Fourier Transform of streamflow data for seasonal signals extraction

Abdullah Azzam

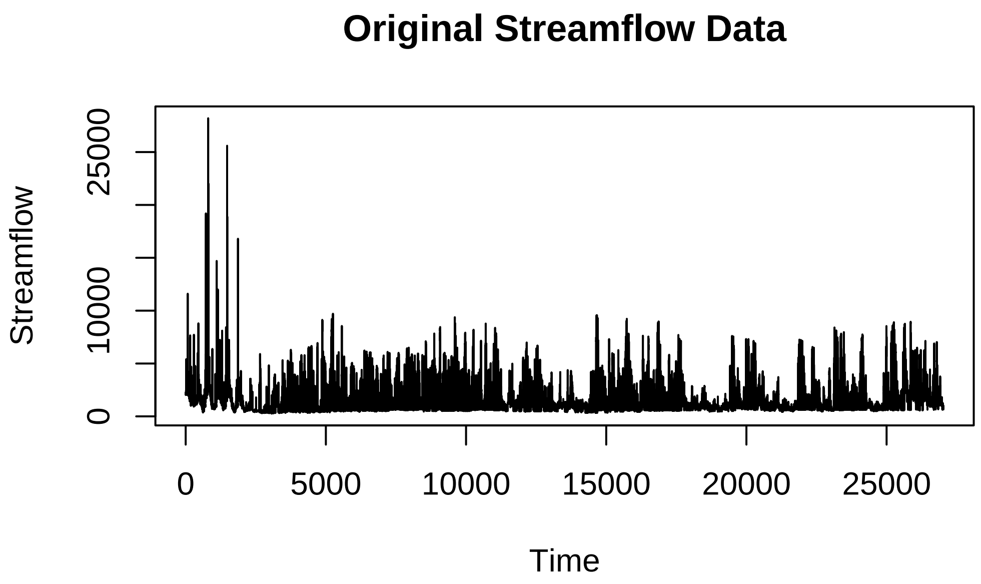
Download the flow data

```r
# Define URL for the data
url <- "https://nwis.waterdata.usgs.gov/nwis/dv?cb_00010=on&cb_00060=on&cb_00065=on&cb_00095=

# Define file paths
file_path <- "usgs_02334430_daily_data.txt"

# Download the files
download.file(url, destfile = file_path, method = "auto")

# Read the data from the first site
data <- read.table(file_path, sep="\t", header=TRUE, stringsAsFactors=FALSE)[-1,]
rownames(data) <- 1:nrow(data)
data$date <- as.Date(data[,3], format="%Y-%m-%d")
colnames(data)[c(22)] <- "q"
data$q <- as.numeric(data[,"q"])
q <- data[, c("q")]
q.timeseries <- data[, c("date", "q")]
```

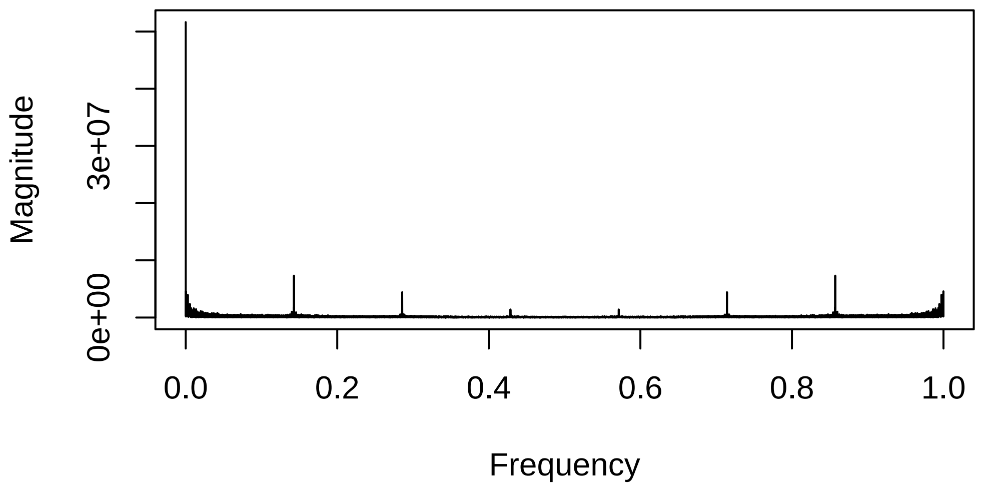
FFT and seasonal signals without noise

```r
library(fftw)

q <- na.omit(q)

# Apply FFT on 'q'
```

```
fft_result <- fft(q, inverse = FALSE)

# Extract the magnitude of the FFT result
fft_magnitude <- abs(fft_result)

# Calculate the frequencies corresponding to the FFT result
frequencies <- seq(0, length(q) - 1) / length(q)

# Plot the original streamflow data
plot(q, type = "l", xlab = "Time", ylab = "Streamflow", main = "Original Streamflow Data")
```

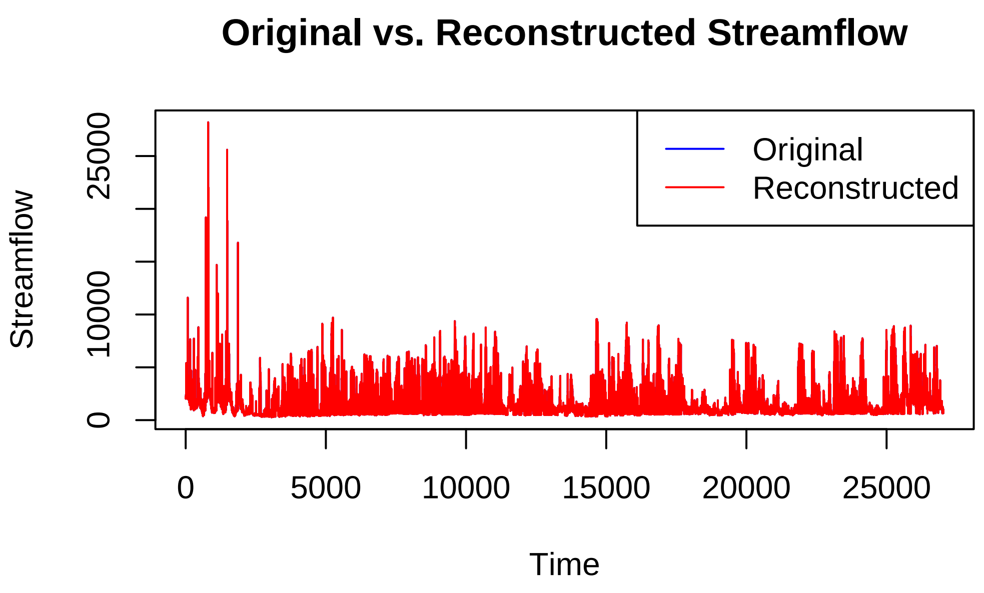


```
# Plot the FFT magnitude
plot(frequencies, fft_magnitude, type = "l", xlab = "Frequency", ylab = "Magnitude")
```

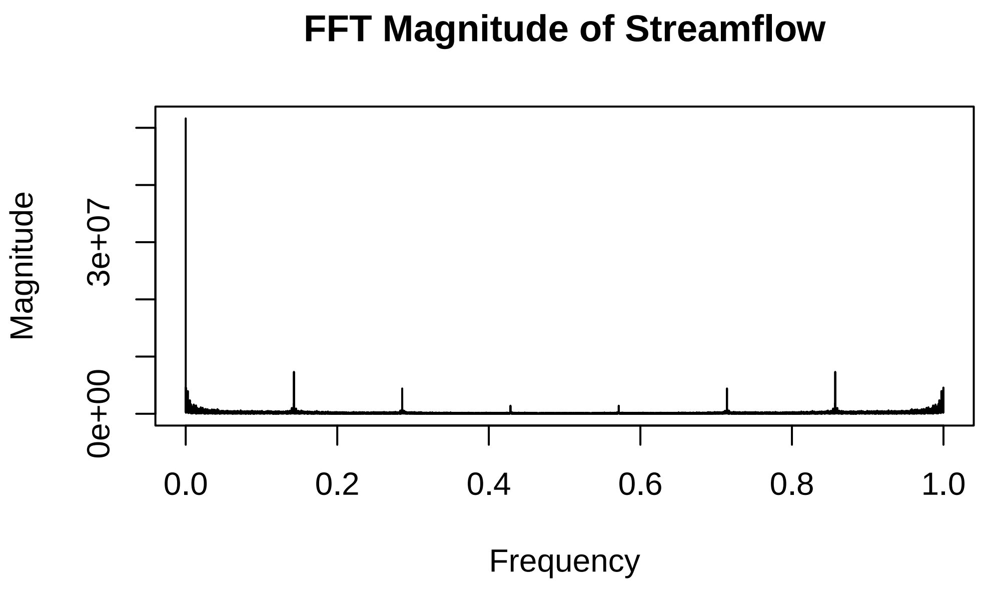


Reconstruction and validation of original timeseries

```
# Load necessary library
library(stats)

q <- na.omit(q)  # Ensuring there are no NA values in the dataset

# Perform FFT on the original data
fft_result <- fft(q, inverse = FALSE)

# Extract the magnitude and phase from the FFT result
fft_magnitude <- Mod(fft_result)
fft_phase <- Arg(fft_result)

# Perform inverse FFT to reconstruct the signal
reconstructed_signal <- fft(fft_result, inverse = TRUE)
reconstructed_signal <- Re(reconstructed_signal) / length(q)

# Plot the original and reconstructed signals for comparison
plot(q, type = "l", col = "blue", xlab = "Time", ylab = "Streamflow", main = "Original vs. Re
lines(reconstructed_signal, col = "red")
legend("topright", legend = c("Original", "Reconstructed"), col = c("blue", "red"), lty = 1)
```



```
# Calculate frequencies for plotting FFT magnitude
frequencies <- (0:(length(fft_magnitude) - 1)) / length(q)

# Optional: Plot FFT Magnitude
plot(frequencies, fft_magnitude, type = "l", xlab = "Frequency", ylab = "Magnitude", main =
```



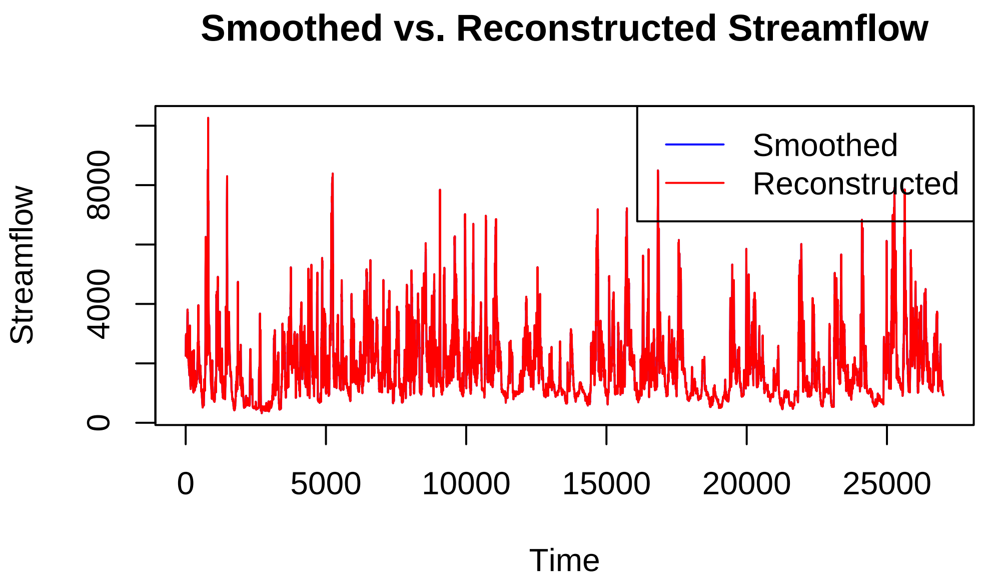
```
# Calculate the correlation between the original and reconstructed data
if (length(q) == length(reconstructed_signal)) {
    correlation_value <- cor(q, reconstructed_signal)
```

```r
    print(paste("Correlation between original and reconstructed data:", correlation_value))
} else {
    print("The lengths of the original and reconstructed data do not match. Cannot calculate
}
```

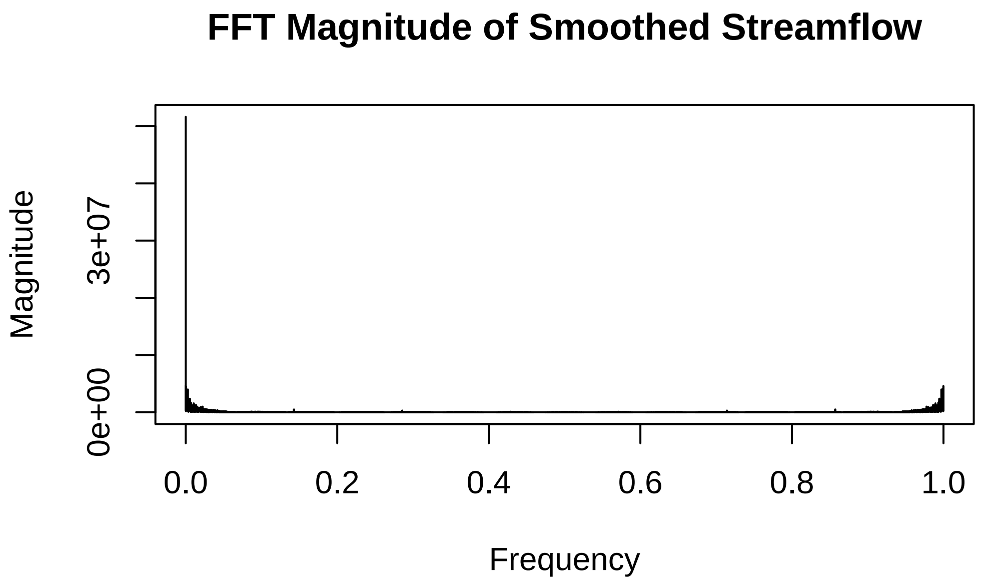
```
[1] "Correlation between original and reconstructed data: 1"
```

Denoising, transforming and reconstructing the denoised data.

```r
# Define the window size for the moving average
window_size <- 15

# Create filter weights for the moving average
filter_weights <- rep(1 / window_size, window_size)

# Apply the moving average filter
q_smoothed <- stats::filter(q, filter_weights, sides = 2)

# Remove NA values that appear at the start and end due to the filter
q_smoothed <- q_smoothed[!is.na(q_smoothed)]

# Perform FFT on the smoothed data
fft_result <- fft(q_smoothed, inverse = FALSE)

# Extract magnitude for analysis (optional step depending on need)
fft_magnitude <- abs(fft_result)

# Calculate frequencies ensuring the length matches the fft_magnitude
frequencies <- (0:(length(fft_magnitude) - 1)) / length(q_smoothed)

# Perform inverse FFT to reconstruct the signal
reconstructed_signal <- fft(fft_result, inverse = TRUE)
reconstructed_signal <- Re(reconstructed_signal) / length(q_smoothed)

# Plot the original, smoothed, and reconstructed signals for comparison
plot(q_smoothed, type = "l", col = "blue", xlab = "Time", ylab = "Streamflow", main = "Smootl
lines(reconstructed_signal, col = "red")
legend("topright", legend = c("Smoothed", "Reconstructed"), col = c("blue", "red"), lty = 1)
```



```r
# Optional: Plot FFT Magnitude
plot(frequencies, fft_magnitude, type = "l", xlab = "Frequency", ylab = "Magnitude", main = '
```



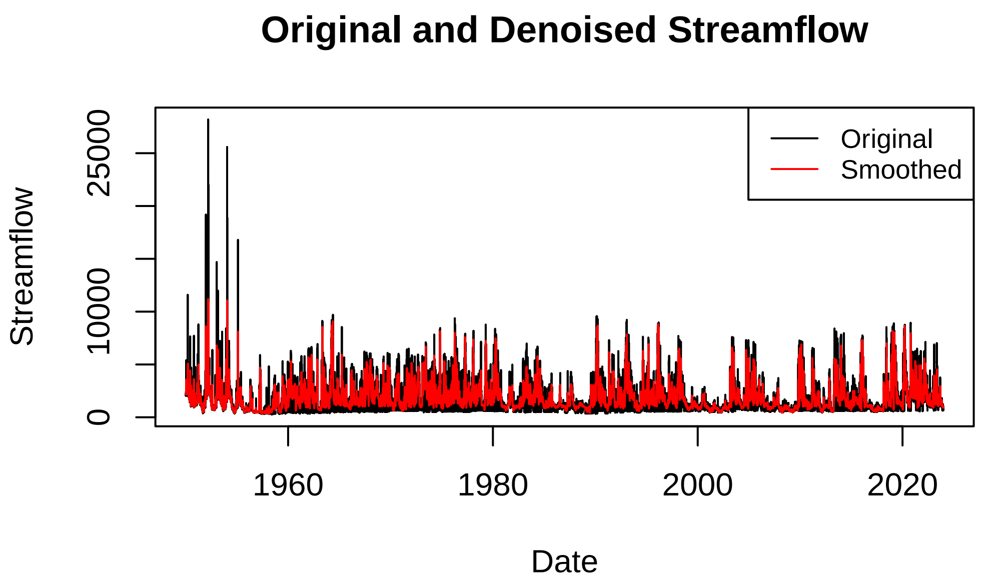
```r
# Ensure both smoothed and reconstructed data have the same length
# This should naturally be the case from the previous steps, but it's good to check
if (length(q_smoothed) == length(reconstructed_signal)) {
    # Calculate the correlation
    correlation_value <- cor(q_smoothed, reconstructed_signal)
    print(paste("Correlation between smoothed and reconstructed data:", correlation_value))
```

```
} else {
    print("The lengths of the smoothed and reconstructed data do not match. Cannot calculate
}
```

```
[1] "Correlation between smoothed and reconstructed data: 1"
```

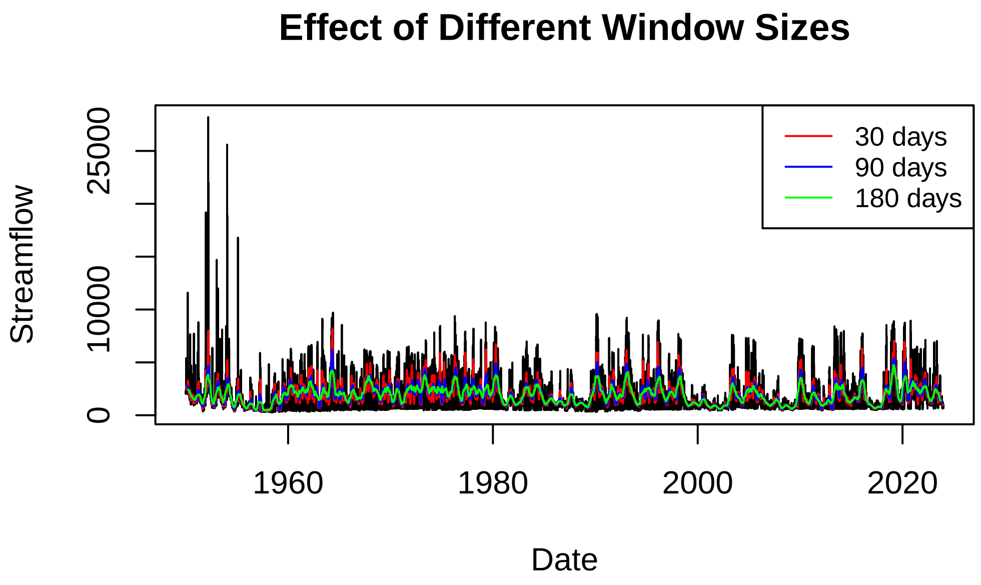
Effect of variable smoothing window sizes

```
# Define the window size and create filter weights
window_size <- 7  # For example, a 30-day moving average
filter_weights <- rep(1 / window_size, window_size)

# Apply the filter using stats::filter to avoid conflicts with dplyr or other packages
q_smoothed <- stats::filter(q.timeseries$q, filter_weights, sides=2)

# Plot original and smoothed data
plot(q.timeseries$date, q.timeseries$q, type="l", xlab="Date", ylab="Streamflow", main="Orig
lines(q.timeseries$date, q_smoothed, col="red")
legend("topright", legend=c("Original", "Smoothed"), col=c("black", "red"), lty=1, cex=0.8)
```



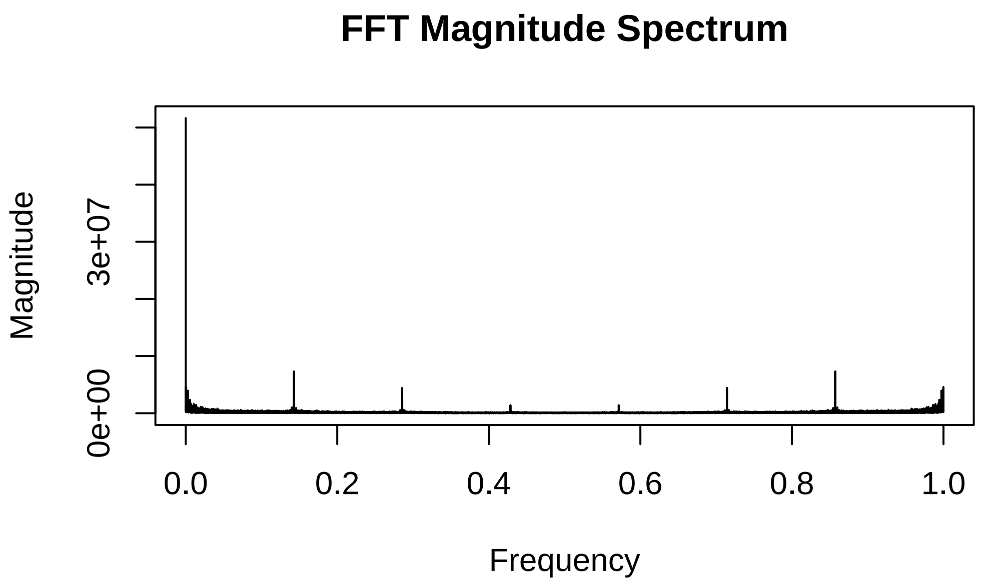
```
# Plot with multiple window sizes
plot(q.timeseries$date, q.timeseries$q, type="l", xlab="Date", ylab="Streamflow", main="Effe
```

```r
# Array of different window sizes
window_sizes <- c(30, 90, 180)

# Colors for different plots
colors <- c("red", "blue", "green")

for (i in seq_along(window_sizes)) {
    filter_weights <- rep(1 / window_sizes[i], window_sizes[i])
    # Use stats::filter to specify the correct function
    q_smoothed <- stats::filter(q.timeseries$q, filter_weights, sides=2)

    # Ensure to handle plotting lines only if lengths match
    if (length(q_smoothed) == length(q.timeseries$date)) {
        lines(q.timeseries$date, q_smoothed, col=colors[i])
    } else {
        warning("Length of smoothed data and date data do not match for window size ", windo
    }
}

# Add a legend to the plot to distinguish between different window sizes
legend("topright", legend=paste(window_sizes, "days"), col=colors, lty=1, cex=0.8)
```

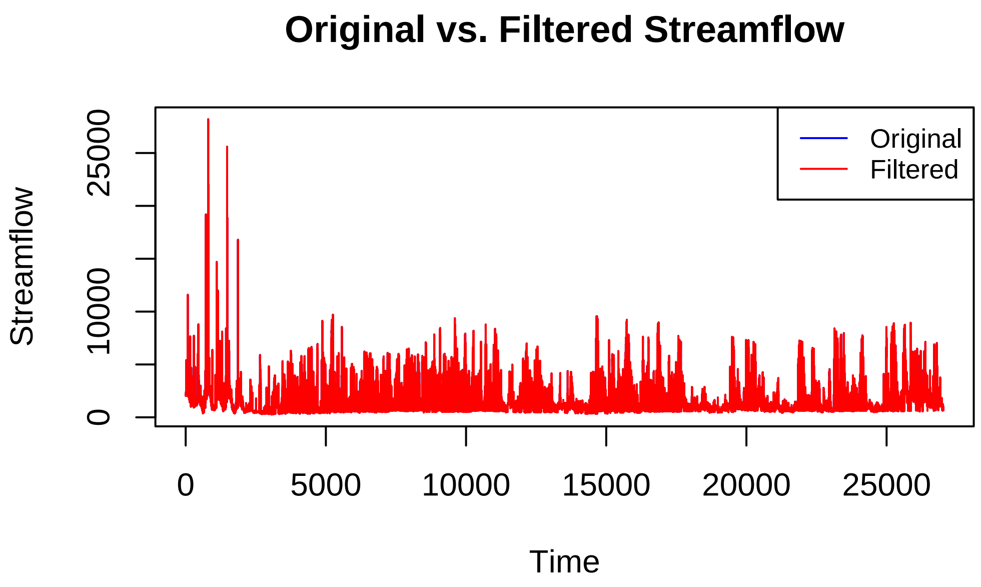


Extract & verify low-flow season

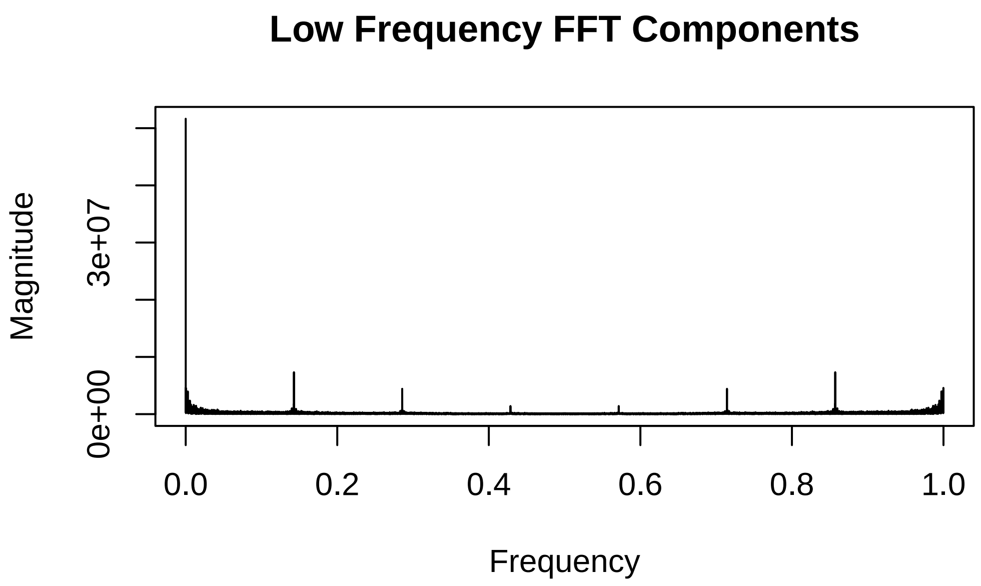
```r
# Compute the FFT
fft_result <- fft(q)

# Calculate the magnitudes of the FFT result
fft_magnitude <- Mod(fft_result)

# Calculate frequencies
frequencies <- seq(0, length(q) - 1) / length(q)

# Plot the magnitudes against the frequencies to visualize the spectrum
plot(frequencies, fft_magnitude, type = "l", main = "FFT Magnitude Spectrum", xlab = "Freque
```



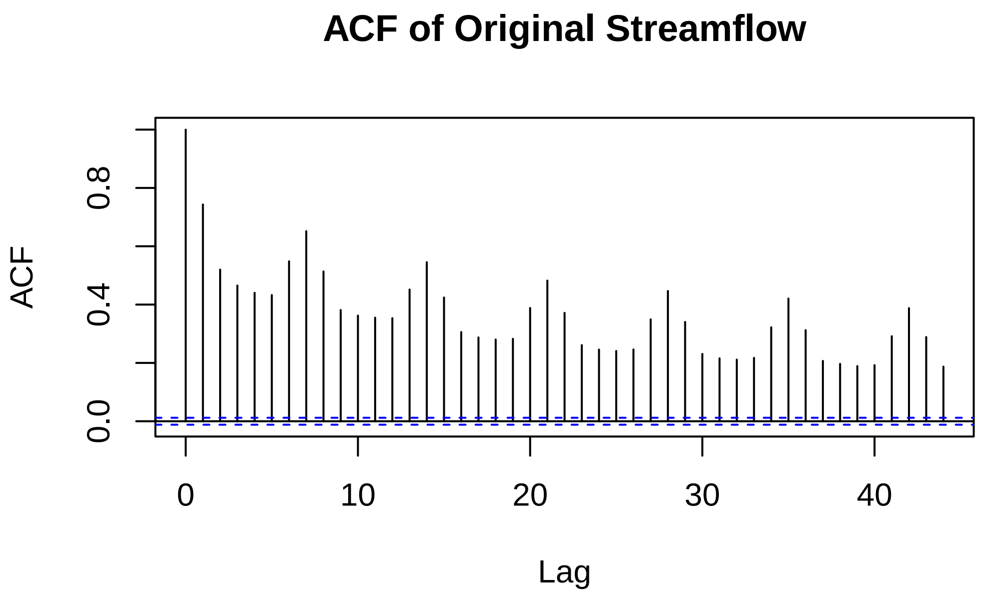
```r
# Example: Zero out all frequencies except the lowest few
low_freq_threshold <- 1000  # Adjust threshold according to your specific needs
fft_filtered <- ifelse(frequencies < low_freq_threshold, fft_result, 0)

# Inverse FFT to reconstruct the filtered signal
filtered_signal <- fft(fft_filtered, inverse = TRUE)
filtered_signal <- Re(filtered_signal) / length(q)

# Plot the original and filtered signals for comparison
plot(q, type = "l", col = "blue", main = "Original vs. Filtered Streamflow", ylab = "Streamfl
lines(filtered_signal, col = "red")
legend("topright", legend=c("Original", "Filtered"), col=c("blue", "red"), lty=1, cex=0.8)
```



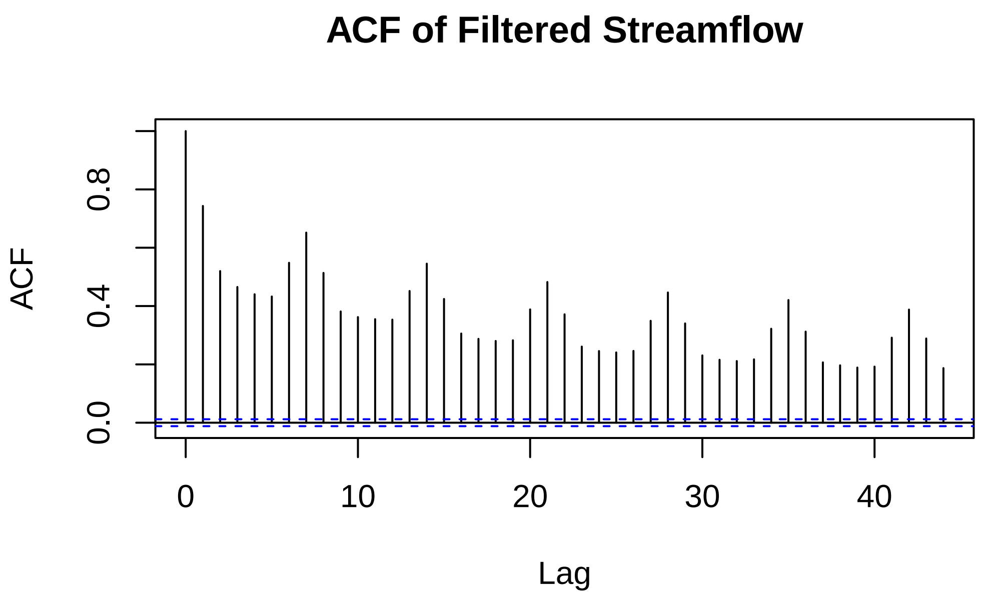
```r
# Plot low-frequency components
low_freq_part <- frequencies < 1000  # Adjust based on data and observation
plot(frequencies[low_freq_part], fft_magnitude[low_freq_part], type = "l", main = "Low Freque
```



```
# Calculate basic statistics
mean_original <- mean(q)
mean_filtered <- mean(filtered_signal)
var_original <- var(q)
var_filtered <- var(filtered_signal)

# Autocorrelation
acf(q, main = "ACF of Original Streamflow", plot = TRUE)
```



```
acf(filtered_signal, main = "ACF of Filtered Streamflow", plot = TRUE)
```


ACF of Filtered Streamflow
ACF
0.0
0.4
0.8
0
10
20
30
40
Lag