-1\)”.

ii. Write a function implementing rejection for \(X\). Recall the algorithm from Q4 PS1:

1. Simulate \(Y \sim \exp(-|x|)\) and \(U \sim U(0,1)\)
2. if \(U < \exp(-y^2/2+|y|-1/2)\) accept \(X\)=\(y\) and stop. Otherwise repeat [1].

Hint: you can do this using a while statement. You should call the function you wrote in Q2.i to simulate \(Y\).

Your function should have no inputs, and return the simulated value of \(X\).
my_rnorm<-function() {
  finished<-FALSE;
  while (!finished) {
    y<-rdbexp();
    finished<-(runif(1)<exp(-y^2/2+abs(y)-0.5))
  }
  return(y)
}

iii. Test your rejection sampler by simulating 1000 samples and checking they are
      normal using the `qqnorm()` function.

> k<-1000; X<-numeric(k);
> for (i in 1:k) X[i]<-my_rnorm();
> qqnorm(X); qqline(X)

Q6. The equation \(0 = x^7 + 10000x^6 + 1.06x^5 + 10600x^4 + 0.0605x^3 + 605x^2 + 0.0005x + 5\)
      has exactly one real root.

      (a) Plot the function to try to get a sense of where the root might be?

\[
\begin{align*}
f &= x^7 + 10000x^6 + 1.06x^5 + 10600x^4 + 0.0605x^3 + 605x^2 + 0.0005x + 5 \\
f(x) &= 7x^6 + 60000x^5 + 5 \cdot 1.06x^4 + 4 \cdot 10600x^3 + 3 \cdot 0.0605x^2 + 2 \cdot 605x + 0.0005 \\
\end{align*}
\]
\[
curve(f(x), from = -20000, to = 20000)
\]

      (b) Write an R function that applies Newton’s method to find the root. The function
          should have 2 arguments: the initial value \(x_0\) and the tolerance value. The
          function should return the estimated solution, the function value at the estimate
          and the number of iterations.

nr1 = function(x, tol = 0.001) {
  k = 0
  while(abs(f(x)) > tol) {
    x = x - (f(x) / f.prime(x))
    k= k + 1
  }
  return(c(x, f(x), k))
}

      (c) What happens when you set \(x_0 = 0\)?

> nr1(0)
[1] -10000      0      1

      (d) What happens when you set \(x_0 = 1\)?

> nr1(1)
[1] -10000      0 368922