Visual Fractions

14 marks

Visual Fractions

Visual fractions can be used to provide a sense of the size of that fraction (provided it is not too small). In the document `VisualFractions.pdf`, you will find an introduction to some graphical primitives that will allow you to draw some visual fractions using circles (the `VisualFractions.Rmd` contains the code in an Rmarkdown file which you might find helpful).

a. *(10 marks)* Read that document and complete the definition of the function `visualFraction`$(\cdots)$.

```
require("grid")
```

```
## Loading required package: grid
```

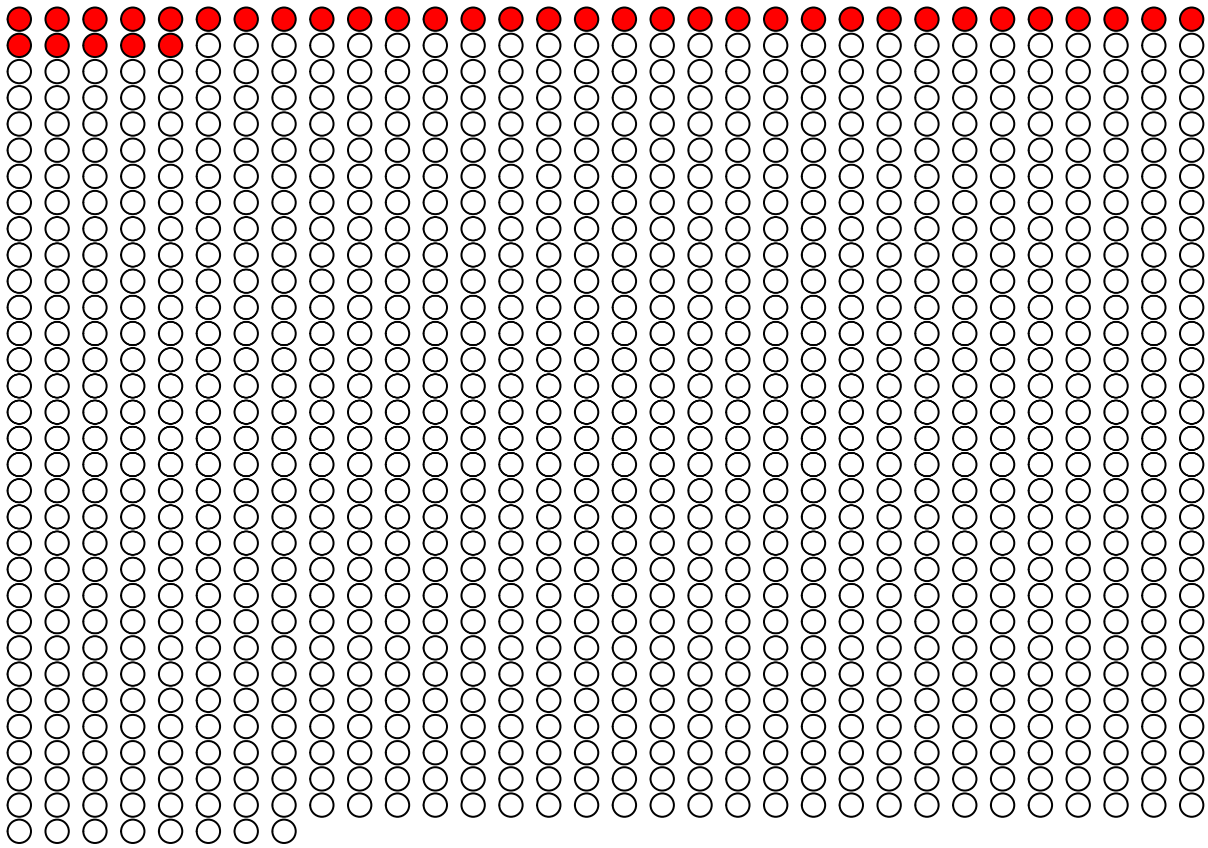
```
xy2grid <- function(x, y) {
  n <- length(x)
  m <- length(y)
  # Return the coordinates of the m x n grid havng
  # locations (x,y) for all x and y
  cbind(rep(x, times=m), rep(y, each=n))
}
#
# For example,
#

## The function will look like:
##
visualFraction <- function(num, # the numerator
                           den, # the denominator
                           numCol="red",
                           # numerator colour
                           denCol="white",
                           # denominator colour
                           random=FALSE,
                           # a logical indicating
                           # whether the numerator values
                           # are to appear at random
                           # locations (if TRUE) or not.
                           ncols = NULL
                           # number of columns to be
                           # used in the array
) {
grid.newpage()
  # begin with some error checking
  #
  #  Check the logical
  if (!is.logical(random))
    stop(paste("random must be TRUE or FALSE, not:",
```

```r
              random))
#
#  Check the numerator
if (!is.numeric(num))
  stop(paste("num must be a number, not", num))
if (length(num) != 1)
  stop(paste("num must be a single number, not of length",
              length(num)))
if (floor(num) != num | num < 0 )
  stop(paste("num must be a non-negative integer, not",
              num))
#
#  Check the denominator
if (!is.numeric(den))
  stop(paste("den must be a number, not", den))
if (length(den) != 1)
  stop(paste("den must be a single number, not of length",
              length(den)))
if (floor(den) != den | den < 0 )
  stop(paste("den must be a non-negative integer, not",
              den))
#
#  Check both
if (num > den)
  stop(paste("num =", num, "> den =", den))
#
#  Check ncols
#
#  Default is NULL, so if user doesn't supply one let's
#  try to make it close to square (default more cols than rows)
if (is.null(ncols)) ncols <- ceiling(sqrt(den))

#  Now check any user supplied value for ncols
if (!is.numeric(ncols))
  stop(paste("ncols must be a number, not", ncols))
if (length(ncols) != 1)
  stop(paste("ncols must be a single number, not of length",
              length(ncols)))
if (floor(ncols) != ncols | ncols < 0 )
  stop(paste("ncols must be a non-negative integer, not",
              ncols))
if (ncols > den )
  stop(paste("ncols =", ncols,"> den =", den))

## If we have ncols columns, we will need
## nrows rows where
nrows <- ceiling(den/ncols)

## We'll also need a radius
## This is size provides spacing for most
radius <- 1/(2*(max(nrows,ncols)+5))

##
```

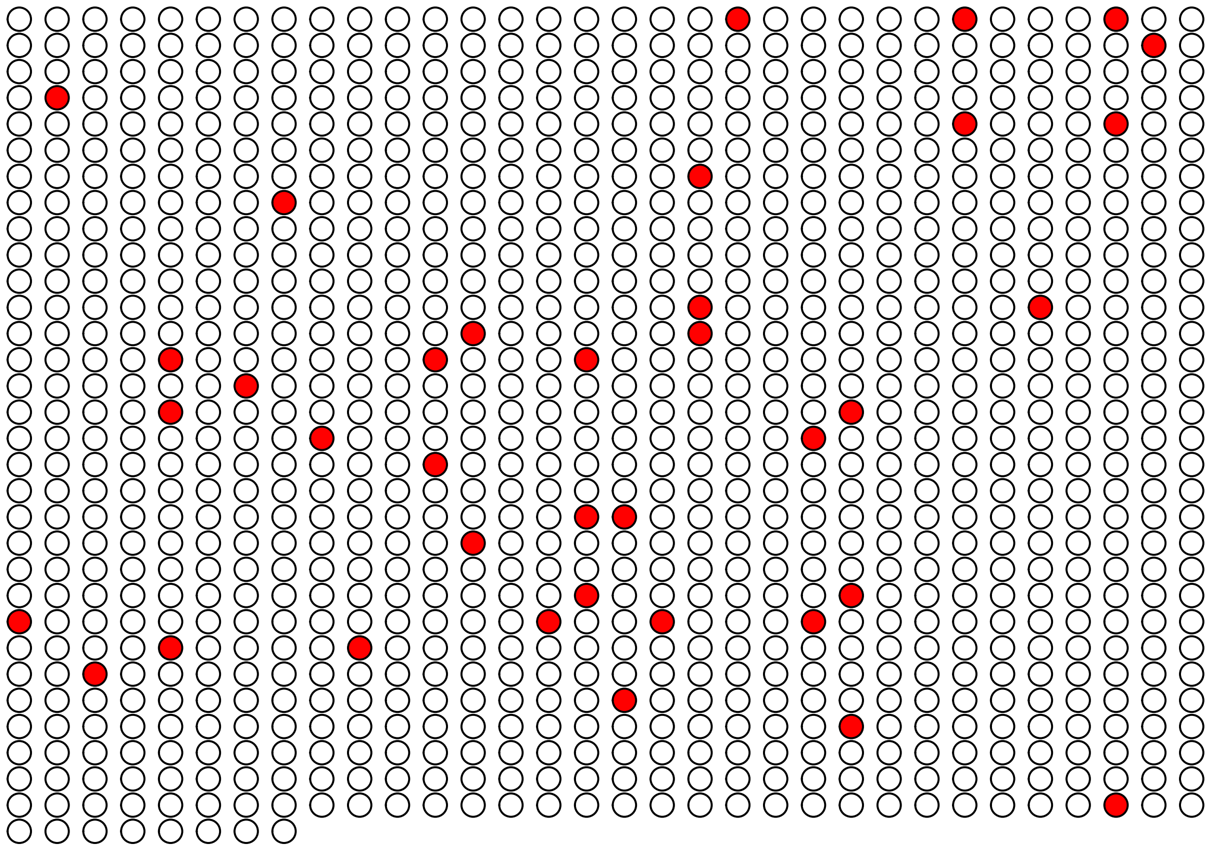
```r
  ## Now it's your turn
  ## The display should be an nrows x ncols array of den circles
  ##
  ## If random=FALSE, the first num circles (from the top left of the
  ## array and proceeding left to right, then top to bottom)
  ## should be coloured numCol, the remainder coloured denCol.
  ##
  ## If random=TRUE, num circles selected at random in the array
  ## should be coloured numCol, the remainder denCol.
  ##
  ## That is, if we index the array 1 to den from top left by row to bottom
  ## right, the indices we would need to colour numCol would be
  if (random) {indices <- sample(1:den, num)} else {indices <- 1:num}
  ##
  ## INSERT YOUR CODE BEL
  coords = xy2grid(1:ncols, 1:nrows)
  coords = coords/(max(nrows, ncols)+1)
  for(i in 1:den){
    if (i %in% indices){
      grid.circle(x=coords[i,1],y=1-coords[i,2],r=radius,gp=gpar(fill=numCol))
    }
    else{
      grid.circle(x=coords[i,1],y=1-coords[i,2],r=radius,gp=gpar(fill=denCol))
    }
  }
}
```

b. *(2 marks)* Show the results of your program on $\frac{3}{100}$ and also on $\frac{37}{1000}$. In each case, show the results **both when** `random` = `FALSE` **and when** `random` = `TRUE`.

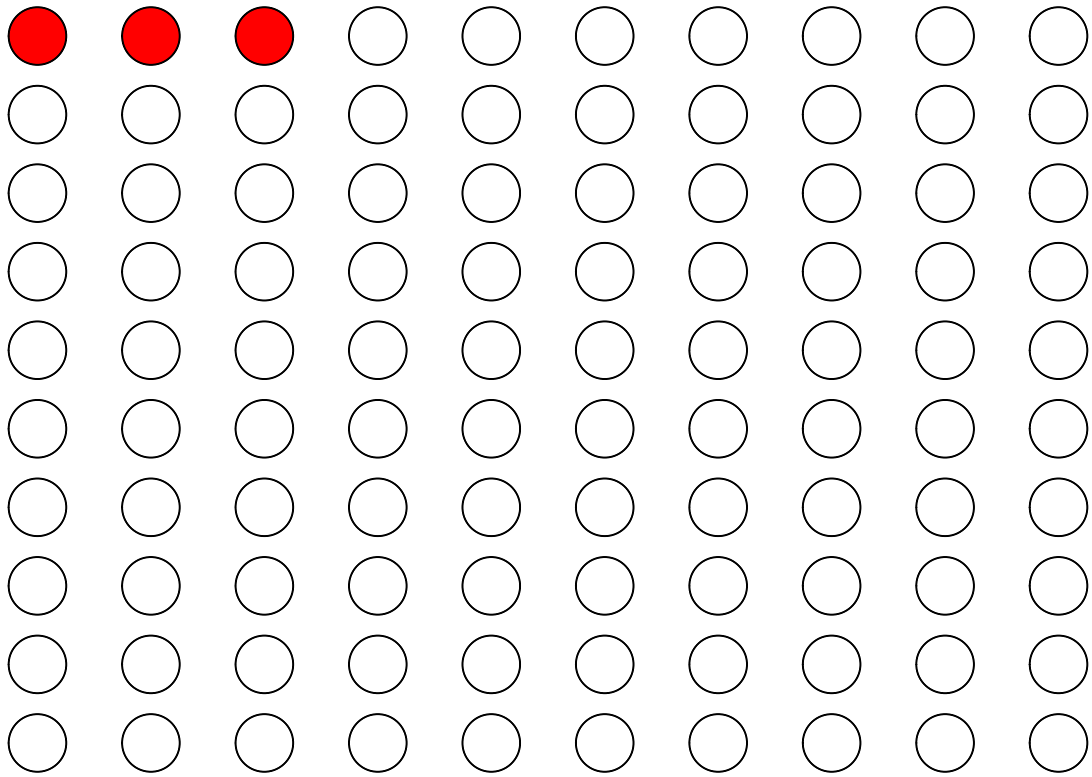
```r
visualFraction(37,1000,random = FALSE)
```



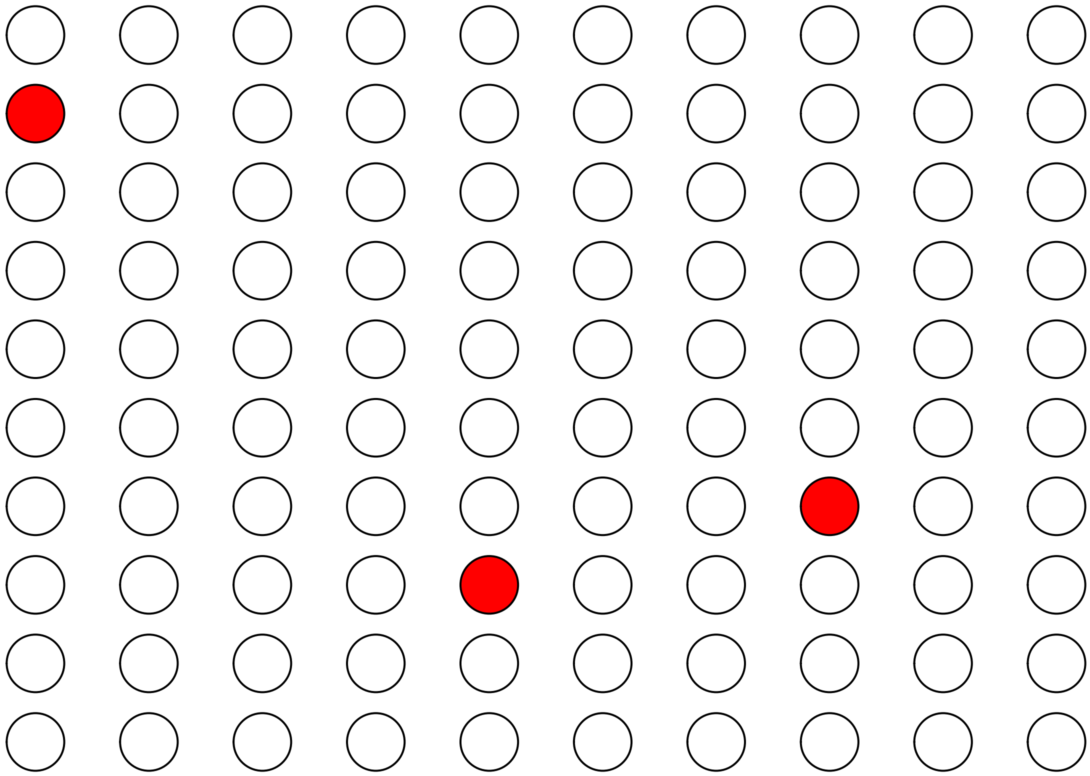
```r
visualFraction(37,1000,random = TRUE)
```



```r
visualFraction(3,100,random = FALSE)
```



```
visualFraction(3,100,random = TRUE)
```



c. *(2 marks)* Explain why the case `random` = `TRUE` might be of interest.

What we are interested here is the density, when there are too many circles and random=FALSE, it's difficult to estimate the ratio of colored points and uncolored point because while it's easy to count how many colored circles are there, it's not easy to estimate how many circle are there in total. however when random=TRUE, we can easily notice there is approximately 1 red circle in every 5*5 square and tell the ratio.