STA 220 - Data and Web Technologies for Data Analysis - midterm

Exercises

1.After executing the command

```
z <- rep(c(rep(x, each = a), rep(y, each = b)), times = t)
```

which element of x or y is contained in the kth element of z?. Suppose that x and y are vectors in R of lengths m and n, respectively. Write a function, `rep_tracker(k, m, n, a, b, t)`, which computes the answer for any positive integers k, m, n, a, b, and t. Your function should return a list whose first element is named "src_vector" and gives the name of the vector ("x" or "y") and whose second element is named "src_index" and gives the integer indicating the original location of the value in either "x" or "y". For example, if the kth element of z was originally the 5 element of y, then your function should return `list(src_vector = "y", src_index = 5)`.

Your may not use the `rep()` function in your solution. Your function should emulate the way that a human might solve this problem using integer arithmetic.

Solution

```
rep_tracker <- function(k, m, n, a, b, t) {
    if (k > (t * (m * a + n * b))) warning("First argument is out of range")
    ##
    ## INSERT YOUR CODE HERE TO DETERMINE v AND i
    ##
    list(src_vector = v, src_index = i)
    }
```

ALERT: Please delete this line and everything below it before submitting your solution.

Hints

It is helpful to think about how a human would work this out. Note that

- `rep(x, each = a)` repeats each element of x a times, producing a vector of length `m*a`.
- `rep(y, each = b)` repeats each element of y b times, producing a vector of length `n*b`.
- `c(rep(x, each = a), rep(y, each = b))` combines these two vectors, producing a vector of length `m*a + n*b`.
  Lets call this vector "w".
- `rep(c(rep(x, each = a), rep(y, each = b)), times = t)` repeats the whole vector w t times, producing a vector of length `t * (m*a + n*b)`.

With this knowledge, we can work backwards to the answer. Suppose for example that k = 130, m = 11, n = 8, a = 2, b = 3, and t = 4.

- The outermost call to `rep()` just makes t = 4 copies of a vector w of length `m*a + n*b` = 46.
- k = 130 divided by 46 gives 2 with a remainder of r = 38. (How would you calculate this remainder in R?)

- These calculations tell us that we're going to skip over 2 copies of w and then look for the 38th element of w.
  Be careful here: what element of w would you be seeking if the remainder calculated in the last step were 0?
- Next, to determine whether we're looking for an element of the original x or y vector, we just need to know whether the remainder from the last step was less than or equal to m*a = 22 or not.
  In the present example r = 38 is greater than 22, so we are looking for an element of y.
- If we are looking for an element of y, then we can subtract m*a from r and ignore x. Otherwise, we can ignore y.
  In the present example, we are looking for an element of y, so r is changed to 16.
- Next, depending on whether we are looking for an element of x or y, we have to use r with either a or b to determine the original index. Let d represent the value of a or b, depending on whether we are looking for an element of x or of y. Thus, in the present example, d = 3.
- Now you just need to figure out how to do the necessary arithmetic with r and d to get the index.

In outline, the function should look something like this. (Again, be careful to do the right thing in the first step if `m*a + n*b` happens to divide evenly into k.

```
calculate the remainder r when k is divided by m*a + n*b
if r is less than or equal to m*a
    set v to "x"
    set d to a
else
    set v to "y"
    set r to r - m*a
    set d to b
endif
use r and d to calculate i
return results
```

Again, you are not allowed to use the `rep()` function in your solution. Thus, a function like the following is *NOT* allowed, even though it returns the correct solution. (Of course there is nothing stopping you from using this version to check that your own solution is working correctly.)

```r
forbidden <- function(k, m, n, a, b, t) {
    if (any(k > (t * (m * a + n * b)))) warning("First argument is out of range")
    xy <- rep(c(rep("x", each = m*a), rep("y", each = n*b)),
              times = t)
    ij <- rep(c(rep(1:m, each = a), rep(1:n, each = b)),
              times = t)
    list(src_vector = xy[k], src_index = ij[k])
    }
```

One last comment: the forbidden version of the function provided above will work correctly if the argument k is an integer *vector*. It is a good exercise to try to modify your solution so that it also works with vector k.

Solution

```r
## I would expect their solution to look something like this.
rep_tracker <- function(k, m, n, a, b, t) {
    if (k > (t * (m * a + n * b))) warning("First argument is out of range")
    r <- k %% (m * a + n * b)
    if (r == 0) r <- m * a + n * b
    if (r <= m*a) {
        v <- "x"
        d <- a
    } else {
        v <- "y"
        r <- r - m*a
        d <- b
    }
    i <- ceiling(r/d)   ## Or:  i <- ((r - 1) %/% d) + 1
    list(src_vector = v, src_index = i)
    }
```

```
## For grading, you can give full credit if they followed the
## guidelines of the assignment and their function is correct (see
## below for how you can test this).  If you take off any points, you
## should insert brief comments in the .Rmd file to explain each
## deduction of points.
##
## If their code is so messy that it's unreadable, you can take off one
## point for that and explain why.
##
## If they have errors in their function, here are my suggestions for
## assigning points:
##
rep_tracker <- function(k, m, n, a, b, t) {
    if (k > (t * (m * a + n * b))) warning("First argument is out of range")
    ## BEGIN 4 point block---------------------------------
    r <- k %% (m * a + n * b)
    if (r == 0) r <- m * a + n * b
    ## END 4 point block---------------------------------
    ## BEGIN 7 point block---------------------------------
    if (r <= m*a) {
        v <- "x"
        d <- a
    } else {
        v <- "y"
        r <- r - m*a
        d <- b
    }
    ## END 7 point block---------------------------------
    ## BEGIN 4 point block---------------------------------
    i <- ceiling(r/d)   ## Or:  i <- ((r - 1) %/% d) + 1
    ## END 4 point block---------------------------------
    list(src_vector = v, src_index = i)
}
## Use your judgement of course.  I hope that they won't introduce any
## mistakes by changing any of the parts that I gave them (the first
## and last couple of lines), but if they do, you should talk off a
## point or two for that as well.
```

```
## After reviewing their code, you can test their solutions by
## generating a few random examples.  It's probably a good idea to
## check some edge cases as well.  Here's a function you can use for
## this.
test_gen <- function(edge = FALSE) {
    m <- sample(1:40, 1)
```

```r
    n <- sample(1:40, 1)
    a <- sample(1:10, 1)
    b <- sample(1:10, 1)
    t <- sample(1:10, 1)
    ## Just choose a random admissible value unless edge is TRUE, in
    ## which case we choose a random edge case.
    if (edge) {
        edge_cases <- (1:t) * (m * a + n * b)
        edge_cases <- c(edge_cases, edge_cases - (m * a + n * b) + 1)
        k <- sample(edge_cases, 1)
    } else {
        k <- sample(1:(t * (m*a + n*b)), 1)
    }
    list(k = k, m = m, n = n, a = a, b = b, t = t)
}
```

```r
## Then just run something like this (you would need to do it a
## several times):
x <- test_gen()
do.call(forbidden, x)   # We know this is correct
do.call(rep_tracker, x) # Is theirs the same?
x <- test_gen(edge = TRUE)
do.call(forbidden, x)
do.call(rep_tracker, x)

## It would be better to do something like this.  If you get an error
## message, you can look at x to see the example where their solution
## failed.
for (i in 1:1000) {
    x <- test_gen()
    error <- !isTRUE(all.equal(do.call(rep_tracker, x), do.call(forbidden, x)))
    if (error) {
        warning("Error detected")
        break
    }
}
```

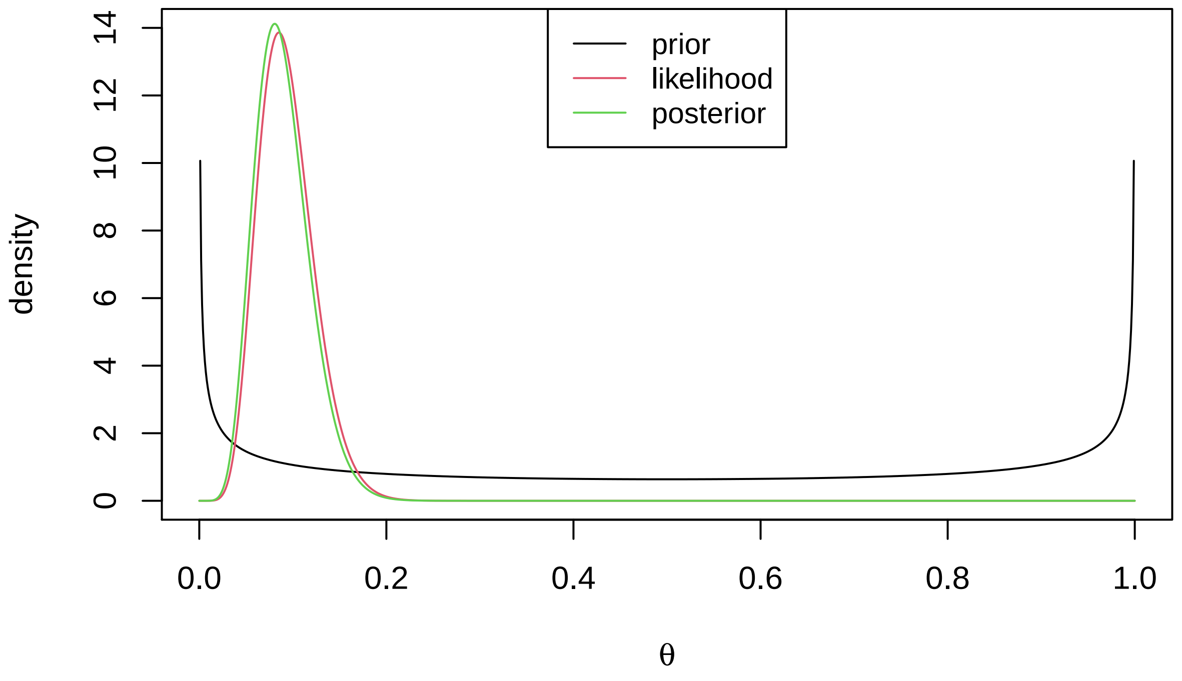
```r
## BTW, here is a vectorized version of the function.  You could probably
## talk about this in the next lab.  It should work if any or
## all of k, m, n, a, b, or t are vectors, with the usual R recycling
## if they are not all of the same length.
vrep_tracker <- function(k, m, n, a, b, t) {
    if (any(k > (t * (m * a + n * b)))) warning("Arguments out of range")
    len_w <- m * a + n * b
    r <- k %% len_w
    r <- ifelse(r == 0, len_w, r)
    is_x <- r <= m*a
    v <- ifelse(is_x, "x", "y")
    r <- ifelse(is_x, r, r - m*a)
    d <- ifelse(is_x, a, b)
    i <- ceiling(r / d)
    list(src_vector = v, src_index = i)
}
```

2. Consider the report: A PHASE 1/2/3, PLACEBO-CONTROLLED, RANDOMIZED, OBSERVER-

BLIND, DOSE-FINDING STUDY TO EVALUATE THE SAFETY, TOLERABILITY, IMMUNOGENICITY, AND EFFICACY OF SARS-COV-2 RNA VACCINE CANDIDATES AGAINST COVID-19 IN HEALTHY INDIVIDUALS (pages 99-101). Let θ be the probability that a subject who fell ill with Covid-19 is from the treatment group and $1-\theta$ the probability that the subject is from the control group. Assuming that 94 subjects fell ill to Covid-19 (with a sample efficacy above 90%) and at most 8 of those 94 subjects were vaccinated. Write a report (Introduction, Methods, Results and Conclusions) assuming:

a. A Beta prior for θ: $p(\theta)$=Beta(a=0.5,b=0.5), where a and b are the shape parameters of the Beta distribution. Plot the prior, likelihood and posterior as function of θ.

```
a = 0.5
b = 0.5
y = 8
n = 94
theta<-seq(0,1,len=1000)
plot(theta,dbeta(theta,a,b),type="l",xlab=expression(theta),
ylab=expression(density),ylim=c(0,14))# prior
lines(theta,dbeta(theta,y+1,n-y+1),type="l",xlab=expression(theta),
ylab=expression(p(theta)),col=2)# likelihood
lines(theta,dbeta(theta,a + y ,b + n - y),type="l",xlab=expression(theta),
ylab="posterior",col=3)# posterior
legend("top",legend=c("prior","likelihood","posterior"),col=c(1,2,3),
cex = 0.9, lwd =1)
```



b. Compute the posterior probability of having a value of $\theta > 0.4118$.

```
1-pbeta(0.4118, a + y, b + n - y)
```

```
## [1] 5.82645e-13
```

c. Compute a 95% credible and confidence intervals.

```
proportion.sample<-y/n
credible<-qbeta(c(0.025, 0.975),  a + y, b + n - y)# posterior
confidence<-c(proportion.sample - 1.96 * sqrt((proportion.sample*(1-proportion.sample))/n),
proportion.sample + 1.96 * sqrt((proportion.sample*(1-proportion.sample))/n))
credible
```

```
## [1] 0.0410614 0.1541568
```

```
confidence
```

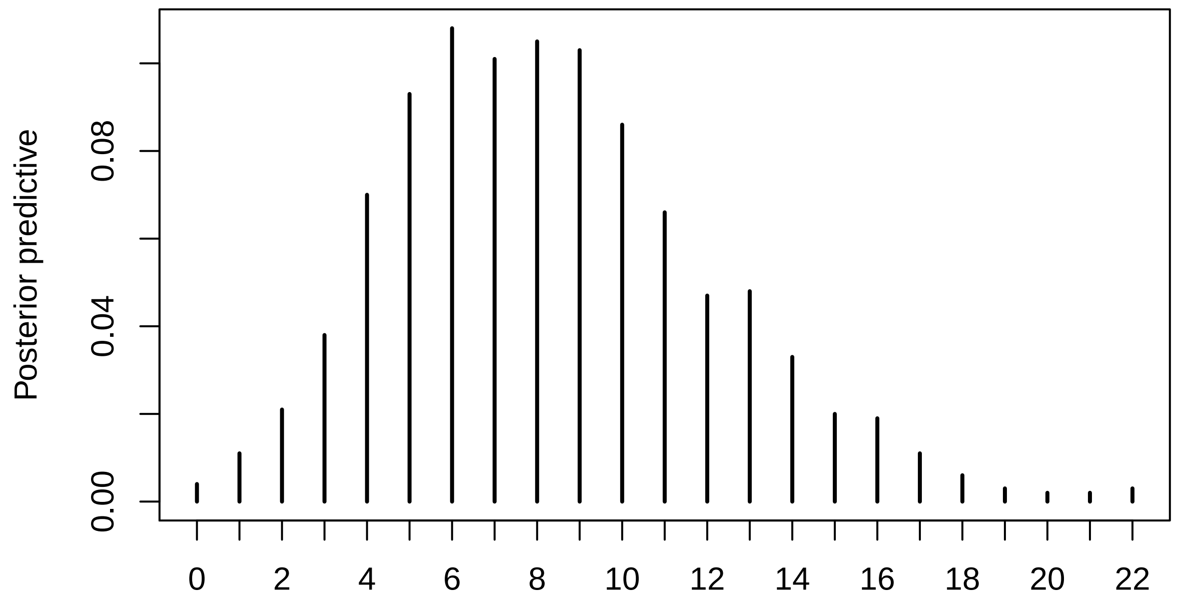
```
## [1] 0.02869607 0.14151670
```

d. Plot the posterior empirical predictive density. If a new sample of 94 subjects with Covid-19 is taken, how many were vaccinated?.

```
predictive.100<- (n*(a + y))/(a + y + b + n - y)
round(predictive.100,digits=0)
```

```
## [1] 8
```

```
theta.s <- rbeta(1000, a + y, b + n - y)
x.p<-rbinom(1000, n, theta.s)# for each theta, random a sample
plot(table(x.p)/1000,xlab="",ylab="Posterior predictive")
```


Posterior predictive