

# MSMS 206 : Practical 03

Ananda Biswas

March 11, 2025

## ➡ Objective

$$X_1, X_2, \dots, X_n \stackrel{\text{iid}}{\sim} \text{Exp}(\text{mean} = \theta).$$

- (i) Using CLT we have to show that  $\bar{X}$  is a CAN estimator for  $\theta$ .
- (ii) We also have to obtain a CAN estimator for  $P[X > t] = e^{-t/\theta}$  and its asymptotic variance.

## ➡ Theory, R Program, Plot and Interpretation

👉 First we shall show that  $\bar{X}$  is a consistent estimator for  $\theta$  i.e.  $P[|\bar{X}_n - \theta| < \epsilon]$  tends to 1 as sample size  $n$  increases.

We generate 10000 samples of size  $n$  and calculate the relative frequency of  $[|\bar{X}_n - \theta| < \epsilon]$ , this is the probability obtained by empirical approach. As sample size  $n$  increases, the empirical probability converges to 1.

We consider  $\theta = 2$ .

```
theta <- 2
```

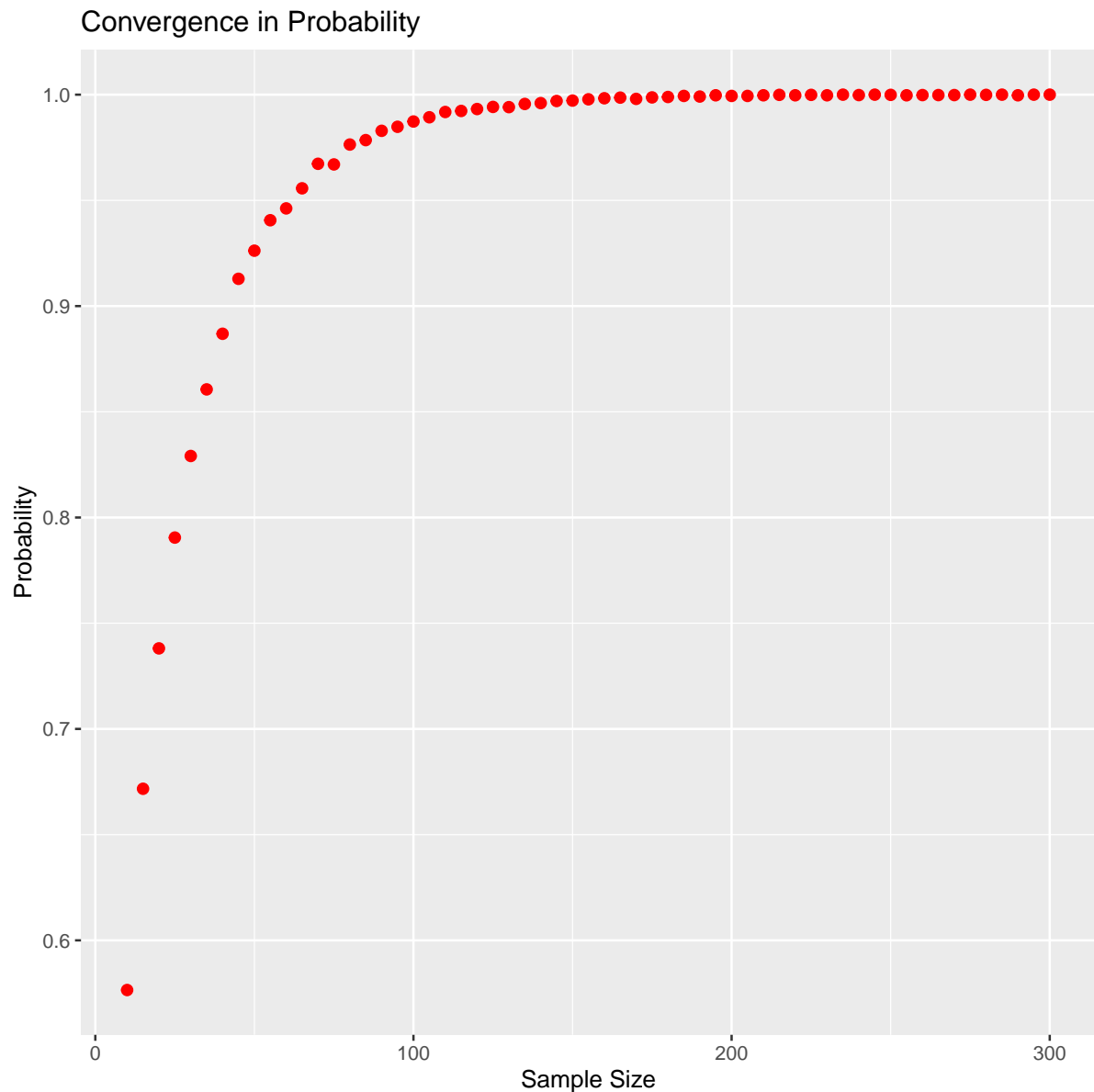
```
prob <- function(size, epsilon){  
  sample_list <- list()  
  for(i in 1:10000){  
    sample_list[[i]] <- rexp(size, rate = 1 / theta)  
  }  
  x_bar <- sapply(sample_list, mean)  
  m <- length(which(abs(x_bar - theta) < epsilon))  
  return(m/10000)  
}
```


Here we take  $\epsilon = 0.5$ .


```
probs <- c()  
for (i in seq(10, 300, 5)) {  
  probs <- append(probs, prob(size = i, epsilon = 0.5))  
}
```

```
df1 <- data.frame(sample_size = seq(10, 300, 5), probability = probs)

df1 %>%
  ggplot(aes(x = sample_size, y = probability)) +
  geom_point(size = 2, col = "red") +
  labs(x = "Sample Size", y = "Probability",
       title = "Convergence in Probability")
```



 As sample size increases, the probability converges to 1. This implies that the sample mean is a consistent estimator of  $\theta$ .

 Now we shall show that  $\bar{X}_n$  has an asymptotic normal distribution.

We generate 10000 samples of size  $n$ , from there we get 10000  $\bar{X}_n$ . We plot their histogram along with density curve. As sample size  $n$  increases, the density curve resembles that of a normal distribution.

First we take  $n = 2$ .

```
sample_list <- list(); size <- 2

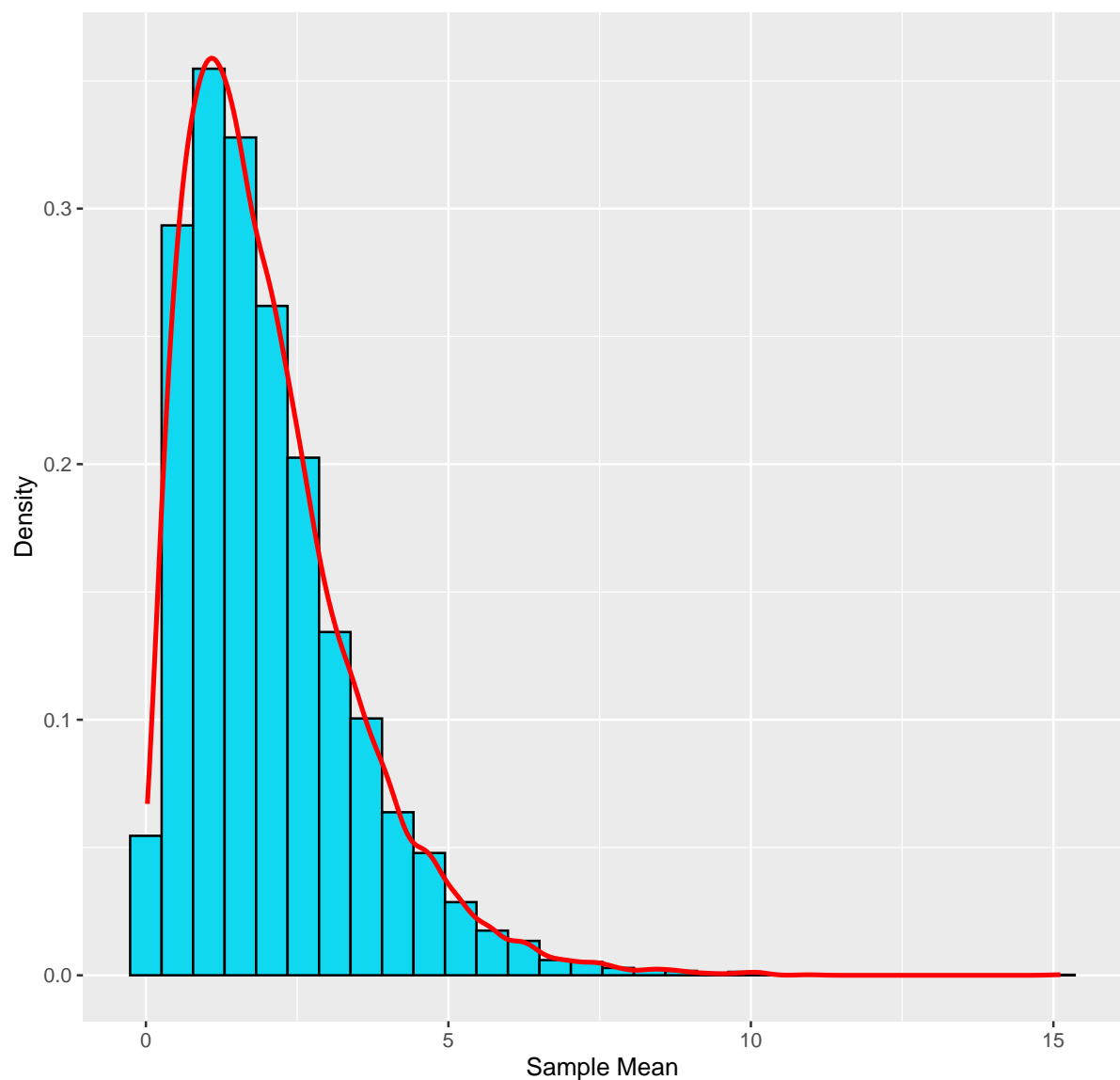
for(i in 1:10000){
  sample_list[[i]] <- rexp(size, rate = 1 / theta)
}

x_bar <- sapply(sample_list, mean)

df2 <- data.frame(means = x_bar)

df2 %>%
  ggplot(aes(x = means)) +
  geom_histogram(aes(y = ..density..), bins = 30, fill = "#0FD8F0", col = "black") +
  geom_density(col = "red", linewidth = 1) +
  labs(x = "Sample Mean", y = "Density",
       title = "Histogram with density curve, sample size = 2")
```

Histogram with density curve, sample size = 2



Next we take  $n = 5$ .

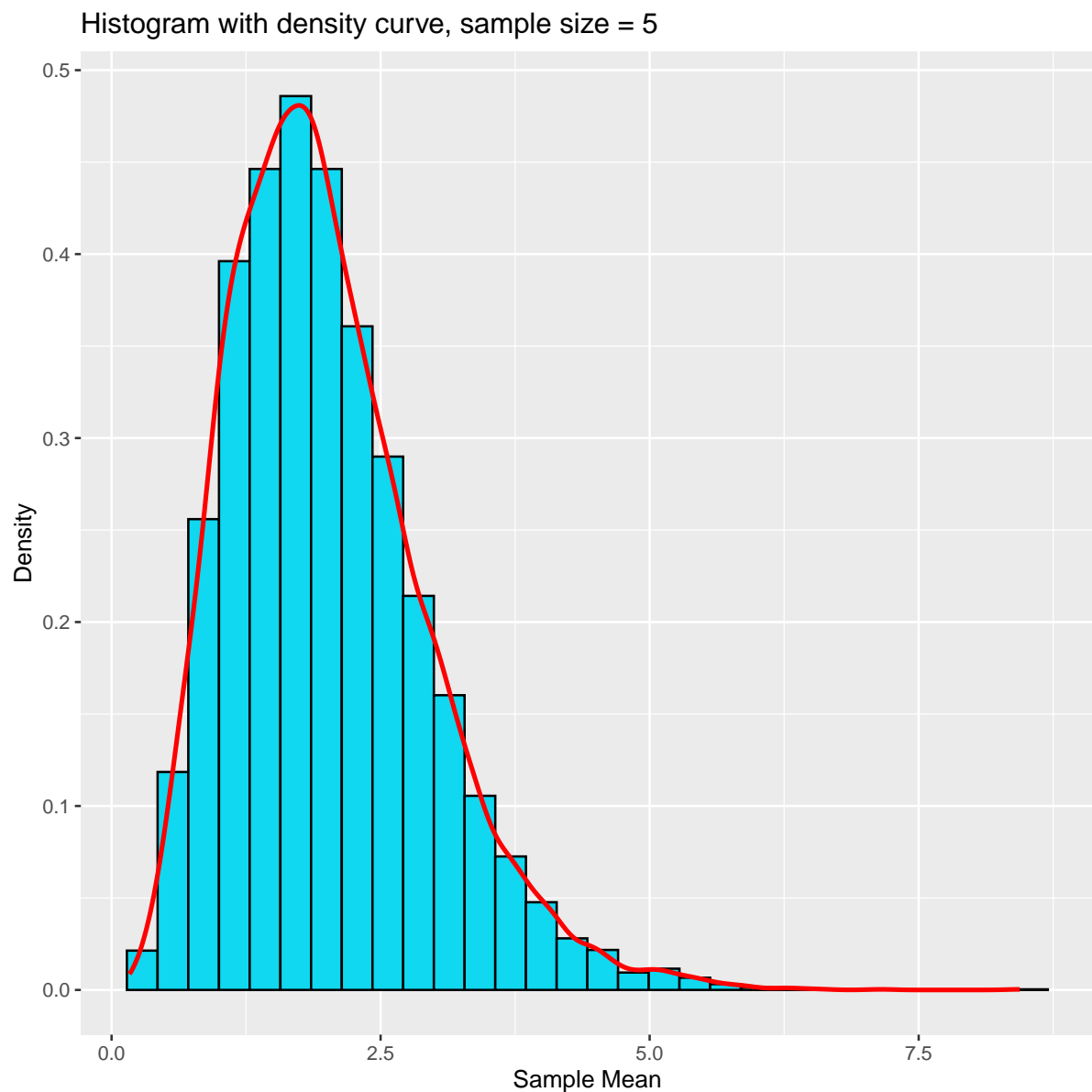
```
sample_list <- list(); size <- 5

for(i in 1:10000){
  sample_list[[i]] <- rexp(size, rate = 1 / theta)
}

x_bar <- sapply(sample_list, mean)

df2 <- data.frame(means = x_bar)

df2 %>%
  ggplot(aes(x = means)) +
  geom_histogram(aes(y = ..density..), bins = 30, fill = "#0FD8F0", col = "black") +
  geom_density(col = "red", linewidth = 1) +
  labs(x = "Sample Mean", y = "Density",
       title = "Histogram with density curve, sample size = 5")
```



Next we take  $n = 10$ .

```
sample_list <- list(); size <- 10

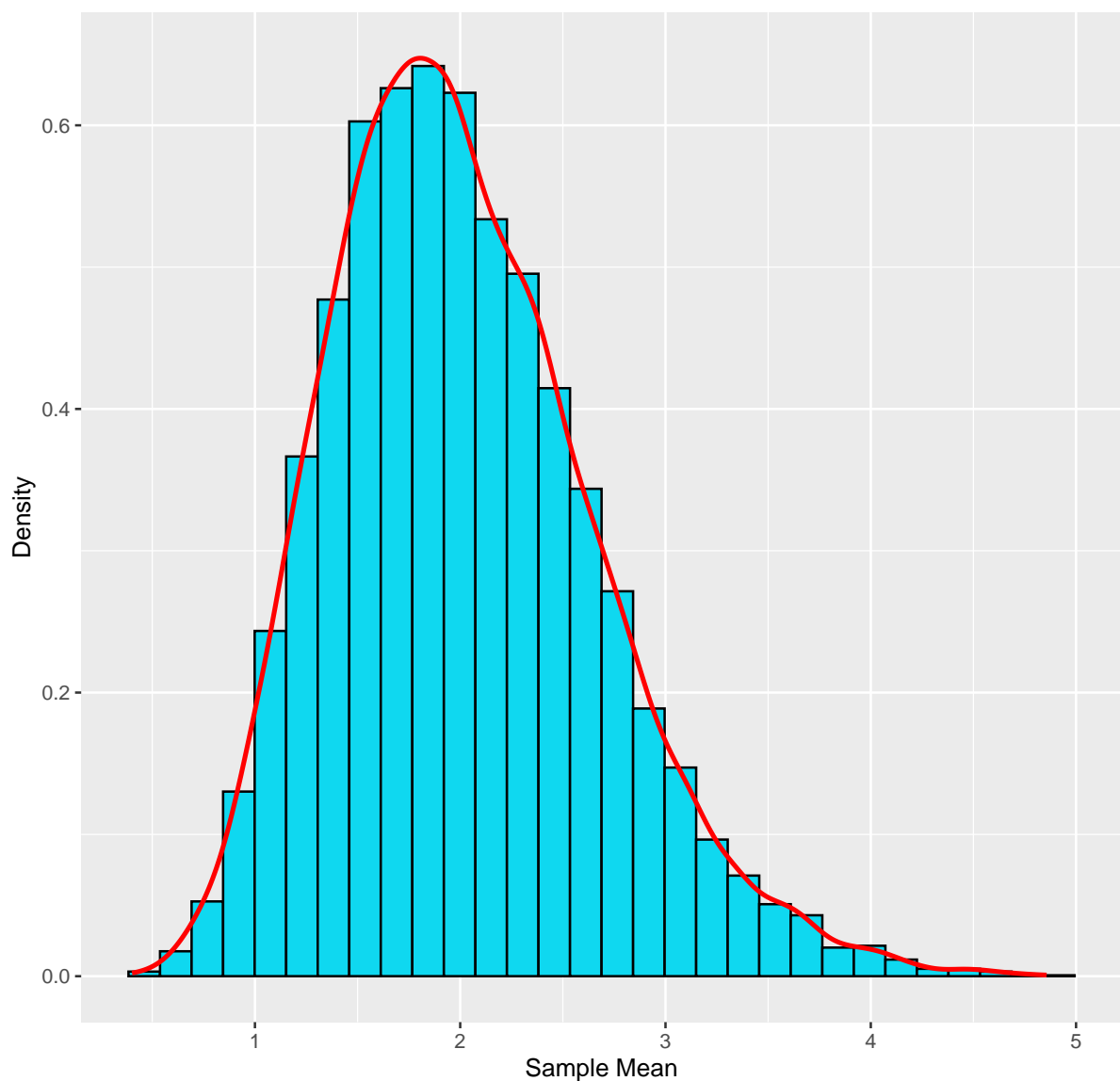
for(i in 1:10000){
  sample_list[[i]] <- rexp(size, rate = 1 / theta)
}

x_bar <- sapply(sample_list, mean)
```

```
df2 <- data.frame(means = x_bar)

df2 %>%
  ggplot(aes(x = means)) +
  geom_histogram(aes(y = ..density..), bins = 30, fill = "#0FD8F0", col = "black") +
  geom_density(col = "red", linewidth = 1) +
  labs(x = "Sample Mean", y = "Density",
       title = "Histogram with density curve, sample size = 10")
```

Histogram with density curve, sample size = 10



Finally we take  $n = 20$ .

```
sample_list <- list(); size <- 20

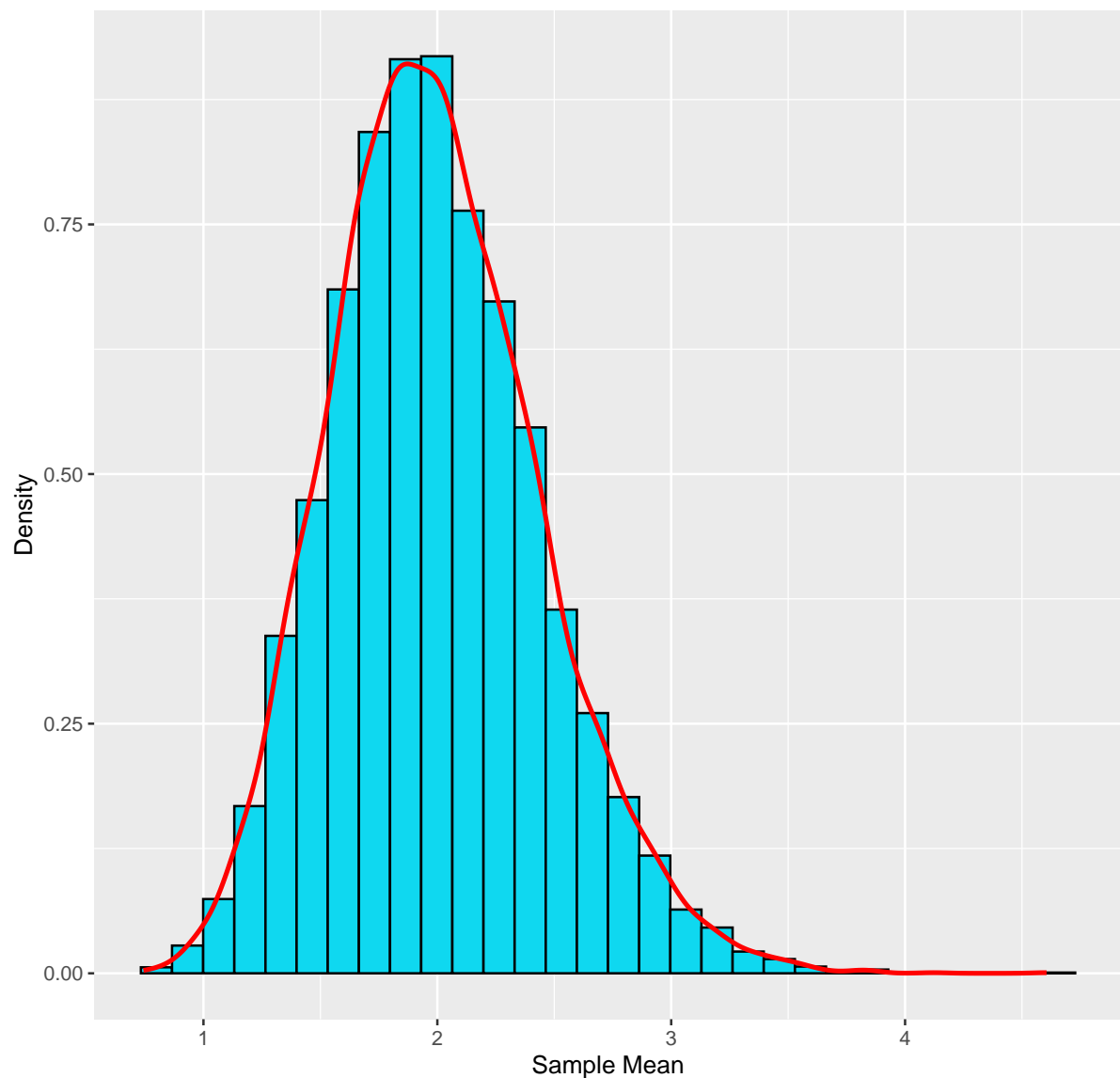
for(i in 1:10000){
  sample_list[[i]] <- rexp(size, rate = 1 / theta)
}


x_bar <- sapply(sample_list, mean)

df2 <- data.frame(means = x_bar)


df2 %>%
  ggplot(aes(x = means)) +
  geom_histogram(aes(y = ..density..), bins = 30, fill = "#0FD8F0", col = "black") +
  geom_density(col = "red", linewidth = 1) +
  labs(x = "Sample Mean", y = "Density",
       title = "Histogram with density curve, sample size = 20")
```

Histogram with density curve, sample size = 20



 The density curve has pretty much resemblance with that of a normal density curve, thus implying convergence in distribution of the sample mean.

Hence it is established that the sample mean is a CAN estimator for  $\theta$ .

 Now we have to obtain a CAN estimator for  $\psi(\theta) = e^{-t/\theta}$ .  $\psi(\theta)$  is differentiable function of  $\theta$ ,  $\frac{d\psi}{d\theta}$  is non-vanishing and continuous. We already have  $\bar{X}$  is a CAN estimator for  $\theta$ . Thus by invariance property of a CAN estimator,  $\psi(\bar{X})$  is a CAN estimator of  $\psi(\theta)$  and

$$\psi(\bar{X}) \sim AN \left( \psi(\theta), \frac{\theta^2}{n} \left( \frac{d\psi}{d\theta} \right)^2 \right).$$

So asymptotic variance of  $\psi(\bar{X})$  is  $\frac{t^2}{n\theta^2} e^{-2t/\theta}$ .


Here we take  $t = 1$  and  $\theta = 2$ .

So  $\psi(\theta) = 0.6065307$ .

```
estimate <- function(n) return(exp(-1/mean(rexp(n, rate = 1/2))))
asv <- function(n) return(exp(-1) / (n * 4))
```

```
df3 <- data.frame(sample_size = seq(5, 50, 5),
                  estimated_psi_theta = sapply(seq(5, 50, 5), FUN = estimate),
                  variance = asv(seq(5, 50, 5)))
df3
```

##	sample_size	estimated_psi_theta	variance
## 1	5	0.3734812	0.018393972
## 2	10	0.5506846	0.009196986
## 3	15	0.5726035	0.006131324
## 4	20	0.6453451	0.004598493
## 5	25	0.6252023	0.003678794
## 6	30	0.6857392	0.003065662
## 7	35	0.4983047	0.002627710
## 8	40	0.5112385	0.002299247
## 9	45	0.5741369	0.002043775
## 10	50	0.5627382	0.001839397

 Thus we get estimates of the parametric function for different sample sizes. The variance of the estimate decreases as sample size increases.

# MSMS 206 : Practical 04

Ananda Biswas

May 8, 2025

## ⊕ Question

Consider a lifetime variable that follows a **Weibull distribution** with shape parameter  $\alpha$  and scale parameter  $\lambda$ . The probability density function of this distribution is given by:

$$f(t; \alpha, \lambda) = \frac{\alpha}{\lambda} \left( \frac{t}{\lambda} \right)^{\alpha-1} e^{-(t/\lambda)^\alpha}, \quad t > 0.$$

The objective is to evaluate the performance of **maximum likelihood estimation (MLE)** for different sample sizes. First, generate random samples of sizes  $n = 60, 80, 100, 120$  and  $140$  from this Weibull distribution. For each sample, estimate the parameters  $\alpha$  and  $\lambda$  using MLE.

Next, compute the **standard errors** of the ML estimates and evaluate their accuracy by estimating the **bias** and the **mean squared error (MSE)**.

## ⊕ Build-up for obtaining MLE

For a sample of size  $n$ , the likelihood function is given by

$$\begin{aligned} L(\alpha, \lambda) &= \prod_{i=1}^n \frac{\alpha}{\lambda} \left( \frac{x_i}{\lambda} \right)^{\alpha-1} e^{-(x_i/\lambda)^\alpha} \\ &= \left( \frac{\alpha}{\lambda} \right)^n \left( \prod_{i=1}^n \frac{x_i}{\lambda} \right)^{\alpha-1} \exp \left\{ - \sum_{i=1}^n \left( \frac{x_i}{\lambda} \right)^\alpha \right\}. \end{aligned}$$

The log-likelihood function is given by

$$l(\alpha, \lambda) = n \log \alpha - n \log \lambda + (\alpha - 1) \sum_{i=1}^n \log \left( \frac{x_i}{\lambda} \right) - \sum_{i=1}^n \left( \frac{x_i}{\lambda} \right)^\alpha.$$

Now,

$$\frac{\partial}{\partial \alpha} l(\alpha, \lambda) = \frac{n}{\alpha} + \sum_{i=1}^n \log \left( \frac{x_i}{\lambda} \right) - \sum_{i=1}^n \left( \frac{x_i}{\lambda} \right)^\alpha \log \left( \frac{x_i}{\lambda} \right) = u(\alpha, \lambda), \text{ say} \quad (1)$$



and

$$\begin{aligned}
\frac{\partial}{\partial \lambda} l(\alpha, \lambda) &= -\frac{n}{\lambda} + (\alpha - 1) \sum_{i=1}^n \frac{\lambda}{x_i} \cdot \left(-\frac{x_i}{\lambda^2}\right) + \sum_{i=1}^n \frac{\alpha \cdot x_i^\alpha}{\lambda^{\alpha+1}} \\
&= -\frac{n}{\lambda} - (\alpha - 1) \frac{n}{\lambda} + \sum_{i=1}^n \frac{\alpha \cdot x_i^\alpha}{\lambda^{\alpha+1}} \\
&= -\frac{n\alpha}{\lambda} + \sum_{i=1}^n \frac{\alpha \cdot x_i^\alpha}{\lambda^{\alpha+1}} = v(\alpha, \lambda), \text{ say.}
\end{aligned}$$

Setting  $v(\alpha, \lambda) = 0$  we get,

$$\begin{aligned}
-\frac{n\alpha}{\lambda} + \sum_{i=1}^n \frac{\alpha \cdot x_i^\alpha}{\lambda^{\alpha+1}} &= 0 \\
\Rightarrow \frac{n}{\lambda} &= \sum_{i=1}^n \frac{x_i^\alpha}{\lambda^{\alpha+1}} \\
\Rightarrow \frac{n}{\lambda} &= \frac{1}{\lambda^{\alpha+1}} \sum_{i=1}^n x_i^\alpha \\
\Rightarrow \lambda^\alpha &= \frac{1}{n} \sum_{i=1}^n x_i^\alpha \\
\therefore \lambda &= \left( \frac{1}{n} \sum_{i=1}^n x_i^\alpha \right)^{\frac{1}{\alpha}} \tag{2}
\end{aligned}$$

Setting  $u(\alpha, \lambda) = 0$  does not yield any closed form solution. So for getting the ML estimate of  $\alpha$ , we resort to numerical methods (here Newton-Raphson method).

Now,

$$u_\alpha(\alpha, \lambda) = \frac{\partial}{\partial \alpha} u(\alpha, \lambda) = -\frac{n}{\alpha^2} - \sum_{i=1}^n \left(\frac{x_i}{\lambda}\right)^\alpha \left[ \log\left(\frac{x_i}{\lambda}\right) \right]^2; \tag{3}$$

At each iteration, with the present value of  $\alpha$  we calculate  $\lambda$  by using (2); then we use the obtained value of  $\lambda$  in (1) and (3) to improve the estimate of  $\alpha$  by Newton-Raphson method.

## ➡ R Program

```
estimate_lambda <- function(s, alpha) mean(s^alpha)^(1/alpha)
```

```
u <- function(alpha, lambda, s){
```

```
  a <- length(s) / alpha
```

```

b <- sum(log(s / lambda))

c <- sum((s / lambda)^alpha * log(s / lambda))

return(a + b - c)
}

```

```

u_alpha <- function(alpha, lambda, s){

  a <- - length(s) / alpha^2

  b <- sum((s / lambda)^alpha * log(s / lambda)^2)

  return(a - b)
}

```

```

estimate_alpha <- function(s, initial, epsilon = 0.0001, iterations = 100){

  alphas <- c(initial)

  for (i in 2:iterations) {
    l <- estimate_lambda(s, alphas[i-1])

    alphas[i] <- alphas[i-1] - u(alphas[i-1], l, s) / u_alpha(alphas[i-1], l, s)

    if(abs((alphas[i] - alphas[i-1])) < epsilon) break
  }

  return(alphas[length(alphas)])
}

```

```

true_alpha <- 3; true_lambda <- 2

```

```

sample_sizes <- c(60, 80, 100, 120, 140)

```

```

alpha_hat = lambda_hat = c()

```

```

for (n in sample_sizes) {

  x <- rweibull(n, shape = true_alpha, scale = true_lambda)


  a <- estimate_alpha(x, 1)

  alpha_hat <- append(alpha_hat, a)

  lambda_hat <- append(lambda_hat, estimate_lambda(x, a))
}

```

```
df1 <- data.frame(Sample_Size = sample_sizes,
                  alpha_hat = alpha_hat,
                  lambda_hat = lambda_hat)
```

 Estimates of the parameters for different sample sizes are as follows :


```
df1
##   Sample_Size alpha_hat lambda_hat
## 1          60  2.859356   2.068752
## 2          80  3.227570   2.024756
## 3         100  3.100148   2.085612
## 4         120  3.113117   2.135652
## 5         140  3.123736   2.045556
```

Now we shall evaluate the accuracy of the estimates.

```
alpha_bias = lambda_bias = alpha_SE = lambda_SE = alpha_MSE = lambda_MSE = c()
```


```
for (k in 1:length(sample_sizes)) {
  alpha_estimates = lambda_estimates = c()
  for (i in 1:100){
    x <- rweibull(sample_sizes[k], shape = true_alpha, scale = true_lambda)
    alpha_estimates[i] <- estimate_alpha(x, 1)
    lambda_estimates[i] <- estimate_lambda(x, alpha_estimates[i])
  }
  alpha_bias[k] <- mean(alpha_estimates) - true_alpha
  lambda_bias[k] <- mean(lambda_estimates) - true_lambda
  alpha_SE[k] <- sd(alpha_estimates)
  lambda_SE[k] <- sd(lambda_estimates)
  alpha_MSE[k] <- mean( (alpha_estimates - true_alpha)^2 )
  lambda_MSE[k] <- mean( (lambda_estimates - true_lambda)^2 )
}
```

```
df2 <- data.frame(sample_sizes,
                  alpha_bias, alpha_SE, alpha_MSE,
                  lambda_bias, lambda_SE, lambda_MSE)
```

 Bias, Standard error and MSE of the estimates for different sample sizes are as follows :

```
df2
##   sample_sizes alpha_bias  alpha_SE  alpha_MSE  lambda_bias  lambda_SE
## 1           60 0.08410573 0.2974252 0.09465089 -0.004768764 0.08186733
## 2           80 0.04029086 0.2849094 0.08198498 -0.004317026 0.08148510
## 3          100 0.07690680 0.2463622 0.06600206  0.005209200 0.08189805
## 4          120 0.01963392 0.2220456 0.04919668 -0.009140331 0.06363508
## 5          140 0.03633982 0.1684986 0.02942845  0.009296586 0.05662430
##      lambda_MSE
## 1 0.006657978
## 2 0.006592059
## 3 0.006667354
## 4 0.004092475
## 5 0.003260675
```

## ➡ Conclusion

 Accuracy of the estimates increase as sample size increases.

## MSMS 206 : Practical 06 - Supplementary Derivations

Ananda Biswas

May 9, 2025

⊕

$$f(t; \alpha, \lambda) = \frac{\alpha}{\lambda} \left( \frac{t}{\lambda} \right)^{\alpha-1} e^{-(t/\lambda)^\alpha}, \quad t > 0.$$

For a sample of size  $n$ , the likelihood function is given by

$$\begin{aligned} L(\alpha, \lambda) &= \prod_{i=1}^n \frac{\alpha}{\lambda} \left( \frac{x_i}{\lambda} \right)^{\alpha-1} e^{-(x_i/\lambda)^\alpha} \\ &= \left( \frac{\alpha}{\lambda} \right)^n \left( \prod_{i=1}^n \frac{x_i}{\lambda} \right)^{\alpha-1} \exp \left\{ - \sum_{i=1}^n \left( \frac{x_i}{\lambda} \right)^\alpha \right\}. \end{aligned}$$

The log-likelihood function is given by

$$l(\alpha, \lambda) = n \log \alpha - n \log \lambda + (\alpha - 1) \sum_{i=1}^n \log \left( \frac{x_i}{\lambda} \right) - \sum_{i=1}^n \left( \frac{x_i}{\lambda} \right)^\alpha.$$

Now,

$$\frac{\partial}{\partial \alpha} l(\alpha, \lambda) = \frac{n}{\alpha} + \sum_{i=1}^n \log \left( \frac{x_i}{\lambda} \right) - \sum_{i=1}^n \left( \frac{x_i}{\lambda} \right)^\alpha \log \left( \frac{x_i}{\lambda} \right) = u(\alpha, \lambda), \text{ say}$$

and

$$\begin{aligned} \frac{\partial}{\partial \lambda} l(\alpha, \lambda) &= -\frac{n}{\lambda} + (\alpha - 1) \sum_{i=1}^n \frac{\lambda}{x_i} \cdot \left( -\frac{x_i}{\lambda^2} \right) + \sum_{i=1}^n \frac{\alpha \cdot x_i^\alpha}{\lambda^{\alpha+1}} \\ &= -\frac{n}{\lambda} - (\alpha - 1) \frac{n}{\lambda} + \sum_{i=1}^n \frac{\alpha \cdot x_i^\alpha}{\lambda^{\alpha+1}} \\ &= -\frac{n\alpha}{\lambda} + \sum_{i=1}^n \frac{\alpha \cdot x_i^\alpha}{\lambda^{\alpha+1}} = v(\alpha, \lambda), \text{ say.} \end{aligned}$$

Now,

$$\begin{aligned}
u_\alpha(\alpha, \lambda) &= \frac{\partial}{\partial \alpha} u(\alpha, \lambda) = -\frac{n}{\alpha^2} - \sum_{i=1}^n \left(\frac{x_i}{\lambda}\right)^\alpha \left[\log\left(\frac{x_i}{\lambda}\right)\right]^2; \\
u_\lambda(\alpha, \lambda) &= \frac{\partial}{\partial \lambda} u(\alpha, \lambda) = \sum_{i=1}^n \frac{\lambda}{x_i} \cdot \left(-\frac{x_i}{\lambda^2}\right) - \sum_{i=1}^n \left[ \frac{(-\alpha)x_i^\alpha}{\lambda^{\alpha+1}} \log\left(\frac{x_i}{\lambda}\right) + \left(\frac{x_i}{\lambda}\right)^\alpha \frac{\lambda}{x_i} \cdot \left(-\frac{x_i}{\lambda^2}\right) \right] \\
&= -\frac{n}{\lambda} + \sum_{i=1}^n \left[ \frac{x_i^\alpha}{\lambda^{\alpha+1}} \left(\alpha \log\left(\frac{x_i}{\lambda}\right) + 1\right) \right].
\end{aligned}$$

Also,

$$\begin{aligned}
v_\alpha(\alpha, \lambda) &= \frac{\partial}{\partial \alpha} v(\alpha, \lambda) = -\frac{n}{\lambda} + \sum_{i=1}^n \left[ \frac{x_i^\alpha}{\lambda^{\alpha+1}} + \frac{\alpha}{\lambda} \left(\frac{x_i}{\lambda}\right)^\alpha \log\left(\frac{x_i}{\lambda}\right) \right] \\
&= -\frac{n}{\lambda} + \sum_{i=1}^n \left[ \frac{x_i^\alpha}{\lambda^{\alpha+1}} \left(\alpha \log\left(\frac{x_i}{\lambda}\right) + 1\right) \right]; \\
v_\lambda(\alpha, \lambda) &= \frac{\partial}{\partial \lambda} v(\alpha, \lambda) = \frac{n\alpha}{\lambda^2} - \sum_{i=1}^n \frac{\alpha(\alpha+1)x_i^\alpha}{\lambda^{\alpha+2}}.
\end{aligned}$$

# MSMS 206 : Practical 01

Ananda Biswas

March 7, 2025



## Question

Fit a multiple linear regression model for the following data-set and obtain the following results.

- (a) estimate of the regression coefficients and  $\sigma$ ,
- (b) confidence interval of the regression coefficients,
- (c) coefficient of determination,
- (d) adjusted coefficient of determination.

$y$	$x_1$	$x_2$
16.68	7	560
11.5	3	220
12.03	3	340
14.88	4	80
13.75	6	150
18.11	7	330
8	2	110
17.83	7	210
79.2	30	1460
21	10	215
13.5	4	255
19.75	6	462
24	9	448
29	10	776
15.35	6	220
19	7	132
9.5	3	36
35.1	17	770
17.9	10	140
52.32	26	810
18.75	9	450
19.83	8	635
10.75	4	150
21.5	5	605
40.33	16	688

④ We fit a multiple linear regression model  $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2$ .

```
raw_data <- read.csv('https://raw.githubusercontent.com/sakunisgithub/data_sets/refs/heads/master/msc_semester_2/tripti_madam_practical_01_data.csv')
```

```
dim(raw_data)
```

```
## [1] 25 3
```

```
names(raw_data)
```


```
## [1] "Y" "X_1" "X_2"
```

```
model1 <- lm(Y ~ X_1 + X_2, data = raw_data)
```

```
model_summary <- summary(model1)
```

```
model_summary
```

```
##
## Call:
## lm(formula = Y ~ X_1 + X_2, data = raw_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -5.776 -0.659  0.164  1.173  7.387
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.326534   1.096063   2.123 0.045279 *
## X_1          1.614726   0.170574   9.466 3.24e-09 ***
## X_2          0.014414   0.003615   3.987 0.000623 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.254 on 22 degrees of freedom
## Multiple R-squared:  0.9597, Adjusted R-squared:  0.956
## F-statistic: 261.9 on 2 and 22 DF, p-value: 4.561e-16
```


 Estimates of regression coefficients are

```
estimates <- model_summary$coefficients[, 'Estimate']
```

```
estimates
```

```
## (Intercept)          X_1          X_2
##  2.32653410  1.61472628  0.01441393
```



 Estimate of error standard deviation  $\sigma$  is

```
sigma_hat <- model_summary$sigma
sigma_hat

## [1] 3.254133
```

Now we shall calculate the confidence intervals of the regression coefficients and the intercept.  $100(1 - \alpha)\%$  confidence interval for  $\beta_j \forall j = 0(1)2$  is given by


$$\left( \hat{\beta}_j - t_{\frac{\alpha}{2}, n-p} \cdot \text{se}(\hat{\beta}_j), \hat{\beta}_j + t_{\frac{\alpha}{2}, n-p} \cdot \text{se}(\hat{\beta}_j) \right).$$

where  $n$  is total number of observations and  $p$  is total number of parameters in the model.

```
std_errors <- model_summary$coefficients[, 'Std. Error']


t_tabulated <- qt(0.025, 22, lower.tail = FALSE)

CI_lower <- estimates - t_tabulated * std_errors
CI_upper <- estimates + t_tabulated * std_errors
```

 95% confidence interval for  $\beta_0$  is


```
CI_lower[1]; CI_upper[1]

## (Intercept)
## 0.0534379
## (Intercept)
## 4.59963
```

 95% confidence interval for  $\beta_1$  is

```
CI_lower[2]; CI_upper[2]

## X_1
## 1.260978
## X_1
## 1.968475
```

 95% confidence interval for  $\beta_2$  is


```
CI_lower[3]; CI_upper[3]

## X_2
## 0.006916106
## X_2
## 0.02191175
```

Now we shall calculate the confidence interval of the error variance  $\sigma^2$ .  $100(1 - \alpha)\%$  confidence interval for  $\sigma^2$  is given by

$$\left( \frac{\text{RSS}}{\chi_{\frac{\alpha}{2}, n-p}^2}, \frac{\text{RSS}}{\chi_{1-\frac{\alpha}{2}, n-p}^2} \right).$$

```
RSS <- sum(model_summary$residuals^2) # Residual Sum of Squares
```


 95% confidence interval for  $\sigma^2$  is

```
RSS / qchisq(0.025, 22, lower.tail = FALSE) # lower bound
```

```
## [1] 6.333929
```

```
RSS / qchisq(1-0.025, 22, lower.tail = FALSE) # upper bound
```

```
## [1] 21.21286
```


 95% confidence interval for  $\sigma$  is

```
sqrt(RSS / qchisq(0.025, 22, lower.tail = FALSE)) # lower bound
```

```
## [1] 2.51673
```


```
sqrt(RSS / qchisq(1-0.025, 22, lower.tail = FALSE)) # upper bound
```

```
## [1] 4.605742
```

  $R^2$  for the model is

```
model_summary$r.squared
```

```
## [1] 0.9596944
```

 Adjusted  $R^2$  for the model is

```
model_summary$adj.r.squared
```

```
## [1] 0.9560302
```

## MSMS 206 : Practical 02

Ananda Biswas

April 11, 2025



**Question :** Solve the following linear programming problem.

$$\begin{aligned} \text{Minimize } z &= 2x_1 + x_2 + 4x_3 \\ \text{subject to } & -2x_1 + 4x_2 \leq 4, \\ & x_1 + 2x_2 + x_3 \geq 5, \\ & 2x_1 + x_3 \leq -2, \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

```
library(lpSolve)
```

```
coeff <- c(2, 1, 4)

constraint_mat <- matrix(data = c(-2, 4, 0,
                                   1, 2, 1,
                                   2, 0, 1),
                        nrow = 3, ncol = 3, byrow = TRUE)

constraint_direction <- c("<=", ">=", "<=")

constraint_RHS <- c(4, 5, -2)
```

```
sol <- lp(direction = "min",
          objective.in = coeff,
          const.mat = constraint_mat,
          const.dir = constraint_direction,
          const.rhs = constraint_RHS)
```

```
sol

## Error: no feasible solution found
```

```
sol$status

## [1] 2
```



Solution status = 2 implies there is no feasible solution.



**Question :** Solve the following linear programming problem.

$$\begin{aligned}
 &\text{Maximize } z = 3x_1 + 5x_2 + 4x_3 \\
 &\text{subject to } \begin{aligned}
 &2x_1 + 3x_2 \leq 8, \\
 &2x_2 + 5x_3 \leq 10, \\
 &3x_1 + 2x_2 + 4x_3 \leq 15, \\
 &x_1, x_2, x_3 \geq 0.
 \end{aligned}
 \end{aligned}$$

```

coeff <- c(3, 5, 4)

constraint_mat <- matrix(data = c(2, 3, 0,
                                0, 2, 5,
                                3, 2, 4),
                        nrow = 3, ncol = 3, byrow = TRUE)

constraint_direction <- c("<=", "<=", "<=")

constraint_RHS <- c(8, 10, 15)

```

```

sol <- lp(direction = "max",
  objective.in = coeff,
  const.mat = constraint_mat,
  const.dir = constraint_direction,
  const.rhs = constraint_RHS)

```

```

sol

## Success: the objective function is 18.65854

```

```

sol$status

## [1] 0

```



The optimal value of the objective function is

```

sol$objval

## [1] 18.65854

```



The optimal solution of the linear programming problem is

```

sol$solution

## [1] 2.170732 1.219512 1.512195

```

## MSMS 206 : Practical 03

Ananda Biswas

March 30, 2025



### Question

- (1) Calculate  $L$  and  $U$  such that  $A = LU$  where  $L$  is a lower triangular matrix and  $U$  is an upper triangular matrix for given

$$A = \begin{bmatrix} 1 & 1 & -1 \\ 1 & -2 & 3 \\ 2 & 3 & 1 \end{bmatrix}.$$

- (2) Solve the following system of linear equations using  $LU$  decomposition method.

$$\begin{aligned} x_1 + x_2 - x_3 &= 4 \\ x_1 - 2x_2 + 3x_3 &= -6 \\ 2x_1 + 3x_2 + x_3 &= 7 \end{aligned}$$

⊕ Standard  $LU$  decomposition is possible only possible for square matrices with all leading principal minors being non-zero.

```
check_LPM <- function(A){  
  
  if(dim(A)[1] != dim(A)[2]) stop("Input must be a square matrix.")  
  
  # leading_principal_minors  
  m <- c()  
  
  for (i in 1:dim(A)[1]) {  
    m[i] <- det(as.matrix(A[1:i, 1:i]))  
  }  
  
  if(all((m != 0) == TRUE)){  
    return(TRUE)  
  }else{  
    return(FALSE)  
  }  
}
```

```

LU_decomposer <- function(A){

  if(!check_LPM(A)) stop("All the leading principal minors must be non-zero.")

  I <- matrix(data = 0, nrow = nrow(A), ncol = ncol(A))

  for (i in 1:nrow(I)) {
    I[i, i] <- I[i, i] + 1
  }

  r <- dim(A)[1]
  c <- dim(A)[2]

  i <- 1; j <- 1

  while(j <= c) {

    while(i <= r) {

      if(i != r){
        a1 <- as.matrix(A[(i+1):r, j] / A[i, j])

        a2 <- t(as.matrix(A[i, ]))

        A[(i+1):r, j] <- A[(i+1):r, j] - a1 %*% a2

        I[(i+1):r, j] <- as.vector(a1)

        break
      }
      i <- i + 1
    }
    j <- j + 1

    i <- j
  }

  return(list(I, A))
}

```

*LU\_decomposer()* returns a list containing  $L$  and  $U$  respectively.

```

A <- matrix(data = c(1, 1, -1,
                    1, -2, 3,
                    2, 3, 1),
            nrow = 3, ncol = 3, byrow = TRUE)

```

```
LU_decomposer(A)
```

```
## [[1]]
```

```
##      [,1]      [,2] [,3]
## [1,]    1  0.0000000    0
## [2,]    1  1.0000000    0
## [3,]    2 -0.3333333    1
##
## [[2]]
##      [,1] [,2]      [,3]
## [1,]    1    1 -1.000000
## [2,]    0   -3  4.000000
## [3,]    0    0  4.333333
```

```
L <- LU_decomposer(A)[[1]]
U <- LU_decomposer(A)[[2]]
```

⊕

```
b <- matrix(data = c(4, -6, 7), nrow = 3, ncol = 1, byrow = TRUE)
```

•

$$\begin{aligned} Ax &= b \\ \Rightarrow LUx &= b \\ \Rightarrow x &= U^{-1}L^{-1}b \end{aligned}$$

```
solve(U) %*% solve(L) %*% b
```

```
##      [,1]
## [1,]    1
## [2,]    2
## [3,]   -1
```



The solution of the given system of equations is

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix}.$$



## More LU Decompositions :

☞ Schaums Outline of Linear Algebra : Example 3.22

$$A = \begin{bmatrix} 1 & 2 & -3 \\ -3 & -4 & 13 \\ 2 & 1 & -5 \end{bmatrix}.$$

**LU\_decomposer(A)**

```
## [[1]]
##      [,1] [,2] [,3]
## [1,]    1  0.0    0
## [2,]   -3  1.0    0
## [3,]    2 -1.5    1
##
## [[2]]
##      [,1] [,2] [,3]
## [1,]    1    2   -3
## [2,]    0    2    4
## [3,]    0    0    7
```

☞ Schaums Outline of Linear Algebra : Solved Problems 3.39(a)

$$A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & -4 & 7 \\ -1 & -2 & 1 \end{bmatrix}.$$

**LU\_decomposer(A)**

```
## [[1]]
##      [,1] [,2] [,3]
## [1,]    1  0.0    0
## [2,]    2  1.0    0
## [3,]   -1 -2.5    1
##
## [[2]]
##      [,1] [,2] [,3]
## [1,]    1   -3  5.0
## [2,]    0    2 -3.0
## [3,]    0    0 -1.5
```



☞ Schaums Outline of Linear Algebra : Solved Problems 3.39(b)

$$A = \begin{bmatrix} 1 & 4 & -3 \\ 2 & 8 & 1 \\ -5 & -9 & 7 \end{bmatrix}.$$

```
LU_decomposer(A)
```

```
## Error in LU_decomposer(A): All the leading principal minors must be non-zero.
```

☞ Schaums Outline of Linear Algebra : Solved Problems 3.41

$$A = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & 3 \\ -3 & -10 & 2 \end{bmatrix}.$$

```
LU_decomposer(A)
```

```
## [[1]]
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    2    1    0
## [3,]   -3    4    1
##
## [[2]]
##      [,1] [,2] [,3]
## [1,]    1    2    1
## [2,]    0   -1    1
## [3,]    0    0    1
```

☞ Schaums Outline of Linear Algebra : Supplementary Problems 3.69(a)

$$A = \begin{bmatrix} 1 & -1 & -1 \\ 3 & -4 & -2 \\ 2 & -3 & -2 \end{bmatrix}.$$

```
LU_decomposer(A)
```

```
## [[1]]
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    3    1    0
## [3,]    2    1    1
##
## [[2]]
##      [,1] [,2] [,3]
## [1,]    1   -1   -1
## [2,]    0   -1    1
## [3,]    0    0   -1
```

☞ Schaums Outline of Linear Algebra : Supplementary Problems 3.69(b)

$$A = \begin{bmatrix} 1 & 3 & -1 \\ 2 & 5 & 1 \\ 3 & 4 & 2 \end{bmatrix}.$$

```
LU_decomposer(A)
```

```
## [[1]]
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    2    1    0
## [3,]    3    5    1
##
## [[2]]
##      [,1] [,2] [,3]
## [1,]    1    3   -1
## [2,]    0   -1    3
## [3,]    0    0  -10
```

☞ Schaums Outline of Linear Algebra : Supplementary Problems 3.69(c)

$$A = \begin{bmatrix} 2 & 3 & 6 \\ 4 & 7 & 9 \\ 3 & 5 & 4 \end{bmatrix}.$$

```
LU_decomposer(A)
```

```
## [[1]]
##      [,1] [,2] [,3]
## [1,]  1.0  0.0   0
## [2,]  2.0  1.0   0
## [3,]  1.5  0.5   1
##
## [[2]]
##      [,1] [,2] [,3]
## [1,]    2    3  6.0
## [2,]    0    1 -3.0
## [3,]    0    0 -3.5
```

☞ Schaums Outline of Linear Algebra : Supplementary Problems 3.69(d)

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 7 \\ 3 & 7 & 10 \end{bmatrix}.$$

```
LU_decomposer(A)
```

```
## Error in LU_decomposer(A): All the leading principal minors must be non-zero.
```

# MSMS 206 : Practical 04

Ananda Biswas

April 10, 2025



**Question :** Consider the “Swiss” dataset in MASS package of R. Perform the following clustering algorithms to divide the data-set into clusters.

- (a)  $k$ –means clustering algorithm to divide the data-set into 3 clusters;
- (b) Agglomerative Hierarchical Clustering.

⊕  $k$ –means clustering algorithm to divide the data-set into 3 clusters

```
library(MASS)

dim(swiss)

## [1] 47 6

head(swiss)

##           Fertility Agriculture Examination Education Catholic
## Courtelary      80.2         17.0           15         12      9.96
## Delemont        83.1         45.1            6          9     84.84
## Franches-Mnt    92.5         39.7            5          5     93.40
## Moutier         85.8         36.5           12          7     33.77
## Neuveville     76.9         43.5           17         15      5.16
## Porrentruy     76.1         35.3            9          7     90.57
##           Infant.Mortality
## Courtelary           22.2
## Delemont             22.2
## Franches-Mnt         20.2
## Moutier              20.3
## Neuveville           20.6
## Porrentruy           26.6

kmeans(swiss, 3)

## K-means clustering with 3 clusters of sizes 11, 20, 16
##
## Cluster means:
##   Fertility Agriculture Examination Education Catholic Infant.Mortality
## 1  58.30909   19.50909   25.72727   23.000 22.21455      19.22727
## 2  68.32500   55.90500   17.05000    7.850  7.55000      19.67000
## 3  80.55000   65.51875    9.43750    6.625 96.15000      20.77500
```

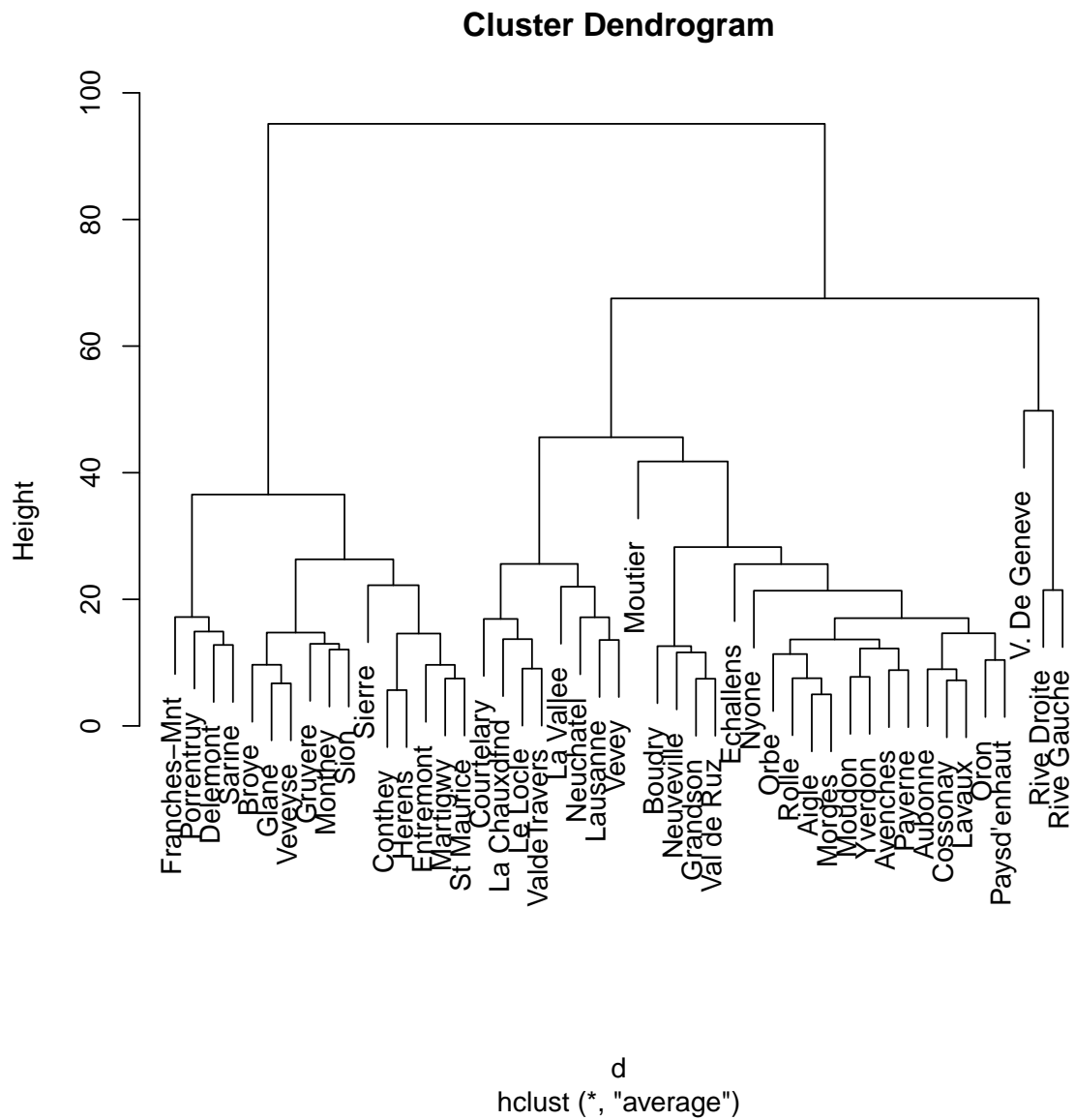
```
##
## Clustering vector:
##   Courtelary      Delemont Franches-Mnt      Moutier      Neuveville      Porrentruy
##           1           3           3           2           2           3
##       Broye       Glane       Gruyere       Sarine       Veveyse       Aigle
##           3           3           3           3           3           2
##       Aubonne     Avenches     Cossonay     Echallens     Grandson     Lausanne
##           2           2           2           2           2           1
##   La Vallee      Lavaux       Morges       Moudon       Nyone       Orbe
##           1           2           2           2           2           2
##       Oron       Payerne Paysd'enhaut       Rolle       Vevey       Yverdon
##           2           2           2           2           1           2
##       Conthey    Entremont       Herens     Martigwy     Monthey     St Maurice
##           3           3           3           3           3           3
##       Sierre     Sion          Boudry La Chauxdfnd     Le Locle     Neuchatel
##           3           3           2           1           1           1
##   Val de Ruz ValdeTravers V. De Geneve Rive Droite Rive Gauche
##           2           1           1           1           1
##
## Within cluster sum of squares by cluster:
## [1] 9116.894 5966.297 6532.906
## (between_SS / total_SS =  81.8 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

## ⊕ Agglomerative Hierarchical Clustering

```
d <- dist(swiss)

x <- hclust(d, method = "average")

plot(x)
```



# MSMS 206 : Practical 05

Ananda Biswas

April 19, 2025



**Question :** Perform  $k$ -means clustering for  $\{2, 4, 10, 12, 3, 20, 30, 11, 25\}$  for  $k = 2$ . Assume 2 and 4 as initial cluster centroids.

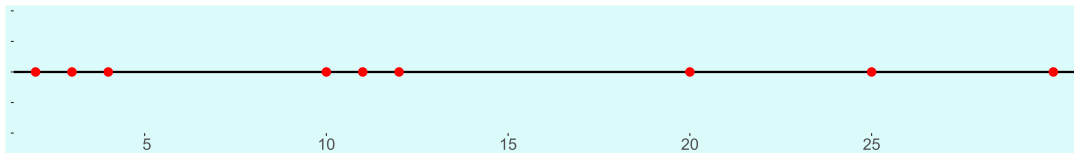
⊕ After a choice of initial centroids, the  $k$ -means clustering algorithm is as follows :

- (1) calculate the distance of each data-point from each of the centroids
- (2) assign each of the data-points to its closest centroid
- (3) relocate the centroids to the average location of the data-points of similar group

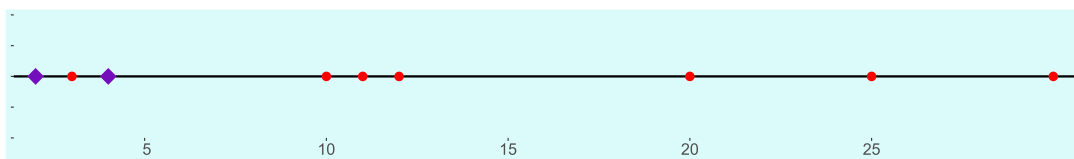
And we repeat this procedure until the assignments don't change after the centroid locations were recomputed.

```
df <- data.frame(x = c(2, 4, 10, 12, 3, 20, 30, 11, 25))
```

Let us have a look at the data-points.



Now We put the initial centroids.



```
m <- dim(df)[1] # number of data-points
n <- dim(df)[2] # dimension of data-points

k <- 2 # number of clusters
```

```
X <- as.matrix(df)
```

Now we initialize the centroids as 2 and 4.

```
centroid <- matrix(data = c(2,
                             4),
                   nrow = k, ncol = 1, byrow = TRUE)
```

We now deploy our  $k$ -means clustering algorithm. We created a list named *iteration\_record()* for visualization of the process that will come later.

```
cluster <- c()

iteration_record <- list()

repeat{
  dist_mat <- matrix(0, nrow = m, ncol = k)

  for (i in 1:k) {
    d <- apply(X, 1, FUN = function(x) return(x - centroid[i, ]))

    d <- matrix(d, nrow = m, ncol = n, byrow = TRUE)

    dist_mat[,i] <- sqrt(diag( d %*% t(d) ) )
  }


  cluster <- apply(dist_mat, 1, FUN = function(x) return(which(x == min(x))[1]))

  new_centroid <- matrix(data = 0, nrow = k, ncol = n)

  for (i in 1:k) {
    new_centroid[i, ] <- mean(X[which(cluster == i), ])
  }

  iteration_record <- append(iteration_record,
                             list(list(mat = cbind(X, dist_mat, cluster),
                                           new_centroid = new_centroid)))

  if(any(centroid - new_centroid != 0)){
    centroid <- new_centroid
  } else{
    break
  }
}
```

 The final clustering of the data-points is as follows :

```
cluster
## [1] 1 1 1 1 1 2 2 1 2
```

```
length(iteration_record)
```

```
## [1] 5
```

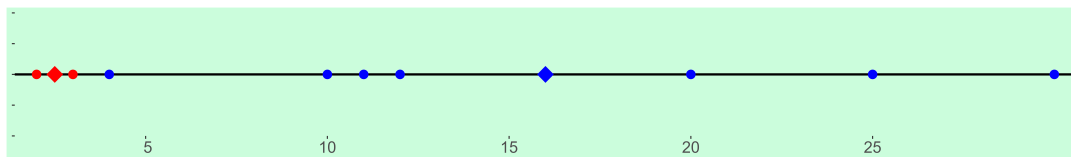
There were 5 iterations, we shall take a look at them one by one.

### 👉 Iteration 1

```
iteration_record[[1]]$mat
```

```
##          x distance_from_centroid_1 distance_from_centroid_2 cluster
## [1,]    2                      0                      2          1
## [2,]    4                      2                      0          2
## [3,]   10                      8                      6          2
## [4,]   12                     10                      8          2
## [5,]    3                      1                      1          1
## [6,]   20                     18                     16          2
## [7,]   30                     28                     26          2
## [8,]   11                      9                      7          2
## [9,]   25                     23                     21          2
```

The data-points along with relocated centroids are as follows :

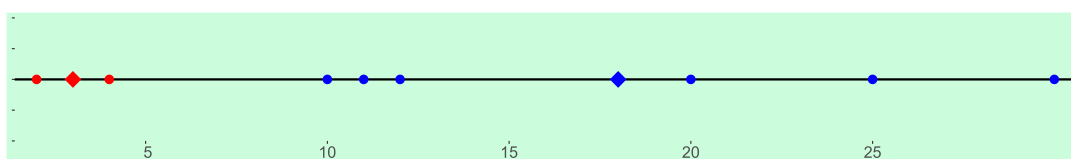


### 👉 Iteration 2

```
iteration_record[[2]]$mat
```

```
##          x distance_from_centroid_1 distance_from_centroid_2 cluster
## [1,]    2                      0.5                     14          1
## [2,]    4                      1.5                     12          1
## [3,]   10                      7.5                      6          2
## [4,]   12                      9.5                      4          2
## [5,]    3                      0.5                     13          1
## [6,]   20                     17.5                      4          2
## [7,]   30                     27.5                     14          2
## [8,]   11                      8.5                      5          2
## [9,]   25                     22.5                      9          2
```

The data-points along with relocated centroids are as follows :



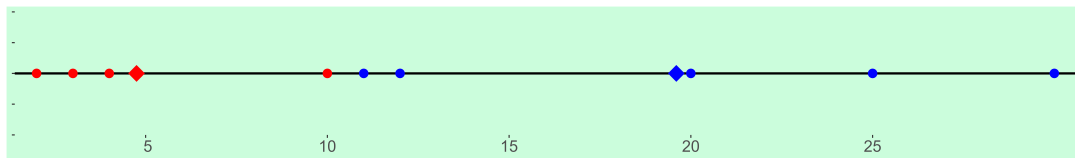


### Iteration 3

```
iteration_record[[3]]$mat
```

##		x	distance_from_centroid_1	distance_from_centroid_2	cluster
##	[1,]	2	1	16	1
##	[2,]	4	1	14	1
##	[3,]	10	7	8	1
##	[4,]	12	9	6	2
##	[5,]	3	0	15	1
##	[6,]	20	17	2	2
##	[7,]	30	27	12	2
##	[8,]	11	8	7	2
##	[9,]	25	22	7	2

The data-points along with relocated centroids are as follows :

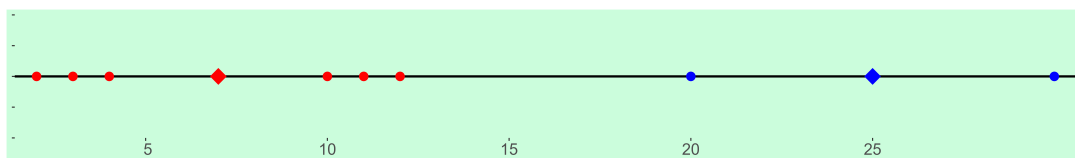


### Iteration 4

```
iteration_record[[4]]$mat
```

##		x	distance_from_centroid_1	distance_from_centroid_2	cluster
##	[1,]	2	2.75	17.6	1
##	[2,]	4	0.75	15.6	1
##	[3,]	10	5.25	9.6	1
##	[4,]	12	7.25	7.6	1
##	[5,]	3	1.75	16.6	1
##	[6,]	20	15.25	0.4	2
##	[7,]	30	25.25	10.4	2
##	[8,]	11	6.25	8.6	1
##	[9,]	25	20.25	5.4	2

The data-points along with relocated centroids are as follows :

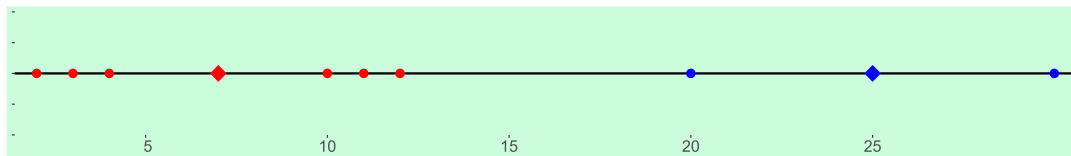


## 📖 Iteration 5

```
iteration_record[[5]]$mat
```

##		x	distance_from_centroid_1	distance_from_centroid_2	cluster
##	[1,]	2	5	23	1
##	[2,]	4	3	21	1
##	[3,]	10	3	15	1
##	[4,]	12	5	13	1
##	[5,]	3	4	22	1
##	[6,]	20	13	5	2
##	[7,]	30	23	5	2
##	[8,]	11	4	14	1
##	[9,]	25	18	0	2

The data-points along with relocated centroids are as follows :



📝 We notice that there is no change in location centroids from Iteration 4 to Iteration 5. So the process stops and we get our final set of clusters.

# MSMS 206 : Practical 06

Ananda Biswas

April 19, 2025



**Question :** Use  $k$ -means clustering to divide *iris* dataset into 3 clusters.

⊕ After a choice of initial centroids, the  $k$ -means clustering algorithm is as follows :

- (1) calculate the distance of each data-point from each of the centroids
- (2) assign each of the data-points to its closest centroid
- (3) relocate the centroids to the average location of the data-points of similar group

And we repeat this procedure until the assignments don't change after the centroid locations were recomputed.

```
df <- iris[, -5]
```

```
dim(df)
```

```
## [1] 150 4
```

```
m <- dim(df)[1] # number of data-points  
n <- dim(df)[2] # dimension of data-points  
  
k <- 3 # number of clusters
```

```
X <- as.matrix(df)
```

Now we initialize the centroids as 3 randomly chosen data-points.

```
random_index <- sample(m, k)  
  
centroid <- X[random_index, ]
```

We now deploy our  $k$ -means clustering algorithm.

```
cluster <- c()  
  
repeat{  
  dist_mat <- matrix(0, nrow = m, ncol = k)
```



# MSMS 206 : Practical 07

Ananda Biswas

April 19, 2025



**Question :** For the following distance matrix, perform hierarchical clustering and plot a dendrogram.

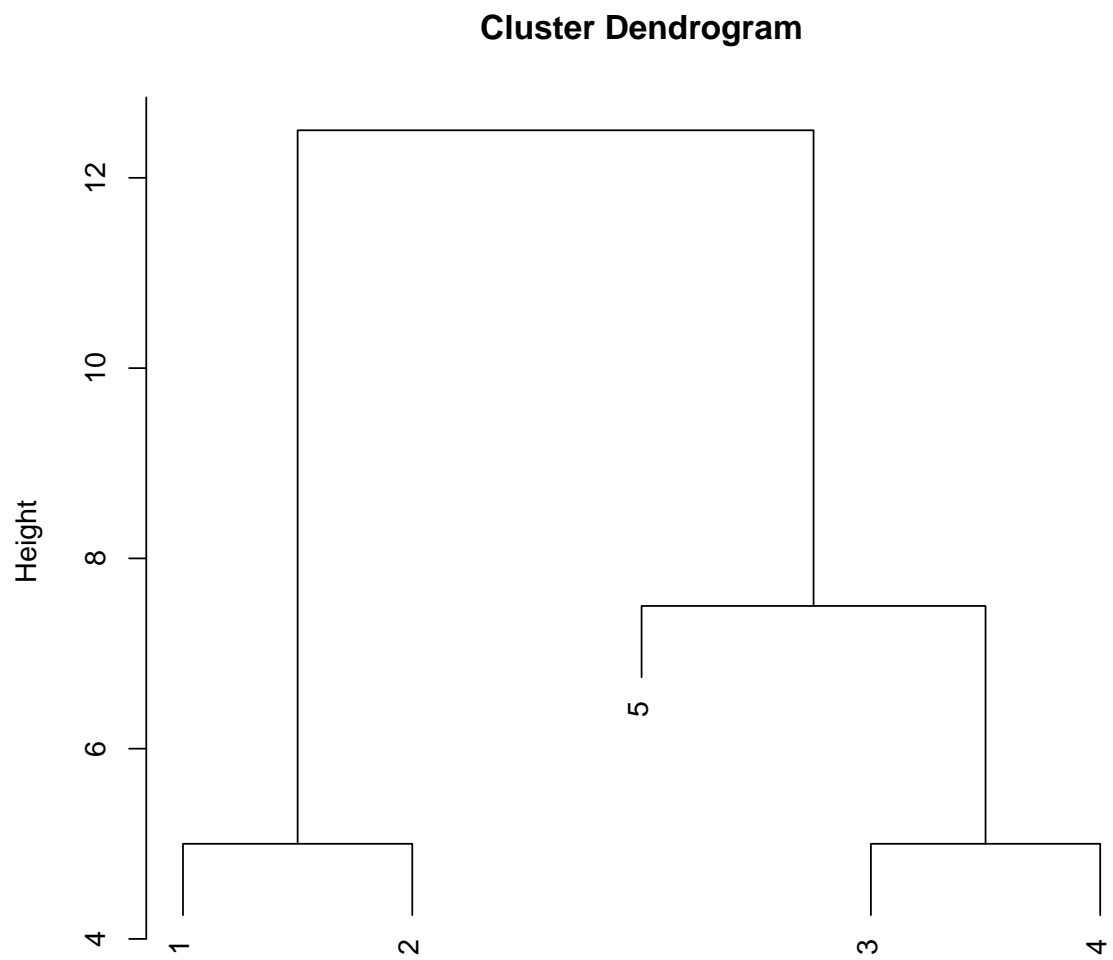
$$\begin{bmatrix} 0 & 5 & 10 & 15 & 20 \\ & 0 & 5 & 10 & 15 \\ & & 0 & 5 & 10 \\ & & & 0 & 5 \\ & & & & 0 \end{bmatrix}$$

➡ Agglomerative Hierarchical Clustering

```
d <- matrix(c(0, 5, 10, 15, 20,
              5, 0, 5, 10, 15,
              10, 5, 0, 5, 10,
              15, 10, 5, 0, 5,
              20, 15, 10, 5, 0), nrow = 5, ncol = 5, byrow = TRUE)
```

```
x <- hclust(as.dist(d), method = "average")
```

```
plot(x, xlab = "")
```



```
hclust (*, "average")
```

# MSMS 206 : Practical 08

Ananda Biswas

May 6, 2025



**Question :** Obtain an initial basic feasible solution of the following transportation problem :

	$D_1$	$D_2$	$D_3$	$D_4$	$a_i$
$O_1$	19	30	50	12	7
$O_2$	70	30	40	60	10
$O_3$	40	10	60	20	18
$b_j$	5	8	7	15	

## ⊕ Transportation Problem

```
library(lpSolve)
```

```
supply <- c(7, 10, 18)
demand <- c(5, 8, 7, 15)
```

```
cost_matrix <- matrix(c(19, 30, 50, 12,
                        70, 30, 40, 60,
                        40, 10, 60, 20), nrow = 3, ncol = 4, byrow = TRUE)
```


```
solution <- lp.transport(cost.mat = cost_matrix,
                        direction = "min",
                        row.signs = rep("=", nrow(cost_matrix)),
                        row.rhs = supply,
                        col.signs = rep("=", ncol(cost_matrix)),
                        col.rhs = demand)
```



The allotment matrix is

```
solution$solution
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    5    0    0    2
## [2,]    0    3    7    0
## [3,]    0    5    0   13
```

 The cost for the above allotment is

```
solution$objval
```

```
## [1] 799
```



# MSMS 206 : Practical 09

Ananda Biswas

May 6, 2025



**Question :** Fit a decision tree model using the **Loan Defaulters' Dataset** given as follows :

Home Owner	Married Status	Defaulted	Annual Income(\$)
yes	single	no	125000
no	married	no	100000
no	single	no	70000
yes	married	no	120000
no	divorcee	yes	95000
no	married	no	60000
yes	divorcee	no	220000
no	single	yes	85000
no	married	no	75000
no	single	yes	90000

Will a divorcee home owner with annual income \$120000 default in his loan ?

## ⊕ Building Decision Tree Model

```
library(rpart)
library(rpart.plot)
```

```
df <- read.csv('https://raw.githubusercontent.com/sakunisgithub/data_sets/refs/heads/master/msc_semester_2/loan_defaulter_data.csv')
```

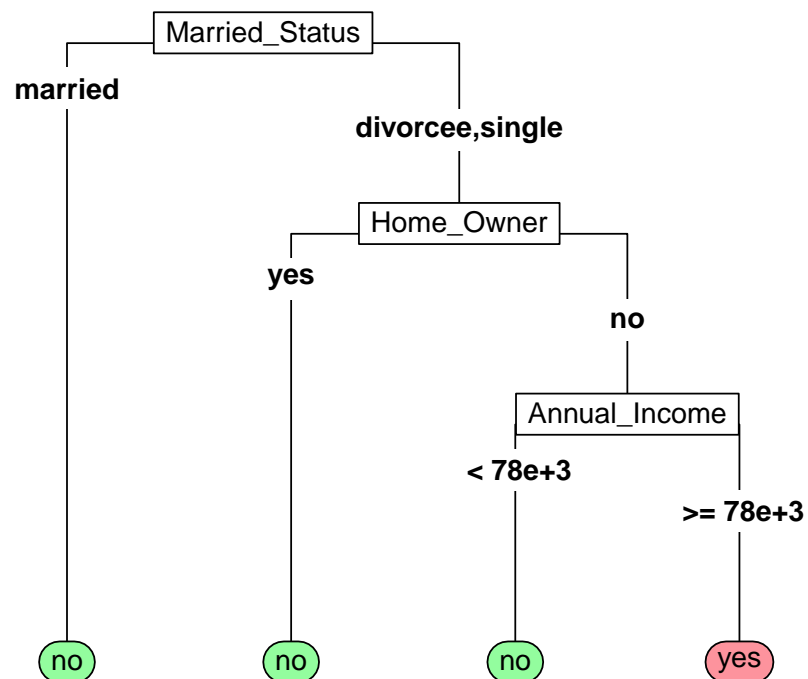
```
tree_model <- rpart(Defaulted ~ .,
  data = df,
  method = "class",
  parms = list(split = "information"),
  control = rpart.control(minsplit = 2,
    minbucket = 1,
    cp = 0.01))
```

```
print(tree_model)
```

```
## n= 10
##
## node), split, n, loss, yval, (yprob)
##      * denotes terminal node
##
## 1) root 10 3 no (0.7000000 0.3000000)
##    2) Married_Status=married 4 0 no (1.0000000 0.0000000) *
##    3) Married_Status=divorcee,single 6 3 no (0.5000000 0.5000000)
##      6) Home_Owner=yes 2 0 no (1.0000000 0.0000000) *
##      7) Home_Owner=no 4 1 yes (0.2500000 0.7500000)
##        14) Annual_Income< 77500 1 0 no (1.0000000 0.0000000) *
##        15) Annual_Income>=77500 3 0 yes (0.0000000 1.0000000) *
```

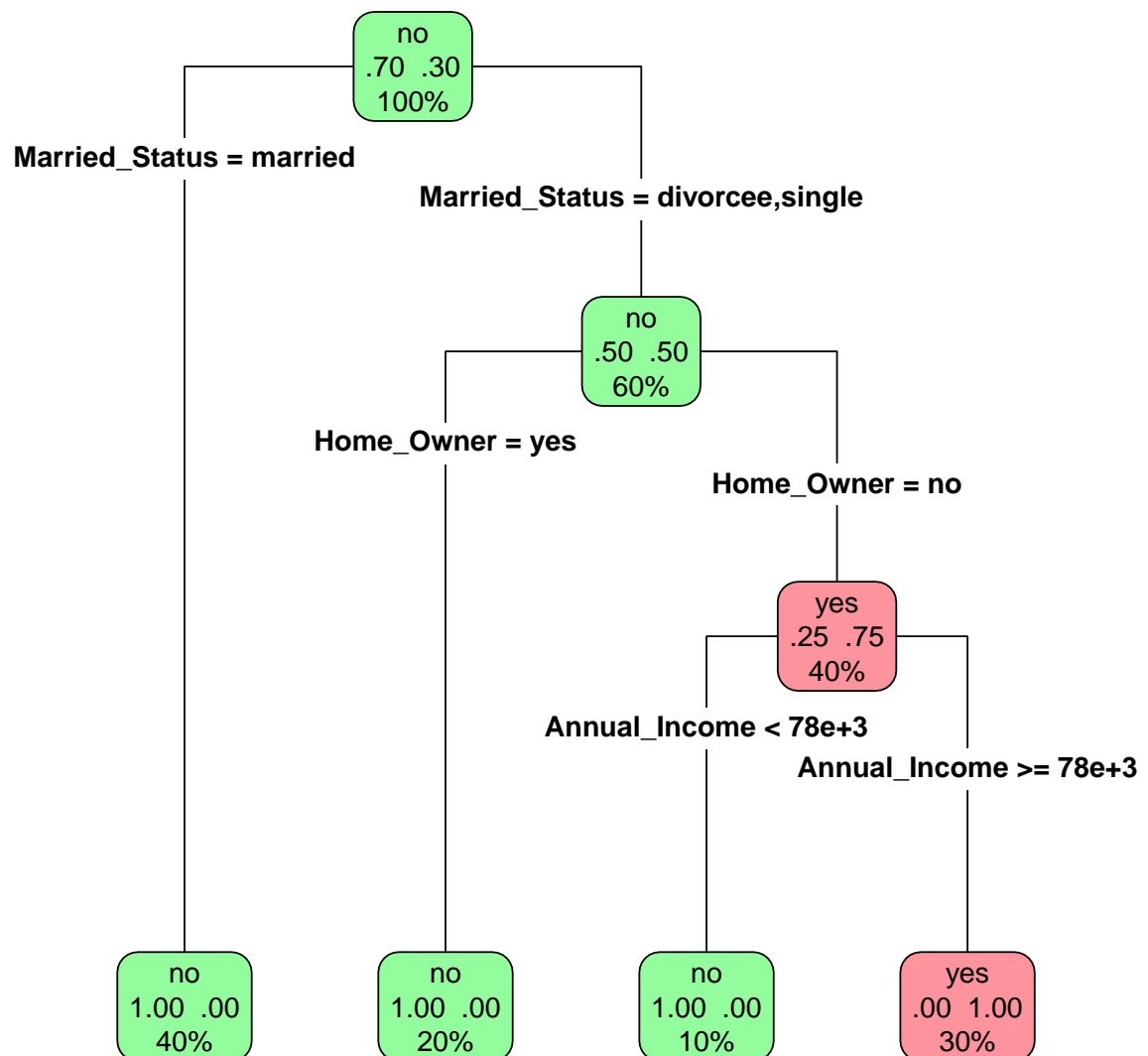
## ④ Visualizing the Decision Tree Model

```
rpart.plot(tree_model,  
  type = 5,  
  extra = 0,  
  box.palette = c("#93fd9e", "#fd939d"))
```



The following diagram displays labels at all nodes, giving a comprehensive idea how the tree was made.

```
rpart.plot(tree_model,
  type = 4,
  extra = 104,
  clip.right.labs = FALSE,
  box.palette = c("#93fd9e", "#fd939d"))
```



### ⊕ Prediction on New Example

```
new_example <- data.frame(Home_Owner = "yes",  
                          Married_Status = "divorcee",  
                          Annual_Income = 120000)
```

```
predict(tree_model, new_example)
```

```
##    no yes  
## 1  1   0
```

Our decision tree model predicts that a divorcee home owner with annual income 120000 dollars will not default in his loan.

# MSMS 206 : Practical 10

Ananda Biswas

May 6, 2025



**Question :** In a certain social mobility study, the population under consideration was divided into three income groups: upper, middle and lower. It has been found that 70% of sons of upper income group parents themselves become upper income, 20% middle income and 10% lower income. Of the sons of middle-income parents, 30% move to upper income group, 50% remain in middle income group and 20% become lower. Of the sons of lower income parents, 10% move to upper income group, 20% to middle income group and 70% remain lower. Draw up a matrix to represent these movements.

At a certain point of time, the population is found to have 10% men in upper income group, 50% in middle income group and 40% in lower. Assuming that each man has one son and one grandson, what will be the group composition of grandsons?

## ➡ Transition Probability Matrix

```
TPM <- matrix(c(0.7, 0.2, 0.1,
                0.3, 0.5, 0.2,
                0.1, 0.2, 0.7), nrow = 3, ncol = 3, byrow = TRUE)

states <- c("upper", "middle", "lower")

rownames(TPM) <- states
colnames(TPM) <- states
```



The transition probability matrix is

```
TPM

##          upper middle lower
## upper    0.7     0.2   0.1
## middle   0.3     0.5   0.2
## lower    0.1     0.2   0.7
```

## ➡ Future Distribution from Initial Distribution

```
X_0 <- matrix(c(0.1, 0.5, 0.4), nrow = 1)
X_1 <- X_0 %*% TPM
X_2 <- X_1 %*% TPM
```



The group composition of grandsons is

```
X_2

##          upper middle lower
## [1,] 0.326   0.305 0.369
```

# MSMS 206 : Practical 11

Ananda Biswas

May 8, 2025



**Question :** Write a program to generate random sample from Poisson Process with parameter  $\lambda$ .

Also write programs to generate  $X(t)$  if

(a)  $P(X(t) = k) = \binom{k + \alpha - 1}{k} \left(\frac{\beta}{t + \beta}\right)^\alpha \left(\frac{t}{t + \beta}\right)^k; \quad k = 0, 1, 2, \dots$

(b)  $P(X(t) = k) = \left(\frac{t}{t + \mu}\right)^k \left(\frac{\mu}{t + \mu}\right); \quad k = 0, 1, 2, \dots$

⊕ A realization of Poisson Process


We take  $\lambda = 2$  and observe the Poisson Process upto time  $T = 5$ .

```
T <- 5  
  
lambda <- 2
```


We observe the arrival times upto time  $T$ , for that we generate exponentially distributed random numbers with rate  $\lambda$  until total time becomes  $T$ . The count of the arrival times form a realization of a Poisson Process with parameter  $\lambda$ .

```
times <- c(0)  
  
i <- 2  
  
sum_times <- 0  
  
while(sum_times < T){  
  
  times[i] <- round(rexp(1, rate = lambda), digits = 2)  
  
  sum_times <- sum(times)  
  
  i <- i + 1  
}  
  
occurrence_times <- cumsum(times[-length(times)])  
  
x <- 0:(length(occurrence_times)-1)
```

```
df1 <- data.frame(States = x, Occurrence_Time = occurrence_times)
```

 States and their arrival times are as follows :

```
df1
##      States Occurrence_Time
## 1         0          0.00
## 2         1          0.09
## 3         2          0.74
## 4         3          0.83
## 5         4          2.51
## 6         5          2.62
## 7         6          3.36
## 8         7          4.13
## 9         8          4.17
## 10        9          4.39
## 11       10          4.47
## 12       11          4.62
## 13       12          4.85
```

 A realization of a Poisson Process with  $\lambda = 2$  in  $(0, 5]$  is as follows :

$$X(t) = \begin{cases} 0, & \text{if } 0 \leq t < 0 \\ 1, & \text{if } 0 \leq t < 0.09 \\ 2, & \text{if } 0.09 \leq t < 0.74 \\ \vdots & \vdots \\ 11, & \text{if } 4.62 \leq t < 4.85 \\ 12, & \text{if } 4.85 \leq t \leq 5 \end{cases}$$

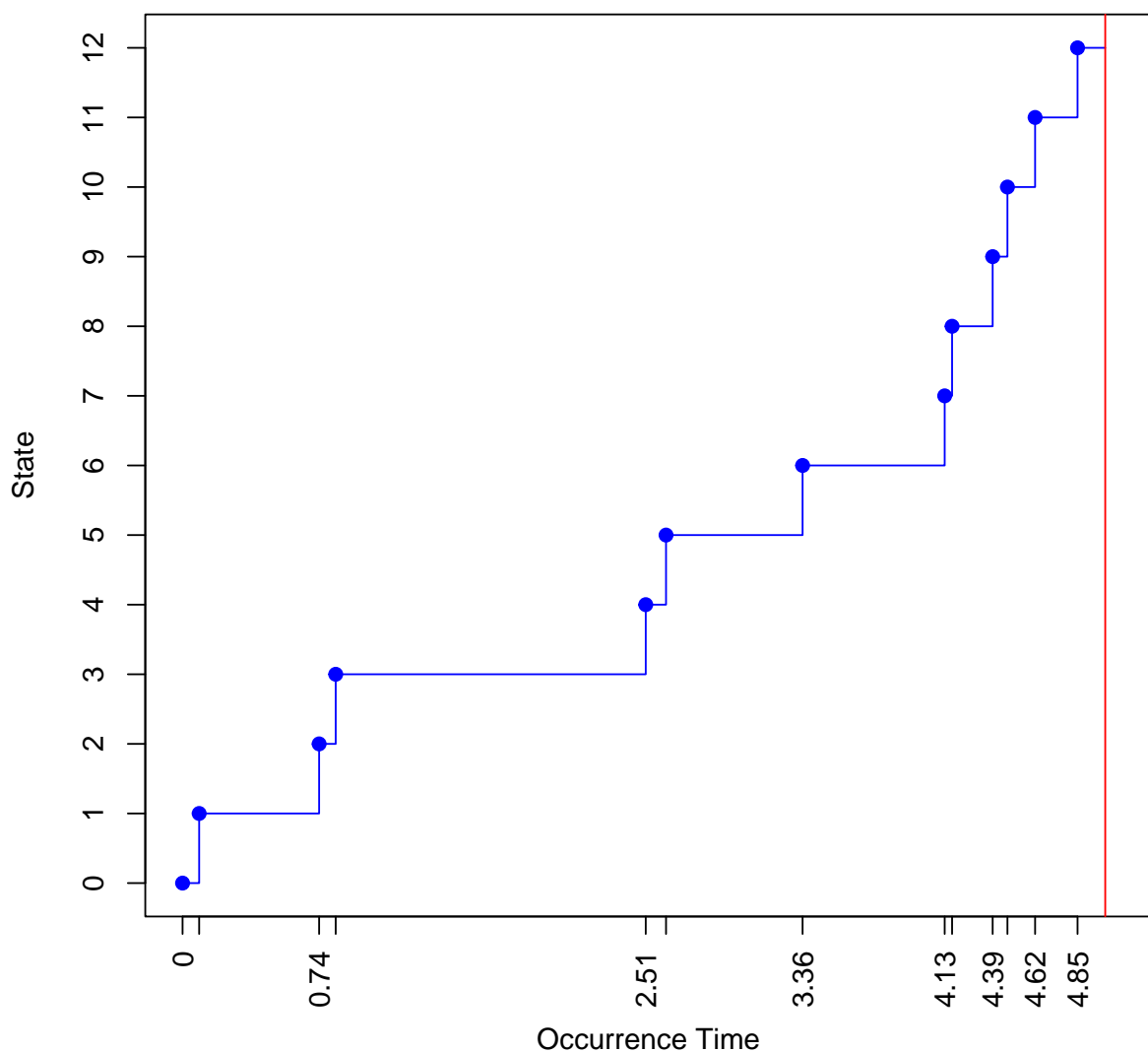


## ➔ Visualization

```
plot(c(x, x[length(x)])) ~ c(occurrence_times, T),
     type = "s",
     col = "blue",
     xaxt = "n",
     yaxt = "n",
     xlim = c(0, T + 0.05),
     xlab = "Occurrence Time",
     ylab = "State",
     main = paste("Realization of Poisson Process with lambda = ", lambda))

axis(1, at = occurrence_times, labels = occurrence_times, las = 2)
axis(2, at = x, labels = x)
points(occurrence_times, x, cex = 1, col = "blue", pch = 19)
abline(v = T, col = "red")
```

**Realization of Poisson Process with lambda = 2**



⊕  $X(t) \sim \text{Negative Binomial}$

$$P(X(t) = k) = \binom{k + \alpha - 1}{k} \left( \frac{\beta}{t + \beta} \right)^\alpha \left( \frac{t}{t + \beta} \right)^k ; \quad k = 0, 1, 2, \dots$$

$$\text{i.e. } X(t) \sim \text{Negative Binomial} \left( \alpha, \frac{\beta}{t + \beta} \right).$$

Previously  $\lambda$  was a fixed parameter. Now  $\lambda$  will be sampled from  $\text{Gamma}(\text{shape} = \alpha, \text{rate} = \beta)$ . Here we take  $\alpha = 2, \beta = 1$ .

```
alpha <- 2; beta <- 1
```

```
T <- 5
```

```
lambda <- rgamma(1, shape = alpha, rate = beta)
```

```
times <- c(0)
```

```
i <- 2
```

```
sum_times <- 0
```

```
while(sum_times < T){
```

```
  times[i] <- round(rexp(1, rate = lambda), digits = 2)
```

```
  sum_times <- sum(times)
```

```
  i <- i + 1
```

```
}
```

```
occurrence_times <- cumsum(times[-length(times)])
```

```
x <- 0:(length(occurrence_times)-1)
```

```
df2 <- data.frame(States = x, Occurrence_Time = occurrence_times)
```

```
df2
```

```
##   States Occurrence_Time
## 1      0             0.00
## 2      1             0.26
## 3      2             1.67
```

⊕  $X(t) \sim \text{Geometric}$

$$P(X(t) = k) = \left(\frac{t}{t + \mu}\right)^k \left(\frac{\mu}{t + \mu}\right); \quad k = 0, 1, 2, \dots \text{ i.e. } X(t) \sim \text{Geometric}\left(\frac{\mu}{t + \mu}\right).$$

Previously  $\lambda$  was a fixed parameter. Now  $\lambda$  will be sampled from  $\text{Gamma}(\text{shape} = 1, \text{rate} = \mu) \Leftrightarrow \text{Exp}(\text{rate} = \mu)$ .

Here we take  $\mu = 2$ .

```
mu <- 2
```

```
T <- 5
```

```
lambda <- rgamma(1, shape = 1, rate = mu)
```

```
times <- c(0)
```

```
i <- 2
```

```
sum_times <- 0
```

```
while(sum_times < T){
```

```
  times[i] <- round(rexp(1, rate = lambda), digits = 2)
```

```
  sum_times <- sum(times)
```

```
  i <- i + 1
```

```
}
```

```
occurrence_times <- cumsum(times[-length(times)])
```

```
x <- 0:(length(occurrence_times)-1)
```

```
df3 <- data.frame(States = x, Occurrence_Time = occurrence_times)
```

```
df3
```

```
##      States Occurrence_Time
## 1         0             0.00
## 2         1             0.72
## 3         2             3.23
## 4         3             3.50
## 5         4             4.80
```