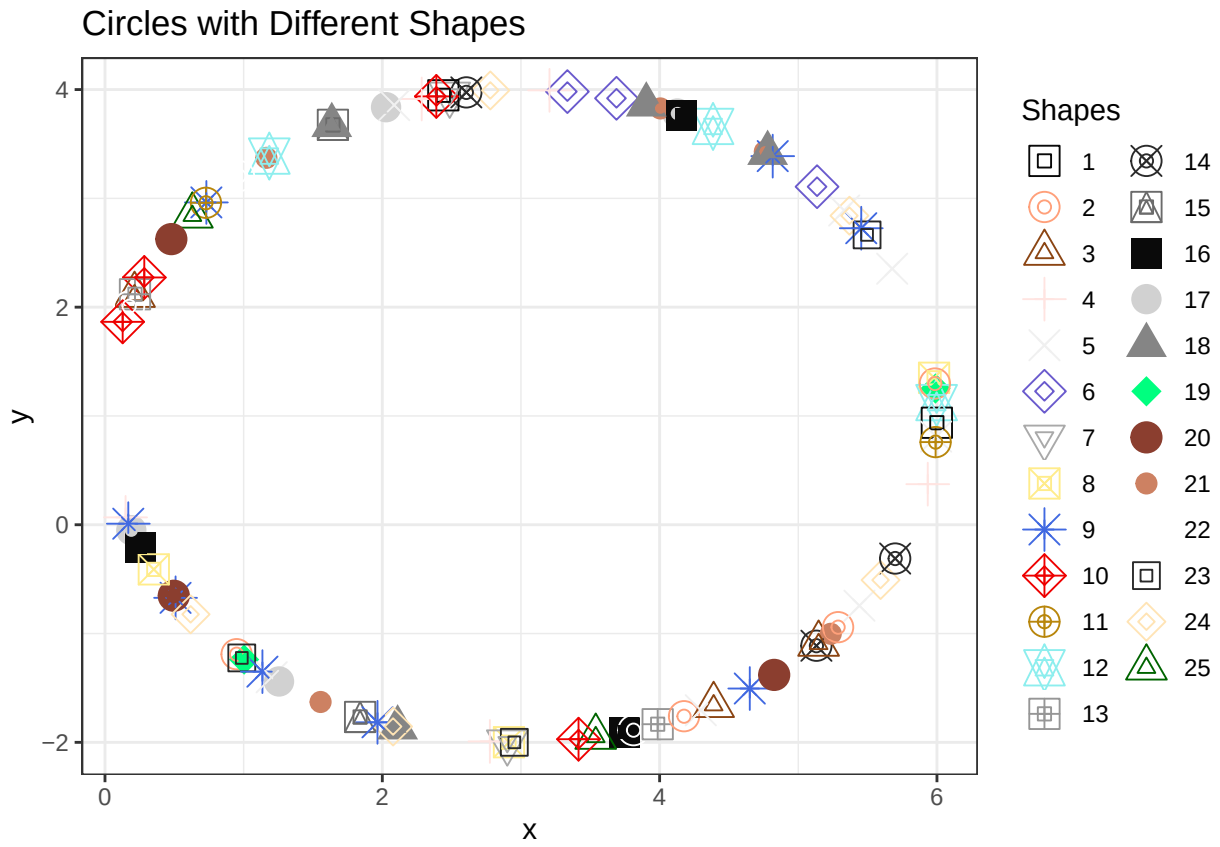
HW1_Lili_Kostanyan

2023-09-26

PROBLEM 1

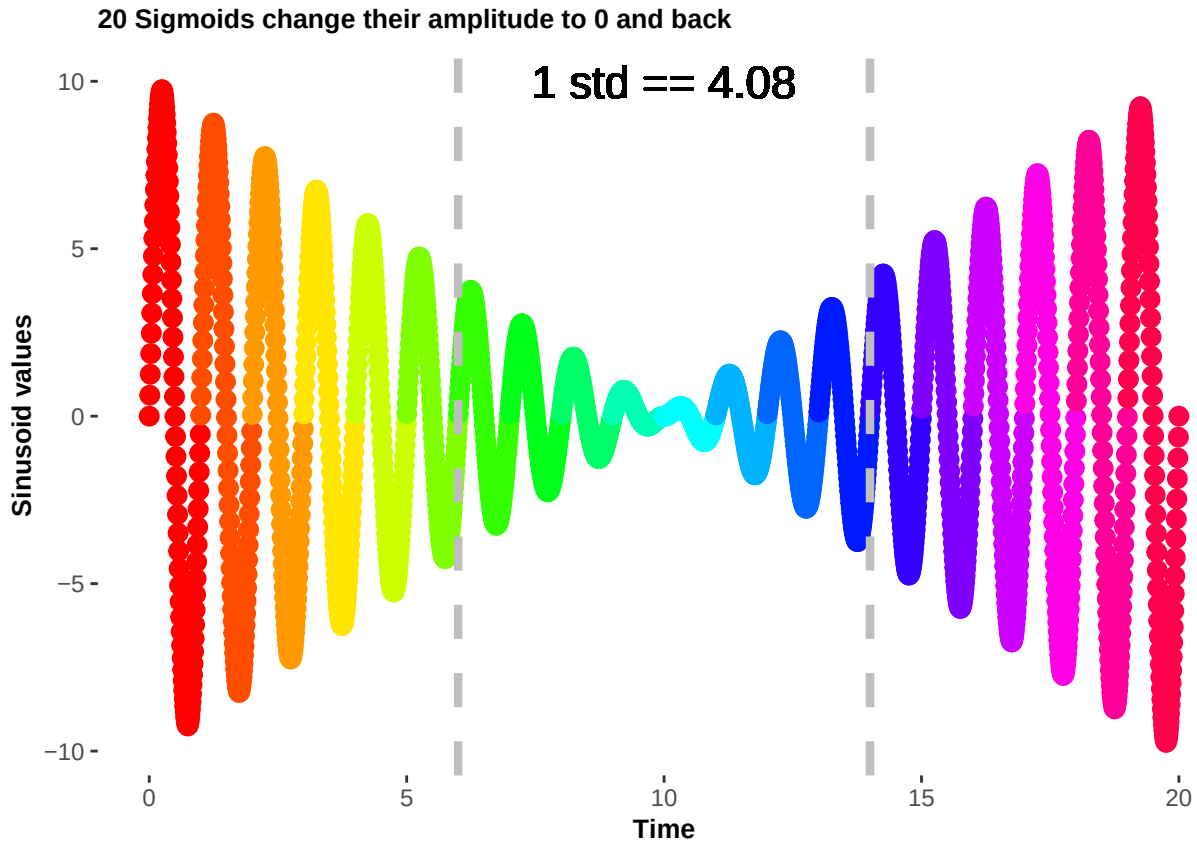
```
library(ggplot2)
# Set a global seed for reproducibility
set.seed(2023)

# Number of data points (N) and number of classes (M)
N <- 100
M <- 25  # Change M to 25 for 25 shape types

# Generate data for each class
data_list <- lapply(1:M, function(class_id) {

  # Generate random angles for the data points in the current class
  angles <- runif(N/M, (class_id - 1) * (2 * pi / M), class_id * (2 * pi / M))
  angles
  # Calculate x and y coordinates to form a circle
  x <- 3 + sqrt(9) * cos(angles)
  y <- 1 + sqrt(9) * sin(angles)

  # Create a data frame for the current class
  class_data <- data.frame(
    x = x,
    y = y,
    shape_type = sample(1:M, N/M, replace = TRUE)  # Add shape type column
  )
  return(class_data)
})

circle_data <- do.call(rbind, data_list)
circle_data_filtered <- circle_data[!(abs(circle_data$x - 0) < 0.1 | abs(circle_data$x - 5.8) < 0.1), ]

# Define colors for each shape type
shape_colors <- sample(colors(), M)

# Create ggplot with manually specified shapes and double shapes
ggplot(circle_data_filtered, aes(x = x, y = y, shape = factor(shape_type), color = factor(shape_type))) +
  geom_point(aes(shape = factor(shape_type), color = factor(shape_type)), size = 5) +
  geom_point(aes(shape = factor(shape_type), color = factor(shape_type)), size = 2) +
  scale_shape_manual(values = 0:M, name = "Shapes") +
  scale_color_manual(values = shape_colors, name = "Shapes") +
  theme_bw() +
  labs(x = "x", y = "y") +
  ggtitle("Circles with Different Shapes")
```



PROBLEM 2

```r
# Load the ggplot2 library
library(ggplot2)

n <- 2000
x <- seq(0, 20, length.out = n)

# Initialize an empty vector to store y values
y <- numeric(0)

# Initialize an amplitude variable
amplitude <- 10

# Initialize a step variable for the amplitude change
step <- 10 / (n / 2)

# Iterate through each x value
for (i in 1:length(x)) {
  # Calculate the corresponding y value for the sinusoidal function with the given amplitude
  y_value <- amplitude * sin(2 * pi * x[i])

  # Append the y value to the vector
  y <- c(y, y_value)

  # Adjust the amplitude based on the pattern
  if (i <= n / 2) {
```

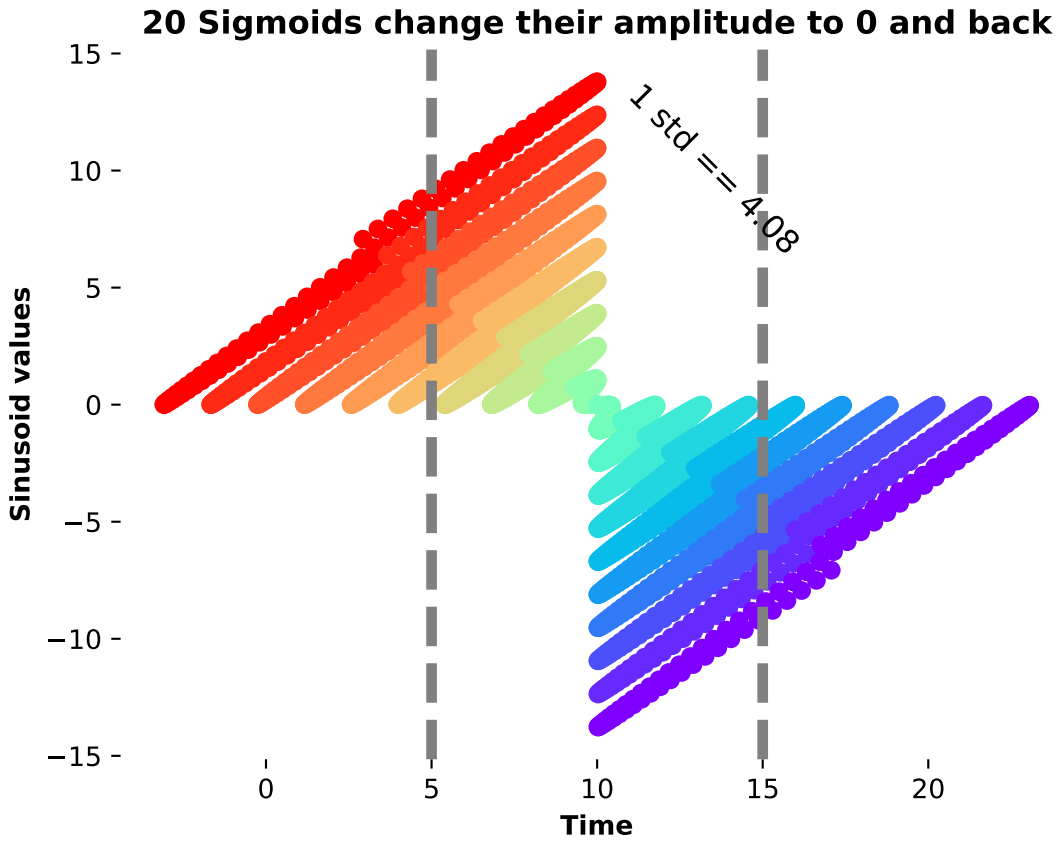
```
    amplitude <- amplitude - step
  } else {
    amplitude <- amplitude + step
  }
}

# Create a data frame with x and y values
sin_coordinates <- data.frame(
  x_values = x,
  y_values = y,
  color_groups = rep(1:(length(x) / 100), each = 100)
)

# Create a gradient color palette with rainbow colors
num_colors <- 20
palette <- rainbow(num_colors)

# Create the ggplot for the sigmoid plot
ggplot(sin_coordinates, aes(x = x_values, y = y_values, color = as.factor(color_groups))) +
  labs(
    title = '20 Sigmoids change their amplitude to 0 and back',
    x = 'Time',
    y = 'Sinusoid values '
  ) +
  geom_point(size = 3) +
  geom_text(aes(x = 10, y = 10, label = "1 std == 4.08"), color = 'black', size = 6) +
  theme_classic() +
  geom_vline(xintercept = seq(6, 14, by = 8), linetype = 2, color = "gray", linewidth = 1.5) +
  scale_color_manual(values = palette) +
  guides(color = "none") + theme(
    axis.line = element_blank(),
    axis.title = element_text(size = 10, face = "bold"),
    plot.title = element_text(size = 10, face = "bold"),
  )
```



PROBLEM 3

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

# Original code for generating the sinusoidal data
n = 2000
x = np.linspace(0, 20, num=n)

y = []

amplitude = 10

step = 10 / (n / 2)

for i in range(len(x)):
    y_value = amplitude * np.sin(2 * np.pi * x[i])
    y.append(y_value)

    if i <= n / 2:
        amplitude -= step
    else:
        amplitude += step

# Create a DataFrame with original data
sin_coordinates = pd.DataFrame({
```

```
    'x_values': x,
    'y_values': y,
    'color_groups': np.repeat(range(1, int(len(x) / 100) + 1), repeats=100)
})

# Define the rotation angle in degrees
rotation_angle_deg = -45

# Convert the rotation angle to radians
rotation_angle_rad = np.radians(rotation_angle_deg)

# Define the center point for rotation
center_x = 10
center_y = 0

# Apply rotation matrix to the coordinates
x_rotated = (sin_coordinates['x_values'] - center_x) * np.cos(rotation_angle_rad) - (sin_coordinates['y
y_rotated = (sin_coordinates['x_values'] - center_x) * np.sin(rotation_angle_rad) + (sin_coordinates['y

# Update the DataFrame with rotated coordinates
sin_coordinates['x_values'] = x_rotated
sin_coordinates['y_values'] = y_rotated

# Rest of your plotting code
num_colors = 20
rainbow_colors = plt.cm.rainbow(np.linspace(0, 1, num_colors))

plt.scatter(sin_coordinates['x_values'], sin_coordinates['y_values'], c=sin_coordinates['color_groups']
plt.title('20 Sigmoids change their amplitude to 0 and back', fontweight='bold')
plt.xlabel('Time', fontweight='bold')
plt.ylabel('Sinusoid values', fontweight='bold')
plt.axvline(x=5, linestyle='--', color='gray', linewidth = 4)
plt.axvline(x=15, linestyle='--', color='gray', linewidth = 4)
plt.text(13.5, 10, '1 std == 4.08', color='black', size=12, ha = 'center', va = 'center',rotation = -45

#plt.gca().set_ylim(-10, 10)
#plt.gca().set_aspect('equal', adjustable='box')

plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['bottom'].set_visible(False)
plt.gca().spines['left'].set_visible(False)


plt.colorbar().remove()
plt.show()
```



PROBLEM 4

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Read the datasets
usd_df = pd.read_csv('USD_AMD Historical Data.csv')
euro_df = pd.read_csv('EUR_AMD Historical Data.csv')

# Data Exploration and Summary for USD Exchange Rates
usd_summary = usd_df.describe()
usd_missing_values = usd_df.isnull().sum()

# Data Exploration and Summary for EURO Exchange Rates
euro_summary = euro_df.describe()
euro_missing_values = euro_df.isnull().sum()

# Print Statistical Analysis
print("Summary of USD Exchange Rates:")
```

```
## Summary of USD Exchange Rates:
```

```
print(usd_summary)
```

```
##         Unnamed: 0        Price         Open         High          Low
## count  2782.000000  2782.000000  2782.000000  2782.000000  2782.000000
## mean   1390.500000   460.344831   460.483853   461.998763   459.338163
## std     803.238549    38.422286    38.617548    38.392110    38.635356
## min       0.000000   383.000000   385.650000   385.900000   384.250000
## 25%     695.250000   414.000000   413.000000   415.020000   412.325000
## 50%    1390.500000   478.000000   478.110000   479.000000   477.000000
## 75%    2085.750000   484.297500   484.550000   485.365000   483.500000
## max    2781.000000   534.000000   534.500000   535.110000   534.500000
```

```
print("\nMissing Values in USD Exchange Rates:")
```

```
##
## Missing Values in USD Exchange Rates:
```

```
print(usd_missing_values)
```

```
## Unnamed: 0       0
## Date             0
## Price            0
## Open             0
## High             0
## Low              0
## Vol.          1736
## Change %         0
## Currency         0
## dtype: int64
```

```
print("\nSummary of EURO Exchange Rates:")
```

```
##
## Summary of EURO Exchange Rates:
```

```
print(euro_summary)
```

```
##              Price         Open         High          Low  Vol.
## count  2782.000000  2782.000000  2782.000000  2782.000000   0.0
## mean    533.938131   534.076032   537.431219   531.385579   NaN
## std      51.402615    51.284563    51.241054    51.422525   NaN
## min     384.520000   385.455000   389.950000   383.745000   NaN
## 25%     523.697500   523.555000   526.966250   520.805000   NaN
## 50%     541.255000   540.960000   544.677500   538.300000   NaN
## 75%     561.236250   560.600000   563.833750   557.612500   NaN
## max     640.900000   645.295000   646.330000   640.870000   NaN
```

```
print("\nMissing Values in EURO Exchange Rates:")
```

```
##
## Missing Values in EURO Exchange Rates:
```

```
print(euro_missing_values)
```

```
## Date           0
## Price          0
## Open           0
## High           0
## Low            0
## Vol.        2782
## Change %       0
## dtype: int64
```

```
# Create subplots with non-overlapping layouts
plt.figure(figsize=(18, 6))

# Plot 1: Histogram of USD Exchange Rates
plt.subplot(1, 3, 1)
sns.histplot(usd_df['Price'], bins=20, kde=True)
plt.xlabel('USD Exchange Rate')
plt.ylabel('Frequency')
plt.title('Histogram of USD')

# Plot 2: Histogram of EURO Exchange Rates
plt.subplot(1, 3, 2)
sns.histplot(euro_df['Price'], bins=20, kde=True)
plt.xlabel('EURO Exchange Rate')
plt.ylabel('Frequency')
plt.title('Histogram of EURO')

# Plot 3: Boxplot of Exchange Rates
plt.subplot(1, 3, 3)
sns.boxplot(data=[usd_df['Price'], euro_df['Price']], palette=['pink', 'turquoise'])
plt.xticks([0, 1], ['USD', 'EURO'])
```

```
## ([<matplotlib.axis.XTick object at 0x1510f3970>, <matplotlib.axis.XTick object at 0x1510f3610>], [Te
```

```
plt.ylabel('Price')
plt.title('Boxplot of Exchange Rates')

# Convert Date column to datetime
usd_df['Date'] = pd.to_datetime(usd_df['Date'])
euro_df['Date'] = pd.to_datetime(euro_df['Date'])

# Group datasets by year and calculate the mean
usd_mean_by_year = usd_df.groupby(usd_df['Date'].dt.year)['Price'].mean().reset_index()
euro_mean_by_year = euro_df.groupby(euro_df['Date'].dt.year)['Price'].mean().reset_index()

# Reproduce the given plot
plt.figure(figsize=(12, 6))
plt.plot(usd_mean_by_year['Date'], usd_mean_by_year['Price'], label='1 USD to AMD', color='blue')
```

```
plt.plot(euro_mean_by_year['Date'], euro_mean_by_year['Price'], label='1 EURO to AMD', color='orange')
plt.fill_between(usd_mean_by_year['Date'], usd_mean_by_year['Price'], euro_mean_by_year['Price'], where=
plt.fill_between(usd_mean_by_year['Date'], usd_mean_by_year['Price'], euro_mean_by_year['Price'], where=
plt.xlabel('Year')
plt.ylabel('Price')
plt.title('Comparison of USD and EURO Prices to AMD')
plt.legend()
plt.grid(True)

# Show Plots
plt.tight_layout()
plt.show()
```