DSA3361 - Workshop 1

Hui Teng/Yiqun/Qian Jiang

2022-08-23

Contents

Structure of Workshop (mini-lecture)

1. Sampling distributions of test statistics/point estimates
2. Bootstrap confidence intervals
3. Permutation tests

Learning Outcomes

We will learn to:

1. Generate the sampling distribution for some point estimate.
2. Construct the Bootstrap confidence intervals.
3. Apply permutation tests.

Setting Up Rstudio

Here's a checklist:

1. Have you created an Rstudio project for this class?
    - Create a new folder `DSA3361` and create a new Rstudio project inside it.
2. Have you downloaded all the data, source codes and slides from Canvas?
    - Create `data`, `src` and `slides` folders within the main folder and dump the Canvas contents there.

Installing and Using Libraries

Libraries extend the functionality of R. They are essentially a collection of functions. Before **using** a library, we need to load it.

These are the libraries we need for today:

```
library(ggplot2)
library(dplyr)
library(stats)
library(fitdistrplus)
library(vcd)
library(Hmisc)
library(fGarch)
library(ufs)
library(tinytex)
```

If you encounter an error such as this, it means that the library has not been installed yet.

```
Error in library(fitdistrplus) : there is no package called 'fitdistrplus'
```

Task 1: Sampling Distributions and Non-parametric Boostrap CIs

In the videos, we discussed that the classical way of calculating p-values and confidence intervals (CIs) depends on the theoretical characterisation of the sampling distribution. This approach is effective but the accuracy of p-values or CIs depends on the distributional assumptions, some of which might be violated in practice. Also, it limits the possible types of statistics that we could use.

The main idea of permutation tests and bootstrapping tests is to **simulate** the sampling distribution, which requires less assumptions. For example, we don't need the sample size to be large, and we do not require that the population data follow normal distribution. Another key benefit is that we can (almost) use any test statistic that we want. A test statistic is also referred to as a point estimate.

The core of the non-parametric bootstrapping and permutation tests is the re-sampling method. The difference is that the non-parametric bootstrapping approach is to re-sample from the original sample with replacement; the permutation test approach is to re-sample from the original sample without replacement.

In this module, we will not focus on the parametric bootstrapping approach. Feel free to check with us if you have any question or feel interested.

We will use the following examples to illustrate the concept of point estimates, and generate the sampling distributions.

1-1: Case Study: E&E Courses Fees Data

Some unit of NUS offers Executive Education courses that offer in-demand skills and knowledge according to technological and business trends. We have obtained data from past courses. We are interested to study the spending of our learners, in terms of the mean course fees spent in each discipline.

Let us first read in the dataset.

```r
# If your working directory is "***/DSA3361",
df_fees <- read.csv("data/NUS_E&E_fees.csv")

# If your working directory is "***/DSA3361/src",
df_fees <- read.csv("../data/NUS_E&E_fees.csv")

str(df_fees)
head(df_fees)
summary(df_fees)
```

We have 875 learners, each of which was enrolled for one or multiple courses in one discipline only. The column `Fees` summarises the total course fees of each learner.

Let us inspect the dataset by checking how many disciplines are involved.

```r
df_fees0 <- df_fees
df_fees$Discipline <- as.factor(df_fees$Discipline)
str(df_fees)
levels(df_fees$Discipline)
summary(df_fees)
```

This dataset involves 4 disciplines: `Artificial Intelligence`, `Data Science`, `Software Engineering`, `StartUp & Tech Development`.

> Question: Which disciplines are more popular, with regards to the number of participants?

1-2: Point Estimates

Choose the Point Estimate

It is natural to use the observed sample's mean fees as a point estimate for each discipline.

```r
# compare the mean scores of the four disciplines
(mean_AI <- mean(df_fees[df_fees$Discipline == "Artificial Intelligence",2]))
(mean_DS <- mean(df_fees[df_fees$Discipline == "Data Science",2]))
(mean_SE <- mean(df_fees[df_fees$Discipline == "Software Engineering",2]))
(mean_STD <- mean(df_fees[df_fees$Discipline == "StartUp & Tech Development",2]))

# alternative way of comparing the mean fees
tapply(df_fees$Fees, df_fees$Discipline, mean)
```

It can be seen that AI has the largest sample mean. Our ultimate target is to infer the population parameter, namely, the population mean spending of each discipline. The following discussion will focus on AI for convenience.

```r
fees_AI <- df_fees[df_fees$Discipline == "Artificial Intelligence",2]

hist(fees_AI)
mean(fees_AI)
sd(fees_AI)
```

The distribution of the (total) course fees of AI is right skewed, which has a longer right tail. You may check the distribution types of course fees of other disciplines. Alternatively, we can use ggplot() to inspect the four histograms simultaneously.

```r
# Y axis is count:
ggplot(df_fees, aes(x = Fees, fill = Discipline)) + geom_histogram() +
      facet_wrap(~Discipline)

# Y axis is density:
ggplot(df_fees, aes(x = Fees, fill = Discipline)) + geom_histogram(aes(y=..density..)) +
      facet_wrap(~Discipline)
```

What do you observe?

1-3: Sampling Distribution

Let us study the sampling distribution of the test/sample statistics, the mean course fees. Recall that a sampling distribution is the distribution of all possible values of a sample statistic from samples of a given sample size from a given population.

The following is a simple example of how sampling distribution could be generated, if the population data is known.

```r
set.seed(1234)
pop_income <- rsnorm(10000, mean=10000, sd=3000, xi=2)  # population data
hist(pop_income, main = "Population Distribution of Household Income")
# set the population parameter, the mean household income
para_pop <- 10000

n_rep <- 1000  # number of random samples
sample_size <- 30
sampling_dist_income <- numeric(n_rep)
```

```r
set.seed(1)
for (i in 1:n_rep){
  sample1 <- sample(pop_income, size = sample_size, replace = FALSE)
  sampling_dist_income[i] <- mean(sample1)
}

hist(sampling_dist_income, main = "Sampling Distribution of Sample Mean")
mean(sampling_dist_income)
mean(sampling_dist_income) - para_pop   # the variability due to the simulation, which could be reduced
```

> Question: how do we reduce the above variability?

The above histogram is not the exact sampling distribution but an approximation. The larger the number of random samples (i.e., n_rep but not the sample size), the closer the histogram will be to the exact one. Another

We can think of the sample distribution as some chart describing how sample statistic varies from one study to another. The population distribution shows the variability of the individual observation, while the sampling distribution shows the variability of the sample mean in repeated studies.

1-4: Boostrap Distribution

In most real practices, we do not have the population data; hence, we use either computational or mathematical structures to estimate the sampling distribution, in order to describe the expected variability of th sample statistic. The classical approach of hypothesis testing or calculating confidence intervals is to gauge the sampling distribution mathematically, under a list of assumptions. It is actually effective and cost-minimal if the required assumptions are satisfied.

For cases where some of the classical assumptions are violated, alternative solutions could be the bootstrapping approach or the permutation tests, which require minimal assumption(s). The main idea is to generate some approximated version of the exact sampling distribution by computational simulations. (Monte Carlo Sampling)

The following is to apply the non-parametric bootstrap method to generate the sampling distribution of sample mean for AI. We will refer to the distribution as the bootstrap distribution of means.

```r
n_rep <- 1000
boot_fees_AI <- numeric(n_rep)

set.seed(1)
for (i in 1:n_rep){
  boot_sample <- sample(fees_AI, size = length(fees_AI), replace = TRUE)
  boot_fees_AI[i] <- mean(boot_sample)
}

hist(boot_fees_AI,
     main = "NP Bootstrap distribution of mean (Fees for AI)")
abline(v=mean(fees_AI), lty = 2, lwd = 3, col="red")

mean(boot_fees_AI)                 #mean X-bar*
sd(boot_fees_AI)                   #SE*

mean(fees_AI)
```

```r
sd(fees_AI)

mean(boot_fees_AI) - mean(fees_AI)  #bias
```

How do we reduce such bias? What is your recommendation?

Note that the (simulated) bootstrap distribution is centered around the original sample mean, which is not necessarily close to the unknown population parameter. Therefore, the bootstrap method is not used to get better estimates of the parameter value; instead, it is useful for quantifying the behavior of a parameter estimate, such as, standard error, skewness, or for calculating confidence intervals.

Evaluating the Bootstrap Distribution

We will use two approaches to evaluate whether the bootstrap distribution is normal, or at least symmetric.

Approach 1:

```r
fit_norm_object1 <- fitdist(fees_AI, distr = "norm")
qqcomp(fit_norm_object1, main = "Sample Observations")
denscomp(fit_norm_object1)

fit_norm_object2 <- fitdist(boot_fees_AI, distr = "norm")
qqcomp(fit_norm_object2, main = "Bootstrap Distribution")
denscomp(fit_norm_object2)
```

The bootstrap distribution is to some extent right skewed.

Approach 2:

```r
qqnorm(fees_AI, main = "Sample Observations")
qqline(fees_AI, col = "red")

qqnorm(boot_fees_AI, main = "Bootstrap Distribution")
qqline(boot_fees_AI, col = "red")
```

1-5: Compute and Plot the 95% Confidence Intervals

The general template of confidence interval is as follows:

$$\begin{aligned}\textsf{Confidence Interval} &= \textsf{Point Estimate} \pm \textsf{Margin of Error}\\ &= \textsf{Point Estimate} \pm \textsf{Critical Value} * \textsf{Standard Error}\end{aligned}$$

We first construct the classical SE (standard error method) confidence interval, which shares the same set of assumptions as one sample t-test.

```r
mean(fees_AI)
sd(fees_AI)
(ct_value <- qt(0.975, df = length(fees_AI) - 1))
(moe <- ct_value * sd(fees_AI)/sqrt(length(fees_AI)))
```

```r
ci.lb <- round(mean(fees_AI) - moe, 3)
ci.ub <- round(mean(fees_AI) + moe, 3)

# classical SE CI
ci_classical <- formatCI(c(ci.lb,ci.ub))
print(paste0("The 95% classical SE CI for mean is ", ci_classical))
```

For the classical approach, standard error is typically calculated using the sample size(s) and the sample (or population) standard deviation(s).

We next construct the bootstrap CI using the standard error method, where standard error is equal to standard deviation of the collection of the bootstrap sample statistics. This is also referred to as **Bootstrap SE Confidence Interval**.

We assume the bootstrap distribution is normal; hence, we set the critical value as the normal-based one, 1.96.

```r
mean(boot_fees_AI)
sd(boot_fees_AI)
moe <- 1.96 * sd(boot_fees_AI)

ci.lb2 <- round(mean(boot_fees_AI) - moe, 3)
ci.ub2 <- round(mean(boot_fees_AI) + moe, 3)

# Bootstrap SE (standard error) CI
(ci_boot_se <- formatCI(c(ci.lb2,ci.ub2)))
print(paste0("The 95% Bootstrap SE CI for mean is ", ci_boot_se))
```

We checked earlier that the bootstrap distribution (mean of AI group) is slightly right skewed. The normality assumption here is possibly violated. Let us double check by calculating the percentage of data below and above the CI range.

```r
percent_below <- sum(boot_fees_AI < ci.lb2) / length(boot_fees_AI)
percent_above <- sum(boot_fees_AI > ci.ub2) / length(boot_fees_AI)

print(paste0("The proportion below the lower limit CI is ",percent_below))
print(paste0("The proportion above the upper limit CI is ",percent_above))
```

We conclude that

- about 1.9% of the resample means are below the bootstrap mean -1.96SE;
- about 2.7% of the resample means are above the bootstrap mean +1.96SE.

In this case, relying on the CLT would be less accurate.

Alternatively, we can construct 95% Bootstrap CI for the mean, using the **Percentile** method, i.e., 95% **Bootstrap Percentile Confidence Interval**.

```r
ci_boot_perc <- round(quantile(boot_fees_AI, c(0.025, 0.975)),2)
formatCI(ci_boot_perc)

print(paste0("The 95% Bootstrap Percentile CI for mean is ", formatCI(ci_boot_perc)))
```

```
hist(boot_fees_AI)
abline(v=ci_boot_perc[1], lty = 2, lwd = 3, col="red")
abline(v=ci_boot_perc[2], lty = 2, lwd = 3, col="red")

# center
mu <- quantile(boot_fees_AI, 0.5)
# left margin
mu - ci_boot_perc[1]
# right margin
ci_boot_perc[2] - mu
```

In conclusion:

- A good confidence interval for the mean need not be symmetrical: an endpoint would be farther from the sample mean in the direction of any extreme values (including outliers).
- When the bootstrap distribution is right skewed, the interval will likely stretch far to the right to control the chance of missing the true parameter around 2.5%.

- Conversely, there is less risk of missing the true mean on the lower side, so the left endpoint need not be as far away from the sample mean.

Plot the Confidence Intervals

Let us plot the above calculated 95% confidence intervals, using the `errbar()` function.

```
x <- c(1:3)
y <- c(mean(fees_AI), mean(boot_fees_AI),
       as.numeric(quantile(boot_fees_AI, 0.5)))
lower_bound <- c(ci.lb, ci.lb2, as.numeric(ci_boot_perc[1]))
upper_bound <- c(ci.ub, ci.ub2, as.numeric(ci_boot_perc[2]))

errbar(x, y, lower_bound, upper_bound, ylim = c(0, 10000),
       errbar.col = "blue", main="Webinar attendance")

errbar(x, y, lower_bound, upper_bound, ylim = c(3300, 4700),
       xaxt = "n",  # hide the original x label
       errbar.col = "blue", main="Webinar attendance")
axis(1, at = 1:3, labels = c("classical", "Bootstrap SE", "Bootstrap Percentile"))
abline(h=4640, lty=2, lwd = 2, col = "red")
abline(h=3338.6, lty=2, lwd = 2, col = "red")
```

When the sample size is large, the difference between classical, bootstrap SE and bootstrap Percentile confidence intervals is likely minimal.

The bootstrap method is effective for cases that sample size is small, or the test statistic is a bit unusual (say, ratio of two sample variances), for which the classical approach could be very troublesome.

Hands-on (Optional)

Besides `errbar()`, we can also use functions from `ggplot2` library to generate multiple confidence intervals in one chart.

For convenience, we will only calculate the classical normal-based confidence intervals.

```r
# calculate the mean, sd and sample size for each discipline
discipline_fee_means <- group_by(df_fees, Discipline) %>%
                        summarise(av_fee= mean(`Fees`, na.rm = TRUE),
                                  sd_fee = sd(`Fees`, na.rm = TRUE),
                                  n= n(), .groups="drop") %>%
                                  arrange(desc(av_fee))

# calculate the average fees and upper / lower limit for the 95% normal-based confidence intervals
mutate(discipline_fee_means, ylower= av_fee - 1.96*sd_fee/sqrt(n),
       yupper = av_fee + 1.96*sd_fee/sqrt(n)) %>%
  ggplot(aes(x=Discipline, y=av_fee)) +
  geom_point(col="red", size=3) +
  geom_errorbar(aes(ymin=ylower, ymax=yupper), size=1,width=0.2) +
  labs(x = "Favorite Discipline", y = "Average Fee") +
  coord_flip() +
  theme(axis.text = element_text(size = 14))
```

Optional: You could change the above code to calculate the 95% bootstrap percentile confidence interval for another statistic, median.

Summary

1. Bootstrap is very powerful in situations when we wish to find standard errors, perform hypothesis tests, and compute confidence intervals for new statistics/point estimators. (Median, ratio of two means/medians, ratio of two variances, etc.)

Task 2: Permutation Testing

Another Case Study: Programme Vs. Math Score

We will be working with the following dataset:

- school_programmes.csv - It contains information on 60 students who were randomly selected from three different programmes: General, Vocational and Academic. We are interested to determine whether the programme type is associated with the math scores.

```r
# If your working directory is "***/DSA3361",
df_school <- read.csv("data/school_programmes.csv")

# If your working directory is "***/DSA3361/src",
df_school <- read.csv("../data/school_programmes.csv")
```

2-1: Data Exploration and Visualisation

We first check the data structure and the overall summary.

```r
str(df_school)
summary(df_school)
```

Let us use the `factor()` function to transform the `prog` and `gender` variables into factors.

```r
df_school$prog <- factor(df_school$prog, levels = c("General", "Vocational", "Academic"))

# Hands-on:
df_school$gender <- factor(_____, levels = _____)

str(df_school)
summary(df_school)
```

Let us compare the mean scores of the three programme groups.

```r
(mean_Aca <- mean(df_school[df_school$prog == "General",3]))
(mean_Gen <- mean(df_school[df_school$prog == "Vocational",3]))
(mean_Voc <- mean(df_school[df_school$prog == "Academic",3]))
```

We can also use the `tapply()` function to generate the mean scores easily.

```r
tapply(df_school$math, df_school$prog, mean)

# Hands-on: Calculate the scores' sd for the three groups
tapply(_____)
```

```r
boxplot(math ~ prog, data = df_school, col = c("tomato", "cyan", "skyblue"))
```

2-2: ANOVA Test

To compare the three mean scores, we can conduct an ANOVA test.

1. Null hypothesis: the three groups' mean scores are the same.
2. Alternative hypothesis: the three groups' mean scores are different.

```r
one.way <- aov(math ~ prog, data = df_school)
summary(one.way)
```

The p-value from the ANOVA test is 0.0279.

Check the Assumptions

There are four assumptions for ANOVA test:

1. Each sample group is drawn from a normally distributed population.
2. All sample groups are randomly selected and independent.
3. Variance across groups are the same.
4. The residuals are normally distributed for each population.

As observed earlier, the largest sd is larger than twice of the smallest sd; hence, the variances are different.

Also, from the boxplot, the three populations are likely skewed.

Therefore, not all the assumptions are satisfied. However, we don't know how much it may affect the accuracy of the p-value.

Hands-on (Optional)

As the normality assumption does not hold, we can also apply the non-parametric version of ANOVA test, Kruskal-Wallis H test.

```r
kruskal.test(math ~ prog, data = df_school) # p-value = 0.01545
```

2-3: Permutation Testing (1st Attempt)

Recall the key steps of implementing a permutation test:

0. Choose a test statistic.
1. Define a function, which generates a permutation sample, and calculates the test statistic for the permutation sample.
2. Use the `replicate()` function to repeat the above simulation multiple times, say, 1,000.

3. Generate a histogram of the 1,000 test statistics that approximates the exact permutation distribution.
4. Calculate the p-value as the proportion of the test statistics that are at least as extreme as the observed one.

Define a Test Statistic

Our first attempt is to pick the F value used in ANOVA test.

$$F = \frac{\text{Mean Sum of Squares between the Groups}}{\text{Mean Sum of Residual Squares}}$$

The mathematical form is a bit complicated. We can fetch the F value from the summary of the ANOVA test table.

```r
# to fetch the sample's test statistic: F value
(original <- summary(one.way)[[1]]$`F value`[1])
```

Define a Function to Simulate One Permutation Sample

```r
# Some preprations
treatment <- df_school$prog
outcome <- df_school$math

permutation.test <- function(treatment, outcome){
    # Generate a permutation sample by reallocating the group order
    treatment_p <- sample(treatment, size = _____, replace = _____)
    # Construct the data frame for the permutation sample
    df_p <- data.frame(prog = treatment_p,  math = outcome)
    # Conduct ANOVA test for the permutation sample
    one.way_p <- aov(_____, data = _____)
    # Output the F value
    _____
}
```

Generate Multiple Permutation Samples and the Permutation Distribution

Let us use the `replicate()` function to run the simulations 1,000 times.

```r
set.seed(123)
test <- replicate(_____ , _____)

hist(test, main = "Sampling Distribution by Permutation") # Generate the histogram of all the 1,000 tes
abline(v=original, lty = 2, lwd = 3, col="red") # Mark the original sample's test statistic

# Calculate the two-tailed p-value
mean(_____)
```

2-4: Permutation Testing (2nd Attempt)

One beauty of permutation testing is that you can almost choose any test statistic that you want.

In our case, we can define the test statistic as the sum of the absolute differences of the three groups' means.

$$\text{Test Statistic} = |\text{Mean}_1 - \text{Mean}_2| + |\text{Mean}_2 - \text{Mean}_3| + |\text{Mean}_3 - \text{Mean}_1|$$

Let us calculate the new test statistic in the following way.

```r
(mean_list <- tapply(df_school$math, df_school$prog, mean))
a <- mean_list[1]
b <- mean_list[2]
c <- mean_list[3]
(original2 <- as.numeric(abs(_____) + abs(_____) + abs(_____)))
```

```r
permutation.test2 <- function(treatment, outcome){
    # Generate a permutation sample by reallocating the group order
    # It is equivalent to re-sampling without replacement
    treatment_p <- sample(treatment, size= length(treatment), replace=FALSE)
    df_p <- data.frame(prog = treatment_p, math = outcome)
    mean_p <- tapply(_____, _____, mean)
    a <- mean_p[1]
    b <- mean_p[2]
    c <- mean_p[3]
    return(abs(_____) + abs(_____) + abs(_____))  # return the test statistic of the permutation sample
}
```

```r
set.seed(123)
test2 <- replicate(1000, permutation.test2(treatment, outcome))

hist(test2) # generate the permutation distribution for the new statistic
abline(v=original2, lty = 2, lwd = 3, col="red")

mean(abs(test2) >= abs(original2)) # p-value is 0.017
```

Hands-on (Optional)

1. You can increase the number of simulations to see how the permutation distribution and the p-value change.

2. You can remove the `set.seed` part, and run the remaining lines repeatedly to see how the chart and the p-value are updated.
3. Do you have any other suggestions for the possible test statistic (for this ANOVA test)?

Summary/Key Takeaways

1. Sampling distribution may not be normal, especially when the sample size is small or when the population data do not follow normal distribution.
2. Bootstrap and permutation tests are preferred if you wish to use a new test statistic, or if the sample size is not large.

References

1. Chihara, L. & Hesterberg, T. (2019). Mathematical statistics with resampling in R, Chapter 3 & 5.
2. Mine Çetinkaya-Rundel and Johanna Hardin. Introduction to Modern Statistics, Chapter 11, 17 & 20.

Appendix

Use `boot` library to generate the non-parametric bootstrap confidence intervals

```
library(boot)

# wrong way
bootstrap <- boot(fees_AI, mean, R = 10000)

# Define function to take in an index i as a second argument
meanfun <- function(data, i){
  d <- data[i, ] # allows boot to select sample
  return(mean(d, na.rm = TRUE))
}

set.seed(1) # for reproducibility, i.e., the results would be the same for anyone who run the same code

boot_fees_AI2 <- boot(as.data.frame(fees_AI),
                    statistic=meanfun,
                    R=10000)
boot_fees_AI2

boot.ci(boot.out = boot_fees_AI2, type = c("norm", "basic", "perc"))

# compare with the earlier CIs
print(paste0("The 95% classical SE CI for mean is ", ci_classical))
print(paste0("The 95% Bootstrap SE CI for mean is ", ci_boot_se))
print(paste0("The 95% Bootstrap Percentile CI for mean is ", formatCI(ci_boot_perc)))
```

Why the numbers are still different, even after using the same seed?

Another Case Study: SERIS - Fitting Gamma Distribution to a Discrete Variable

- `Dataset_SERIS_weekly.csv` - This contains the number of maintenance events at SERIS monitoring stations in 2020.

```
# Let us import the seris dataset
seris <- read.csv("data/Dataset_SERIS_weekly.csv")
# Or,
seris <- read.csv("../data/Dataset_SERIS_weekly.csv")
```

Fitting a Discrete Variable

To fit a discrete variable, there are three steps: (using Poisson as an example)

1. fit_object <- fitdist(data, distr = "pois")
2. fitted_curve <- dpois(x = *support*, lambda = parameter) * sum(observed)
3. rootogram(x = observed, fitted = fitted_curve)

In the video 2-4, as the support is small, say, from 1 to 12, we will exactly use the above template, where `observed` is the frequency table with bins of size 1.

Now, let us try a more complex example, where the support of the discrete variable is large. Instead of bin size 1, we will use a histogram with larger bins.

Let us use the SERIS dataset that is imported earlier. The key variable is the numbers of SERIS weekly maintenance events.

We also assume the maintenance events occur independently, namely, each week's number is independent of other weeks'.

```
str(seris)
maint_events <- seris$n
hist(maint_events)
```

As it is a discrete variable with no clear support, it is reasonable to assume that the variable follows a Poisson distribution. Let's fit the Poisson distribution to it and see if it looks alright.

```
poisson_fit <- fitdist(maint_events, "pois")
poisson_fit                       # only one parameter: lambda
(lambda <- poisson_fit$estimate) # the bracket is to display the value

mean(maint_events)    # for Poisson distribution, mean is equal to lambda
```

This is not a coincidence as the mean of a Poisson distribution is equal to its lambda value.

Rootograms

A rootogram shows if a count variable is well-fitted to a discrete distribution. It essentially involves comparing fitted count values to observed ones.

```
hist(maint_events, breaks=12)

hist_e <- hist(maint_events, breaks=12, plot=FALSE)
brks   <- hist_e$breaks
counts <- hist_e$counts
brks
counts
```

What we need is to generate the pmf of the fitted Poisson distribution, which will be stored at `fitted_probs`.

```
brks[-1]
brks[-12]
fitted_probs <- ppois(brks[-1], lambda=lambda) -
                    ppois(brks[-12], lambda=lambda)

rootogram(counts, fitted_probs * sum(counts))
```

So, Poisson distribution does not fit the `maint_events` variable.

> Optional Task: you can increase the number of breaks for the histogram, and re-run the above code. What happens to the rootogram?

Just like the continuous variable case, we can also use the `denscomp()` function to compare the fitted pmf to the observed count values.

```
denscomp(poisson_fit)

# the center of the fitted values are around the lambda value, 15.
# Yet, half of the observed values are less than or equal to 10.
# So, the Poisson fits poorly to the dataset.
```

This chart overlays three graphics:

1. The fitted Poisson pmf in red.
2. The histogram of the data
3. The empirical pmf of the observed data. For instance, at 41, the height of the black line is about 0.02, which comes from the fact that there was one week in the data with 41 events. Thus $1/52 \approx 0.02$.

It is clear that a Poisson fit is not appropriate for this data. We want to "fit" a distribution with the goal of simulating from it. But the Poisson is the only discrete pmf with infinite support that we know about..

What should we do?

Fitting Gamma Distribution to a Discrete Variable

```
gamma_fit <- fitdist(maint_events, "gamma")
gamma_fit

qqcomp(gamma_fit)
denscomp(gamma_fit)
```

This seems to fit the histogram better, but the problem is that, if we generate random variables from the fitted gamma distribution, it will return us non-integer values. Let us try the following strategy:

1. Generate a continuous variable, X, from fitted Gamma distribution.
2. If one value of X is 12.5, we will randomly replace it to any number between 11 and 15.

```
r_my_pmf <- function(n) {      # the size of the new sample generated from
  X <- rgamma(n, shape=gamma_fit$estimate[1], rate=gamma_fit$estimate[2])
  X %/% 5 * 5 + sample(1:5, size=n, replace=TRUE)
}

test_x <- r_my_pmf(52)    # We simulate the sequence of weekly numbers
                          # for the next 52 weeks
hist(test_x, breaks=12)
```

Suppose X is 21.4, the code of `X %/% 5` outputs 4, as 21.4 contains 4 copies of 5. Hence, `X %/% 5 * 5` outputs 20. The `sample()` part outputs a random value between 1 and 5. Therefore, the final value could be any value between 21 and 25.

So, the functions `r_my_pmf` roughly simulates a new sample of size `n` which follows a new distribution. It is a discrete variable with support 0, 1, 2, ... The precise pmf can be mathematically derived, but it is not necessary. In the next section, we will use `r_my_pmf` to run simulations.

There are other options to fit a distribution when the Poisson fails:

1. Zipf distribution
2. Log-series
3. Use a smoothed version of the empirical pmf.

But they all require reading up, perhaps some derivations. Today, our goal is convince you that it is worthwhile to do so, while providing you with something quick.

Using the Fitted Distribution to Run Simulations

Part of this course is about using simulation. Simulating involves repeated generation of random variables, followed by averaging. The Law of Large Numbers guarantees us that the sample average will be close to the mean of the distribution that the numbers came from.

For example, the following code is to simulate samples in order to estimate the mean of the standard normal distribution.

The number inside the brackets following `rnorm` can be taken as the sample size. We want to test how the sample size affects the estimate.

```
X <- rnorm(100)  # by default, mean = 0, sd = 1
mean(X)

X <- rnorm(100000)
mean(X)
```

The larger the sample size, the closer the sample mean to the true population mean. We shall carry the same spirit for running simulations. The general principle is to run the simulations multiple times, and take the mean of the estimates.

SERIS simulation

Suppose now we are satisfied with using`r_my_pmf` to generate weekly counts in the new year. When we call this:

```
r_my_pmf(1)
```

It returns a simulated observation of the number of events in some week.

One Scenario:

As a policy-maker, you are interested in knowing what the total number of events in a year will be. We can answer this using simulation:

```
generate_and_sum <- function(n) {
  X <- r_my_pmf(n)
  sum(X)
}

generate_and_sum(52)    # one simulation for one year

generate_and_sum(52)
generate_and_sum(52)
# the estimate, the sum for one year, is essentially a random variable
```

Of course, the above sum value may vary greatly if you run it multiple times. By law of large numbers, in order to get an estimate that is closer to the true target value, we can generate 1,000 simulations (which is like 1000 years of data), and take the mean of all the sum values.

```
output <- replicate(1000, generate_and_sum(52))
mean(output)
hist(output)
```

Let us use `set.seed()` to control the outcome.

```
set.seed(123)
output <- replicate(1000, generate_and_sum(52))
quantile(output, 0.025)
quantile(output, 0.975)
```

Roughly speaking, we are 95% confident that the annual sum of maintenance reports is between 632 and 1018.

Hands-on (Optional):

To check the assumption of independence: each week's number is independent of other weeks'.

Is it true in our case?

```
# Pls try it out here.
```