Statistical Computing 1, Stat 590 Homework 5a Name: Answer Key
Fall 2015 Prof. Erik B. Erhardt

Part I. (140 points) Do all calculations in LaTeX + R + knitr. Insert computer text output and graphics to support what you are saying. For this assignment, all R code should well commented and be visible (`echo=TRUE`) in the document where you have written it.

(70pts) **1. Multinomial sampling**

Suppose $\underset{\sim}{X} = \{X_1, X_2, \ldots, X_k\}$ is a discrete random variable. It is easy to see that the joint distribution of $\underset{\sim}{X} = \{X_1, X_2, \ldots, X_k\}$ can be obtained through a series of conditional distributions:

$$\begin{aligned}
\Pr[X_1 = x_1, X_2 = x_2, \ldots, X_k = x_k] &= \Pr[X_k = x_k | X_1 = x_1, X_2 = x_2, \ldots, X_{k-1} = x_{k-1}] \\
&\quad \times \Pr[X_1 = x_1, X_2 = x_2, \ldots, X_{k-1} = x_{k-1}] \\
\Pr[X_1 = x_1, X_2 = x_2, \ldots, X_{k-1} = x_{k-1}] &= \Pr[X_{k-1} = x_{k-1} | X_1 = x_1, X_2 = x_2, \ldots, X_{k-2} = x_{k-2}] \\
&\quad \times \Pr[X_1 = x_1, X_2 = x_2, \ldots, X_{k-2} = x_{k-2}] \\
&\vdots
\end{aligned}$$

That is,

$$\begin{aligned}
\Pr[X_1 = x_1, X_2 = x_2, \ldots, X_k = x_k] &= \Pr[X_1 = x_1] \\
&\quad \times \prod_{i=2}^{k} \Pr[X_i = x_i | X_1 = x_1, X_2 = x_2, \ldots, X_{i-1} = x_{i-1}].
\end{aligned}$$

When $\underset{\sim}{X} = \{X_1, X_2, \ldots, X_k\}$ has a Multinomial$(m, \underset{\sim}{\theta})$ distribution, with $\underset{\sim}{\theta} = (\theta_1, \ldots, \theta_k)$, it can be shown that

$$\begin{aligned}
X_1 &\sim \text{Binomial}(m, \theta_1) \\
X_2 | X_1 = x_1 &\sim \text{Binomial}(m - x_1, \theta_2^*), \\
&\qquad \text{where } \theta_2^* = \theta_2/(1 - \theta_1) \\
&\vdots \\
X_i = x_i | X_1 = x_1, X_2 = x_2, \ldots, X_{i-1} = x_{i-1} &\sim \text{Binomial}(m - (x_1 + \cdots + x_{i-1}), \theta_i^*), \\
&\qquad \text{where } \theta_i^* = \theta_i/(1 - \theta_1 - \cdots - \theta_{i-1}) \\
&\vdots
\end{aligned}$$

To be precise here, you need to recognize that:

- If $x_1 + \cdots + x_{i-1} = m$ (the total sample size), then

$$X_i = x_i | X_1 = x_1, X_2 = x_2, \ldots, X_{i-1} = x_{i-1} \sim \text{Binomial}(m - m, \theta_i^*)$$

means $X_i = 0$ with probability 1. Similarly, $X_{i+1} = \cdots = X_k = 0$ conditional on previous X_is.

- For the last cell X_k, we are constrained so that $x_1 + \cdots + x_k = m$ with probability 1 and the conditional distribution

$$X_k = x_k | X_1 = x_1, X_2 = x_2, \ldots, X_{k-1} = x_{k-1} \sim \text{Binomial}(m - (x_1 + \cdots + x_{k-1}), \theta_k^*),$$
$$\text{where } \theta_k^* = \theta_k / (1 - \theta_1 - \cdots - \theta_{k-1})$$
$$= \theta_k / \theta_k = 1.$$

  This implies that given $X_1, X_2, \ldots, X_{k-1}$, that $x_k = m - (x_1 + \cdots + x_{k-1})$ with probability 1, that is, all "successes" in this Binomial distribution!

This characterization of the multinomial distribution is commonly used to generate random samples from a multinomial distribution, given a routine to generate binomial random variables.

(a) (10 pts) For this problem, I want you to write a function that will generate a *single* multinomial sample given an input sample size m and a probability vector $\underset{\sim}{\theta} = (\theta_1, \ldots, \theta_k)$. Some thoughts:

- You may assume the input arguments are "permissible".
- The algorithm is naturally programmed using a `for()` loop or a `while()` loop.
- You need to be concerned if, for example, $x_1 + \cdots + x_{k-1} = m$ (it can't go above it). If this condition holds, you need to break out of the loop and return, assuming all x_is are initialized to zero. With this in mind, I believe the `while()` loop may be more transparent, because you can loop *while* this condition is not satisfied, given you first generate x_1.
- As an aside, a function `f()` that returns the k-by-1 vector x can have the following structure:

```
## This function demonstrates a particular structure.
## It is not intended to run as it is.

f <- function( inputs ) {
  x <- rep(0, k)
  ...
  x[1] <- s
  ...
  for (i in 2:(k-1)) {
    x[i] <- something
    if ( condition ) {
      some commands
      return(x)
    }
  }
  x[k] <- something
  return(x)
}
```

  The important point here is that whenever you have a `return()` statement in a function, the execution of the function ceases, and the current value of `x` is returned. If you choose to use a `for()` loop to generate a multinomial random variable x then your code will likely mimic this structure.

- You can generate a Binomial(m, p) random variable with `rbinom()`. Or, if you prefer, you can write your own Binomial generator.

Solution:

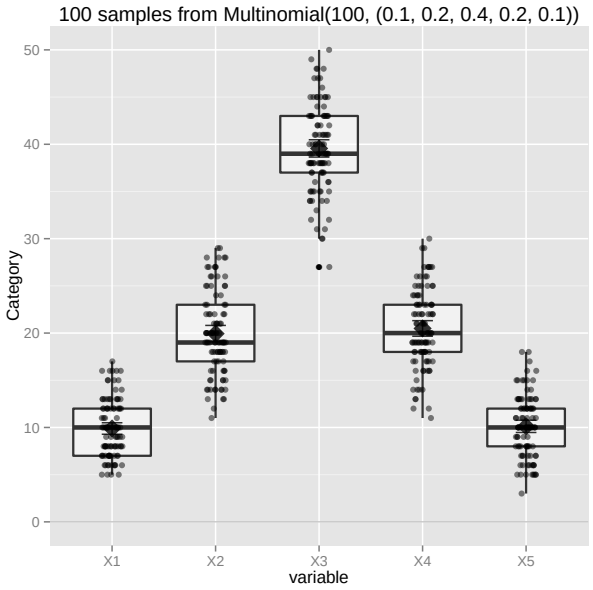
```
# A helpful function for checking permissibility of theta
f.dmulti.error.check <- function(theta) {
  if (any(theta < 0) | any(theta > 1)) {
    warning("Proportions in theta must between 0 and 1")
    return(1)
  }
  if (abs(sum(theta) - 1) > 1e-10 ) {
    warning("Sum of theta vector is not 1")
    return(1)
  }
  if (length(theta) < 2 ) {
    warning("theta must be a vector of at least 2")
    return(1)
  }
  return(0)
}

## Multinomial random deviates via conditional binomial samples
# n = number of samples to draw
# m = total count
# theta = row vector of proportions.
# returns a matrix with each sample in a row.
f.dmulti <- function(theta, m, n = 1) {

  # basic error checking
  if(f.dmulti.error.check(theta) == 1) { return(NULL) }
  if (m <= 0) {
    warning("m must be greater than 0")
    return(NULL)
  }

  k <- length(theta) # number of categories
  x <- matrix(NA, ncol = k, nrow = n) # init return matrix

  # conditional probatilities for all steps
  theta.cond <- c(theta[1], theta[2:k]/(1 - cumsum(theta)[1:(k - 1)]))

  # for each of the first k - 1 bins
  for (i.k in 1:(k - 1)) {
    # determine how many observations we have left to allocate
    m.temp <- m - apply(x, 1, sum, na.rm = TRUE) # remaining sample size to draw
    # draw a binonmial rv with m.temp trials with the binomial conditional probabilities
    x[, i.k] <- rbinom(n, m.temp, theta.cond[i.k])
  }
  # put the rest of the sample in the last bin
  m.temp <- m - apply(x, 1, sum, na.rm = TRUE) # remaining sample size to draw
  x[, k] <- m.temp

  return(x)
```

}

(b) (10 pts) Present some visual evidence that the function above "works correctly". What you might choose to do is generate 100 or so samples $\underset{\sim}{X} = \{X_1, X_2, \ldots, X_k\}$ for a given m and $\underset{\sim}{\theta} = (\theta_1, \ldots, \theta_k)$. Noting that $X_i \sim \text{Binomial}(m, \theta_i)$, you might make a histogram of the generated counts X_i for the ith cell and compare the shape to that of the binomial pdf (using `pbinom()`) for $i = 1, \ldots, k$. You might think of how to turn counts for the histogram into proportions and plot those with the binomial probabilities (together).

Solution: Looks perfect.

```
theta <- c(0.1, 0.2, 0.4, 0.2, 0.1)
x <- f.dmulti(theta, m = 100, n = 100)

library(reshape2)
x.long <- melt(data.frame(x))
## No id variables; using all as measure variables
# Plot the data using ggplot
library(ggplot2)
p <- ggplot(x.long, aes(x = variable, y = value))
p <- p + geom_hline(aes(yintercept = 0),
                    colour = "black", linetype = "solid", size = 0.2, alpha = 0.3)
# boxplot, size=.75 to stand out behind CI
p <- p + geom_boxplot(size = 0.75, alpha = 0.5)
# points for observed data
p <- p + geom_point(position = position_jitter(w = 0.1, h = 0), alpha = 0.5)
# diamond at mean for each group
p <- p + stat_summary(fun.y = mean, geom = "point", shape = 18, size = 6, alpha = 0.8)
# confidence limits based on normal distribution
p <- p + stat_summary(fun.data = "mean_cl_normal", geom = "errorbar",
                      width = .2, alpha = 0.8)
p <- p + labs(title = "100 samples from Multinomial(100, (0.1, 0.2, 0.4, 0.2, 0.1))")
p <- p + ylab("Frequency")
p <- p + ylab("Category")
print(p)
```



(c) (20 pts) For each combination of

n = number of Multinomial samples = 500, 1000

m = Multinomial sample size = 25, 100, 500

k = number of cells = 3, 6, 9, 12, 15

compute n samples from a multinomial distribution with given m and $\theta_1 = \cdots = \theta_k = k^{-1}$ (equal cell probabilities). Keeping a record of the total clock time needed to complete this task for a given n, m, and k. For each value of n, make a plot of the time(m, k) values. For example, with time on the vertical axis and k on the horizontal axis, plot the times where each m is grouped and connected a line of different colors and line types (solid, dashed, etc.).
Discussed the results. You might also consider plotting time$(m, k)/n$ in the same manner (average time per sample).
Note that there are at least three ways to measure the clock time for a process:

- see the example at the bottom of help page for `?proc.time`
- see package `rbenchmark`
- see package `microbenchmark`

Solution: Perform benchmarking:

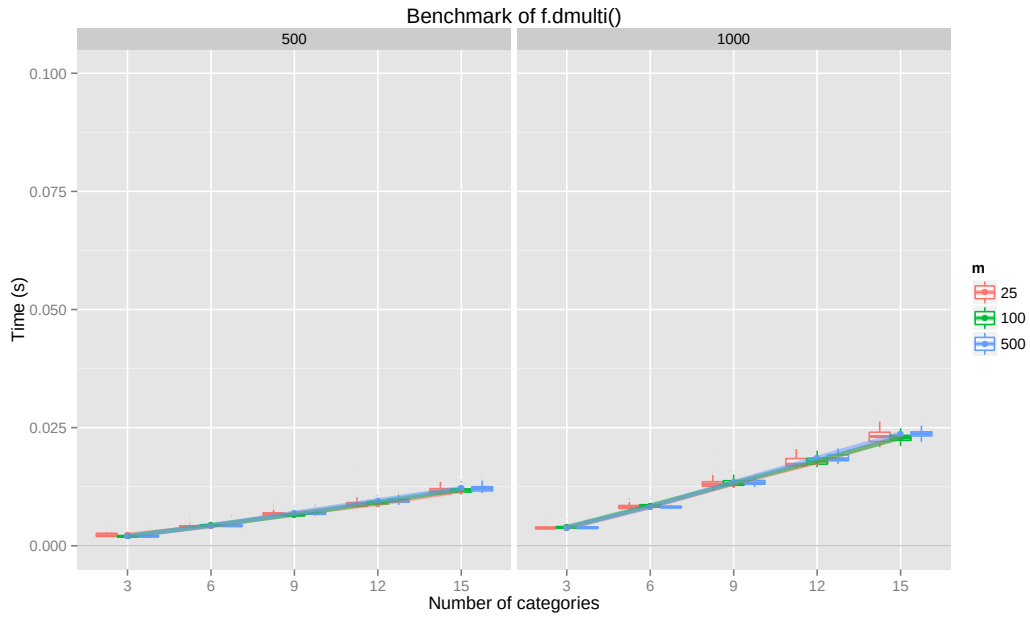
```
library(microbenchmark)
R <- 100 # reps of each condition
time.f.dmulti <- NULL # init location to store time results
for (i.n in c(500, 1000)) {
  for (i.m in c(25, 100, 500)) {
    for (i.k in c(3, 6, 9, 12, 15)) {
      # define theta
      theta <- rep(1/i.k, i.k)
      # perform benchmark
      mb.out <- microbenchmark(f.dmulti(theta, m = i.m, n = i.n), times = R)
```

```
      # bind results to previous results
      time.f.dmulti <- rbind(time.f.dmulti
                             ,data.frame(
                                   n = i.n
                                 , m = i.m
                                 , k = i.k
                                 , time = mb.out$time/1e9))
                                   # convert nanoseconds to seconds

    }
  }
}
time.f.dmulti$n <- factor(time.f.dmulti$n)
time.f.dmulti$m <- factor(time.f.dmulti$m)
time.f.dmulti$k <- factor(time.f.dmulti$k)
```

Plot results:

```
library(plyr)
# Calculate the cell means for each (n, m, k) combination
#   this is a 5% trimmed mean (5% trimmed off both ends of distribution)
time.f.dmulti.mean.nmk <- ddply(time.f.dmulti, .(n, m, k), summarise, mean = mean(time, trim = (
#time.f.dmulti.mean.nmk

# Interaction plots, ggplot
p <- ggplot(time.f.dmulti, aes(x = k, y = time, colour = m))
p <- p + geom_hline(aes(yintercept = 0), colour = "black"
                  , linetype = "solid", size = 0.2, alpha = 0.3)
p <- p + geom_boxplot(alpha = 0.5, outlier.size=0.1)
p <- p + geom_point(data = time.f.dmulti.mean.nmk, aes(y = mean), size = 2)
p <- p + geom_line(data = time.f.dmulti.mean.nmk, aes(y = mean, group = m), size = 1.5, alpha =
p <- p + scale_y_continuous(limits = c(0,0.1))  # same scale for both plots
p <- p + facet_grid(. ~ n)
p <- p + labs(title = "Benchmark of f.dmulti()")
p <- p + ylab("Time (s)")
p <- p + xlab("Number of categories")
print(p)
```

```
## Warning:  Removed 1 rows containing non-finite values (stat_boxplot).
## Warning:  Removed 3 rows containing non-finite values (stat_boxplot).
```



The time per sample is plotted. There is a linear increase with number of categories k and number of samples n. However, there is little difference for increases in the size of each sample m.

(d) (10 pts) Consider the following pseudo-code, where $\underset{\sim}{\theta} = (\theta_1, \ldots, \theta_k)$ is a vector of probabilities with $\theta_i > 0$ and $\sum_i^k \theta_i = 1$. Suppose I do the following, given $\underset{\sim}{\theta}$ is defined: First, define $\underset{\sim}{\theta}$ as a 1-by-k vector and define scalar m, then define the following function:

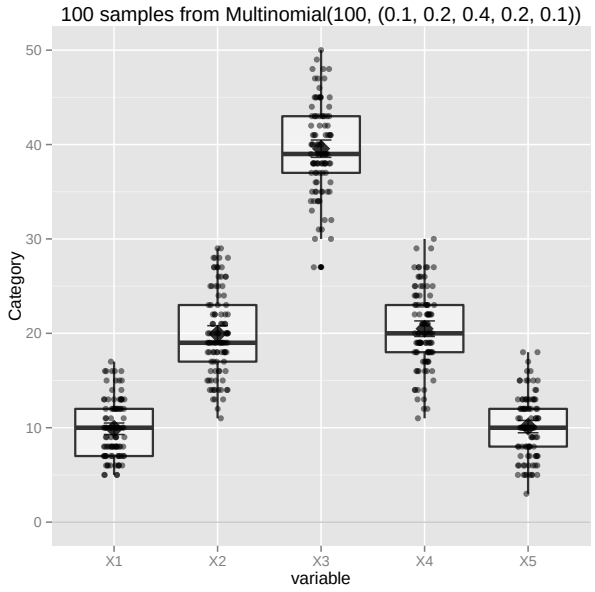
```
f.dmultigen <- function(m, theta) {
  theta.c <- c(0, cumsum(theta))
  u <- runif(m)
  x <- hist(u, breaks = theta.c, plot = FALSE)$counts
  return(x)
}

# or, (probably) faster, all n samples at once
f.dmultigen2 <- function(theta, m, n = 1) {
  theta.c <- c(0, cumsum(theta))
  x <- t(apply(matrix(.bincode(runif(m * n), breaks = theta.c), nrow = n), 1, tabulate))
  return(x)
}
```

Explain what this code is doing step-by-step and argue why this algorithm generates a single Multinomial$(m, \underset{\sim}{\theta})$ sample.

Solution: Function `f.multigen()` works because it randomly distributes m items to k bins, with bin i having probability θ_i.
And it works correctly.

```
theta <- c(0.1, 0.2, 0.4, 0.2, 0.1)
x <- f.dmultigen2(theta, m = 100, n = 100)
```

```
## Error in eval(expr, envir, enclos):  could not find function "f.dmultigen2"
library(reshape2)
x.long <- melt(data.frame(x))
## No id variables; using all as measure variables
# Plot the data using ggplot
library(ggplot2)
p <- ggplot(x.long, aes(x = variable, y = value))
p <- p + geom_hline(aes(yintercept = 0),
                    colour = "black", linetype = "solid", size = 0.2, alpha = 0.3)
# boxplot, size=.75 to stand out behind CI
p <- p + geom_boxplot(size = 0.75, alpha = 0.5)
# points for observed data
p <- p + geom_point(position = position_jitter(w = 0.1, h = 0), alpha = 0.5)
# diamond at mean for each group
p <- p + stat_summary(fun.y = mean, geom = "point", shape = 18, size = 6, alpha = 0.8)
# confidence limits based on normal distribution
p <- p + stat_summary(fun.data = "mean_cl_normal", geom = "errorbar",
                      width = .2, alpha = 0.8)
p <- p + labs(title = "100 samples from Multinomial(100, (0.1, 0.2, 0.4, 0.2, 0.1))")
p <- p + ylab("Frequency")
p <- p + ylab("Category")
print(p)
```



(e) (10 pts) Using the new code (or a modification of it) in part (d), repeat the analysis done in part (c) and compare the results. Which method, if either, is faster for these combinations of n, m, and k?

Solution: Perform benchmarking:

```
library(microbenchmark)
# R <- 100 # reps of each condition
time.f.dmultigen2 <- NULL # init location to store time results
```

```
for (i.n in c(500, 1000)) {
  for (i.m in c(25, 100, 500)) {
    for (i.k in c(3, 6, 9, 12, 15)) {
      # define theta
      theta <- rep(1/i.k, i.k)
      # perform benchmark
      mb.out <- microbenchmark(f.dmultigen2(theta, m = i.m, n = i.n), times = R)

      # bind results to previous results
      time.f.dmultigen2 <- rbind(time.f.dmultigen2
                           ,data.frame(
                                 n = i.n
                               , m = i.m
                               , k = i.k
                               , time = mb.out$time/1e9))
                                 # convert nanoseconds to seconds

    }
  }
}
## Error in microbenchmark(f.dmultigen2(theta, m = i.m, n = i.n), times = R): could
not find function "f.dmultigen2"
time.f.dmultigen2$n <- factor(time.f.dmultigen2$n)
time.f.dmultigen2$m <- factor(time.f.dmultigen2$m)
time.f.dmultigen2$k <- factor(time.f.dmultigen2$k)
```

Plot results:

```
library(plyr)
# Calculate the cell means for each (n, m, k) combination
#   this is a 5% trimmed mean (5% trimmed off both ends of distribution)
time.f.dmultigen2.mean.nmk <- ddply(time.f.dmultigen2, .(n, m, k), summarise, mean = mean(time,
## Error in if (empty(.data)) return(.data):  missing value where TRUE/FALSE needed
#time.f.dmultigen2.mean.nmk

# Interaction plots, ggplot
p <- ggplot(time.f.dmultigen2, aes(x = k, y = time, colour = m))
## Error:  ggplot2 doesn't know how to deal with data of class list
p <- p + geom_hline(aes(yintercept = 0), colour = "black"
                  , linetype = "solid", size = 0.2, alpha = 0.3)
p <- p + geom_boxplot(alpha = 0.5, outlier.size=0.1)
p <- p + geom_point(data = time.f.dmultigen2.mean.nmk, aes(y = mean), size = 2)
## Error in do.call("layer", list(mapping = mapping, data = data, stat = stat, :  object
'time.f.dmultigen2.mean.nmk' not found
p <- p + geom_line(data = time.f.dmultigen2.mean.nmk, aes(y = mean, group = m), size = 1.5, alpl
## Error in do.call("layer", list(mapping = mapping, data = data, stat = stat, :  object
'time.f.dmultigen2.mean.nmk' not found
p <- p + scale_y_continuous(limits = c(0,0.1))  # same scale for both plots
p <- p + facet_grid(. ~ n)
p <- p + labs(title = "Benchmark of f.dmultigen2()")
p <- p + ylab("Time (s)")
```

```
p <- p + xlab("Number of categories")
print(p)
## Error in layout_base(data, cols, drop = drop):  At least one layer must contain
all variables used for facetting
```

The time per sample is plotted. There is no increase with number of categories k but linear increase with number of samples n. and size of each sample m.

(f) (10 pts) Devise a good way of modifying the strategy in part (a) to generate n Multinomial samples per call to the function. That is, pass n as input to the function and return n samples.
Do the same for the strategy in part (d).
Does this modification impact time(m, k), the time needed to generate n samples from Multinomial$(m, \underset{\sim}{\theta})$? If so, how (you don't have to redo part (c), just consider a few cases)?

Solution: Already did this, and presumably my vectorized code is faster than using a `for()` loop.

(40pts) **2. Multinomial hypothesis testing**

Suppose $\underset{\sim}{X} = \{X_1, X_2, \ldots, X_k\}$ has a Multinomial$(m, \underset{\sim}{\theta})$ distribution, where $\underset{\sim}{\theta} = (\theta_1, \ldots, \theta_k)$ such that $\theta_i > 0$ and $\sum_i^k \theta_i = 1$. We wish to test $H_0 : \theta_1 = \theta_{01}, \ldots, \theta_k = \theta_{0k}$, versus $H_1 :$ not H_0. Two standard test statistics are the Pearson statistic

$$P = \sum_{i=1}^{k} \frac{(x_i - m\theta_{0i})^2}{m\theta_{0i}}$$

and the likelihood ratio statistic

$$G^2 = 2\sum_{i=1}^{k} x_i \log_e\left(\frac{x_i}{m\theta_{0i}}\right),$$

where $0 \log_e(0) \equiv 0$. For large n, if H_0 is true, then both P and $G^2 \dot{\sim} \chi^2_{k-1}$. A standard large-sample test is to reject H_0 based on the p-value

$$\begin{aligned} \text{p-value}(P) &= \Pr(\chi^2_{k-1} > P_{\text{obs}}) \qquad \text{or} \\ \text{p-value}(G^2) &= \Pr(\chi^2_{k-1} > G^2_{\text{obs}}), \end{aligned}$$

that is, the area under the χ^2_{k-1} curve to the right of observed values of P and G^2.

(a) (10 pts) Write separate functions to compute P and G^2 given vector inputs $\underset{\sim}{X} = \{X_1, X_2, \ldots, X_k\}$ and $\underset{\sim}{\theta}_0 = (\theta_{01}, \ldots, \theta_{0k})$.

- assuming input vectors are "permissible", that is, $X_i \geq 0$ and $\theta_{0i} > 0$.
- calculations should be based on vector or matrix calculations, not `for()` loops
- for G^2, since $x_i \log_e(x_i) \equiv 0$ when $x_i = 0$, you might think of identifying the elements of $\{X_1, X_2, \ldots, X_k\}$ that are positive and basing the statistic solely on

these elements. Alternatively, you might think of slightly changing the definition to have components $x_i \log_e((x_i + \varepsilon)/m\theta_{0i})$ where ε is a very small number, say $\varepsilon = 10^{-10}$.

Solution:

```
# A helpful function for checking permissibility of x and theta
f.error.check <- function(theta, x) {
  # basic error checking
  if (dim(x)[2] != dim(theta)[2]) {
    warning("Number columns for x and theta must be the same")
    return(1)
  }
  if (any(x < 0)) {
    warning("Counts in x must all be nonnegative")
    return(1)
  }
  if (any(theta < 0) | any(theta > 1)) {
    warning("Proportions in theta must between 0 and 1")
    return(1)
  }
  if (abs(sum(theta) - 1) > 1e-10 ) {
    warning("Sum of theta vector is not 1")
    return(1)
  }
  if (length(theta) < 2 ) {
    warning("theta must be a vector of at least 2")
    return(1)
  }
  return(0)
}

# Pearson chi-square statistic and chi-sq p-value
f.P <- function(theta, x) {
  # convert x to a matrix for generality (each row a sample, columns for categories)
  if (is.vector(x) == TRUE) {
    x <- matrix(x, nrow = 1 )
  }
  theta <- matrix(theta, nrow = 1)

  # basic error checking
  if(f.error.check(theta, x) == 1) { return(NULL) }

  m <- apply(x, 1, sum)  # sum of counts in x
  k <- dim(x)[2]         # number of categories
  ones <- matrix(1, ncol = 1, nrow = nrow(x)) # vector of 1s to make multiple rows of theta in
  P <- apply((x - m * ones %*% theta)^2 / (m * ones %*% theta), 1, sum)  # Pearson P statistic
  p.value <- 1 - pchisq(P, k - 1)  # p-value based on chi2 with k-1 df

  temp <- list(P = P, p.value = p.value)
  return(temp)
}
```

```
# Likelihood ratio statistic and chi-sq p-value
f.G2 <- function(theta, x) {
  # convert x to a matrix for generality (each row a sample, columns for categories)
  if (is.vector(x) == TRUE) {
    x <- matrix(x, nrow = 1 )
  }
  theta <- matrix(theta, nrow = 1)

  # basic error checking
  if(f.error.check(theta, x) == 1) { return(NULL) }

  m <- apply(x, 1, sum)  # sum of counts in x
  k <- dim(x)[2]         # number of categories
  ones <- matrix(1, ncol = 1, nrow = nrow(x)) # vector of 1s to make multiple rows of theta in
  G2 <- 2 * apply(x * log((x + 1e-14)/(m * ones %*% theta)), 1, sum)  # Likelihood ratio G2 sta
  p.value <- 1 - pchisq(G2, k - 1)  # p-value based on chi2 with k-1 df

  temp <- list(G2 = G2, p.value = p.value)
  return(temp)
}
```

(b) (10 pts) Write a script where you can input the cell counts $\underset{\sim}{X} = \{X_1, X_2, \ldots, X_k\}$ and the null probabilities $\underset{\sim}{\theta}_0 = (\theta_{01}, \ldots, \theta_{0k})$. Have your script check whether the input values are "permissible", that is, you need to input k non-negative counts $X_i \geq 0$ and k probabilities $\theta_{01} > 0$ that sum to 1. Given permissible input, the script should call your functions to compute P and G^2, then print out the observed values of P and G^2 with their p-values. The output should include labels with the printing, that is, return summaries and a message to the screen/command window. You can use the `pchisq()` function to compute the χ^2_{k-1} cdf.

Solution: See below.

(c) (10 pts) The Dean of Arts and Sciences at a certain university established grading guidelines of 10% As and Fs, 20% Bs and Ds, and 40% Cs for his faculty. In a statistics class consisting of 117 students, the number of individuals receiving the five letter grades were as follows:

grade	A	B	C	D	F
number	16	50	31	11	9

Does it appear that the professor is following the Dean's recommendation, that is, does it appear that the grades are a random sample from a population with the recommended grade distribution? Use the code designed in the earlier part of the problem to answer this question.

Solution: From the Pearson and Likelihood ratio statistics and p-values (both much less than 0.05), there is strong evidence against the null hypothesis in favor of an alternative that the grade distribution does not follow the Dean's guidelines.

```
x     <- c( 16,  50,  31,  11,   9)
theta <- c(0.1, 0.2, 0.4, 0.2, 0.1)
```

```
out.P <- f.P(theta, x)
out.G2 <- f.G2(theta, x)
```

Pearson:

	P	p.value
1	44.346154	0.000000

Likelihood ratio:

	G2	p.value
1	39.078089	0.000000

(d) (10 pts) Generalize your P and G^2 functions so that they can compute each statistic for multiple samples, given by rows of a matrix. Illustrate their use on the data in this table.

grade	A	B	C	D	F
Prof 1	16	50	31	11	9
Prof 2	10	23	22	20	7
Prof 3	21	10	42	3	1
Prof 4	3	12	31	16	0

Solution: The data suggest that only the grading distribution of Prof 2 is consistent (p-value > 0.05) with the Dean's guidelines.

```
x <- matrix(c(  16, 50, 31, 11,  9
              , 10, 23, 22, 20,  7
              , 21, 10, 42,  3,  1
              ,  3, 12, 31, 16,  0)
            , ncol=5, byrow=TRUE)
theta <- c(0.1, 0.2, 0.4, 0.2, 0.1)

out.P <- f.P(theta, x)
out.G2 <- f.G2(theta, x)
```

Pearson:

	P	p.value
1	44.346154	0.000000
2	7.573171	0.108526
3	44.753247	0.000000
4	10.459677	0.033357

Likelihood ratio:

	G2	p.value
1	39.078089	0.000000
2	7.676726	0.104164
3	45.659082	0.000000
4	16.848874	0.002068

(30pts) **3. Goodness-of-fit and upper-tail probability approximation**

For the goodness-of-fit test, a standard approach is to reject H_0 if

$$P \geq \chi^2_{k-1,1-\alpha}$$

where

$$\Pr[\chi^2_{k-1} \geq \chi^2_{k-1,1-\alpha}] = \alpha,$$

that is, $\chi^2_{k-1,1-\alpha}$ is the upper α percentile of the χ^2_{k-1} distribution, and α is the desired size of the test. A similar rule is used for G^2. Because the χ^2_{k-1} is only an approximation, neither

$$\Pr(P \geq \chi^2_{k-1}) \quad \text{nor} \quad \Pr(G^2 \geq \chi^2_{k-1})$$

may not be exactly equal to α in small samples. This problem seeks to answer "how good is the approximation?" for a few settings. In practice, $\alpha = 0.01$, 0.05, or 0.10, so let's restrict attention to these three cases. The `qchisq()` function can be used to compute the quantile

$$t = \chi^2_{k-1,1-\alpha}$$

for any choices of k and α.

I would like you to write a script that uses the crude MC estimate (using the same stream of random numbers) to calculate

$$\Pr(P \geq \chi^2_{k-1}) \quad \text{and} \quad \Pr(G^2 \geq \chi^2_{k-1})$$

for the specified values of t (based on α and k), using the same k, m, and $\underset{\sim}{\theta}$ combinations considered in Problem 2, and the number n of simulated samples so that the margin-of-error (MOE) in the estimated probabilities does not exceed (approximately) 0.01.

Summarize the results in tabular form, giving values for α (nominal level), m, k, $\underset{\sim}{\theta}$, plus the two crude MC estimates plus their estimated standard error (that is, the square root of the estimated variance). Also give the estimate for $(\Pr(P \geq t) - \Pr(G^2 \geq t))$ and its standard error. For these parameter combinations, does it appear that the upper-tail approximation to either P or G^2 is accurate? Discuss the results.

Solution: We are effectively asking for the mean and SE(mean) of the p-values calculated using the Pearson and Likelihood methods.

First I set up the simluation and create data.frame `df.cv` with the results.

```
# init location to store results
n.runs <- 3*5*3
df.gof.tail.prob <- data.frame(m             = rep(NA, n.runs)
                             , k             = rep(NA, n.runs)
                             , alpha         = rep(NA, n.runs)
                             , P.p           = rep(NA, n.runs)
                             , P.p.SE        = rep(NA, n.runs)
                             , G2.p          = rep(NA, n.runs)
```

```
                                , G2.p.SE        = rep(NA, n.runs)
                                , diff.P.G2.p    = rep(NA, n.runs)
                                , diff.P.G2.p.SE = rep(NA, n.runs)
                                 )

R <- 1e4 # reps of each condition

i.runs <- 0
for (i.m in c(25, 100, 500)) {
  for (i.k in c(3, 6, 9, 12, 15)) {
    ## DEBUG
    # i.m=25; i.k=3

    # define theta
    theta <- rep(1/i.k, i.k)

    # draw samples
    sam <- f.dmulti(theta, m = i.m, n = R)

    # calculate statistics
    out.P  <- f.P (theta, sam)
    out.G2 <- f.G2(theta, sam)
    diff.P.G2.p <- out.P$p.value - out.G2$p.value

    for (i.a in c(0.01, 0.05, 0.1)) {
      i.runs <- i.runs + 1

      # Converting p-values to 1 or 0 for whether less than alpha
      P.Hyp.decision <- as.numeric(out.P$p.value < i.a)
      G2.Hyp.decision <- as.numeric(out.G2$p.value < i.a)

      # Est and SEs
      P.p     <- mean(P.Hyp.decision)
      P.p.SE  <- sd(P.Hyp.decision) / sqrt(R)
      G2.p    <- mean(G2.Hyp.decision)
      G2.p.SE <- sd(G2.Hyp.decision) / sqrt(R)

      # Diff
      diff.P.G2.p    <- P.p - G2.p
      diff.P.G2.p.SE <- sqrt(P.p.SE^2 + G2.p.SE^2)

      # data.frame with results
      df.gof.tail.prob[i.runs, ] <- c(i.m, i.k, i.a
                                      , P.p
                                      , P.p.SE
                                      , G2.p
                                      , G2.p.SE
                                      , diff.P.G2.p
                                      , diff.P.G2.p.SE
                                       )

    }
  }
```

```
}
str(df.gof.tail.prob)
```

```
## 'data.frame': 45 obs. of  9 variables:
##  $ m             : num  25 25 25 25 25 25 25 25 25 25 ...
##  $ k             : num  3 3 3 6 6 6 9 9 9 12 ...
##  $ alpha         : num  0.01 0.05 0.1 0.01 0.05 0.1 0.01 0.05 0.1 0.01 ...
##  $ P.p           : num  0.0092 0.0513 0.0997 0.0076 0.051 ...
##  $ P.p.SE        : num  0.000955 0.002206 0.002996 0.000869 0.0022 ...
##  $ G2.p          : num  0.0134 0.0676 0.0997 0.0165 0.0804 ...
##  $ G2.p.SE       : num  0.00115 0.00251 0.003 0.00127 0.00272 ...
##  $ diff.P.G2.p   : num  -0.0042 -0.0163 0 -0.0089 -0.0294 -0.0302 -0.0087 -0.0294 -0.0557 -0.008
##  $ diff.P.G2.p.SE: num  0.00149 0.00334 0.00424 0.00154 0.0035 ...
```

Convert data frame to long format and create a plot.

```
# easier to do manually by creating a seperate df for each statistic
#   and stacking them

df.gof.tail.prob.temp <- df.gof.tail.prob
colnames(df.gof.tail.prob.temp)[4:9] <- c("p", "p.SE")

df.gof.tail.prob.long <-
  rbind(
    data.frame(df.gof.tail.prob.temp[,c(1, 2, 3, 4, 5)], Stat = "P")
  , data.frame(df.gof.tail.prob.temp[,c(1, 2, 3, 6, 7)], Stat = "G2")
  , data.frame(df.gof.tail.prob.temp[,c(1, 2, 3, 8, 9)], Stat = "diff.P.G2")
  )


df.gof.tail.prob.long$m     <- factor(df.gof.tail.prob.long$m    )
df.gof.tail.prob.long$k     <- factor(df.gof.tail.prob.long$k    )
df.gof.tail.prob.long$alpha <- factor(df.gof.tail.prob.long$alpha)

str(df.gof.tail.prob.long)
```

```
## 'data.frame': 135 obs. of  6 variables:
##  $ m    : Factor w/ 3 levels "25","100","500": 1 1 1 1 1 1 1 1 1 1 ...
##  $ k    : Factor w/ 5 levels "3","6","9","12",..: 1 1 1 2 2 2 3 3 3 4 ...
##  $ alpha: Factor w/ 3 levels "0.01","0.05",..: 1 2 3 1 2 3 1 2 3 1 ...
##  $ p    : num  0.0092 0.0513 0.0997 0.0076 0.051 ...
##  $ p.SE : num  0.000955 0.002206 0.002996 0.000869 0.0022 ...
##  $ Stat : Factor w/ 3 levels "P","G2","diff.P.G2": 1 1 1 1 1 1 1 1 1 1 ...
```

Plot observed tail probability with what was expected. Error bars are ± 1 SE.

Summary of results: At $m = 500$, both statistics have converged to the correct tail probability. P does not appear biased for either sample size or number of categories. The G^2 statistic appears conservative (larger p-values) for smaller sample sizes, and the bias increases as the number of categories k increases.

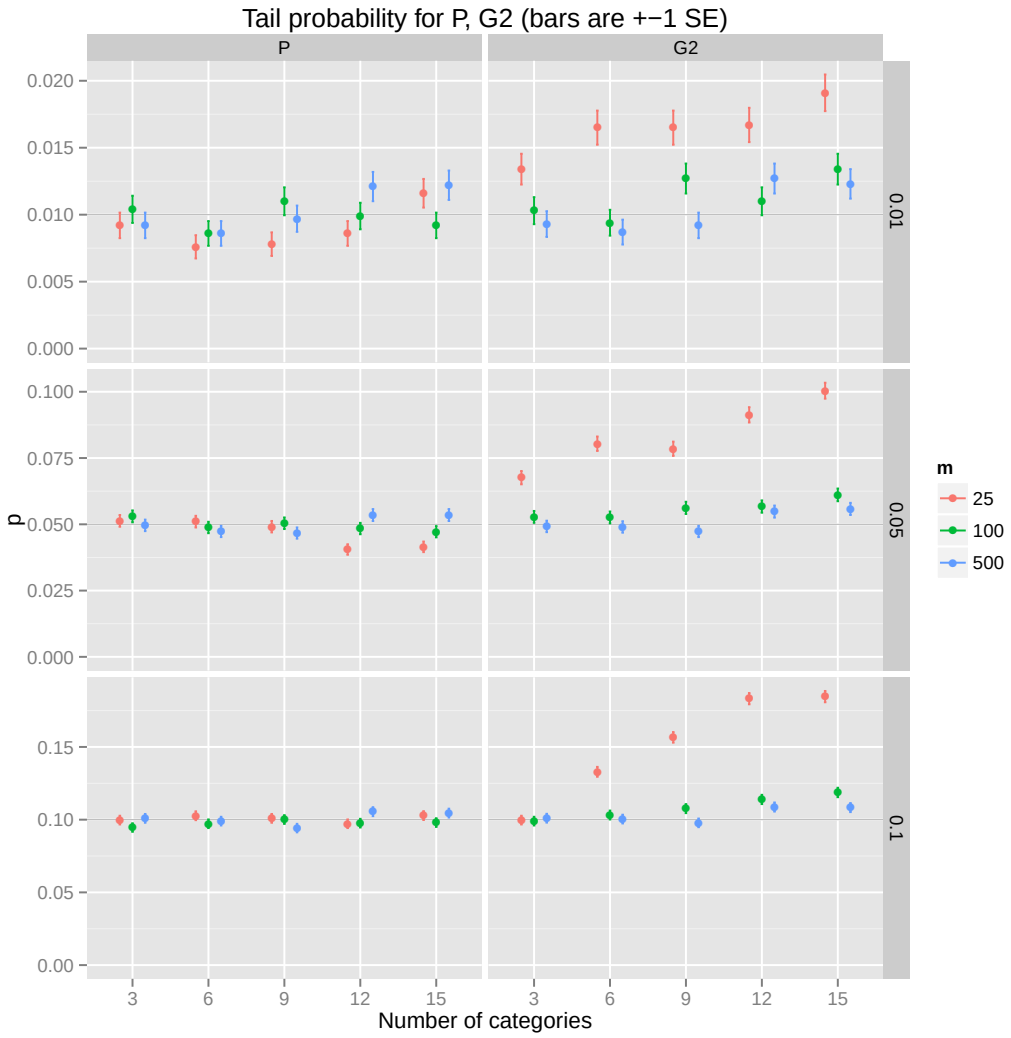
```
# Interaction plots, ggplot
library(ggplot2)
```

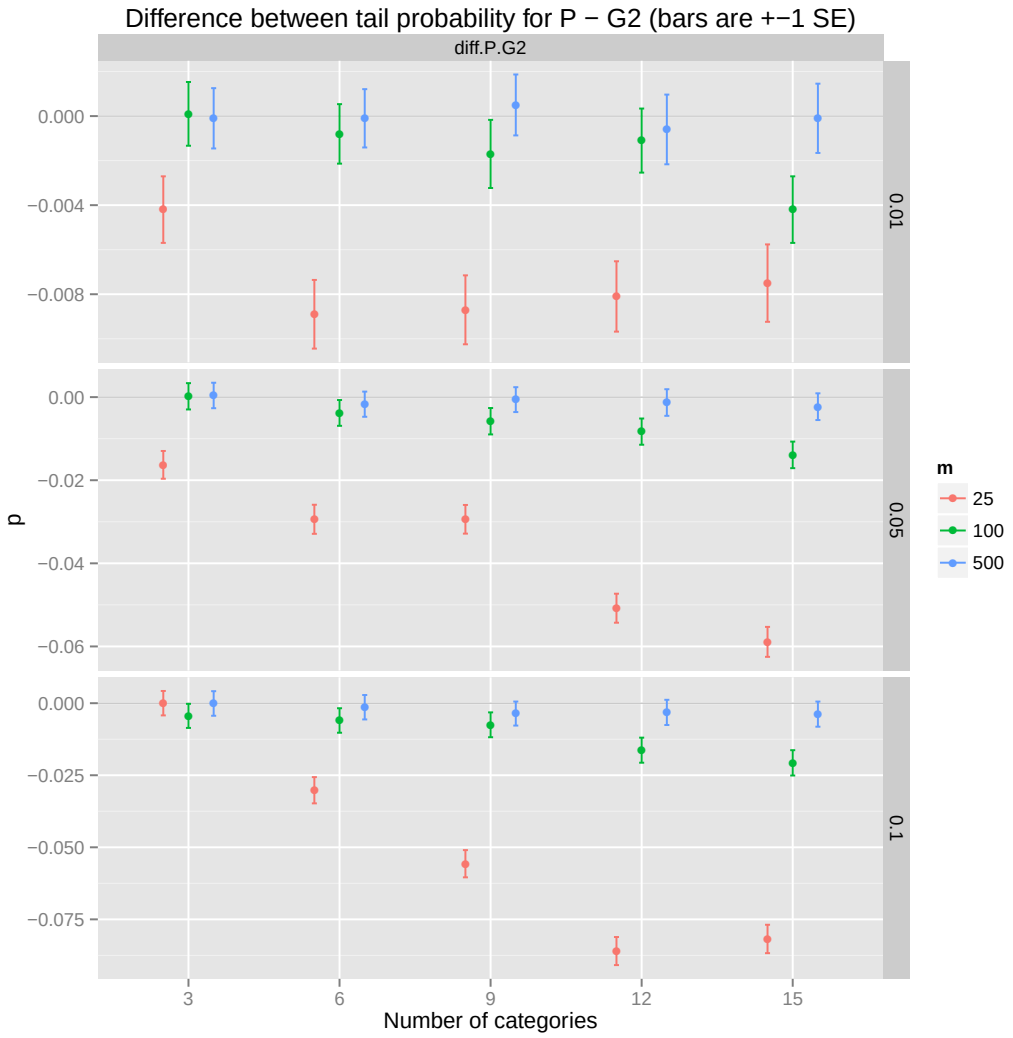
```
  # The errorbars overlapped, so use position_dodge to move them horizontally
  pd <- position_dodge(0.5) # move them .05 to the left and right

p <- ggplot(subset(df.gof.tail.prob.long, Stat %in% c("P", "G2")), aes(x = k, y = p, colour = m))
p <- p + geom_hline(aes(yintercept = as.numeric(as.character(alpha))), colour = "black"
                  , linetype = "solid", size = 0.2, alpha = 0.3)
p <- p + geom_errorbar(aes(ymin = p - p.SE, ymax = p + p.SE, colour = m)
                       , width = 0.1, position = pd)
p <- p + geom_point(position = pd)
p <- p + facet_grid(alpha ~ Stat, scales = "free")
p <- p + expand_limits(y = 0)
p <- p + labs(title = "Tail probability for P, G2 (bars are +-1 SE)")
p <- p + ylab("p")
p <- p + xlab("Number of categories")
print(p)
```

```
## ymax not defined:  adjusting position using y instead
## ymax not defined:  adjusting position using y instead
## ymax not defined:  adjusting position using y instead
## ymax not defined:  adjusting position using y instead
## ymax not defined:  adjusting position using y instead
## ymax not defined:  adjusting position using y instead
```

```
# difference
p <- ggplot(subset(df.gof.tail.prob.long, !(Stat %in% c("P", "G2"))), aes(x = k, y = p, colour = m)
p <- p + geom_hline(aes(yintercept = 0), colour = "black"
                  , linetype = "solid", size = 0.2, alpha = 0.3)
p <- p + geom_errorbar(aes(ymin = p - p.SE, ymax = p + p.SE, colour = m)
                       , width = 0.1, position = pd)
p <- p + geom_point(position = pd)
p <- p + facet_grid(alpha ~ Stat, scales = "free")
p <- p + expand_limits(y = 0)
p <- p + labs(title = "Difference between tail probability for P - G2 (bars are +-1 SE)")
p <- p + ylab("p")
p <- p + xlab("Number of categories")
print(p)
```

```
## ymax not defined:  adjusting position using y instead
## ymax not defined:  adjusting position using y instead
## ymax not defined:  adjusting position using y instead
```


Tail probability for P, G2 (bars are +−1 SE)
P
G2
0.01
0.05
0.1
p
0.020
0.015
0.010
0.005
0.000
0.100
0.075
0.050
0.025
0.000
0.15
0.10
0.05
0.00
3
6
9
12
15
Number of categories
m
25
100
500


Difference between tail probability for P − G2 (bars are +−1 SE)
diff.P.G2
0.01
0.05
0.1
0.000
−0.004
−0.008
0.00
−0.02
−0.04
−0.06
0.000
−0.025
−0.050
−0.075
p
3
6
9
12
15
Number of categories
m
25
100
500