PS02 by Emily Han

2025-01-20

1. The exponential distribution

The exponential probability density function is defined as: $f_e(x;\theta) = \theta \exp(-\theta x)$.

1a.

Loosely speaking, the "support" of a random variable is the set of admissible values, or values that have positive probability. More formally, the support of a function is the subset of the function's domain that are not mapped to zero. If $X \sim f_e(x;\theta)$ what is the support of X?

The support of the function is when $\theta > 0$.

1b.

Derive the log-likelihood for n independent observations under the probability model f_e.

$$L(\theta; x_1, x_2, .., x_n) = \prod_{i=1}^{n} \theta e^{-\theta x_i}$$

$$logL(\theta; x) = nlog(\theta) - \theta \sum_{i=1}^{n} x_i$$

1c.

Derive an analytic expression for θ, the MLE.

$$\frac{d}{d\theta}\left(n\log(\theta) - \theta\sum_{i=1}^{n} x_i\right) = \frac{n}{\theta} - \sum_{i=1}^{n} x_i$$

$$\frac{n}{\theta} - \sum_{i=1}^{n} x_i = 0$$

$$\hat{\theta} = \frac{n}{\sum_{i=1}^{n} x_i}$$

1d.

```
library(formatR)
set.seed(5)
x <- rexp(10000, 5)
```

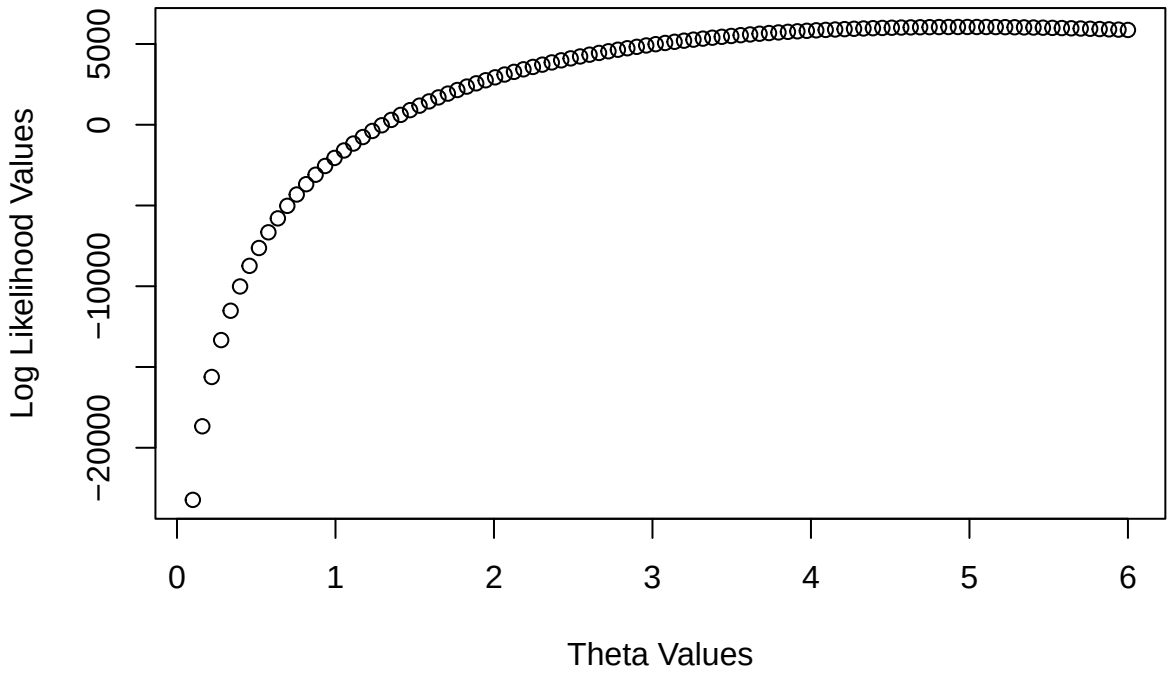
i. Program the exponential log likelihood function into R; call this function exp.ll.

```
exp.ll <- function(theta,x){
  n <- length(x)
  log_likelihood <- n * log(theta) - theta* sum(x)
  return(log_likelihood)
}
```

ii. Plot the log-likelihood as a function of θ given x. Eyeballing the graph, what is the approximate value of the maximizer?

```
theta_values <- seq(0.1, 6, length.out = 100)
log_ll_val <- exp.ll(theta_values, x)


plot(theta_values,log_ll_val, xlab = "Theta Values", ylab = "Log Likelihood Values")
```



The approximate value of the maximizer is 5.2.

iii. Calculate $\hat{\theta}$ given x using the analytic result you derived in part (c).

```
mle <- function(i){
  n <- length(i)
  theta_hat <- n / (sum(i))
```

```
  return(theta_hat)
}

theta_hat <- mle(x)
```

iv. Let $\tilde{\theta} = 6$. Calculate the likelihood ratio for $\hat{\theta} \mid x$ and $\tilde{\theta} \mid x$.

```
log_val1 <- exp.ll(theta_hat, x)
log_val2 <- exp.ll(6, x)



likelihood_ratio <- log_val1 - log_val2
likelihood_ratio
```

```
## [1] 184.9271
```

v. Calculate an approximation to $\hat{\theta}$ as well as the standard error around $\hat{\theta}$ using optim. Use 1 as your starting value. You will do this twice, once using the BFGS algorithm and once using SANN. Report the total computational time required for each.

Maximum Likelihood Estimation using BFGS

```
start1 <- Sys.time()
mle.fit <- optim(
  par = c(1),               # Starting values for the parameters
  fn = exp.ll,              # Function to minimize (log-likelihood)
  x = x,
  method = "BFGS",             # Optimization method (Broyden-Fletcher-Goldfarb-Shanno algorithm)
  control = list(
    trace = TRUE,              # Show optimization progress
    maxit = 1000,              # Maximum number of iterations
    fnscale = -1               # Negate the function to maximize instead of minimize
  ),
  hessian = TRUE               # Return the Hessian matrix
)
```

```
## initial  value 2008.061622
```

```
## Warning in log(theta): NaNs produced
## Warning in log(theta): NaNs produced
## Warning in log(theta): NaNs produced
## Warning in log(theta): NaNs produced
## Warning in log(theta): NaNs produced
## Warning in log(theta): NaNs produced
## Warning in log(theta): NaNs produced
```

```
## iter  10 value -6054.151950
## iter  10 value -6054.152023
## final  value -6054.152023
## converged
```

```r
end1 <- Sys.time()
tot_time1 <- end1 - start1
print(paste("Total time for BFGS Method:", tot_time1))
```

```
## [1] "Total time for BFGS Method: 0.00493597984313965"
```

```r
mle.fit$par
```

```
## [1] 4.980155
```

Maximum Likelihood Estimation using SANN

```r
start2 <- Sys.time()
mle.fit2 <- optim(
  par = c(1),               # Starting values for the parameters
  fn = exp.ll,              # Function to minimize (log-likelihood)
  x = x,
  method = "SANN",             # Optimization method (Simulated Annealing algorithm)
  control = list(
    trace = TRUE,              # Show optimization progress
    maxit = 1000,              # Maximum number of iterations
    fnscale = -1               # Negate the function to maximize instead of minimize
  ),
  hessian = TRUE               # Return the Hessian matrix
)
```

```
## sann objective function values
## initial       value 2008.061622
## iter      999 value -6054.152030
## final         value -6054.152030
## sann stopped after 999 iterations
```

```r
end2 <- Sys.time()
tot_time2 <-  end2 - start2
print(paste("Total time for SANN Method:", tot_time2))
```

```
## [1] "Total time for SANN Method: 0.0255000591278076"
```

```r
mle.fit2$par
```

```
## [1] 4.979811
```

2. Maximizing a multivariate function

2a.

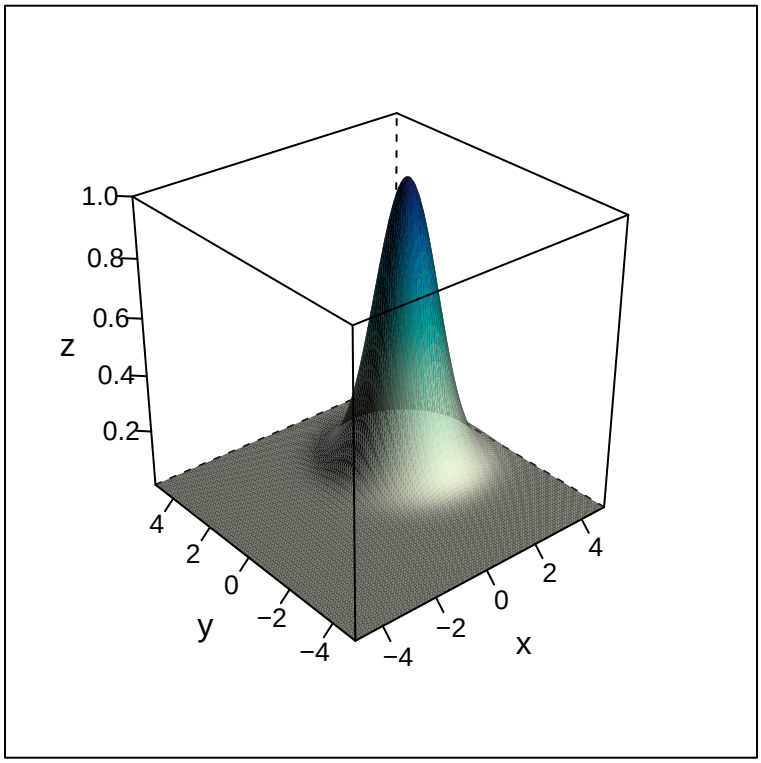
Use the following code to implement and visualize this function. Just eye-balling it, at what values of x and y does it look like the function achieves a maximum? Use ChatGPT or a similar tool to walk you through the code if you do not understand what the code is doing.

```r
mvn <- function(xy) {
    x <- xy[1]
    y <- xy[2]
    z <- exp(-0.5 * ((x - 2)^2 + (y - 1)^2))
    return(z)
}

# install.packages('lattice')
library(lattice)

y <- x <- seq(-5, 5, by = 0.1)
grid <- expand.grid(x, y)
names(grid) <- c("x", "y")
grid$z <- apply(grid, 1, mvn)

wireframe(z ~ x + y, data = grid, shade = TRUE, light.source = c(10, 0, 10), scales = list(arrows = FALS
```



It looks like the function achieves a maximum at x = 2 and y = 1.

2b.

Use optim() to find the values $(x^\star, y^\star)$ that maximize this joint density. Use c(1,0) as your starting values and method="BFGS". Then find the maximum again with the starting value c(5,5), still using method="BFGS". Compare the performance of optim() for these two sets of starting values.

Maximum Likelihood Estimation using c(1,0) as starting value

```r
mle.fit_b1 <- optim(
  par = c(1, 0),              # Starting values for the parameters
  fn = mvn,             # Function to minimize (log-likelihood)
  method = "BFGS",            # Optimization method (Broyden-Fletcher-Goldfarb-Shanno algorithm)
  control = list(
    trace = TRUE,             # Show optimization progress
    maxit = 1000,             # Maximum number of iterations
    fnscale = -1              # Negate the function to maximize instead of minimize
  ),
  hessian = TRUE              # Return the Hessian matrix
)
```

```
## initial  value -0.367879
## final  value -1.000000
## converged
```

```r
mle.fit_b1$par
```

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

Maximum Likelihood Estimation using c(5,5) as starting value

```r
# Maximum Likelihood Estimation using optim
mle.fit_b2 <- optim(
  par = c(5, 5),              # Starting values for the parameters
  fn = mvn,             # Function to minimize (log-likelihood)
  method = "BFGS",            # Optimization method (Broyden-Fletcher-Goldfarb-Shanno algorithm)
  control = list(
    trace = TRUE,             # Show optimization progress
    maxit = 1000,             # Maximum number of iterations
    fnscale = -1              # Negate the function to maximize instead of minimize
  ),
  hessian = TRUE              # Return the Hessian matrix
)
```

```
## initial  value -0.000004
## iter  10 value -0.000004
## iter  20 value -0.000004
## iter  30 value -0.000004
## iter  40 value -0.000004
## iter  50 value -0.000004
## iter  60 value -0.000004
## iter  70 value -0.000004
## iter  80 value -0.000004
## iter  90 value -0.000004
## iter 100 value -0.000004
## iter 110 value -0.000004
## iter 120 value -0.000004
## iter 130 value -0.000004
## iter 140 value -0.000004
## iter 150 value -0.000004
## iter 160 value -0.000004
```

```
## iter 170 value -0.000004
## iter 180 value -0.000004
## iter 190 value -0.000004
## iter 200 value -0.000004
## iter 210 value -0.000004
## iter 220 value -0.000004
## iter 230 value -0.000004
## iter 240 value -0.000004
## iter 250 value -0.000004
## iter 260 value -0.000004
## iter 270 value -0.000004
## iter 280 value -0.000004
## iter 290 value -0.000004
## iter 300 value -0.000004
## iter 310 value -0.000004
## iter 320 value -0.000004
## iter 330 value -0.000004
## iter 340 value -0.000004
## iter 350 value -0.000004
## iter 360 value -0.000004
## iter 370 value -0.000004
## iter 380 value -0.000004
## iter 390 value -0.000004
## iter 400 value -0.000004
## iter 410 value -0.000004
## iter 420 value -0.000004
## iter 430 value -0.000004
## iter 440 value -0.000004
## iter 450 value -0.000004
## iter 460 value -0.000004
## iter 470 value -0.000004
## iter 480 value -0.000004
## iter 490 value -0.000004
## iter 500 value -0.000004
## iter 510 value -0.000004
## iter 520 value -0.000004
## iter 530 value -0.000004
## iter 540 value -0.000004
## iter 550 value -0.000004
## iter 560 value -0.000004
## iter 570 value -0.000004
## iter 580 value -0.000004
## iter 590 value -0.000004
## iter 600 value -0.000004
## iter 610 value -0.000004
## iter 620 value -0.000004
## iter 630 value -0.000004
## iter 640 value -0.000004
## iter 650 value -0.000004
## iter 660 value -0.000004
## iter 670 value -0.000004
## iter 680 value -0.000004
## iter 690 value -0.000004
## iter 700 value -0.000004
```

```
## iter 710 value -0.000004
## iter 720 value -0.000004
## iter 730 value -0.000004
## iter 740 value -0.000004
## iter 750 value -0.000004
## iter 760 value -0.000004
## iter 770 value -0.000004
## iter 780 value -0.000004
## iter 790 value -0.000004
## iter 800 value -0.000004
## iter 810 value -0.000004
## iter 820 value -0.000004
## iter 830 value -0.000004
## iter 840 value -0.000004
## iter 850 value -0.000004
## iter 860 value -0.000004
## iter 870 value -0.000004
## iter 880 value -0.000004
## iter 890 value -0.000004
## iter 900 value -0.000004
## iter 910 value -0.000004
## iter 920 value -0.000004
## iter 930 value -0.000004
## iter 940 value -0.000004
## iter 950 value -0.000004
## iter 960 value -0.000004
## iter 970 value -0.000004
## iter 980 value -0.000004
## iter 990 value -0.000004
## iter1000 value -0.000004
## final  value -0.000004
## stopped after 1000 iterations
```

```
mle.fit_b2$par
```

```
## [1] 4.988302 4.984402
```

2c.

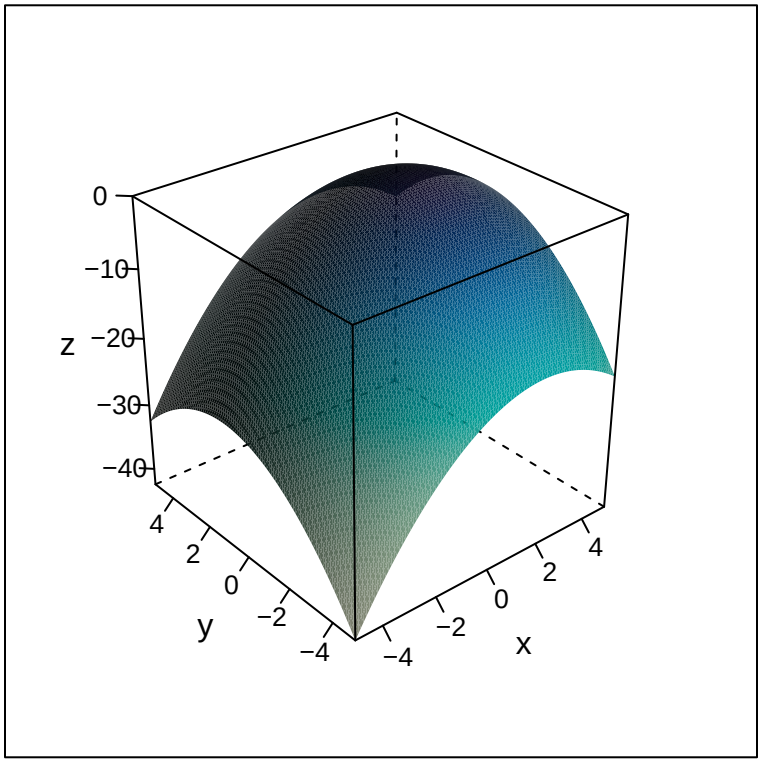
Alter the provided R function so that it returns the (natural) logarithm of $f(x, y)$ Alter the code to plot the log likelihood. Then repeat the two procedures you performed in section (b), only as applied to the log likelihood. Describe any differences in your results, and then provide a brief explanation for those differences.

```
mvn_log <- function(xy) {
    x <- xy[1]
    y <- xy[2]
    z <- log(exp(-0.5 * ((x - 2)^2 + (y - 1)^2)))
    return(z)
}


y <- x <- seq(-5, 5, by = 0.1)
```

```r
grid <- expand.grid(x, y)
names(grid) <- c("x", "y")
grid$z <- apply(grid, 1, mvn_log)

wireframe(z ~ x + y, data = grid, shade = TRUE, light.source = c(10, 0, 10), scales = list(arrows = FALS
```



Maximum Likelihood Estimation of log-function using c(1,0) as starting value

```r
# Maximum Likelihood Estimation using optim
mle.fit_b3 <- optim(
  par = c(1, 0),              # Starting values for the parameters
  fn = mvn_log,              # Function to minimize (log-likelihood)
  method = "BFGS",            # Optimization method (Broyden-Fletcher-Goldfarb-Shanno algorithm)
  control = list(
    trace = TRUE,             # Show optimization progress
    maxit = 1000,             # Maximum number of iterations
    fnscale = -1              # Negate the function to maximize instead of minimize
  ),
  hessian = TRUE              # Return the Hessian matrix
)
```

```
## initial  value 1.000000
## final  value -0.000000
## converged
```

```r
mle.fit_b3$par
```

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

Maximum Likelihood Estimation of log-function using c(5,5) as starting value

```r
mle.fit_b4 <- optim(
  par = c(5, 5),              # Starting values for the parameters
  fn = mvn_log,              # Function to minimize (log-likelihood)
  method = "BFGS",            # Optimization method (Broyden-Fletcher-Goldfarb-Shanno algorithm)
  control = list(
    trace = TRUE,             # Show optimization progress
    maxit = 1000,             # Maximum number of iterations
    fnscale = -1              # Negate the function to maximize instead of minimize
  ),
  hessian = TRUE              # Return the Hessian matrix
)
```

```
## initial  value 12.500000
## final  value -0.000000
## converged
```

```r
mle.fit_b4$par
```

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

When the c(1,0) was used as starting values, estimating maximum likelihood for both original functions and log function was successful since the starting point is close to the maximum. However, when c(5,5) was used as starting value, estimating maximum likelihood wasn't successful for the original function although it was successful for the log function.

In the original function, starting at c(5,5) places the optimizer start in the flat region and the optimizer struggles to find the direction of the maximum due minimal differences in slope of the function (gradient). On the other hand, the log function smooths the steep curvature and amplifies the gradients in flat regions without changing the maximum. Therefore, the optimizer was able to successfully navigate to the true maximum value even when the starting values was far.

3. The normal variance

3a.

Derive $\hat{\sigma}^2$, the MLE of the variance of the normal distribution. Hint: begin with the log-likelihood of the Normal distribution given in the text.

Log-likelihood function of the Normal distribution [(1.4) from textbook]:

$$-2\log L = n\log\sigma^2 + \frac{\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2}{\sigma^2}$$

Divide by -2 and take derivative: $\frac{\partial}{\partial\sigma^2}(-\frac{n}{2}\log\sigma^2 - \frac{\sum_{i=1}^{n}(y_i-\beta_0-\beta_1 x_i)^2}{2\sigma^2})$

Derivative of the first term: $\frac{\partial}{\partial \sigma^2}(-\frac{n}{2}\log\sigma^2) = -\frac{n}{2}\cdot\frac{1}{\sigma^2} = -\frac{n}{2\sigma^2}$

Derivative of the second term: $\frac{\partial}{\partial \sigma^2}(-\frac{\sum_{i=1}^{n}(y_i-\beta_0-\beta_1 x_i)^2}{2\sigma^2}) = -\frac{\sum_{i=1}^{n}(y_i-\beta_0-\beta_1 x_i)^2}{2}\cdot -\frac{1}{(\sigma^2)^2} = \frac{\sum_{i=1}^{n}(y_i-\beta_0-\beta_1 x_i)^2}{2(\sigma^2)^2}$

Answer:

$$\hat{\sigma}^2 = -\frac{n}{2\sigma^2} + \frac{\sum_{i=1}^{n}(y_i-\beta_0-\beta_1 x_i)^2}{2(\sigma^2)^2}$$

Use of ChatGPT and other generative AI tools

I certify that I use ChatGPT in this assignment for converting the mathematical expressions and equations to LaTex format in addition to asking for assistance in reporting computational time. Although it was quiet helpful with converting into LaTex format, it can be distracting as sometimes spit out the calculation of the equation without being prompted.

Link to Chat Thread

```
sessionInfo()
```

```
## R version 4.4.2 (2024-10-31)
## Platform: x86_64-apple-darwin20
## Running under: macOS Ventura 13.7.2
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.4-x86_64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.4-x86_64/Resources/lib/libRlapack.dylib;  LAPACK v
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/Los_Angeles
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] lattice_0.22-6 formatR_1.14
##
## loaded via a namespace (and not attached):
##  [1] compiler_4.4.2    fastmap_1.2.0     cli_3.6.3         tools_4.4.2
##  [5] htmltools_0.5.8.1 rstudioapi_0.17.1 yaml_2.3.10       rmarkdown_2.28
##  [9] highr_0.11        grid_4.4.2        knitr_1.48        xfun_0.48
## [13] digest_0.6.37     rlang_1.1.4       evaluate_1.0.1
```