1. (Random matrix). Use the following

```r
n <- 4
mu <- 10
sigma <- 2
A <- matrix(rnorm(n^2,mean=mu,sd=2),n,n)
```

to generate a random matrix.

(a) Based on the random matrix with \( n = 4 \) given above, calculate the determinant and the log determinant of \( A \) with the sign reported for the log determinant.

**Solution:** I generate the matrix. I use `det` to compute the determinant and `determinant` to compute the log determinant of \( A \). I have \( \text{det}(A) = 10696.05 \) and \( \log[\text{det}(A)] = 9.2776 \) (your answer may be different but they should be close to mine).

```r
> n <- 4
> mu <- 0
> sigma <- 3
> A <- matrix(rnorm(n^2,mean=mu,sd=2),n,n)
> diag(A) <- 10
> det(A)
[1] 10696.05
> determinant(A)
$modulus
[1] 9.277629
attr("logarithm")
[1] TRUE
$sign
[1] 1
attr("class")
[1] "det"
```

(b) Calculate SVD of \( A \) for the matrix your derive in part (a). Let the singular values be denoted by \( d_1, d_2, d_3, d_4 \), respectively. Compute \( \prod_{i=1}^{4} d_i, \sum_{i=1}^{4} \log(|d_i|) \), and \( \prod_{i=1}^{4} \text{sign}(d_i) \). Compare those with the answers in part (a).

**Solution:** I followed the instruction and found that \( \text{det}(A) = \prod_{i=1}^{4} d_i, \log[\text{det}(A)] = \sum_{i=1}^{4} \log(|d_i|) \), and the sign of \( \text{det}(A) \) is \( \prod_{i=1}^{4} \text{sign}(d_i) \).
(c) Choose \( n = 500 \) and also calculate the determinant and the log determinant of \( A \). Provide your findings.

**Solution:** I found that it is impossible to compute \( \det(A) \) by R because it is overflowed. We can still compute \( \log[\det(A)] \) (the sign may be either negative or positive in your answer, but that is fine).

```r
> n <- 500
> mu <- 0
> sigma <- 3
> A <- matrix(rnorm(n^2, mean=mu, sd=2),n,n)
> diag(A) <- 10
> det(A)
[1] -Inf
> determinant(A)
$modulus
[1] 1661.116
attr(,"logarithm")
[1] TRUE
$sign
[1] -1
attr(,"class")
[1] "det"
```

(d) Simply explain how to compute the log determinant of a large matrix by SVD without using the determinant value.

**Solution:** If the size of \( A \) is large, then we can compute SVD of \( A \). Then \( \log[\det(A)] \) is reported by \( \sum_{i=1}^{n} \log(|d_i|) \) and \( \prod_{i=1}^{n} \text{sign}(d_i) \), where \( d_1, \cdots, d_n \) are singular values reported by SVD. This method does not need the computation of the determinant value.

2. It is known that \( \mu = E(X) = \lambda \) and \( \sigma^2 = V(X) = \lambda \) if \( X \sim \text{Poisson}(\lambda) \). You can use
simulation to confirm the two formulae. In particular, for a selected $\lambda$, you can independently generate $n$ data from $Poisson(\lambda)$, denoted by $x_1, x_2, \ldots, x_n$, respectively. You then look the absolute difference between $\bar{x}$ and $\mu$, and the difference between $s^2$ and $\sigma^2$, respectively, and check whether they are close to 0. Please use this method to investigate the two formulae for $\lambda = 10, 20, 50, 100$ with $n = 10^4, 10^5, 10^6$, respectively. You need to list your result as a table created by R (you table should have 12 rows because there are $4 \times 3 = 12$ combinations of $\lambda$ and $n$). You also need to draw a conclusion.

**Solution:** For each combination ($\lambda, n$), I generated data from $Poisson(\lambda)$ and calculated the sample mean (mean) and the sample variance (s2). I then calculated the absolute difference between the sample mean and $\lambda$, and the difference between the sample variance and $\lambda$. I found all are close to 0. Therefore, I conclude that the two formulae are correct.

```r
> lambda.o <- c(10,20,50,100)
> n.o <- c(1e4,1e5,1e6)
> result <- matrix(0,length(lambda.o)*length(n.o),6)
> for(i in 1:length(lambda.o)){
+   for(j in 1:length(n.o)){
+     lambda <- lambda.o[i]
+     n <- n.o[j]
+     x <- rpois(n,lambda)
+     x.mean <- mean(x)
+     x.var <- var(x)
+     result[(i-1)*length(n.o)+j,] <- c(lambda,n,x.mean,
+       x.var,abs(x.mean-lambda),abs(x.var-lambda))
+   }
+ }
> dimnames(result)[[2]] <- c("lambda","n","mean","s2","d.mu","d.sigmasq")
> round(result,4)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>lambda</td>
<td>n</td>
<td>mean</td>
<td>s2</td>
<td>d.mu</td>
<td>d.sigmasq</td>
</tr>
<tr>
<td>[1,]</td>
<td>10</td>
<td>1e+04</td>
<td>10.0163</td>
<td>9.9714</td>
<td>0.0163</td>
</tr>
<tr>
<td>[2,]</td>
<td>10</td>
<td>1e+05</td>
<td>10.0103</td>
<td>10.0110</td>
<td>0.0110</td>
</tr>
<tr>
<td>[3,]</td>
<td>10</td>
<td>1e+06</td>
<td>10.0020</td>
<td>10.0123</td>
<td>0.0123</td>
</tr>
<tr>
<td>[4,]</td>
<td>20</td>
<td>1e+04</td>
<td>19.9704</td>
<td>19.7313</td>
<td>0.0296</td>
</tr>
<tr>
<td>[5,]</td>
<td>20</td>
<td>1e+05</td>
<td>20.0089</td>
<td>20.1021</td>
<td>0.1021</td>
</tr>
<tr>
<td>[6,]</td>
<td>20</td>
<td>1e+06</td>
<td>20.0032</td>
<td>19.9946</td>
<td>0.0032</td>
</tr>
<tr>
<td>[7,]</td>
<td>50</td>
<td>1e+04</td>
<td>49.8791</td>
<td>50.9496</td>
<td>0.1209</td>
</tr>
<tr>
<td>[8,]</td>
<td>50</td>
<td>1e+05</td>
<td>50.0066</td>
<td>50.4972</td>
<td>0.0066</td>
</tr>
<tr>
<td>[9,]</td>
<td>50</td>
<td>1e+06</td>
<td>49.9856</td>
<td>49.9464</td>
<td>0.0114</td>
</tr>
<tr>
<td>[10,]</td>
<td>100</td>
<td>1e+04</td>
<td>99.8518</td>
<td>99.2092</td>
<td>0.1482</td>
</tr>
<tr>
<td>[11,]</td>
<td>100</td>
<td>1e+05</td>
<td>99.9768</td>
<td>99.6703</td>
<td>0.0232</td>
</tr>
</tbody>
</table>

3
3. We can use R to numerically compute the (local) maximum of a function. In this example, we calculate the (local) maximizer of \( f(x) = -2x^8 + 4x^6 + 2x + 6 \). You can do it by:

(a) obtain an initial guess of the maximizer, denoted by \( x_0 \). Then, you update \( x_0 \) by \( x_0 \leftarrow x_0 - f'(x_0)/f''(x_0) \), which is called the Newton-Raphson method, where \( f'(x) \) is the derivative and \( f''(x) \) is the second-order derivative of \( f(x) \). To implement the Newton-Raphson method, you need (i) obtain \( f'(x) \) and \( f''(x) \) (ii) use the plot of \( f(x) \) for the derivation of the initial \( x_0 \). There are two maximizers of \( f(x) \). Please derive those and confirm that the difference between your answer and the true maximizer is not over 0.0001 in absolute values.

**Solution:** First, I obtain \( f'(x) \) and \( f''(x) \), and they are

\[
\begin{align*}
  f'(x) &= -16x^7 + 24x^5 + 2 \\
  f''(x) &= -112x^6 + 120x^4.
\end{align*}
\]

Then, I implement the Newton-Raphson method and obtain two maximizers. By the plot of \( f(x) \) for \( x \in [-1.5, 1.5] \). I find that the first maximizer is around \(-1.3\) and the second maximizer is around \(4.15\). I choose them as initial guesses of the maximize respectively. Then, I derive the two maximizers. The first solution is \( x_1 = -1.204444 \). The second solution is \( x_2 = -1.241899 \). To know whether the difference is not over 0.0001. I calculate \( f(x_1 \pm 0.0001) \) and \( f(x_2 \pm 0.0001) \). Both are lower than \( f(x_1) \) and \( f(x_2) \), respectively. Therefore, I conclude that the difference between my answer and the true maximizer is not over 0.0001 in absolute values.

```r
> f <- function(x){
+   SS <- -2*x^8+4*x^6+2*x+6
+   SS
+ }
> x <- seq(-1.5,1.5,0.01)
> y <- f(x)
> plot(x,y,type="l")
>
> df <- function(x){
+   SS <- -16*x^7+24*x^5+2
+   SS
+ }
> d2f <- function(x){
+   SS <- -112*x^6+120*x^4
+   SS
```
> x0 <- -1.3
> ff <- f(x0)
> ff1 <- df(x0)
> ff2 <- d2f(x0)
> x0 <- x0-ff1/ff2
> x0
[1] -1.232849
> ff
[1] 6.392622
> ff1
[1] 13.28731
> ff2
[1] -197.8706
> while(abs(ff1)>1e-6){
+ ff <- f(x0)
+ ff1 <- df(x0)
+ ff2 <- d2f(x0)
+ x0 <- x0-ff1/ff2
+ }
> x0
[1] -1.204444
> ff
[1] 6.945172
> ff1
[1] 9.237056e-14
> ff2
[1] -89.39179
> x1 <- x0-0.0001
> x2 <- x0+0.0001
> c(f(x1),f(x2))-f(x0)
> 
> x0 <- 1.5
> ff <- f(x0)
> ff1 <- df(x0)
> ff2 <- d2f(x0)
> x0 <- x0-ff1/ff2
> x0

5
while(abs(ff1)>1e-6){
+  ff <- f(x0)
+  ff1 <- df(x0)
+  ff2 <- d2f(x0)
+  x0 <- x0-ff1/ff2
+  }
}