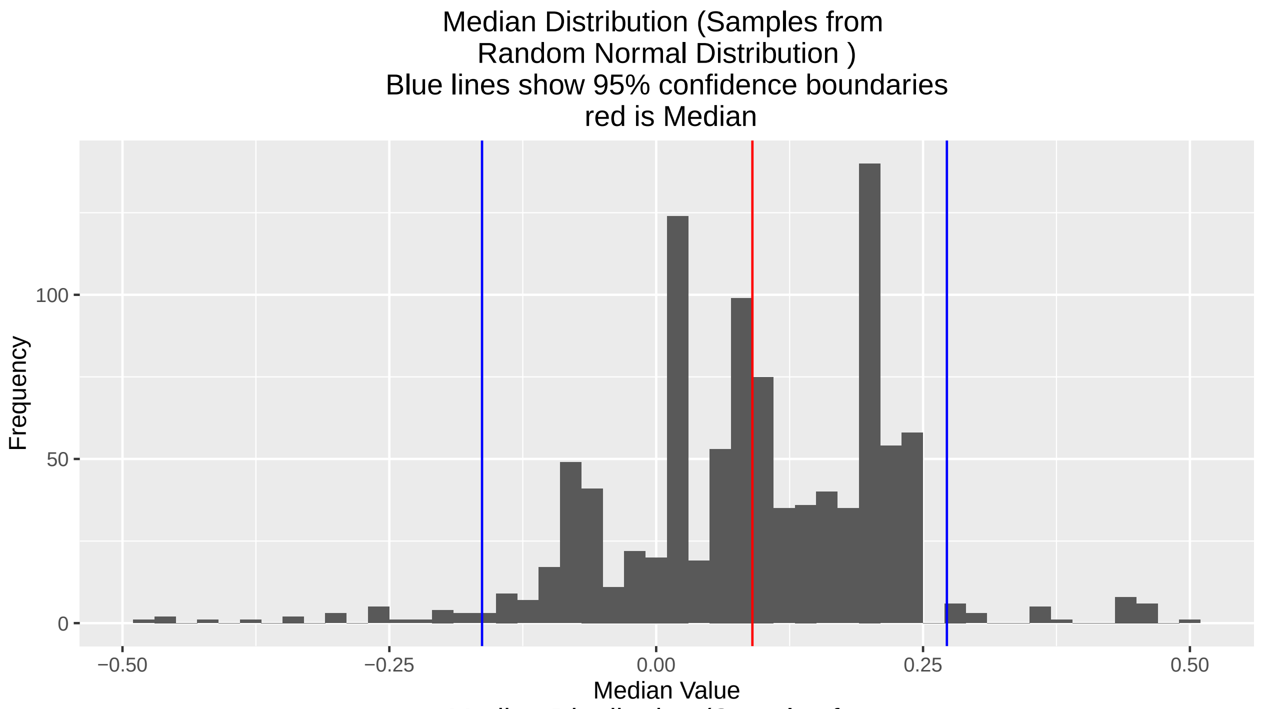
W203_Supplemental_4_I

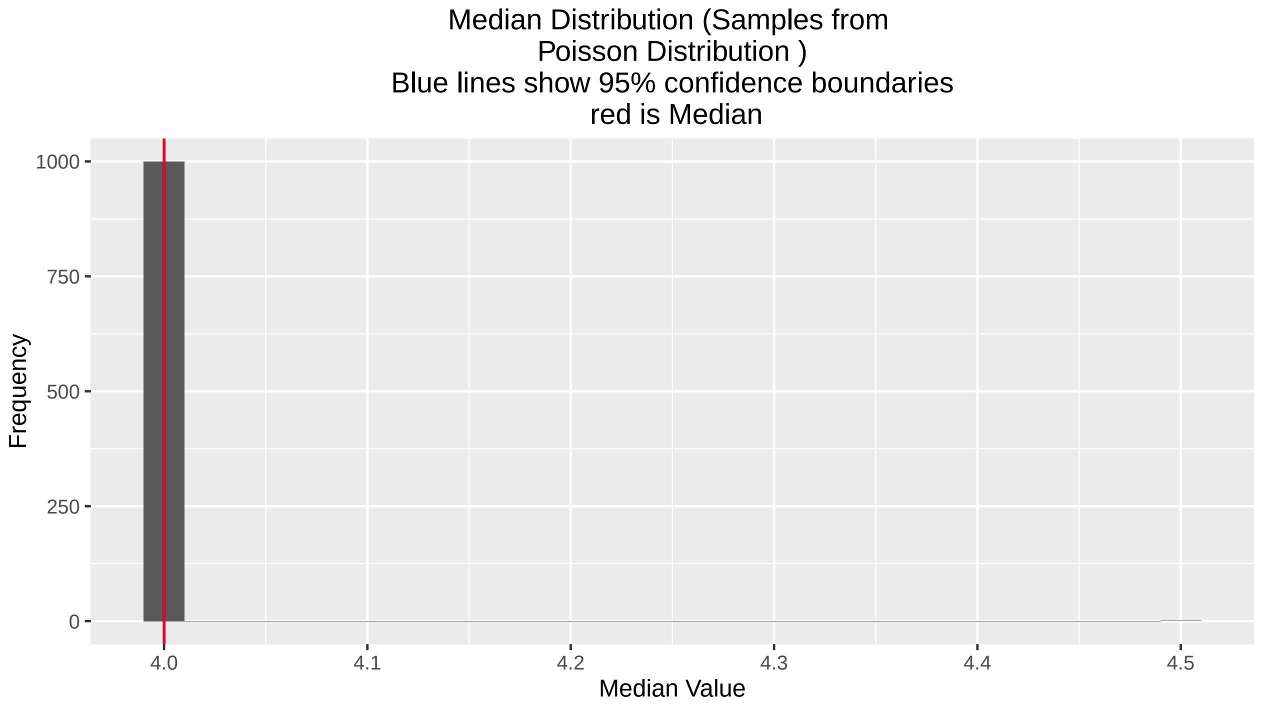
Natarajan Shankar

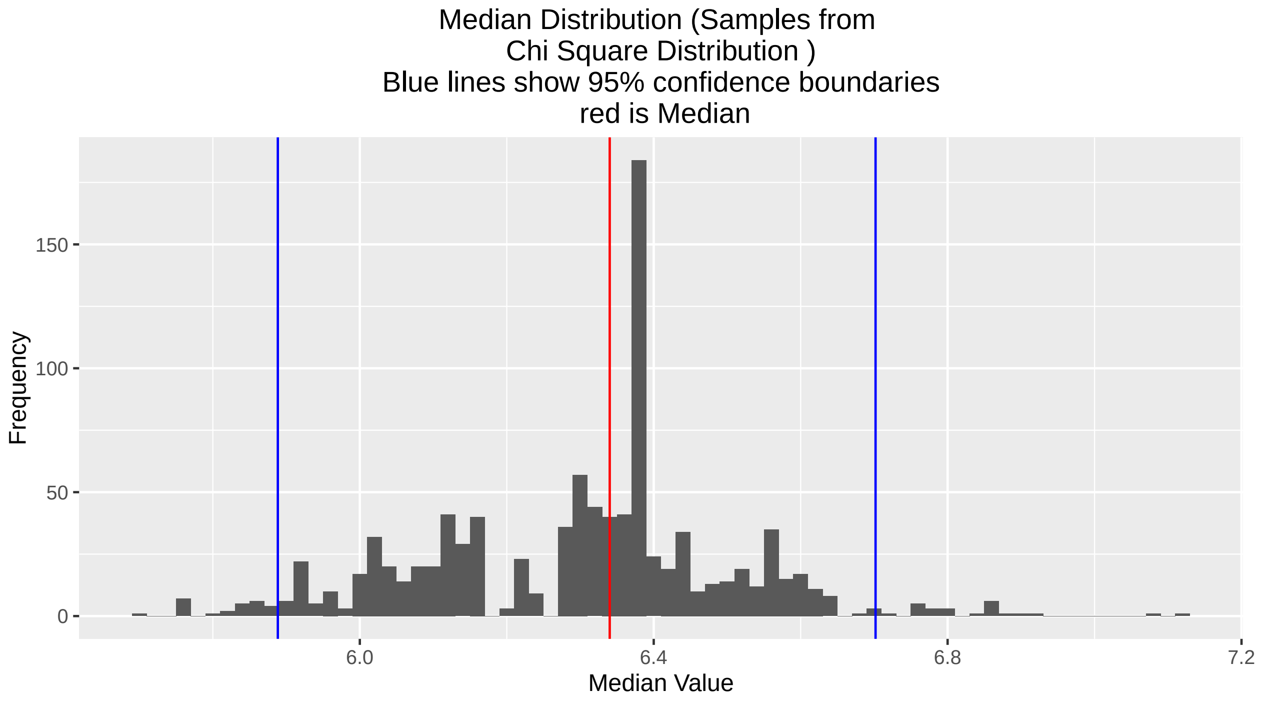
July 20, 2016

```r
# Natarajan Shankar
# W 203: Supplement Exercie 4
# July 20th

# Problem 1: BOOTSTRAPPING
#

# Play it safe and set a seed value to get repeatable results
set.seed(314156)

# Pull in needed libraries
library(ggplot2)

# Chose number of samples to be 1001. An odd number helps read the meadian vector easie
NBS <- 1001

# Set the confidence interval (CI) at plus/minus 95% (or 0.05 total outside the CI),
# split evenly across two tails i.e. 0.025 at each tail
LOW_CI <- 0.05/2
HIGH_CI <- 1 - LOW_CI

# FUNCTION: bootstrap_median()
# Inputs: Number of iterations, NBS and  the parent sample
# Iteration processing to extract many bootstrap samples
bootstrap_median <- function (iteration, parent_sample) {

  # Take a child sample from the parent sample, with replacement.
  # Child sample has the same length as the parent sample
  child_sample <- sample(parent_sample, size = length(parent_sample), replace = TRUE)

  # Calculate the median of the new sample
  child_median <- median(child_sample)

  # return the calculated mean
  return(child_median)
}


# Main bootstrap processing function
#   1. Takes a parent sample (from one of Random Normal, Chi Square, Binomial)
#   2. Takes the number of bootstrap iteraions (NBS)
#   3. Iterates over each sample and extracts a median
#   4. Returns a collection of bootstrapped medians and CI interval
median_bootstrap <- function(parent_sample, NBS) {

  # Compute the median across NBS sub samples
  median_distribution <- sapply(1:NBS, bootstrap_median, parent_sample = parent_sample)
  # Sort the medians from high to low
  median_distribution <- sort(median_distribution)

  # compute the 95% confidence interval
  low_CI <- median_distribution[round(NBS * LOW_CI)]
  high_CI <- median_distribution[round(NBS * HIGH_CI)]

  # Function returns the median distribution and confidence intervals
  return(list(median_distribution, c(low_CI, high_CI)))
}


# Test the median bootstrap using 3 kinds of data distributions
# 1. Random Normal
# 2. Poisson
# 3. Chi Square
# Plot results for each
test_code <- function() {
  # Random Normal Distribution
    parent_sample <- rnorm(n=100, mean=0, sd =1)
    results <- median_bootstrap(parent_sample, NBS)
    distribution_hist(results, "Random Normal Distribution")

  # Poisson Distribution
    parent_sample <- rpois(300, 4)
    results <- median_bootstrap(parent_sample, NBS)
    distribution_hist(results, "Poisson Distribution")

  # Chi Square Distribution
    parent_sample <- rchisq(300, df=7)
    results <- median_bootstrap(parent_sample, NBS)
    distribution_hist(results, "Chi Square Distribution")
}



# Use one common plot routine that can be used by the test code.
# Customize the header strings so that each plat is distinguished by title
distribution_hist <- function (results, type = "") {
    resultsDF <- data.frame(results[[1]])
    myPlot <- ggplot(resultsDF, aes(results[[1]]))
    myPlot <- myPlot + geom_histogram(binwidth=0.02)
    myPlot <- myPlot +
      ggtitle(paste ("Median Distribution (Samples from \n", type, ")",
                "\nBlue lines show 95% confidence boundaries\n", "red is Median")) +
                labs(x="Median Value", y="Frequency") +
                geom_vline(xintercept = results[[2]][1], color = "blue") +
                geom_vline(xintercept= results[[2]][2], color="blue") +
                geom_vline(xintercept= results[[1]][round(1001/2)], color="red")
    plot(myPlot)
}

# Test the code above
```

```
test_code()
```


Median Distribution (Samples from
Random Normal Distribution)
Blue lines show 95% confidence boundaries
red is Median
Frequency
100
50
0
−0.50
−0.25
0.00
0.25
0.50
Median Value


Median Distribution (Samples from
Poisson Distribution)
Blue lines show 95% confidence boundaries
red is Median
Frequency
1000
750
500
250
0
4.0
4.1
4.2
4.3
4.4
4.5
Median Value



Median distribution of samples with the Normal Distribution as parent show equal distribution around Median and are not clustered

Median distribution of samples with the Poisson distribution as parent show binary values as expected, clustered around the Median

Median distribution of samples with the Chi Square distribution as parent also shows even distribution around Median, not clustered

```r
# Problem 2: OPTIMIZATION PROBLEM
#

# Function : isEmpty()
# Support routine to determine whether a vector is empty
# Used after cleaning up negative numbers fromm user input.
# If after cleanup of negative numbers, the vector is null, do not process further
isEmpty <- function(x) {
  return(length(x)==0)
}


# Function: objf()
# Scaffolding Cost function for Optim
# Take 3 numbers, x, r, and k. Raise r to the power of k and compare with X
# Return square of the compare

objf <- function(r, numbers, k) {

  cost = (numbers - (r^k)) ^2

  return(cost)
}

# Function: rootk()
# Function to find the root k of a positive number using numerical optimization
# Accept a vecor of numbers, numbers
# Return root k for all the numbers in the vector
rootk <- function(numbers, k, start = NULL) {

  if (!isEmpty(x <- numbers[numbers < 0])) {
    cat("\nCaution 0001 :  negative numbers in vector, being removed... \n\n\n")
    numbers <- numbers[numbers >= 0]

    # Catch illegal k values (does not work via rmarkdown, needs investigation!!)
    if (k < 0) {
      stop("\nERROR 001: Root request is invalid (below 0), Processing stopped")
    }

    # Stop if no numbers to process after negative number cleanup
    if(isEmpty(numbers))
        stop("\nERROR 002: Input had negative numbers,
                    \ncleaned up number vector is empty!, Processing stopped")
  }

  # In case the user does not provide a starting approximation, start at 1
  is.null(start)
    start = 1

  # User vector has positive numbers and has been sanitized
  # apply Optim() to all members in the vector. Return a vecor of results
  # Use method BGFS for optimized processing
  #
  roots <- sapply(numbers, function(i) {optim(par = i,
                              numbers = i, objf, k = k, method="BFGS", control = list(maxit = 400))$par})
    result <- list(numbers, roots)

  # For pretty printing of final message
  if (k ==1)
      tag <- "st"
  else if (k ==2)
      tag <- "nd (square)"
  else if (k == 3)
      tag <- "rd (cube)"
  else
      tag <- "th level"

    names(result) <- c("Processed Numbers", paste ("Corresponding",
                            paste(k, tag, sep =""), "Root for each processed number"))
    return(result)

}
```

Test case 1 : One element vector

```r
rootk(4, 2)
```

```
## $`Processed Numbers`
## [1] 4
##
## $`Corresponding 2nd (square) Root for each processed number`
## [1] -2
```

- Note the negative root, accurate but interesting, larger start value does not change convergence on negative root

Test case 2 : two element vector

```r
rootk(125,3, start = NULL)
```

```
## $`Processed Numbers`
## [1] 125
##
## $`Corresponding 3rd (cube) Root for each processed number`
## [1] 5
```

Test case 3 : One element, large number, big root

```r
rootk(134217728,9)
```

```
## $`Processed Numbers`
## [1] 134217728
##
## $`Corresponding 9th level Root for each processed number`
## [1] 8
```

Test case 4 : 3 element vector, one negative

```r
rootk(c(25, -125, 96), 3)
```

```
##
## Caution 0001 :  negative numbers in vector, being removed...

## $`Processed Numbers`
## [1] 25 96
##
## $`Corresponding 3rd (cube) Root for each processed number`
## [1] 2.924018 4.578857
```

Test case 5 : 3 element vector, negative root request

"STOP" WORKS VIA CONSOLE, DOES NOT WORK VIA RMARKUP, NEEDS INVESTIGATION

```r
rootk(c(25, 125, 96), -3)
```

```
## $`Processed Numbers`
## [1]  25 125  96
##
## $`Corresponding -3th level Root for each processed number`
## [1]  24.99962 125.00000  95.99999
```

Test case 6 : 3 element vector, all 3 negative

```r
rootk(c(-25, -125, -96), 5)
```

```
##
## Caution 0001 :  negative numbers in vector, being removed...

## Error in rootk(c(-25, -125, -96), 5):
## ERROR 002: Input had negative numbers,
##
## cleaned up number vector is empty!, Processing stopped
```