ECON526: Problem Set 3

Jesse Perla

Name: _____

Student ID: _____

Instructions

- Ensure you modify the field above with your **name and student number above immediately**
- Modify directly and do not rename the file as Canvas will automatically append your name to it.
- Submit both the `.ipynb` and an exported `html` version of this file.
- See here for instructions.
- Do not change the filenames. Canvas automatically appends your name/student number to submitted files.

Setup

Feel free to use the following packages (and we have added a few convenience imports)

```
import numpy as np
import matplotlib.pyplot as plt
import scipy
from numpy.linalg import cond, matrix_rank, norm
from scipy.linalg import inv, solve, det, eig, lu, eigvals
from scipy.linalg import solve_triangular, eigvalsh, cholesky
```

Question 1

Question 1.1

Prove that if you have an eigendecomposition of $A = Q\Lambda Q^{-1}$, then $A \cdot A = A^2 = Q\Lambda^2 Q^{-1}$ where Λ^2 is the componentwise square of the eigenvalues. Hint: use the fact that the Q is orthonormal

Answer:

(double click to edit your answer)

Question 1.2

There is nothing special in part 1.1 about taking the square. Just as you would define the square root of a number x as a $x^{1/2}$ such that $x^{1/2} \times x^{1/2} = x$, we can do the same thing with a matrix square root $A^{1/2}$ is such that $A^{1/2} \cdot A^{1/2} = A$.

Use the eigendecomposition and take inspiration from part Q1.1 to find $A^{1/2}$ for the A given below. Verify that $A^{1/2} \cdot A^{1/2} \approx A$.

```
A = np.array([[2, 1], [1, 2]])
# modify here to calculate A^{1/2}, i.e. where A^{1/2} A^{1/2} = A
```

Question 1.3

Consider that both $(-2) \times (-2) = 2 \times 2 = 4$. There may be multiple square roots of a matrix, and many matrices do not have a square root. For example, For example, the matrix $A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ does not have a square root.

Take this matrix and mention a property of it that suggests why this matrix looks suspicious and does not have a unique square root or why the method might fail

```
A = np.array([[0, 1], [0, 0]])
# modify here, add comments and an explanation on what the problems here would be
```

Question 1.4

Argue that if a matrix is symmetric positive definite then its square root will also be symmetric, positive definite.

Answer:

(double click to edit your answer)

Question 2

Question 2.1

Take a random variable with the multivariate normal distribution

$$Y \sim N(\mu, \Sigma)$$

with $\mu \equiv \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ and $\Sigma \equiv \begin{bmatrix} 0.1 & 0.05 \\ 0.05 & 0.1 \end{bmatrix}$

Take 10,000 draws using this distribution. Hint: See `np.random.multivariate_normal` in the lecture notes.

```
N = 10000
# modify here
# mu = ...
# Sigma = ...
# Y_draws = ...
```

Question 2.2

Take the `Y_draws` above, calculate the mean and covariance matrix, and compare them to μ and Σ to see how close they are. Hint: use `np.cov(Y_draws, rowvar=False)` to get the covariance matrix. What is a natural way to compare how close matrices or vectors are?

```
# modify code here
# make sure to state how you are comparing vectors/matrices
```

Answer:

(double click to add an explanation here)

Question 2.3

A key property of Gaussian random variables is that for any $Y \sim N(\mu, \Sigma)$ we can write it as a function of a unit multivariate normal $X \sim N(0, I)$ and a matrix A such that $Y = \mu + CX$. This is called the Cholesky decomposition of Σ.

1. Find the Cholesky of the covariance matrix Σ, C. You can use upper or lower Cholesky, but be consistent
2. Draw the same N number of draws from $X \sim N(0, I)$ as the previous parts of Q2
3. Transform those draws Y
4. Compare the mean and covariance of those transformed Y to μ and Σ to see how well the transformation did

```
# modify code here
```

Question 2.4

The same transformation works with the matrix square root. That is, if $Y \sim N(\mu, \Sigma)$, then $Y = \mu + \Sigma^{1/2}X$ where $X \sim N(0, I)$ where $\Sigma^{1/2}$ is the matrix square root of Σ.

Repeat the Q2.3 using this transformation instead. That is,

1. Find the square root of the covariance matrix Σ, $\Sigma^{1/2}$. Hint: You can do this yourself using previous sections, use the `scipy` function `scipy.linalg.sqrtm` or use the scipy function `scipy.linalg.matrix_power` with the argument `0.5`.
2. Draw the same N number of draws from $X \sim N(0, I)$ as the previous parts of Q2
3. Transform those draws to form Y samples
4. Compare the mean and covariance of those transformed Y to μ and Σ to see how well the transformation did

```
# modify code here
```

Question 3

Question 3.1

Take a new mean μ and covariance matrix, Σ. You want to do the same thing as previous examples (i.e., find a matrix, A such that $Y = \mu + AX$ for $X \sim N(0, I)$).

Take the matrix below, its eigenvalues, and the results of calling Cholesky. Explain why you probably shouldn't be using it in this case.

```
mu = np.array([0, 1, 1])
Sigma = np.array([
    [26, 21, 6],
    [21, 17, 5],
    [6, 5, 2]
]) / 30.0
C = cholesky(Sigma, lower=True)
print(C)
print(f"Eigenvalues(Sigma) = {eigvalsh(Sigma)}")
```

```
[[9.30949336e-01 0.00000000e+00 0.00000000e+00]
 [7.51920618e-01 3.58057437e-02 0.00000000e+00]
 [2.14834462e-01 1.43222975e-01 5.75915904e-08]]
Eigenvalues(Sigma) = [-4.09003431e-17  1.95131000e-02  1.48048690e+00]
```

Answer:

(double click to add an explanation here)

Question 3.2

The Cholesky is suspicious, but what about the matrix square root?

```
Sigma_sqrt = scipy.linalg.sqrtm(Sigma)
print(f"Sigma^-1 = \n{Sigma_sqrt}")
print(f"Sigma^-1 real only = \n{np.real(Sigma_sqrt)}")
```

```
Sigma^-1 =
[[0.7208559 +4.03468980e-09j 0.57388853-5.37958640e-09j
  0.13298645+1.34489660e-09j]
 [0.57388853-5.37958640e-09j 0.4659534 +7.17278187e-09j
  0.14214799-1.79319547e-09j]
 [0.13298645+1.34489660e-09j 0.14214799-1.79319547e-09j
  0.16963261+4.48298867e-10j]]
Sigma^-1 real only =
[[0.7208559  0.57388853 0.13298645]
 [0.57388853 0.4659534  0.14214799]
 [0.13298645 0.14214799 0.16963261]]
```

Any comments on whether this would be a reasonable way to transform the random variable into a $Y = \mu + \Sigma^{-1}X$ for $X \sim N(0, I)$?

Answer:

(double click to add an explanation here)

Question 3.3

Your friend suggests trying an SVD of the covariance matrix, which is always defined. Here are the results for $USV^T = \Sigma$,

```
U, S, Vh = scipy.linalg.svd(Sigma) # returns V' not V
print(f"U =\n {U}")
print(f"S =\n {S}")
print(f"V' =\n {Vh}")
```

```
U =
 [[-0.76451208  0.26337697  0.58834841]
 [-0.61865349 -0.04339637 -0.78446454]
 [-0.18107771 -0.96371641  0.19611614]]
S =
 [1.48048690e+00 1.95131000e-02 8.63766899e-18]
V' =
 [[-0.76451208 -0.61865349 -0.18107771]
 [ 0.26337697 -0.04339637 -0.96371641]
 [-0.58834841  0.78446454 -0.19611614]]
```

Your friend then shows you the following

```
U_truncated = U[:, :2] # truncating to only use the first two columns
S_truncated = np.diag(np.sqrt(S[:2])) # first two singular values
A = U_truncated @ S_truncated # using first two singular values
print(f"shape(A) = {A.shape}")
print(f"A =\n{A}")
print(f"A @ A.T =\n{A @ A.T}")
print(f"||A A' - Sigma|| = {norm(A @ A.T - Sigma)}")
```

```
shape(A) = (3, 2)
A =
[[-0.93022207  0.03679094]
 [-0.75274824 -0.00606201]
 [-0.22032678 -0.13462087]]
```

```
A @ A.T =
[[0.86666667 0.7        0.2       ]
 [0.7        0.56666667 0.16666667]
 [0.2        0.16666667 0.06666667]]
||A A' - Sigma|| = 9.899056296961976e-16
```

Can you explain why this is a great observation and how it might be useful for transforming the random variable $Y = \mu + A\hat{X}$ for some unit normal $\hat{X}$?

Answer:

(double click to add an explanation here)

Question 3.4

Finally, your friend uses this to generate a bunch of random values to check if the random variable works relative to the original and compares the sqrt version to the new one (Ignore any warnings here about diving by zero/etc.)

```
N = 100000
Y_draws = np.random.multivariate_normal(mu,Sigma, size=N) # unit normal
X_draws = np.random.multivariate_normal(np.zeros(3), np.eye(3), size=N) # unit normal
Y_draws_sqrt = mu + X_draws @ Sigma_sqrt.T # sqrt version
X_hat_draws = np.random.multivariate_normal(np.zeros(2), np.eye(2), size=N) # unit normal
Y_draws_hat = mu + X_hat_draws @ A.T # Using the A from the SVD
print(f"||mean(Y_draws) - mu|| = {norm(np.mean(Y_draws, axis=0) - mu)}")
print(f"||mean(Y_draws_sqrt) - mu|| = {norm(np.mean(Y_draws_sqrt, axis=0) - mu)}")
print(f"||mean(Y_draws_hat) - mu|| = {norm(np.mean(Y_draws_hat, axis=0) - mu)}")
print(f"||cov(Y_draws) - Sigma|| = {norm(np.cov(Y_draws, rowvar=False) - Sigma)}")
print(f"||cov(Y_draws_sqrt) - Sigma|| = {norm(np.cov(Y_draws_sqrt, rowvar=False) - Sigma)}")
print(f"||cov(Y_draws_hat) - Sigma|| = {norm(np.cov(Y_draws_hat, rowvar=False) - Sigma)}")
```

```
||mean(Y_draws) - mu|| = 0.001966352968676589
||mean(Y_draws_sqrt) - mu|| = 0.001211045001229686
||mean(Y_draws_hat) - mu|| = 0.0008009724714110563
||cov(Y_draws) - Sigma|| = 0.0003591056637926922
||cov(Y_draws_sqrt) - Sigma|| = 0.010943840626177693
||cov(Y_draws_hat) - Sigma|| = 0.0008272157071567913
```

This seems to have worked great. In fact, simulating with the new processes seems closer to the true mean/variance than directly using the Y but your friend is confused. When they simulated `Y_draws` and `Y_draws_sqrt` they were drawing 3-dimensional random normals, but when they did `Y_draws_hat` it was only 2-dimensional. How can this be and still deliver the same results? Explain it to your friend.

Answer:

(double click to add an explanation here)