Week 3 Exercises

Exercise 1 Mean v. median

Suppose y follows a normal model with mean μ and SD σ. Then avg(y) is the *best* unbiased estimator of μ. But median(y) is also an unbiased estimator of μ. Use a Monte Carlo simulation to show how avg(y) is a better estimator of μ than median(y).

The average is better because the SE of the average estimator is smaller than the SE of the median estimator.

```
# parameters
mu <- 0
sigma <- 1
N <- 10 # sample size of y
n_mc_sims <- 10000   # monte carlo repetitions

# containers
mean_hat <- numeric(n_mc_sims)
median_hat <- numeric(n_mc_sims)

# simulate many data sets y; compute and store estimates for each
for (i in 1:n_mc_sims) {
  y <- rnorm(N, mean = mu, sd = sigma)
  mean_hat[i]   <- mean(y)
  median_hat[i] <- median(y)
}

sd(mean_hat)  # se of avg estimator
```

```
[1] 0.3188053
```

```r
sd(median_hat)  # se of median estimator
```

```
[1] 0.37633
```

Exercise 2 Asymptotics as approximations

For a large number of repetitions, do the following:

1. Simulate a data set y with $N = 50$ observations from the exponential distribution with $\lambda = 2$.
2. Compute the point estimate $\hat{\lambda} = \frac{1}{\text{avg}(y)}$. Store this point estimate.

Compare the mean and SD of the simulated point estimates to the theoretical mean and SD of the *asymptotic* sampling distribution. We care, of course, about the *actual* sampling distribution, but we have to use an *asymptotic approximation* in practice. Discuss the differences and whether they seem important or unimportant.

Hint: For the exponential model, the asymptotic sampling distribution of $\hat{\lambda}$ is $\hat{\lambda} \overset{a}{\sim} N\left(\lambda, \frac{\lambda^2}{N}\right)$.

Optional: Go beyond the mean and SD. Use a table of quantiles of the simulated point estimates and theoretical distribution and/or plots of the CDFs to compare the simulated (i.e., actual) and theoretical distributions.

Solution

```r
# load packages
library(tinytable)
options(tinytable_tt_digits = 3)
options(tinytable_latex_placement = "H")

# set seed for reproducibility
set.seed(1234)

# parameters
lambda <- 2
N <- 50  # sample size
R <- 10000  # number of mc repetitions

# loop
lambda_hat <- numeric(R)  # container to store results
for (i in 1:R) {
```

```
 y <-  rexp(N, rate = lambda)  # simulate data set
 lambda_hat[i] <- 1/mean(y)  # compute and store estimate
}

# empirical summaries
emp_mean <- mean(lambda_hat)
emp_sd   <- sd(lambda_hat)

# asymptotic
asy_mean <- lambda
asy_sd   <- sqrt(lambda^2 / N)

# data frame of results
res <- data.frame(
  Statistic   = c("Mean", "SD"),
  Empirical   = c(emp_mean, emp_sd),
  Asymptotic  = c(asy_mean, asy_sd)
)

# print table
tt(
  res,
  digits  = 3,
  align   = "lrr",
)
```

Statistic	Empirical	Asymptotic
Mean	2.04	2
SD	0.293	0.283

```
# quantile comparison table
probs <- c(0.01, 0.05, 0.10, 0.25, 0.50, 0.75, 0.90, 0.95, 0.99)
emp_q <- quantile(lambda_hat, probs = probs)
asy_q <- qnorm(probs, mean = asy_mean, sd = asy_sd)

# combine into data frame
quant <- data.frame(
  p = probs,
  empirical = emp_q,
  asymptotic = asy_q,
```

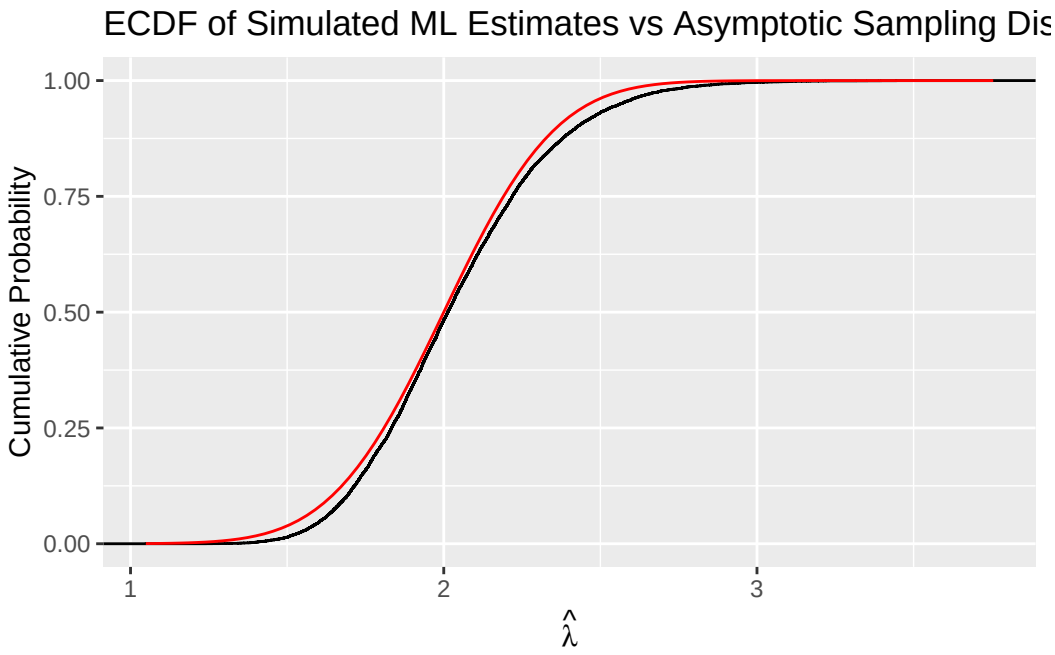
```
  difference = emp_q - asy_q
)

# make table
tt(
  quant,
  digits  = 3,   # no rounding for the text column; 3 decimals for numbers
  align   = "lrr",
)
```

p	empirical	asymptotic	difference
0.01	1.47	1.34	0.1302
0.05	1.61	1.53	0.073
0.1	1.69	1.64	0.0503
0.25	1.83	1.81	0.0242
0.5	2.01	2	0.013
0.75	2.22	2.19	0.028
0.9	2.43	2.36	0.0628
0.95	2.56	2.47	0.0995
0.99	2.83	2.66	0.176

```
# load packages
library(tidyverse)

# plot ecdf and asymptotic distribution
gg_data <- data.frame(lambda_hat = lambda_hat)
ggplot(gg_data, aes(x = lambda_hat)) +
  stat_ecdf() +
  stat_function(fun = function(x) pnorm(x,
                                        mean = asy_mean,
                                        sd = asy_sd),
                color = "red") +
  labs(
    title = "ECDF of Simulated ML Estimates vs Asymptotic Sampling Distribution",
    x = expression(hat(lambda)),
    y = "Cumulative Probability"
  )
```



Discussion

- **Center (bias).** The empirical mean of $\hat{\lambda}$ is about 2.04, compared to the true $\lambda = 2.000$. This is a difference of roughly 2%. At $N = 50$, this upward bias is present but quite small.
- **Spread.** The empirical SD is about 0.293, while the asymptotic SD is 0.283. This difference is 3.7% of the asymptotic value, so the asymptotic variance formula provides an accurate description of variability.
- **Quantiles and tails.** For the median, the empirical value is 2.013 compared to an asymptotic prediction of 2. At the 1% and 99% quantiles, the differences are about 0.13 and 0.176, respectively, which correspond to differences of only a few percent relative to the true scale. The ECDF and normal CDF curves are nearly indistinguishable except in the tails.
- **Bottom line.** With $N = 50$, the asymptotic normal distribution closely matches the finite-sample distribution of $\hat{\lambda}$. The bias is about 2%, the SD differs by about 4%, and quantile differences are small. The asymptotic normal approximation is accurate and reliable for practical use at this sample size.

Exercise 3 German Tank Problem

Suppose a discrete uniform distribution from 0 to K. The pmf is $f(x; K) = \frac{1}{K+1}, \quad x \in \{0, 1, \ldots, K\}$. The ML estimator is $\hat{K} = \max(y)$. The method of moments estimator is $\hat{K} = 2 \times \text{avg}(y)$. Use a Monte Carlo simulation to evaluate bias and variance of each estimator for

a small ($N = 3$), medium ($N = 25$), and large sample size ($N = 1,000$). Which estimator do you recommend?

Note: This model violates several of the regularity conditions, so the usual asymptotic guarantees do not hold.

Solution

```r
# load packages
library(tinytable)
library(dplyr)
options(tinytable_tt_digits = 3)
options(tinytable_latex_placement = "H")

# set seed for reproducibility
set.seed(1234)

# parameters
K  <- 100               # true K
Ns <- c(3, 25, 1000)    # sample sizes
R  <- 10000             # number of mc repetitions

# container: list of data frames
res_list <- list()

# loop over sample sizes
for (N in Ns) {
  K_hat_mle <- numeric(R)   # container for MLE
  K_hat_mom <- numeric(R)   # container for MOM

  for (i in 1:R) {
    y <- sample(0:K, N, replace = TRUE)  # simulate data set
    K_hat_mle[i] <- max(y)                # MLE
    K_hat_mom[i] <- 2 * mean(y)           # MOM
  }

  # empirical summaries
  emp_mean_mle <- mean(K_hat_mle)
  emp_sd_mle   <- sd(K_hat_mle)
  emp_mean_mom <- mean(K_hat_mom)
  emp_sd_mom   <- sd(K_hat_mom)

  # store results for this N
  res_list[[as.character(N)]] <- data.frame(
```

```
    N          = N,
    Estimator  = c("MLE", "MOM"),
    Mean       = c(emp_mean_mle, emp_mean_mom),
    SD         = c(emp_sd_mle,   emp_sd_mom),
    Bias       = c(emp_mean_mle - K, emp_mean_mom - K),
    MSE        = c(mean((K_hat_mle - K)^2), mean((K_hat_mom - K)^2))
  )
}

# combine all into one data frame
res <- bind_rows(res_list)

# print table
tt(
  res,
  digits = 5,
  align  = "lrrrrr",
)
```

N	Estimator	Mean	SD	Bias	MSE
3	MLE	75.055	19.7364	-24.9454	1011.7588
3	MOM	99.952	33.59	-0.048467	1128.1803
25	MLE	96.665	3.6737	-3.3353	24.6191
25	MOM	100.191	11.7466	0.191144	138.0057
1000	MLE	100	0	0	0
1000	MOM	99.993	1.8362	-0.007073	3.3712

Notice that the ML estimator is severely biased downward in small samples. But it also has a much smaller variance. It's not clear how to tradeoff these two. However, it's worth noting the the severely biased ML estimator has a *smaller* mean-squared error.

Exercise 4 Jensen's Inequality

Suppose the usual exponential model. The ML estimate of the rate λ is $\hat{\lambda} = \frac{1}{\text{avg}(y)}$. Using the invariance property, the ML estimate of the mean $\frac{1}{\lambda}$ is $\hat{\mu} = \frac{1}{\hat{\lambda}}$. Show analytically that the ML estimate of the mean is unbiased, while the ML estimate of the rate is biased.

Jensen's Inequality. If g is a convex function and X is a random variable with $\mathbb{E}[X]$ finite, then $\mathbb{E}[g(X)] \geq g(\mathbb{E}[X])$, with strict inequality unless X is degenerate (i.e., constant almost surely).

Hint: First show that $\hat{\mu}$ is unbiased. Then use Jensen's inequality to show that $\hat{\lambda}$ is biased upward. You do not need to find the size of the bias, just use Jensen's inequality to show that the ML estimate of the rate is biased. To use Jensen's inequality, let $g(x) = 1/x$, and notice that $g(x)$ is convex on $(0, \infty)$ (since $g''(x) = 2/x^3 > 0$).

Solution

1) Unbiasedness of the ML estimate of the mean

$\hat{\mu} = \text{avg}(y) = \frac{1}{N}\sum_{i=1}^{N} y_i$ is the sample mean of iid observations. $\mathbb{E}[y_i] = \frac{1}{\lambda}$. Then

$$\begin{aligned}
\mathbb{E}[\hat{\mu}] &= \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N} y_i\right] && \text{(definition of } \hat{\mu}) \\
&= \frac{1}{N} \cdot \mathbb{E}\left[\sum_{i=1}^{N} y_i\right] && \text{(linearity)} \\
&= \frac{1}{N} \cdot \left[\sum_{i=1}^{N} \mathbb{E}(y_i)\right] && \text{(sum of expectations)} \\
&= \frac{1}{N} \cdot \left[\sum_{i=1}^{N} \frac{1}{\lambda}\right] && \text{(each has mean } 1/\lambda) \\
&= \frac{1}{N} \cdot \left[N \cdot \frac{1}{\lambda}\right] && \text{(sum } N \text{ identical terms)} \\
&= \frac{1}{\lambda} && \text{(simplify)} \\
&= \mu && \text{(definition of } \mu)
\end{aligned}$$

Because $\mathbb{E}(\hat{\mu}) = \mu$, $\hat{\mu}$ is unbiased estimate of the mean.

2) Bias of the ML estimate of the rate

$\hat{\lambda} = \dfrac{1}{\text{avg}(y)} = \left(\dfrac{1}{N}\sum_{i=1}^{N} y_i\right)^{-1}$. Let $g(x) = 1/x$, and notice that $g(x)$ is convex on $(0, \infty)$ (since $g''(x) = 2/x^3 > 0$). Then

$$
\begin{aligned}
\mathbb{E}[\widehat{\lambda}] &= \mathbb{E}\left[\frac{1}{\text{avg}(y)}\right] && \text{(definition of } \widehat{\lambda}\text{)} \\
&= \mathbb{E}\left[g(\text{avg}(y))\right] && \text{(definition of } g \text{ above)} \\
&> g(\mathbb{E}[\text{avg}(y)]) && \text{(Jensen's inequality, } g \text{ convex)} \\
&= \frac{1}{\mathbb{E}[\text{avg}(y)]} && \text{(apply } g\text{)} \\
&= \frac{1}{\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}(y_i)} && \text{(linearity of expectation)} \\
&= \frac{1}{\frac{1}{N}\sum_{i=1}^{N}\frac{1}{\lambda}} && \text{(each } y_i \text{ has mean } 1/\lambda\text{)} \\
&= \frac{1}{\frac{1}{N}\cdot N\cdot\frac{1}{\lambda}} && \text{(sum of } N \text{ identical terms)} \\
&= \lambda && \text{(simplify)}
\end{aligned}
$$

Since the inequality is strict for finite N (the distribution of $\text{avg}(y)$ is non-degenerate), $\mathbb{E}[\widehat{\lambda}] > \lambda$. Thus, the ML estimate of the **rate** is biased upward.

Exercise 5 Poisson

Suppose a Poisson model. The ML estimate of the mean parameter λ is $\widehat{\lambda} = \text{avg}(y)$. Find an estimate of the SE using the observed Fisher information.

Solution

The log-likelihood is $\ell(\lambda) = \sum_{i=1}^{N}\left[y_i \log\lambda - \lambda - \log(y_i!)\right]$.

First, find the second derivative with respect to λ.

$$\ell(\lambda) = \sum_{i=1}^{N} \left[y_i \log \lambda - \lambda - \log(y_i!) \right] \qquad \text{Poisson log-likelihood}$$

$$\frac{\partial \ell}{\partial \lambda} = \sum_{i=1}^{N} \left(y_i \cdot \frac{1}{\lambda} - 1 - 0 \right) \qquad \text{use } \frac{d}{d\lambda} \log \lambda = \frac{1}{\lambda},\ \frac{d}{d\lambda}(-\lambda) = -1,\ \log(y_i!) \text{ is constant}$$

$$= \sum_{i=1}^{N} \left(\frac{y_i}{\lambda} - 1 \right) = \frac{1}{\lambda} \sum_{i=1}^{N} y_i - N \qquad \text{collect terms}$$

$$\frac{\partial^2 \ell}{\partial \lambda^2} = \frac{\partial}{\partial \lambda} \left(\frac{1}{\lambda} \sum_{i=1}^{N} y_i - N \right) \qquad \text{differentiate again}$$

$$= \sum_{i=1}^{N} \left(-\frac{y_i}{\lambda^2} \right) - 0 \qquad \text{use } \frac{d}{d\lambda} \left(\frac{1}{\lambda} \right) = -\frac{1}{\lambda^2},\ N \text{ is constant}$$

$$= -\sum_{i=1}^{N} \frac{y_i}{\lambda^2}.$$

The observed Fisher information is

$$\mathcal{J}_{\text{obs}}(\lambda) = -\frac{\partial^2 \ell}{\partial \lambda^2} = \sum_{i=1}^{N} \frac{y_i}{\lambda^2}.$$

Asymptotically, $\text{Var}(\hat{\lambda}) \approx \mathcal{J}_{\text{obs}}(\lambda)^{-1}$.[1]

To find the estimate of the SE, replace λ with $\hat{\lambda}$:

$$\widehat{\text{SE}}(\hat{\lambda}) = \sqrt{\mathcal{J}_{\text{obs}}(\hat{\lambda})^{-1}} = \sqrt{\frac{\hat{\lambda}}{N}} = \sqrt{\frac{\text{avg}(y)}{N}}.$$

Exercise 6 `operations`

The code below loads the number of enforcement operations for each district in Bogota from Holland (2015) from the {crdata} package. See `?crdata::holland2015` for details.

[1] Rather than "approximately equals" ($\approx$), it's more technically correct to write "asymptotically equals." However, we treat "asymptotically equals" to mean "approximately" in practice. Given the practical orientation of the course, it seems better to write "approximately" here.

```r
# load holland2015 data set from crdata package
holland2015 <- crdata::holland2015

# get operations in bogota only
ops <- holland2015$operations[holland2015$city == "bogota"]

# print
ops
```

```
 [1] 15 10 15 20  8  4  8 16  4 28  2  8  4  4  4  4  8  3  4
```

A Poisson model.

Task: Using the Poisson estimators from the previous question, find the ML estimate and SE estimate of λ for a Poisson model.

But before moving on, compare the SD of the predictive distribution (no need to simulate, it depends only and directly on λ) and the SD of the data. Why might this be a problem for the SE estimate?

A negative-binomial model.

For the Poisson model, the mean and variance of the outcome are directly linked. In fact, both equal λ. For most outcomes (almost all?), the variance is *much* larger than the average. This means a huge mismatch between the variance of the outcomes and the model.

The negative binomial model has an additional parameter r that allows the variance to be larger than (but not smaller than) the mean. For almost all datasets, this is a good default.

The negative binomial distribution is often parameterized by a mean $\mu > 0$ and a dispersion (or "size") parameter $r > 0$.[2] Its probability mass function is

$$\Pr(Y = y) \;=\; \frac{\Gamma(y + r)}{\Gamma(r)\, y!} \left(\frac{r}{r + \mu}\right)^{r} \left(\frac{\mu}{r + \mu}\right)^{y}, \qquad y = 0, 1, 2, \ldots,$$

which has mean μ and variance $\mu + \mu^2/r$. In R, the base function `dnbinom()` directly supports this mean–size parameterization when the `mu` argument is supplied (e.g., `dnbinom(y, size = r, mu = mu)`).

Task. Use `optim()` to estimate the mean and its SE for the `ops` variable. Compare to the Poisson estimates.

Solution.

[2] There are other common parameterizations that are less interpretable for our purposes.

First, fit the Poisson model and compute the ML estimate and SE of λ without using `optim()`, following the observed-information derivation above.

```
# load packages
library(tidyverse)
library(tinytable)

# load holland2015 data set from crdata package
holland2015 <- crdata::holland2015

# get operations in bogota only
ops <- holland2015$operations[holland2015$city == "bogota"]

# estimate lambda with ml
N <- length(ops)
lambda_hat_pois <- mean(ops); lambda_hat_pois
```

```
[1] 8.894737
```

```
se_lambda_hat_pois  <- sqrt(lambda_hat_pois / N); se_lambda_hat_pois
```

```
[1] 0.6842105
```

```
# compare predictive SD to data SD
sd_pred_pois <- sqrt(lambda_hat_pois)   # predictive SD under Poisson
sd_data      <- sd(ops)                 # empirical SD

# summarize results in a small table
res_pois <- data.frame(
  Statistic = c("Predictive SD ", "Data SD"),
  Value     = c(sd_pred_pois, sd_data)
)

tt(res_pois, digits = 3, align = "lr")
```

Statistic	Value
Predictive SD	2.98
Data SD	6.94

Now use `optim()` to fit a negative-binomial model. Compare to the Poisson estimates.

```r
# negative-binomial log-likelihood (mean-size parameterization)
nb_ll_fn <- function(theta, y) {
  mu <- theta[1]
  r  <- theta[2]
  sum(dnbinom(y, size = r, mu = mu, log = TRUE))
}

# function to fit nb model
est_nb <- function(y) {
  est <- optim(
    par     = c(3, 1),   # starting values
    fn      = nb_ll_fn,
    y       = y,
    control = list(fnscale = -1),
    hessian = TRUE
  )

  info_obs <- -est$hessian          # observed information
  var_hat  <- solve(info_obs)        # covariance matrix estimate

  if (est$convergence != 0) print("Model did not converge!")

  list(theta_hat = est$par, var_hat = var_hat)
}

# fit nb model to ops
fit_nb <- est_nb(ops)

# nb mean estimate and its se
mu_hat_nb <- fit_nb$theta_hat[1]; mu_hat_nb
```

```
[1] 8.894568
```

```r
se_mu_hat_nb  <- sqrt(fit_nb$var_hat[1,1]); se_mu_hat_nb
```

```
[1] 1.434298
```

```r
# comparison table
res_compare <- data.frame(
  Model      = c("Poisson", "Negative-binomial"),
```

```
  Parameter  = c("lambda",  "mu"),
  Estimate   = c(lambda_hat_pois, mu_hat_nb),
  SE         = c(se_lambda_hat_pois,  se_mu_hat_nb)
)

tt(res_compare, digits = 3, align = "l l r r")
```

Model	Parameter	Estimate	SE
Poisson	lambda	8.89	0.684
Negative-binomial	mu	8.89	1.434

Exercise 7 Exponential

Suppose an exponential model of y. We know that the ML estimate of the rate λ is $\widehat{\lambda} = \frac{1}{\text{avg}(y)}$. By the invariance property, the ML estimate of the mean $\mu = \frac{1}{\lambda}$ is $\widehat{\mu} = \frac{1}{\widehat{\lambda}}$.

1. Use the observed Fisher information to find a closed-form estimate of the SE of $\widehat{\lambda}$.
2. Then use the delta method to find a closed-form estimate of the SE of $\widehat{\mu}$.

Hint: For an exponential model with rate λ, the density is $f(y;\lambda) = \lambda e^{-\lambda y}$, so $\ell(\lambda) = \sum (\log \lambda - \lambda y_i)$.

Solution

The log-likelihood is

$$\ell(\lambda) \;=\; \sum_{i=1}^{N} \big[\log \lambda \;-\; \lambda y_i\big].$$

First, find the second derivative with respect to λ.

$$\ell(\lambda) = \sum_{i=1}^{N} \big[\log\lambda - \lambda y_i\big] \qquad \text{Exponential log-likelihood}$$

$$\frac{\partial \ell}{\partial \lambda} = \sum_{i=1}^{N} \left(\frac{1}{\lambda} - y_i\right) \qquad \text{use } \frac{d}{d\lambda}\log\lambda = \frac{1}{\lambda},\ \frac{d}{d\lambda}(-\lambda y_i) = -y_i$$

$$= \frac{N}{\lambda} \ - \ \sum_{i=1}^{N} y_i \qquad \text{collect terms}$$

$$\frac{\partial^2 \ell}{\partial \lambda^2} = \frac{\partial}{\partial \lambda}\left(\frac{N}{\lambda} - \sum_{i=1}^{N} y_i\right) \qquad \text{differentiate again}$$

$$= -\frac{N}{\lambda^2} \ - \ 0 \qquad \text{use } \frac{d}{d\lambda}\left(\frac{1}{\lambda}\right) = -\frac{1}{\lambda^2},\ \sum y_i \text{ is constant.}$$

The observed Fisher information is

$$\mathcal{J}_{\text{obs}}(\lambda) = -\frac{\partial^2 \ell}{\partial \lambda^2} = \frac{N}{\lambda^2}.$$

To find the estimate of the SE, replace λ with $\hat{\lambda}$:

$$\widehat{\text{SE}}(\hat{\lambda}) = \sqrt{\mathcal{J}_{\text{obs}}(\hat{\lambda})^{-1}} = \sqrt{\frac{\hat{\lambda}^2}{N}} = \frac{\hat{\lambda}}{\sqrt{N}} = \frac{1}{\sqrt{N}\ \text{avg}(y)}.$$

Now suppose we use the invariance property to obtain $\hat{\mu} = \dfrac{1}{\hat{\lambda}}$. Then we can use the delta method to estimate the SE with $\tau(\lambda) = \dfrac{1}{\lambda}$.

The first derivative is $\tau'(\lambda) = -\frac{1}{\lambda^2}$.

Plugging in, we have

$$\widehat{\text{Var}}(\hat{\mu}) \approx \big[\tau'(\hat{\lambda})\big]^2 \cdot \widehat{\text{Var}}(\hat{\lambda}) \ = \ \left(\frac{1}{\hat{\lambda}^2}\right)^2 \cdot \frac{\hat{\lambda}^2}{N}$$

$$= \frac{1}{N\,\hat{\lambda}^2}.$$

And the standard error is

$$\widehat{\text{SE}}(\hat{\mu}) \ \approx \ \sqrt{\frac{1}{N\,\hat{\lambda}^2}} \ = \ \frac{1}{\sqrt{N}\,\hat{\lambda}} \ = \ \frac{\text{avg}(y)}{\sqrt{N}}.$$

Exercise 8 Delta method, by hand

Suppose you obtain the ML estimate $\hat{\theta} = (\hat{a}, \hat{b}) = (2, 1)$ with an estimated covariance matrix of

$$\widehat{\text{Var}}(\hat{\theta}) = \widehat{\text{Var}}(\hat{a}, \hat{b}) = \begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{bmatrix}.$$

However, you want to estimate the quantity of interest $q = \tau(a, b) = \dfrac{a}{b}$. Find $\hat{q}$ using the invariance property and estimate the SE of $\hat{q}$ using the delta method. (Find the gradient analytically.) Confirm your work with R.

Solution

First, find the gradient. For $\tau(a, b) = a/b$,

$$\nabla\tau(a, b) = \begin{bmatrix} \frac{\partial}{\partial a}(a/b) \\ \frac{\partial}{\partial b}(a/b) \end{bmatrix} = \begin{bmatrix} 1/b \\ -a/b^2 \end{bmatrix}.$$

Plugging in $(\hat{a}, \hat{b}) = (2, 1)$, we have $\nabla\tau(\hat{a}, \hat{b}) = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$.

Now do the required matrix multiplication.

$$\widehat{\text{Var}}[\tau(\hat{\theta})] \approx \overbrace{\begin{bmatrix} 1 & -2 \end{bmatrix}}^{\nabla\tau(\hat{\theta})^\top} \left(\underbrace{\begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{bmatrix}}_{\widehat{\text{Var}}(\hat{\theta})} \underbrace{\begin{bmatrix} 1 \\ -2 \end{bmatrix}}_{\nabla\tau(\hat{\theta})} \right)$$

$$= \begin{bmatrix} 1 & -2 \end{bmatrix} \begin{bmatrix} 1 \cdot 1 + \frac{1}{2} \cdot (-2) \\ \frac{1}{2} \cdot 1 + 1 \cdot (-2) \end{bmatrix} = \begin{bmatrix} 1 & -2 \end{bmatrix} \begin{bmatrix} 0 \\ -1.5 \end{bmatrix} = 0 + 3 = 3.$$

Thus $\widehat{\text{Var}}[\hat{\tau}] = 3$ and $\widehat{\text{SE}}(\hat{\tau}) = \sqrt{3}$.

We can confirm our work with R.

```
# given values
a_hat <- 2
b_hat <- 1
V_hat <- matrix(c(1, 1/2,
                  1/2, 1), nrow = 2, byrow = TRUE)

# gradient at (a_hat, b_hat) for tau(a,b) = a/b
```

```r
grad <- c(1 / b_hat, -a_hat / b_hat^2)  # c(1, -2)

# delta method
var_tau_hat <- grad %*% V_hat %*% grad
se_tau_hat  <- sqrt(var_tau_hat)

var_tau_hat  # 3
```

```
     [,1]
[1,]    3
```

```r
se_tau_hat   # sqrt(3)
```

```
         [,1]
[1,] 1.732051
```

Exercise 9 (α, β) to SD

In the notes, we saw how to use the invariance principle and the delta method to obtain an ML estimate of the mean μ and its SE from ML estimates of α and β and their estimated covariance matrix.

Similarly, use the invariance principle and the delta method to find an ML estimate of the SD σ and its SE for the `batting_average` variable in Lahman's data. Recall that $\sigma = \sqrt{\dfrac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}}$ for the beta distribution. Use a numerical gradient.

Solution.

First, find the ML estimates of α and β and estimate their covariance.

```r
# log-likelihood function (using dbeta)
beta_ll_fn <- function(theta, y) {
  alpha <- theta[1]
  beta  <- theta[2]
  sum(dbeta(y, shape1 = alpha, shape2 = beta, log = TRUE))
}

# function to fit beta model
est_beta <- function(y) {
  est <- optim(
```

```
    par     = c(2, 2),      # decent starting values
    fn      = beta_ll_fn,
    y       = y,
    control = list(fnscale = -1),
    method  = "BFGS",
    hessian = TRUE
  )

  info_obs <- -est$hessian            # observed information (note the negative)
  var_hat  <- solve(info_obs)         # covariance matrix estimate

  if (est$convergence != 0) print("Model did not converge!")

  list(theta_hat = est$par, var_hat = var_hat)
}

# load packages
library(tidyverse)
library(Lahman)  # data from Lahman's baseball database

# create data frame with batting average
bstats <- battingStats() |>
  filter(yearID == 2023, AB >= 100) |>            # 2023; at least 100 AB
  transmute(player_id = playerID, batting_average = BA) |>
  drop_na() |>
  arrange(desc(batting_average))

# estimate beta model using the batting average data
fit <- est_beta(bstats$batting_average)
fit$theta_hat
```

```
[1]  37.07655 114.92550
```

```
fit$var_hat
```

```
          [,1]     [,2]
[1,]  5.964783 18.40870
[2,] 18.408705 57.83667
```

To find $\hat{\sigma}$ plug $(\hat{\alpha}, \hat{\beta})$ into $\sigma = \tau(\alpha, \beta)$.

```r
# compact notation for readability (matches the notes)
a <- fit$theta_hat[1]  # alpha_hat
b <- fit$theta_hat[2]  # beta_hat

# sigma_hat via invariance property
sigma_hat <- sqrt( (a*b) / ((a + b)^2 * (a + b + 1)) )
sigma_hat
```

```
[1] 0.03471841
```

We use a numerical gradient for

$$\tau(\theta) = \sigma(\alpha, \beta) = \sqrt{\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}}.$$

Recall the delta method $\widehat{\mathrm{Var}}[\tau(\hat{\theta})] \approx \nabla\tau(\hat{\theta})^\top \widehat{\mathrm{Var}}(\hat{\theta})\nabla\tau(\hat{\theta})$. Here, we use a numerical $\nabla\tau(\hat{\theta})$ computed using `numDeriv::grad()`.

```r
library(numDeriv)   # for numerical gradients

# define tau(theta) = sigma(alpha, beta)
sigma_fn <- function(theta) {
  a <- theta[1]; b <- theta[2]
  sqrt( (a*b) / ((a + b)^2 * (a + b + 1)) )
}

# numerical gradient of tau at (a, b)
grad_sigma <- grad(func = sigma_fn, x = fit$theta_hat)
grad_sigma   # 2 x 1 gradient vector evaluated at (a, b)
```

```
[1]  0.0001263342 -0.0001908174
```

```r
# delta method variance and SE
var_hat_sigma <- t(grad_sigma) %*% fit$var_hat %*% grad_sigma
se_hat_sigma  <- sqrt(var_hat_sigma)

var_hat_sigma
```

```
             [,1]
[1,] 1.313558e-06
```

```
se_hat_sigma
```

```
              [,1]
[1,] 0.001146105
```

Exercise 10 Turnout rates

Load a tibble version of the 2024 turnout data from the UF Election Lab into R.

Model the proportion of the voting-eligible population that voted using a beta distribution. Estimate the parameters of the beta distribution α and β as well as the mean $\mu = \frac{\alpha}{\alpha+\beta}$ and the SD $\sigma = \sqrt{\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}}$ }$. Use a parametric bootstrap to estimate the SE of each.

Solution

```
# load packages
library(tidyverse)
library(tinytable)

# load data as tibble
turnout_2024 <- tribble(
  ~state,               ~vep_turnout,
  "Alabama",            0.5893,
  "Alaska",             0.6378,
  "Arizona",            0.6360,
  "Arkansas",           0.5348,
  "California",         0.6206,
  "Colorado",           0.7314,
  "Connecticut",        0.6708,
  "Delaware",           0.6702,
  "District of Columbia", 0.6357,
  "Florida",            0.6671,
  "Georgia",            0.6826,
  "Hawaii",             0.5027,
  "Idaho",              0.6344,
  "Illinois",           0.6325,
  "Indiana",            0.5869,
  "Iowa",               0.7078,
  "Kansas",             0.6318,
  "Kentucky",           0.6219,
  "Louisiana",          0.6077,
```

```
  "Maine",            0.7424,
  "Maryland",         0.6930,
  "Massachusetts",    0.6803,
  "Michigan",         0.7464,
  "Minnesota",        0.7635,
  "Mississippi",      0.5745,
  "Missouri",         0.6434,
  "Montana",          0.6820,
  "Nebraska",         0.6796,
  "Nevada",           0.6580,
  "New Hampshire",    0.7405,
  "New Jersey",       0.6724,
  "New Mexico",       0.5957,
  "New York",         0.6044,
  "North Carolina",   0.7032,
  "North Dakota",     0.6310,
  "Ohio",             0.6539,
  "Oklahoma",         0.5328,
  "Oregon",           0.7194,
  "Pennsylvania",     0.7143,
  "Rhode Island",     0.6328,
  "South Carolina",   0.6214,
  "South Dakota",     0.6400,
  "Tennessee",        0.5761,
  "Texas",            0.5657,
  "Utah",             0.6415,
  "Vermont",          0.7089,
  "Virginia",         0.7119,
  "Washington",       0.7017,
  "West Virginia",    0.5546,
  "Wisconsin",        0.7664,
  "Wyoming",          0.6128
)

# beta log-likelihood
beta_ll_fn <- function(theta, y) {
  alpha <- theta[1]
  beta <- theta[2]
  ll <- sum(dbeta(y, shape1 = alpha, shape2 = beta, log = TRUE))
  return(ll)
}
```

```r
# create a function to fit the beta model
est_beta <- function(y) {
  est <- optim(par = c(3, 3), fn = beta_ll_fn, y = y,
               control = list(fnscale = -1),
               method = "BFGS") # for >1d problems
  if (est$convergence != 0) print("Model did not converge!")
  res <- list(est = est$par)
  return(res)
}

# ml estimates
ml_est <- est_beta(turnout_2024$vep_turnout)
a <- ml_est$est[1]; a  # alpha
```

```
[1] 39.59784
```

```r
b <- ml_est$est[2]; b  # beta
```

```
[1] 21.29332
```

```r
a / (a + b)  # mean; using a and b for readability
```

```
[1] 0.6503053
```

```r
sqrt( (a * b) / ((a + b)^2 * (a + b + 1)) ) # SD
```

```
[1] 0.06061623
```

```r
# parametric bootstrap
n_bs <- 100
bs_alpha <-
  bs_beta <-
  bs_mu <-
  bs_sigma <-
  numeric(n_bs)  # containers for the estimates

for (i in 1:n_bs) {
  # bootstrap sample
  bs_y <- rbeta(length(y), shape1 = ml_est$est[1], shape2 = ml_est$est[2])
```

```
  # fit model to bootstrapped sample
  bs_est <- est_beta(bs_y)

  # store ml estimates of alpha and beta for readability below
  bs_a <- bs_est$est[1]
  bs_b <- bs_est$est[2]

  # compute and store all qis
  bs_alpha[i] <- bs_a
  bs_beta[i]  <- bs_b
  bs_mu[i]    <- bs_a / (bs_a + bs_b)
  bs_sigma[i] <- sqrt( (bs_a * bs_b) / ((bs_a + bs_b)^2 * (bs_a + bs_b + 1)) )
}

# bootstrap SEs
se_alpha <- sd(bs_alpha)
se_beta  <- sd(bs_beta)
se_mu    <- sd(bs_mu)
se_sigma <- sd(bs_sigma)

# tinytable (avoid underscores in visible labels)
tab_df <- tibble::tibble(
  Parameter     = c("$\\alpha$", "$\\beta$", "$\\mu$", "$\\sigma$"),
  `ML estimate` = c(a_hat = a,
                    b_hat = b,
                    mu_hat = a / (a + b),
                    sigma_hat = sqrt( (a * b) / ((a + b)^2 * (a + b + 1)) )),
  `Bootstrap SE`= c(se_alpha, se_beta, se_mu, se_sigma)
)
tt(
  tab_df,
  digits = 3,
  caption = "Beta model: ML estimates and parametric-bootstrap standard errors.",
)
```

Table 1: Beta model: ML estimates and parametric-bootstrap standard errors.

Parameter	ML estimate	Bootstrap SE
α	39.5978	1.4765
β	21.2933	0.78647
μ	0.6503	0.00209
σ	0.0606	0.00112

Exercise 11 `duration`

Part 1. *Parametric models.*

Model `duration` in the `coalition` data set (from the **brglm2** package) using three different distributions: an exponential distribution, a Weibull distribution, and a log-normal distribution.

For each distribution:

1. Estimate the parameter(s) using maximum likelihood.
2. Estimate the covariance matrix.
3. Estimate the mean using the invariance property.
4. Estimate the SE using the delta method.

Hint: Use `optim()` and `numDeriv::grad()`. Complete each step for the exponential model, then copy-and-paste the code and make the needed changes for the Weibull and log-normal.

Part 2. Classical estimate.

Compute the classical mean and SE estimates by taking the average $\widehat{\mu} = \text{avg}(y)$ and the standard error $\widehat{\text{SE}}(\widehat{\mu}) = \dfrac{\text{SD}(y)}{\sqrt{N}}$.

Part 3.

Compare these four estimates and their standard error estimates. What stands out? Explain.

Some Background.

When we model duration data, we often focus on the hazard function, which describes the relative chance of an event at time t, given that the unit has survived until t. The hazard tells us how the risk of failure changes with time.

The *exponential distribution* is the simplest model. The density is

$$f(t \mid \lambda) = \lambda e^{-\lambda t}, \quad t > 0,\ \lambda > 0,$$

and the mean is $\mathbb{E}[T] = 1/\lambda$. The hazard is $h(t) = \lambda$, which is constant over time. This means the chance of failure in the next instant does not depend on how long the unit has lasted so far. This "memoryless" property makes the exponential convenient, but often unrealistic.

The *Weibull distribution* generalizes the exponential by allowing the hazard to change with time. The density is

$$f(t \mid k, \lambda) = \frac{k}{\lambda}\left(\frac{t}{\lambda}\right)^{k-1} \exp\left[-\left(\frac{t}{\lambda}\right)^{k}\right], \quad t > 0,\ k, \lambda > 0,$$

and the mean is $\mathbb{E}[T] = \lambda\,\Gamma(1 + 1/k)$. The hazard is

$$h(t) = \frac{k}{\lambda}\left(\frac{t}{\lambda}\right)^{k-1}.$$

When $k = 1$, this reduces to the exponential with a constant hazard. If $k > 1$, the hazard increases with time, which might capture processes like aging or wear-out. If $k < 1$, the hazard decreases with time, which might represent situations where early failures are more likely. This flexibility makes the Weibull especially useful.

The *lognormal distribution* assumes that the log of the durations is normally distributed. The density is

$$f(t \mid \mu, \sigma) = \frac{1}{t\sigma\sqrt{2\pi}} \exp\left(-\frac{(\ln t - \mu)^2}{2\sigma^2}\right), \quad t > 0,\ \sigma > 0,$$

and the mean is $\mathbb{E}[T] = \exp(\mu + \frac{1}{2}\sigma^2)$. The hazard is non-monotone: it starts low, rises for a while, and then falls. This "hump-shaped" hazard works well when events are most likely after some waiting time, but less likely if they haven't occurred by then—for example, the adoption of a new technology or the incubation period of a disease.

Solution.

```
# load packages
library(brglm2)
library(numDeriv)

# data
data("coalition", package = "brglm2")
y <- coalition$duration
```

1) Exponential model.

```r
# log-likelihood
log_lik <- function(lambda, y) {
  sum(dexp(y, rate = lambda, log = TRUE))
}

# find ml estimate
fit <- optim(
  par     = 1,
  fn      = log_lik,
  y       = y,
  method  = "BFGS",
  control = list(fnscale = -1),
  hessian = TRUE
)
```

```
Warning in dexp(y, rate = lambda, log = TRUE): NaNs produced
Warning in dexp(y, rate = lambda, log = TRUE): NaNs produced
Warning in dexp(y, rate = lambda, log = TRUE): NaNs produced
Warning in dexp(y, rate = lambda, log = TRUE): NaNs produced
Warning in dexp(y, rate = lambda, log = TRUE): NaNs produced
Warning in dexp(y, rate = lambda, log = TRUE): NaNs produced
Warning in dexp(y, rate = lambda, log = TRUE): NaNs produced
Warning in dexp(y, rate = lambda, log = TRUE): NaNs produced
Warning in dexp(y, rate = lambda, log = TRUE): NaNs produced
Warning in dexp(y, rate = lambda, log = TRUE): NaNs produced
Warning in dexp(y, rate = lambda, log = TRUE): NaNs produced
Warning in dexp(y, rate = lambda, log = TRUE): NaNs produced
Warning in dexp(y, rate = lambda, log = TRUE): NaNs produced
Warning in dexp(y, rate = lambda, log = TRUE): NaNs produced
Warning in dexp(y, rate = lambda, log = TRUE): NaNs produced
Warning in dexp(y, rate = lambda, log = TRUE): NaNs produced
Warning in dexp(y, rate = lambda, log = TRUE): NaNs produced
Warning in dexp(y, rate = lambda, log = TRUE): NaNs produced
Warning in dexp(y, rate = lambda, log = TRUE): NaNs produced
```

```r
lambda_hat <- fit$par

# covariance
info_obs <- -fit$hessian
var_hat  <- solve(info_obs)

# invariance property + delta method
```

```r
tau <- function(lambda) 1 / lambda
mean_hat_exp <- tau(lambda_hat)
g <- grad(tau, x = lambda_hat)  # 1x1 gradient
var_mean <- t(g) %*% var_hat %*% g
se_mean_exp  <- sqrt(var_mean)

# print key results
lambda_hat
```

```
[1] 0.0542343
```

```r
mean_hat_exp
```

```
[1] 18.43851
```

```r
se_mean_exp
```

```
         [,1]
[1,] 1.040192
```

2) Weibull model.

```r
# log-likelihood
log_lik <- function(theta, y) {
  k  <- theta[1]
  s <- theta[2]
  sum(dweibull(y, shape = k, scale = s, log = TRUE))
}

# find ml estimate
fit <- optim(
  par     = c(1, 1),
  fn      = log_lik,
  y       = y,
  method  = "BFGS",
  control = list(fnscale = -1),
  hessian = TRUE
)
```

```
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
Warning in dweibull(y, shape = k, scale = s, log = TRUE): NaNs produced
```

```r
k_hat  <- fit$par[1]
s_hat <- fit$par[2]

# covariance
info_obs <- -fit$hessian
var_hat  <- solve(info_obs)

# invariance property + delta method
tau <- function(theta) {
  k <- theta[1]
  s <- theta[2]
  s * gamma(1 + 1/k)
}
mean_hat_weib <- tau(c(k_hat, s_hat))
g <- numDeriv::grad(tau, x = c(k_hat, s_hat))
var_mean <- t(g) %*% var_hat %*% g
se_mean_weib  <- sqrt(var_mean)

# print key results
```

```r
k_hat
```

```
[1] 1.138367
```

```r
s_hat
```

```
[1] 19.28594
```

```r
mean_hat_weib
```

```
[1] 18.40969
```

```r
se_mean_weib
```

```
          [,1]
[1,] 0.9138076
```

3) Log-normal model.

```r
# log-likelihood
log_lik <- function(theta, y) {
  mu  <- theta[1]
  sig <- theta[2]
  sum(dlnorm(y, meanlog = mu, sdlog = sig, log = TRUE))
}

# find ml estimate
fit <- optim(
  par     = c(0, 1),
  fn      = log_lik,
  y       = y,
  method  = "BFGS",
  control = list(fnscale = -1),
  hessian = TRUE
)
```

```
Warning in dlnorm(y, meanlog = mu, sdlog = sig, log = TRUE): NaNs produced
Warning in dlnorm(y, meanlog = mu, sdlog = sig, log = TRUE): NaNs produced
Warning in dlnorm(y, meanlog = mu, sdlog = sig, log = TRUE): NaNs produced
Warning in dlnorm(y, meanlog = mu, sdlog = sig, log = TRUE): NaNs produced
Warning in dlnorm(y, meanlog = mu, sdlog = sig, log = TRUE): NaNs produced
Warning in dlnorm(y, meanlog = mu, sdlog = sig, log = TRUE): NaNs produced
Warning in dlnorm(y, meanlog = mu, sdlog = sig, log = TRUE): NaNs produced
Warning in dlnorm(y, meanlog = mu, sdlog = sig, log = TRUE): NaNs produced
Warning in dlnorm(y, meanlog = mu, sdlog = sig, log = TRUE): NaNs produced
Warning in dlnorm(y, meanlog = mu, sdlog = sig, log = TRUE): NaNs produced
```

```r
mu_hat  <- fit$par[1]
sig_hat <- fit$par[2]

# covariance
info_obs <- -fit$hessian
var_hat  <- solve(info_obs)

# invariance property + delta method
tau <- function(theta) {
  mu  <- theta[1]
  sig <- theta[2]
  exp(mu + 0.5 * sig^2)
}
mean_hat_lnorm <- tau(c(mu_hat, sig_hat))
g <- numDeriv::grad(tau, x = c(mu_hat, sig_hat))
var_mean <- t(g) %*% var_hat %*% g
se_mean_lnorm  <- sqrt(var_mean)

# print key results
mu_hat
```

```
[1] 2.4355
```

```r
sig_hat
```

```
[1] 1.147172
```

```r
mean_hat_lnorm
```

```
[1] 22.05417
```

```
se_mean_lnorm
```

```
         [,1]
[1,] 1.838451
```

4) Classical.

```
# classical mean and SE of coalition$duration
mean_classical <- mean(coalition$duration)
se_classical   <- sd(coalition$duration) / sqrt(length(coalition$duration))

mean_classical
```

```
[1] 18.4379
```

```
se_classical
```

```
[1] 0.8553566
```

A summary table.

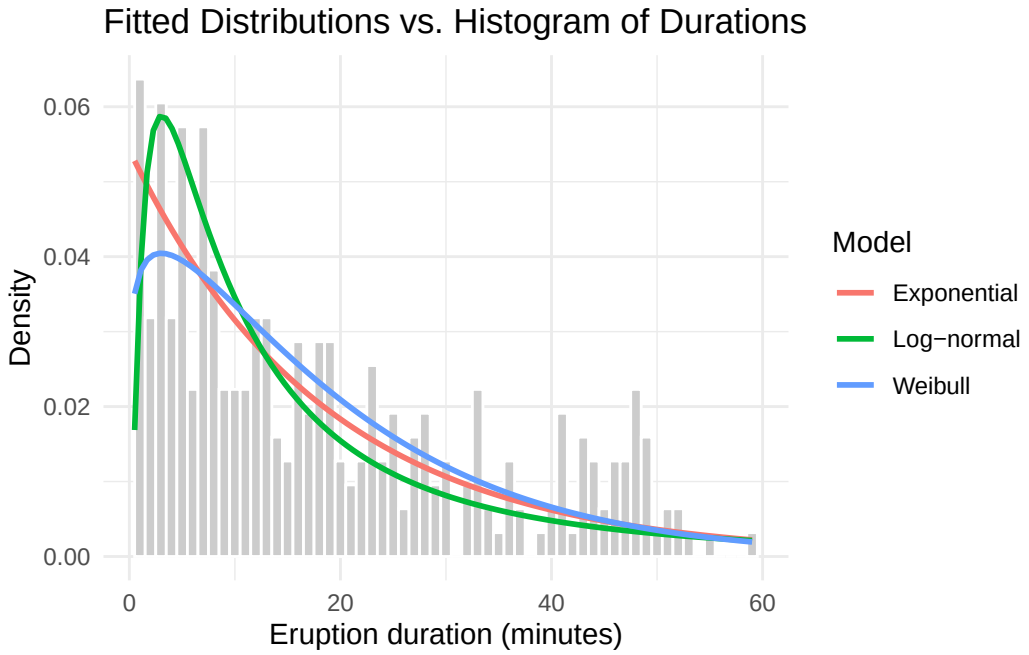
```
library(tinytable)

# collect results including classical
res <- data.frame(
  Model = c("Classical", "Exponential", "Weibull", "Log-normal"),
  Mean  = c(mean_classical, mean_hat_exp, mean_hat_weib, mean_hat_lnorm),
  SE    = c(se_classical,   se_mean_exp,  se_mean_weib,  se_mean_lnorm)
)

# format nicely
tt(
  res,
  digits = 4,
  align  = "lrr"
)
```

Model	Mean	SE
Classical	18.44	0.8554
Exponential	18.44	1.0402
Weibull	18.41	0.9138
Log-normal	22.05	1.8385

However, while these models make different parametric assumptions, they all fit the observed durations moderately well. At least none severely deviates from the observed pattern.



Exercise 12 Two CIs

Consider a simple Bernoulli model for $y \in \{0, 1\}$ with parameter $\pi \in [0, 1]$. Create a small sample with a rare event: take $N = 20$ trials with exactly one success.

Part 1. *Wald confidence interval.*

Compute the ML estimate $\hat{\pi} = \text{avg}(y)$, the Wald SE $\widehat{\text{SE}}(\hat{\pi}) = \sqrt{\hat{\pi}(1-\hat{\pi})/N}$, and the usual 95% Wald confidence interval $\hat{\pi} \pm 1.96 \cdot \widehat{\text{SE}}(\hat{\pi})$. What do you notice about the interval relative to the parameter space $[0, 1]$?

Part 2. *Parametric bootstrap confidence interval.*

Use a *parametric bootstrap* to obtain a 95% percentile confidence interval for π based on the fitted model Bernoulli($\hat{\pi}$). Compare the bootstrap interval with the Wald interval. What stands out? Briefly explain.

Hint: For the bootstrap, repeatedly simulate $y_1^{*(b)}, \ldots, y_N^{*(b)} \overset{iid}{\sim}$ Bernoulli($\hat{\pi}$), recompute $\hat{\pi}^{*(b)} = \text{avg}(y^{*(b)})$, and then take the empirical 2.5$^{\text{th}}$ and 97.5$^{\text{th}}$ percentiles of $\{\hat{\pi}^{*(b)}\}$.

Solution.

```
set.seed(1)

# data: N = 20 with exactly one success
N <- 20
y <- c(1, rep(0, N - 1))  # rare event sample
y <- sample(y)            # shuffle (irrelevant, but tidy)

# ml estimate and wald 95%ci
pi_hat   <- mean(y)
se <- sqrt(pi_hat * (1 - pi_hat) / N)
wald_ci <- pi_hat + c(-1, 1) * 1.96 * se

pi_hat
```

```
[1] 0.05
```

```
se
```

```
[1] 0.04873397
```

```
wald_ci
```

```
[1] -0.04551858  0.14551858
```

The Wald interval extends below 0, which is outside the parameter space $[0, 1]$.

```
# parametric bootstrap w/ percentile 95% ci
n_bs <- 2000
bs_est <- numeric(n_bs)  # a container for the estimates
for (i in 1:n_bs) {
  bs_y <- rbinom(20, size = 1, prob = pi_hat)
  bs_est[i] <- mean(bs_y)
```

```r
}

print(quantile(bs_est, probs = c(0.025, 0.975)), digits = 2)  # 95% ci
```

```
 2.5% 97.5%
 0.00  0.15
```

The Wald interval uses a normal approximation centered at $\hat{\pi}$ with variance estimated by Fisher information. With extreme $\hat{\pi}$ near zero and small N, this approximation is poor and the lower bound of the CI can dip below 0. In contrast, the parametric bootstrap respects the model's support by construction (every $\hat{\pi}^{*(b)} \in [0, 1]$), so the percentile interval stays within $[0, 1]$ and typically provides more reasonable coverage in this setting.