DSCI 6607– Fall 2024

Assignment 1*

```
# Abdallah Chidjou
# Citation: (source of help: googling in general, stackoverflow, and chatgpt)
```

Question 1

Write a python script which simulates tossing an unfair coin where the probability of head is 0.63. Toss the coin 301 times and compute

$$\widehat{p} = \frac{\text{Number of Heads}}{\text{Total number of trials}}$$

Explain your solution. [**10 points**]

```
import random
# The following tossing function takes in two parameters
# (num: number of time the coin is tossed) and
# (pb_head: The probability of getting a head in a single toss)
# and returns the proportion of heads observed
# by dividing the total count of heads by the total number of tosses.

def toss_func(num, pb_head):
    count = 0
    for i in range(num):
        if random.random() < pb_head:
            count += 1
    return count / num

# Number of toss and the probability of head
num1 = 301
pb_head1 = 0.63

# coin simulation
simulation = toss_func(num1, pb_head1)

print(f"Proportion of Heads: {simulation:.4f}")
```

```
## Proportion of Heads: 0.6279
```

*This content is protected and may not be shared, uploaded, or distributed without written permission from Dr. Armin Hatefi.

```
# This code simulates the tossing of a biased coin
#multiple times to estimate the proportion of heads
```

Question 2

We would like to learn an introduction to Bisection root finding method in python. The bisection method finds the root of the function $f(x)$ in interval $[a, b]$ when there is a unique root in the interval and $f(a) > 0$ and $f(b) < 0$.
The method is computed the root iteratively by

- Compute $m = \frac{a+b}{2}$,
- then a is replaced by m if $f(m) > 0$, otherwise b is replaced by m if $f(m) < 0$. Then the method computes the mean of the new points and search for the root. Th method iteratively takes the above steps until it finds the root of the function.

a) Write a python function which takes a, b, f and computes the root of the f function using the bisection method. The method stops when $|f(m)| < 10^{-5}$.

b) Apply your python function from part (a) and find the root of $f(x) = x^3 - 4x^2 - 3$ in interval $[-2, 0]$. Explain your solutions. [**10 points**]

```
# The following bisection function find the root of
# the function f(x) in the interval [a,b]
# where there is an unique root in the
# interval and f(a) > 0 and f(b) < 0

# (a) Write a python function which takes a, b, f and computes
# the root of the f function using the bisection method.
# The method stops when f(m) < 10^-5.
def bisection_func( a, b, f):
    # Ensure that f(a) and f(b) have opposite signs
    if f(a) * f(b) >= 0:
        print("Bisection method fails. f(a) and f(b) must have opposite signs.")
        return None

    # While loop to apply the bisection method
    while (b - a) > 1e-5:
        # Compute the midpoint
        m = (a + b) / 2.0

        # Evaluate the function at the midpoint
        fm = f(m)

        # Check if we found the root
        if abs(fm) < 1e-5:
            return m

        # Decide the next interval
        if f(a) * fm < 0:
            b = m  # Root is in the left half
        else:
            a = m  # Root is in the right half

    # Return the midpoint as the best estimate of the root
    return (a + b) / 2.0

# Define the function f(x) = x^3 - 4x^2 - 3
def f(x):
    return x**3 + 4*x**2 - 3

# (b) Apply the Bisection Method to Find the Root
a = -2
b = 0
root = bisection_func(a, b, f)

print(f"The root found is: {root}")
```

```
## The root found is: -1.0
```

```
# The output of the above code will give the approximate
# root of the function within the given interval.
```

Question 3

Central Limit Theorem emphasizes that the sample mean convergences to population mean as the sample size n is large enough.

a) Write an R function which takes sample **x** and plots the cumulative sample means. Note that the cumulative means are given by a vector of size n including: the mean of the first observation, the mean of the first two observations, ... and finally the mean of the n observations.

b) Generate $n = 1000$ observations with replacement from 'rivers' data set.

c) Apply your function from part (a) to the data of part (b).

d) Show the population mean (the mean of the 'rivers' data set) in your plot in part (c). What do you observe in your plot? Can you intuitively argue how many samples one requires for the convergence of the sample mean? If needed, increase the sample size n. Explain your solutions. [**10 points**]

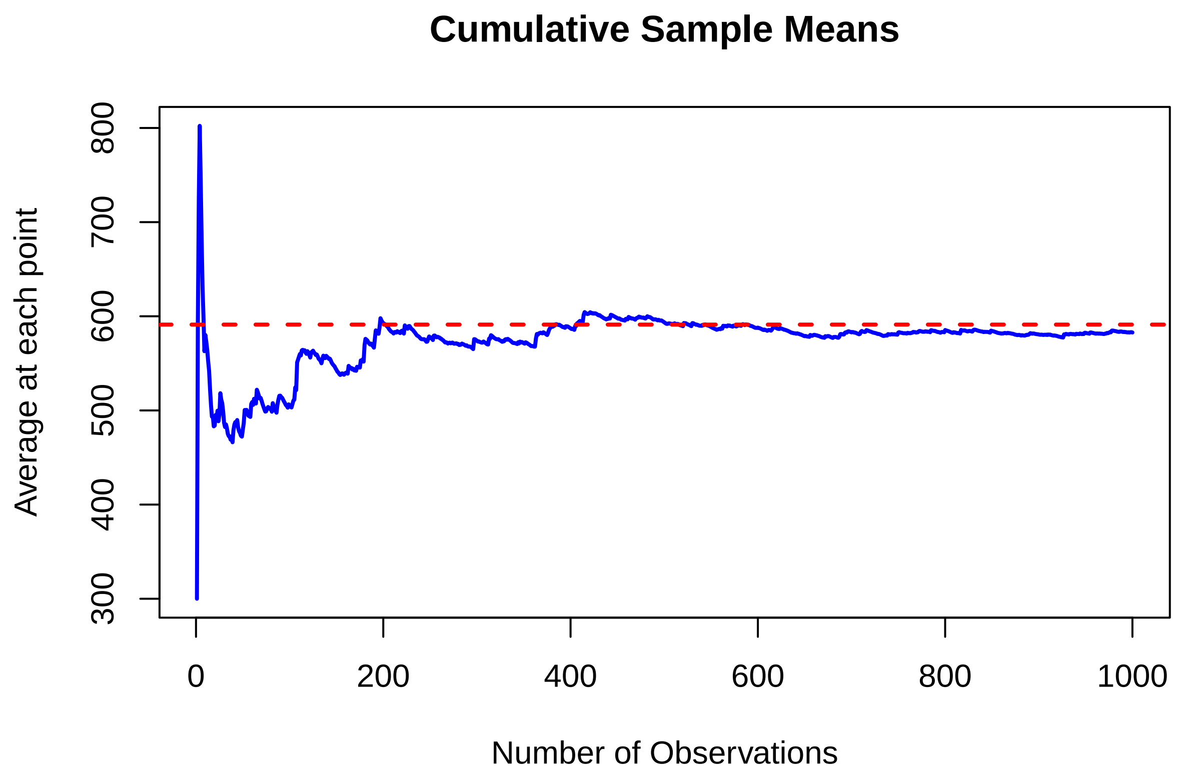
```
data("rivers")
head(rivers)
```

```
## [1] 735 320 325 392 524 450
```

```r
# (a)

# The following function takes sample x and plots the cumulative sample means
cumulative_func <- function(x) {
  # Compute cumulative means using the function cumsum and seq_along
  cum_means <- cumsum(x) / seq_along(x)

  # Plot the cumulative means for visualisation
  # The x-axis represents the number of observations,
  # while the y-axis shows the running average at each point,
  # giving you an idea of how the mean changes over time.
  plot(cum_means, type = 'l', col = 'blue', lwd = 2,
       xlab = 'Number of Observations',
       ylab = 'Average at each point',
       main = 'Cumulative Sample Means')
}
# Load the rivers dataset
data("rivers")
head(rivers)
```

```
## [1] 735 320 325 392 524 450
```

```r
# (b) Generate n = 1000 observations with replacement from `rivers` data set.
sample_data <- sample(rivers, size = 1000, replace = TRUE)

# Apply the function to the sample data
cumulative_means <- cumulative_func(sample_data)

# Calculate the population mean of the rivers dataset
population_mean <- mean(rivers)

# Add the population mean line to the plot
abline(h = population_mean, col = 'red', lwd = 2, lty = 2)
```



```
# The blue line represents the cumulative mean of the sample.
# Initially, the cumulative mean fluctuates heavily, indicating significant
# variation when only a few observations are considered.
# This happens because there isn't enough data for the mean to stabilize.

# As the number of observations increases, the cumulative mean gradually
# stabilizes and converges to a value close to the red dashed line,
# which represents the population mean.

# This visualization effectively demonstrates that, over time,
# the sample mean converges to the population mean as the sample size increases
```

Question 4

The `airquality` data set reports the daily air quality measurements in New York, May to September 1973. The data set includes 153 observations and 6 variables.[**10 points**]

a) In Ridge regression analysis, the coefficients of the regression model

$$y = \beta_1 x_1 + \ldots + \beta_p x_p, \tag{1}$$

are estimated by

$$\widehat{\beta}_R = \left(\mathbf{X}^\top \mathbf{X} - \lambda I\right)^{-1} \mathbf{X}^\top \mathbf{y} \tag{2}$$

where $\mathbf{X}$ is $n \times p$ design matrix (i.e., n observations with p columns) and $\mathbf{y}$ is the response vector of size n, λ is the Ridge parameter and I is the identity matrix of size $p \times p$.

b) Let 'Ozone' be the response variable and design matrix includes 'Solar.R, Wind, Temp' variables where $p = 3$. First normalize all the variables including the explanatory variables and response variable. The normalized version of variable x is computed by

$$z_x = \frac{x - \bar{x}}{\sigma_x},$$

where $\bar{x}$ and σ_x denote the mean and standard deviation of x, respectively.

c) Write an R script that computes $\widehat{\beta}_R$ of the model as described by (2) for a given $\lambda = 0.1$. Hint: You have to estimate them via the matrix computation.

d) Consider the response and three explanatory variables from part (a). Display the box plot of the variables separately. Then explain which measure (mean vs median) should be used to describe the center of the variables. Explain your solutions.

```r
# (a)-
# Load the airquality dataset
data("airquality")

head(airquality)
```

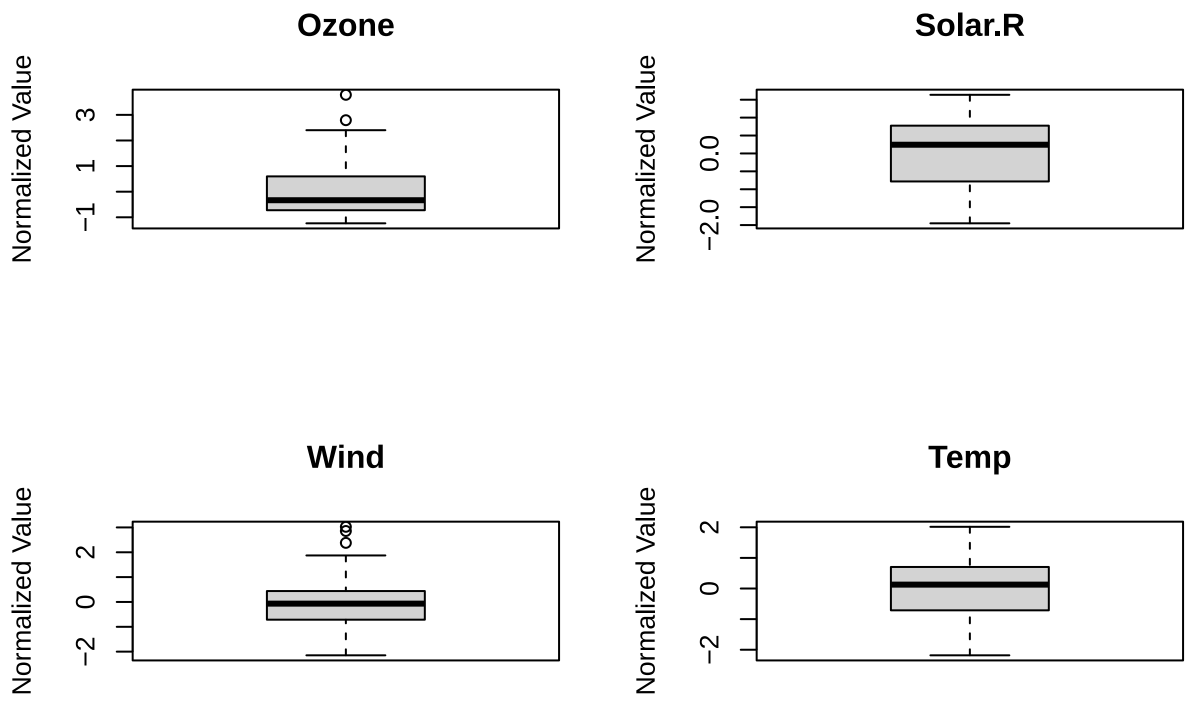
```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

```r
# Remove rows with NA values to make normalization easier
airquality <- na.omit(airquality)

# b- Normalizing the variables
normalize <- function(x) {
  return((x - mean(x)) / sd(x))
}

# Normalize the response variable (Ozone) and predictors (Solar.R, Wind, Temp)
airquality$Ozone <- normalize(airquality$Ozone)
airquality$Solar.R <- normalize(airquality$Solar.R)
airquality$Wind <- normalize(airquality$Wind)
airquality$Temp <- normalize(airquality$Temp)

# Design matrix for predictors, adding a column of ones for the intercept
X <- as.matrix(cbind(1, airquality[, c("Solar.R", "Wind", "Temp")]))
```

```r
# Response vector
y <- as.matrix(airquality$Ozone)

# (c)- Implementing Ridge Regression
ridge_regression <- function(X, y, lambda) {
  # Number of predictors (p)
  p <- ncol(X)

  # Identity matrix of size p
  I <- diag(p)

  # Ridge regression calculation
  beta_hat_R <- solve(t(X) %*% X + lambda * I) %*% t(X) %*% y
  return(beta_hat_R)
}

# Ridge parameter lambda
lambda <- 0.1

# Compute ridge regression coefficients
beta_hat_R <- ridge_regression(X, y, lambda)
print(beta_hat_R)
```

```
##                  [,1]
## 1         1.908196e-17
## Solar.R   1.638378e-01
## Wind     -3.562652e-01
## Temp      4.727975e-01
```

```r
# (d)- Display box plots of the variables
par(mfrow = c(2, 2)) # Set up a 2x2 plotting area
boxplot(airquality$Ozone, main = "Ozone", ylab = "Normalized Value")
boxplot(airquality$Solar.R, main = "Solar.R", ylab = "Normalized Value")
boxplot(airquality$Wind, main = "Wind", ylab = "Normalized Value")
boxplot(airquality$Temp, main = "Temp", ylab = "Normalized Value")
```



```
# Ozone and Wind have outliers, and their distributions are slightly skewed,
# so the median would be the better measure to describe the center of these variables.
# Solar.R and Temp are more symmetric without significant outliers, so the mean would
# be appropriate for describing their central tendency.
```

Question 5

In this question, we plan to learn an introduction to finding the optimum tuning regularization terms. Consider the estimation of the coefficients of the regression in Question 4. [**10 points**]

a) Take a sample of size 100 observations from the ‘airquality‘ data set. Consider this data set henceforth your population. Similar to Question 4, consider the ‘Ozone‘ as your response and your design matrix includes ‘Solar.R, Wind, Temp‘ variables. Divide the population into 2 folds. The training data of size 70 and testing data of size 30. Note that the folds are mutually exclusive such that there is no common observation between them.

b) Use the training fold to estimate the coefficients of the regression from (2) and obtain $\widehat{\beta}_R$. Then predict the response value of test data by

$$\widehat{y} = \widehat{\beta}_R x_{new}$$

where x_{new} is the covariates of the test fold $\widehat{y}$ denotes the predicted responses of the test fold. Finally compute

$$\sqrt{\text{MSE}} = \sqrt{\sum_{i=1}^{30}(y_{i,test} - \widehat{y}_{i,test})^2/100}$$

c) Now write a function which uses the above function and computes the root MSEs for 100 equally spaced selected points for $\lambda \in [-2, 2]$. Find the λ which gives the minimum root MSE? Explain your solutions.

```r
# Load the dataset and remove missing values
data("airquality")
airquality <- na.omit(airquality)

# Set seed for reproducibility
set.seed(42)

# (a)- Take a random sample of 100 observations
rand_sample <- airquality[sample(1:nrow(airquality), 100, replace = FALSE), ]

# Split data into training (70) and testing (30)
train_data <- rand_sample[1:70, ]
test_data <- rand_sample[71:100, ]

# Define response and predictors for training and testing
y_train <- train_data$Ozone
X_train <- as.matrix(cbind(1, train_data[, c("Solar.R", "Wind", "Temp")]))
y_test <- test_data$Ozone
X_test <- as.matrix(cbind(1, test_data[, c("Solar.R", "Wind", "Temp")]))

# (b)- Estimate Coefficients and Predict Test Data
# Ridge Regression Function
ridge_regression <- function(X, y, lambda) {
  # Number of predictors (p)
  p <- ncol(X)

  # Identity matrix of size p
  I <- diag(p)

  # Ridge regression calculation
  beta_hat_R <- solve(t(X) %*% X + lambda * I) %*% t(X) %*% y
  return(beta_hat_R)
}

# Define lambda
lambda <- 0.1

# Estimate coefficients using the training data
beta_hat_R <- ridge_regression(X_train, y_train, lambda)

# Predict the response for the testing data
y_hat_test <- X_test %*% beta_hat_R

# Compute RMSE for the testing data
compute_rmse <- function(y_true, y_pred) {
  sqrt(sum((y_true - y_pred)^2) / length(y_true))
}

# Calculate RMSE for the test set
rmse <- compute_rmse(y_test, y_hat_test)
print(rmse)
```

```
## [1] 18.10227
```

```r
# (c)- Find the $\lambda$ which gives the minimum root MSE
# Define a function to compute RMSE for a given lambda
compute_lambda_rmse <- function(lambda, X_train, y_train, X_test, y_test) {
  beta_hat_R <- ridge_regression(X_train, y_train, lambda)
  y_hat_test <- X_test %*% beta_hat_R
  rmse <- compute_rmse(y_test, y_hat_test)
  return(rmse)
}

# Generate 100 equally spaced lambda values between -2 and 2
lambda_values <- seq(-2, 2, length.out = 100)

# Calculate RMSE for each lambda value
rmse_values <- sapply(lambda_values, function(lambda) {
  compute_lambda_rmse(lambda, X_train, y_train, X_test, y_test)
})

# Find the lambda that gives the minimum RMSE
lambda_min <- lambda_values[which.min(rmse_values)]
min_rmse <- min(rmse_values)

# Print the optimal lambda and corresponding RMSE
print(paste("lambda that gives the minimum RMSE:", lambda_min))
```

```
## [1] "lambda that gives the minimum RMSE: 0.0202020202020203"
```

```r
print(paste("Minimum RMSE:", min_rmse))
```

```
## [1] "Minimum RMSE: 18.0834565086884"
```

```r
# We sampled 100 observations from the airquality dataset
# and split into training and testing sets.
# Estimated Ridge regression coefficients and computed RMSE for the test set.
# Evaluated RMSE over 100 lambda values from -2 to 2
# and found the lambda with the lowest RMSE.
# We are basicaly comparing the performance of RMSE over different values of lambda
```

Question 6

We would like to learn an introduction to Box-Muller transformation. The Box-Muller method simulates data form standard normal distribution from random numbers between $[0, 1]$.

Let u_1 and u_2 are two numbers generated randomly between $[0, 1]$, then x and y as

$$x = \sqrt{-2 * \log(u_1)} \cos(2\pi u_2),$$
$$y = \sqrt{-2 * \log(u_1)} \sin(2\pi u_2)$$

follow from standard normal distribution.

a) Write a python function which takes u_1, u_2 and simulates x and y.

b) Apply your python function from part (a) and generates 1000 observations from standard normal distribution.

c) Plot the histogram of the generated observations form part (b). Compare the histogram with histogram of the true observations from standard normal distribution. Explain your solutions. [**10 points**]

```
import numpy as np
import matplotlib.pyplot as plt

# (a)- Write a python function which takes u1, u2 and simulates x and y
# Box-Muller transformation function
def box_muller(u1, u2):
    # Apply Box-Muller transformation
    x = np.sqrt(-2 * np.log(u1)) * np.cos(2 * np.pi * u2)
    y = np.sqrt(-2 * np.log(u1)) * np.sin(2 * np.pi * u2)
    return x, y


# b- Generate 1000 observations using the Box-Muller function
n = 500  # Number of pairs to generate
u1 = np.random.uniform(0, 1, n)
u2 = np.random.uniform(0, 1, n)

# Apply Box-Muller transformation to each pair
x_values = []
y_values = []
for i in range(n):
    x, y = box_muller(u1[i], u2[i])
    x_values.append(x)
    y_values.append(y)

# Combine the generated x and y values into one list
generated_data = np.array(x_values + y_values)

print()
```

```
print("The total count of elements in the array:",generated_data.size)
```

```
## The total count of elements in the array: 1000
```

```
print()
```

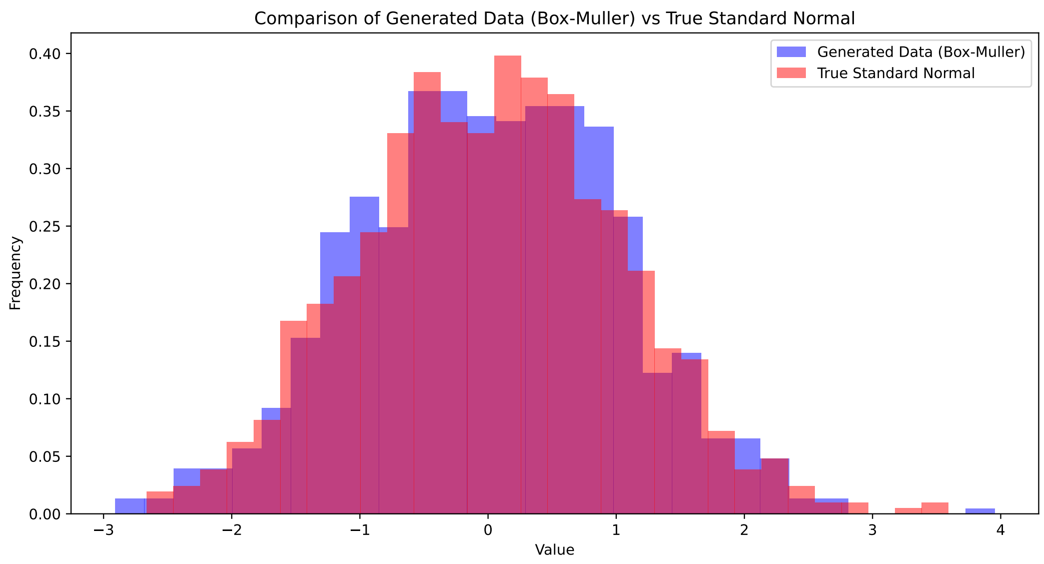
```
print("The first 5 elements of the array:",generated_data[:5])
```

```
## The first 5 elements of the array: [ 0.48686939 -0.4782106  -1.9678229   0.17460553 -0.1349087 ]
```

```
print()
```

```
print()
```

```
# (c)- Plot the histogram of the generated data
# and compare with true standard normal
plt.figure(figsize=(12, 6))

# Histogram of the generated data
plt.hist(generated_data, bins=30, density=True, alpha=0.5, color='blue',
label='Generated Data (Box-Muller)')

# Generate data from standard normal distribution
true_normal_data = np.random.normal(0, 1, 1000)

# Histogram of the true standard normal data
plt.hist(true_normal_data, bins=30, density=True, alpha=0.5,
color='red', label='True Standard Normal')

# Labels and title
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Comparison of Generated Data (Box-Muller) vs True Standard Normal')
plt.legend()
plt.show()
```



```
# The Box-Muller transformation is a method to generate normally
# distributed random variables from uniformly distributed random variables.
# The generated histogram (in blue) should closely resemble the histogram of the standard
# normal distribution (in red). This similarity shows that the Box-Muller transformation
# effectively produces data that follows the standard normal distribution.
# The overlay of the two histograms allows
# to visually and verify that the generated data matches
# the properties of a standard normal distribution-centred
# around 0 with a standard deviation of 1.
# Minor differences may occur due to random
# sampling but should generally be close.
```

Due on Friday, October 15, by 3 pm

Have fun!