SS2864B, 2020

Assignment #4 due to March 20, 11:55pm, 2020

Instructions Submit an electronic version (pdf, words) of your solutions (appropriately annotated with comments, plots, and explanations; notice that neatness counts) to owl. Save all your R codes in one script (or markdown) file with proper comments and submit it as well to owl.

1. Write two R functions which will evaluate polynomials of the form

$$P(x) = c_1 + c_2 x + \cdots + c_{n-1} x^{n-2} + c_n x^{n-1}.$$

   Your functions should take a **vector** x and the vector (called it coef) of polynomial coefficients as arguments and they should return the values of the evaluated polynomial, i.e., if $x = (x_1, \ldots, x_n)$, the return value is a vector of $(P(x_1), \ldots, P(x_n))$.

   (a) The first function is called **directpoly**. You can use **for** loop to loop the sum in the polynomial formula. But you cannot use **for** loop to calculate for different values of x, i.e., the computation for x must be vectorized. Do one error checking on coef input and return a proper error message if its length is less than 2. Test your function with $x = 1:3$ and coef=c(2,17,-3), i.e., $c_1 = 2, c_2 = 17, c_3 = -3$.

   (b) For moderate to large values of n, evaluate of a polynomial at x can be done more efficiently by using *Horner's rule*. This algorithm is:
   Step 1: set output=c_n;
   Step 2: for $i = n-1, \ldots, 1$, set output=output*$x + c_i$;
   Step 3: return output.
   The second function is called **hornerpoly** by using Horner's algorithm. Again you cannot use **for** loop to calculate for different values of x, i.e., the computation for x must be vectorized. Do one error checking on coef input and return a proper error message if its length is less than 2. Test your function with $x = 1:3$ and coef=c(2,17,-3).

   (c) Do some timing comparisons of the functions in (a) and (b). Try the following code

```
system.time(dirpoly(x=seq(-10,10, length=5000000), c(1,-2,2,3,4,6,7,8)))
system.time(hornerpoly(x=seq(-10,10, length=5000000), c(1,-2,2,3,4,6,7,8)))
```

   Run a few times. How faster is the Horner's algorithm? Comment your findings.

2. Write an R function called **my.unif** to generate a sequence of uniform pseudo-random numbers. The argument list should be n—sample size, a—multiplier, c—increment with default value 0, m—modulus, and $x0$—seed. The function should use

$$x_i = (ax_{i-1} + c) \pmod{m}, i = 1, \ldots, n$$

   to generate a vector of $x = (x_1, \ldots, x_n)$ and return value as x/m. Use your function to generate 50 uniform pseudo-random numbers with $a = 172, c = 13, m = 30307$ and initial seed $x_0 = 17218$. Construct a histogram to see if its distribution is like uniform[0,1]. Also generate 50 uniform pseudo-random numbers with $a = 171, c = 51, m = 32767$ and initial seed $x_0 = 2020$. Construct a histogram to see if its distribution is like uniform[0,1].

3. Generate 1000 uniform pseudo-random variates using using the **runif** function, assigning them to a vector called ***U***. Use set.seed(2020).

   (a) Compute the average, variance, and standard deviation of the numbers in ***U***.

   (b) Compare your results with true mean, variance, and standard deviation.

   (c) Compute the proportion of the values of U that are less than 0.6, and compare with the probability that a uniform random variable **U** is less than 0.6.

   (d) Construct a histogram of the values of ***U*** and comment your findings.

4. Construct two R function for generating random numbers from the following density.

   (a) Use the inverse method to implement an R function with inputs n and $a = 1$. The density function is
   $$f(x) = \frac{x}{a^2}e^{-x^2/(2a^2)}, \ a > 0, x \geq 0.$$
   Test your function with $n = 20$ and plot a histogram with $n = 1000$ and the option probability=TRUE. Then add the density curve $f(x)$ with color=2 to the plot. Comment your findings. No looping is allowed.

   (b) Use the reject method to implement an R function with inputs n, M, and $a = 1$, with the same density given in (a). Notice that the range is fixed from 0 to 5 and M is needed to be the max value of $f(x)$. Test your function with $n = 20$ and plot a histogram with $n = 1000$ and the option probability=TRUE. Then add the density curve $f(x)$ with color=2 to the plot. Comment your findings. No looping is allowed.

   (c) Use **system.time** function to record the time of generating a vector of 100,000 random numbers for each function implemented in (a) and (b) respectively. Comment your findings.

5. Let U be a uniform[0,1] random valuable. Use **runif** function to simulate 10000 values of uniform[0,1] for U, called u.

   (a) Estimate $E[U^2]$ and construct 95% confidence interval. Compare with the true value and comment the accuracy.

   (b) Since $E[U^2] = E[U^2 + (1 - U)^2]/2$, this suggests that we can use $v = (u^2 + (1 - u)^2)/2$ to estimate $E(V)$ where $V = (U^2 + (1 - U)^2)/2$. Construct 95% confidence interval. Compare with the result in (a) and comment your findings.

   (c) Similarly, use the fact $E[U^2] = E[(U/2)^2 + (1 - U/2)^2]/2$ to construct another estimator and a 95% confidence interval. Compare with the results in (a) and (b) and comment your findings.