ECS 132 - Project

Teresa Li, sqtli@ucdavis.edu
Wenjing Fu, luffu@ucdavis.edu

2018-05-19

Contents

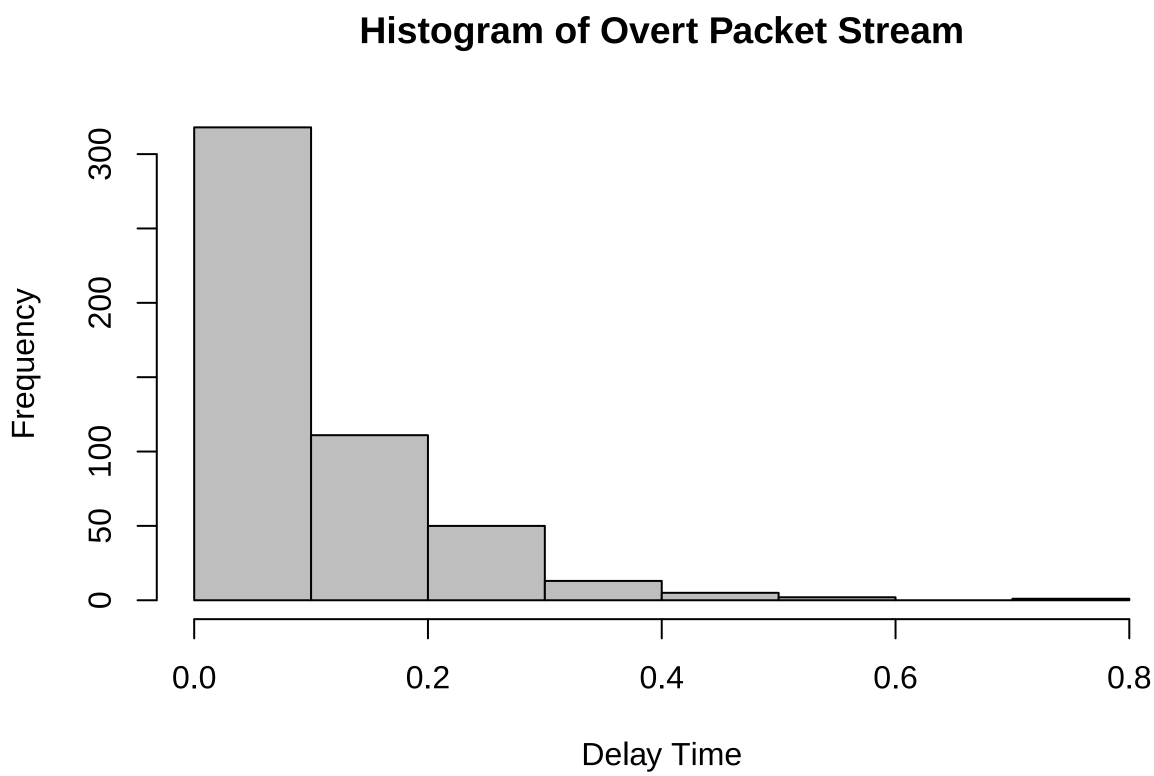
Design

Question 1

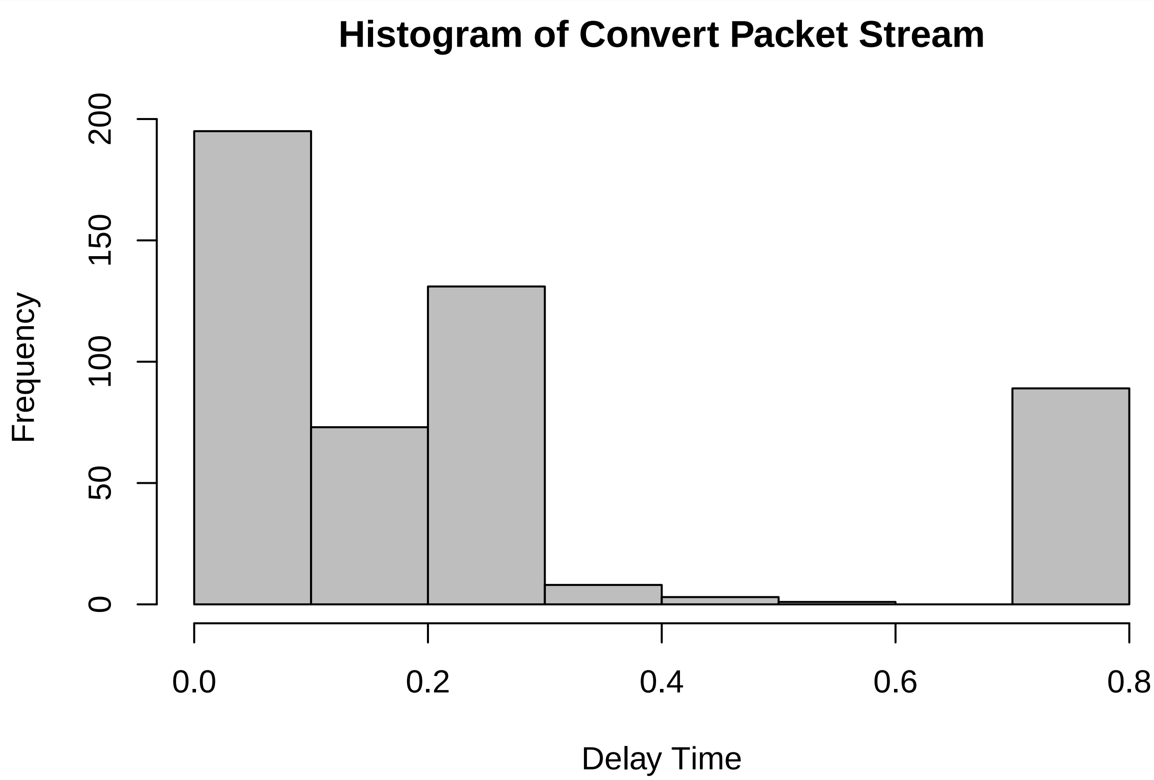
```r
Traffic_data_orig <- read.csv("Traffic_data_orig.csv", header=TRUE)
message <- "this is a secret message"
raw <- charToRaw(message)
time = Traffic_data_orig$Time
num = as.integer(rawToBits(raw))

delays = numeric(length(time) - 1)
for (i in (1:(length(time) - 1))) {
  delays[i] = time[i+1] - time[i]
}

index = 1
bitlen = length(raw)*8
encrpt <- numeric(length(raw)*8)
for (i in (0:(length(raw)-1))) {
  for (j in 1:8) {
    if (num[i*8+j] == 0) {
      encrpt[index] = 0.25
    }
    else {
      encrpt[index] = 0.75
    }
    index = index+1
    j = j-1
  }
}

delays2 = delays
for (i in (1:bitlen)) {
  delays2[i] = encrpt[i]
}
```

Question 2

```r
hist(delays, col='grey', xlab = 'Delay Time',
     main = 'Histogram of Overt Packet Stream')
```




```r
hist(delays2, col='grey', xlab = 'Delay Time',
     main = 'Histogram of Convert Packet Stream')
```



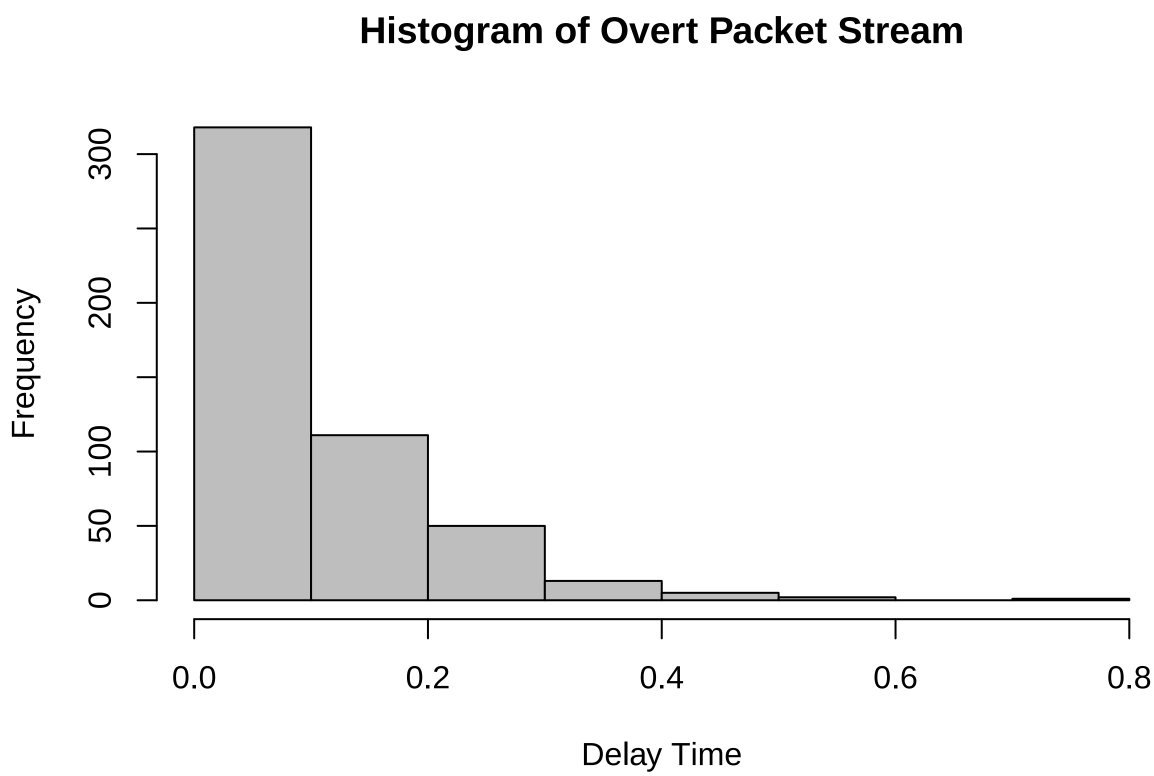
Yes, Eve will be suspicious because it is obvious that the distribution changed.

Question 3

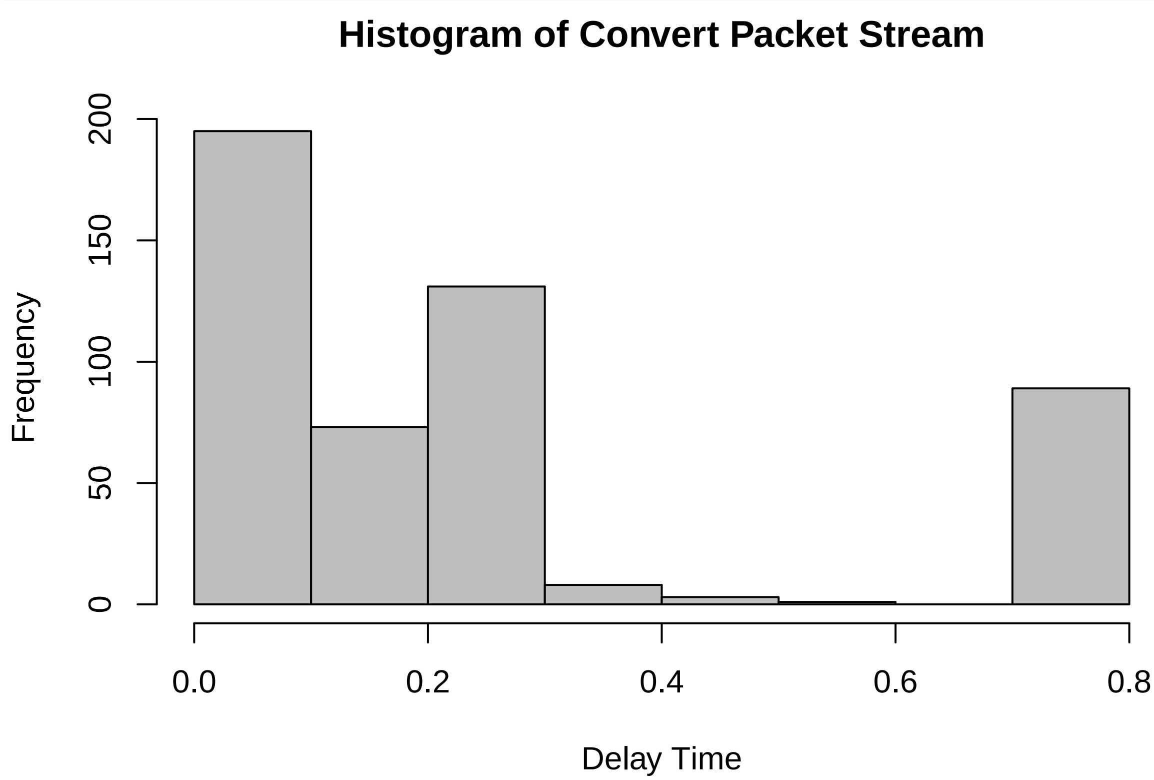
```r
Traffic_data_orig <- read.csv("Traffic_data_orig.csv", header=TRUE)
message <- "this is a secret message"
raw <- charToRaw(message)
time = Traffic_data_orig$Time
num = as.integer(rawToBits(raw))
delays = numeric(length(time) - 1)
for (i in (1:(length(time) - 1))) {
  delays[i] = time[i+1] - time[i]
}
m = median(delays)
max = max(delays)
min = min(delays)

index = 1
bitlen = length(raw)*8
encrpt <- numeric(length(raw)*8)
for (i in (0:(length(raw)-1))) {
  for (j in 1:8) {
    if (num[i * 8 + j] == 0) {
      encrpt[index] = runif(1, min, m)
    }
    else {
      encrpt[index] = runif(1, m, max)
    }
    index = index + 1
    j = j - 1
  }
}

delays3 = delays
for (i in (1:bitlen)) {
  delays3[i] = encrpt[i]
}
```

Question 4

```r
hist(delays, col='grey', xlab = 'Delay Time',
     main = 'Histogram of Overt Packet Stream')
```




```r
hist(delays2, col='grey', xlab = 'Delay Time',
     main = 'Histogram of Convert Packet Stream')
```



I think Eva will not be suspicious.

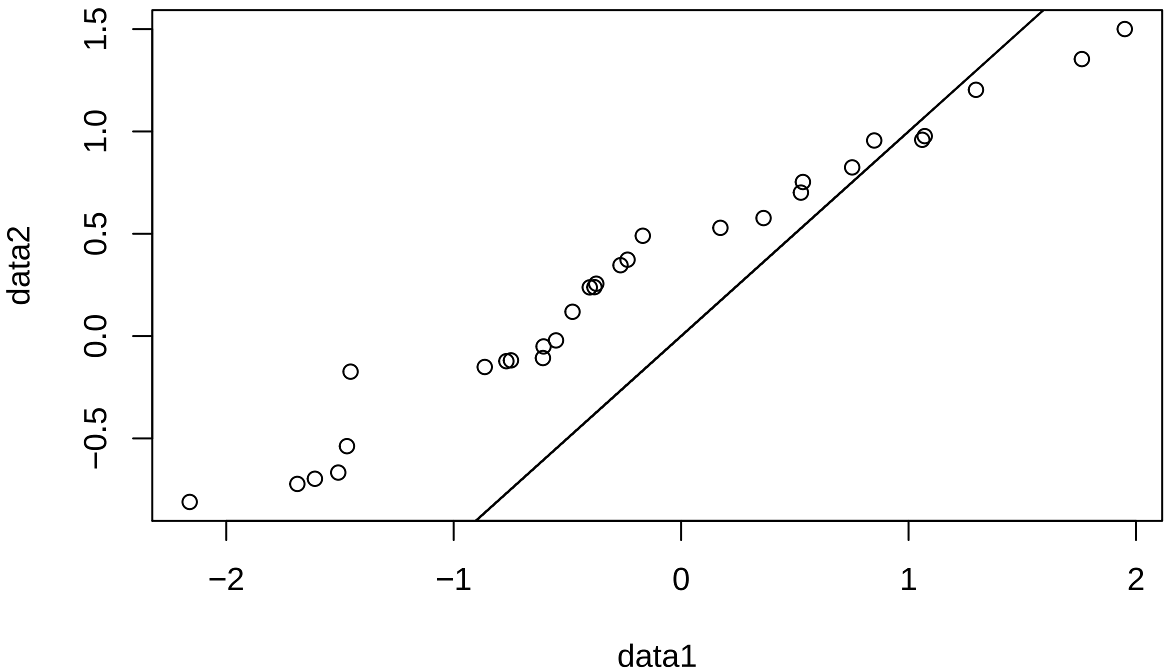
Question 5

1. Instead of generating random number from m to max, and min to m, we can choose one of the existing one from m to max, and min to m.
2.
3.

Detection

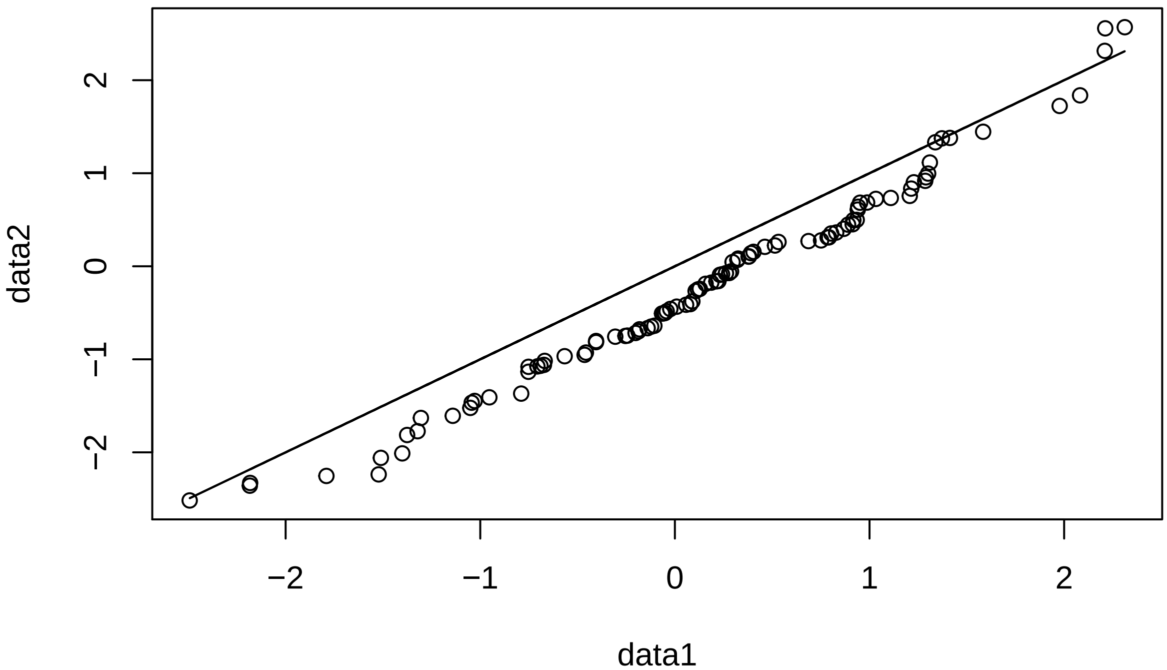
Step 1

```
data1 <- rnorm(30)
data2 <- rnorm(30)
qqplot(data1, data2)
lines(data1, data1, type = 'l')
```

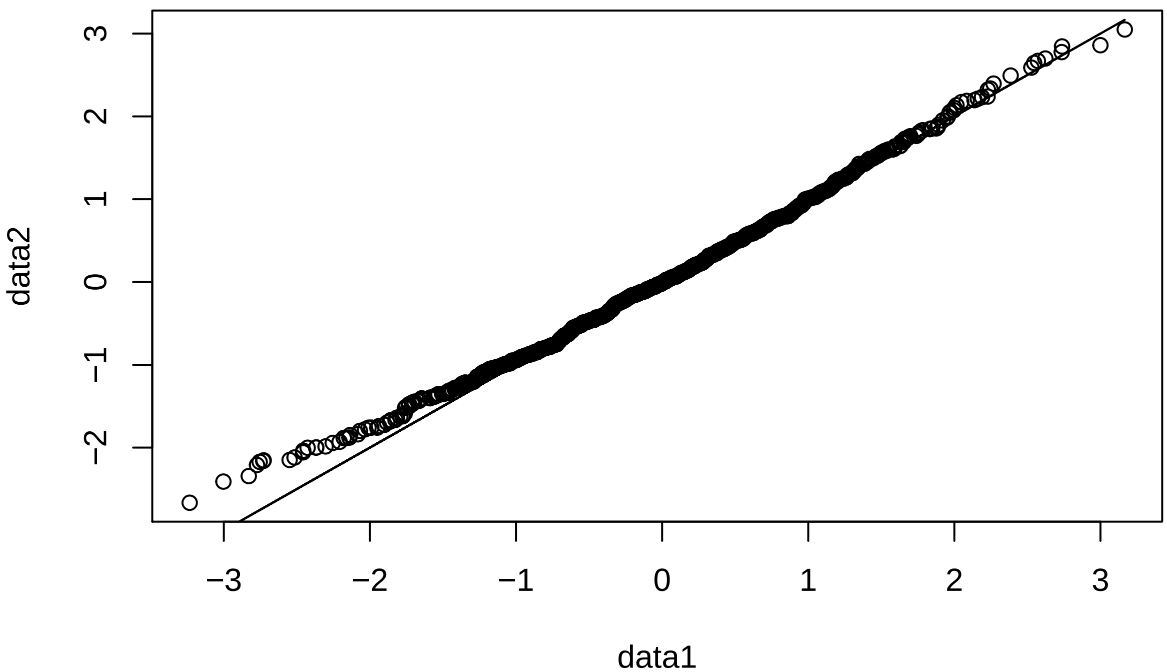


Step 2

```
data1 <- rnorm(100)
data2 <- rnorm(100)
qqplot(data1, data2)
lines(data1, data1, type = 'l')
```



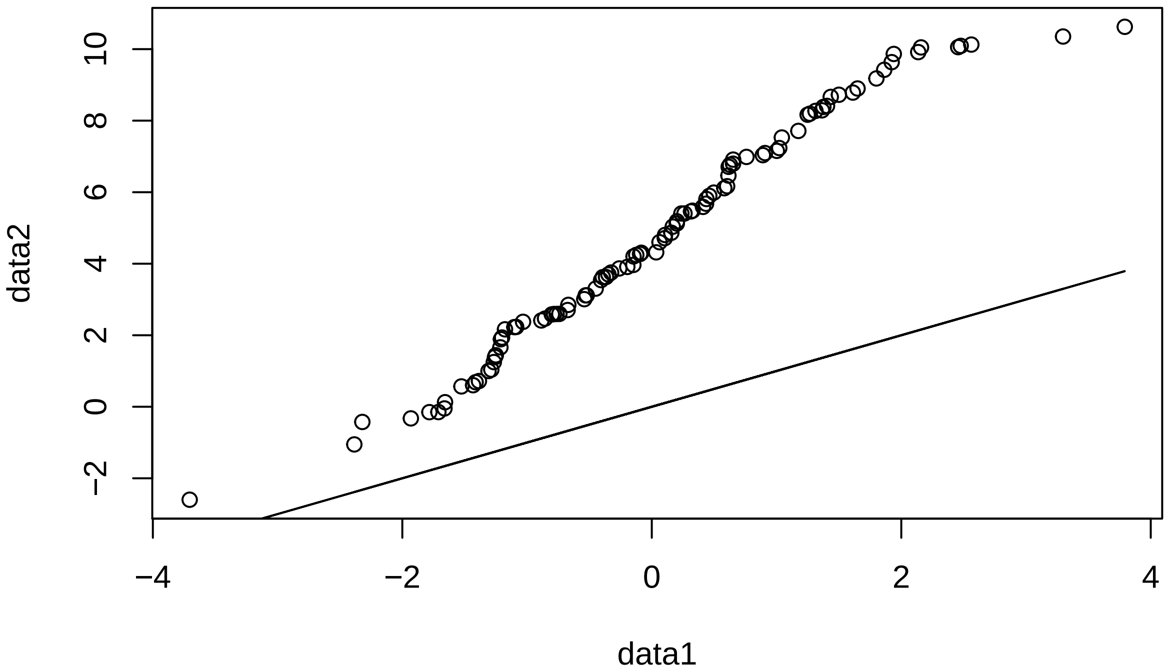

```
data1 <- rnorm(1000)
data2 <- rnorm(1000)
qqplot(data1, data2)
lines(data1, data1, type = 'l')
```



Two plots are directly proportional to each other.

Step 3

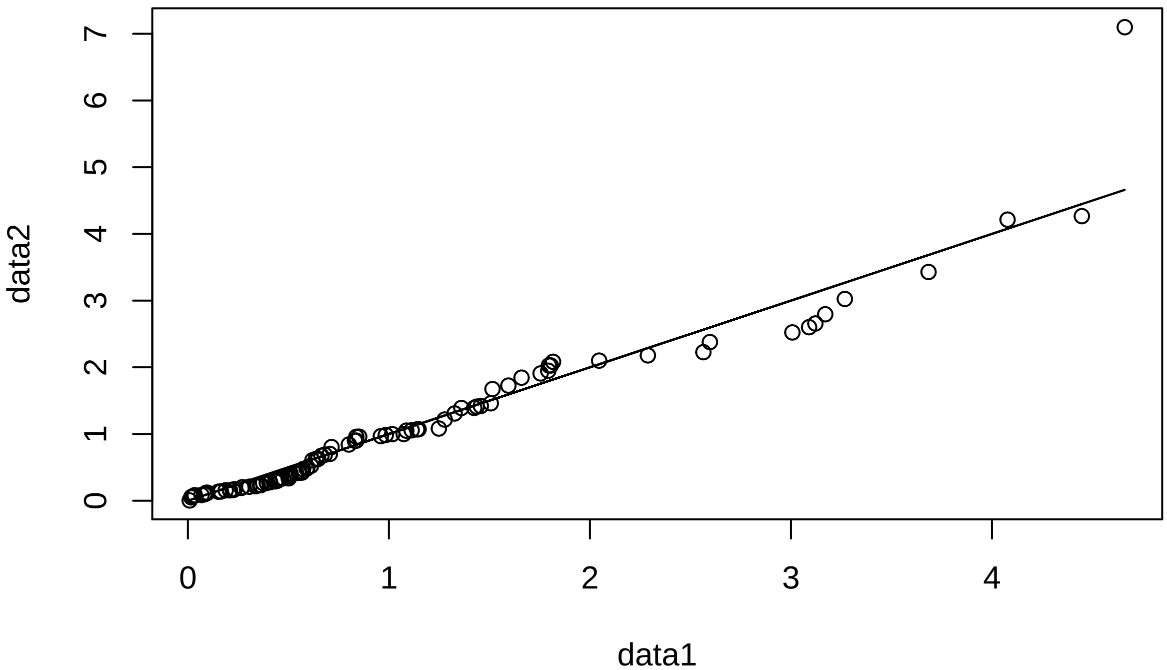
```
data1 <- rnorm(100)
data2 <- rnorm(100, mean = 5, sd = 3)
qqplot(data1, data2)
lines(data1, data1, type = 'l')
```



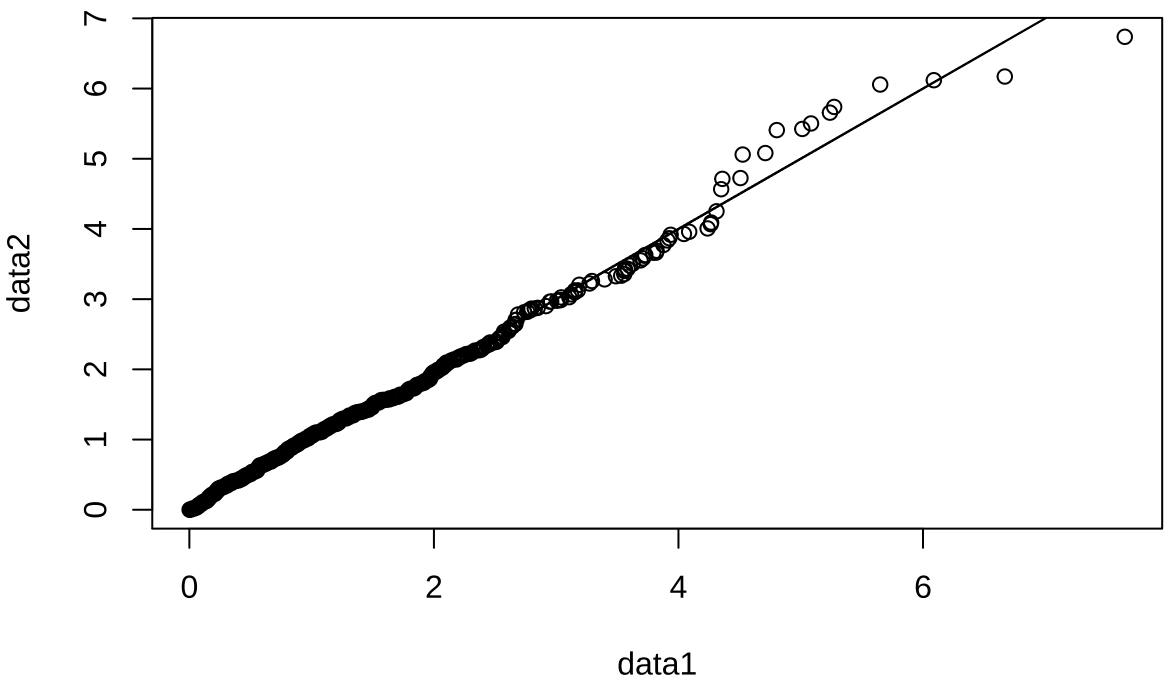
data2 is directly proportional to data1, but the slope is different this time.

Step 4

```
data1 <- rexp(100)
data2 <- rexp(100)
qqplot(data1, data2)
lines(data1, data1, type = 'l')
```



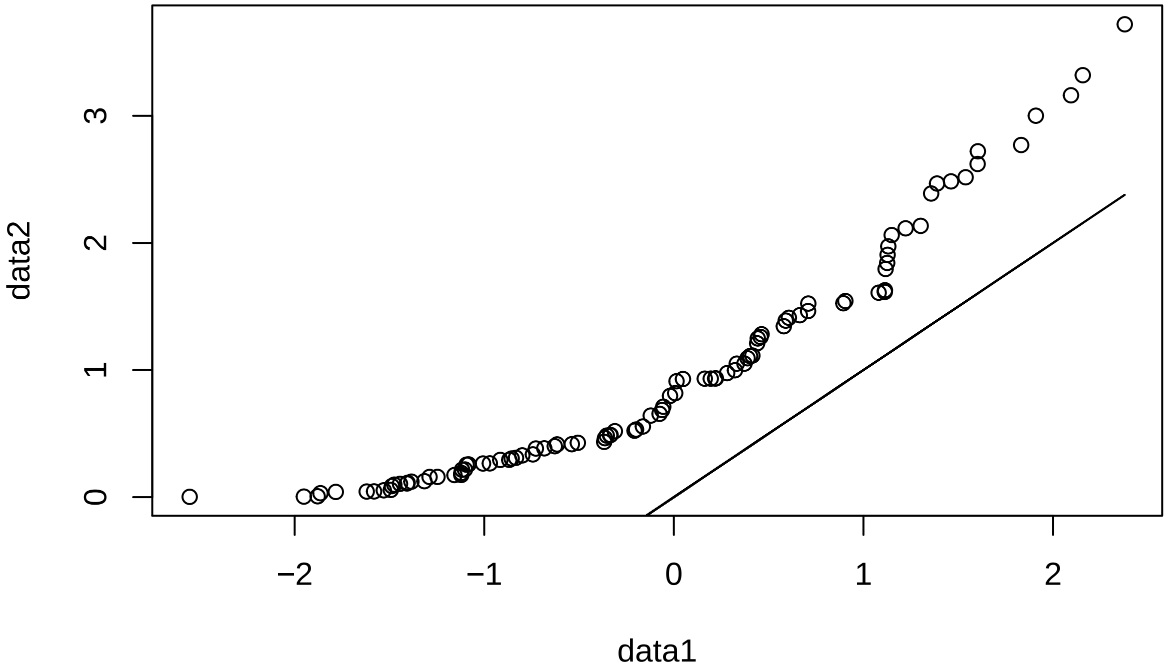

```r
data1 <- rexp(1000)
data2 <- rexp(1000)
qqplot(data1, data2)
lines(data1, data1, type = 'l')
```



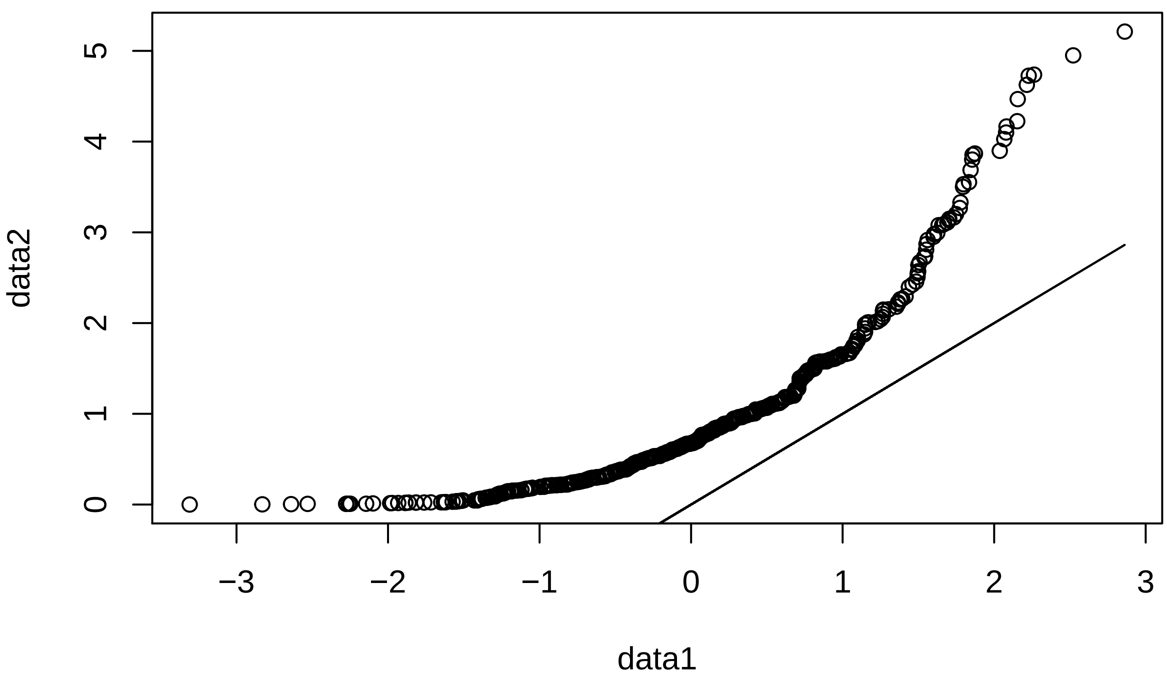
With a bigger size, two data sets are more consistent.

Step 5

```
data1 <- rnorm(100)
data2 <- rexp(100)
qqplot(data1, data2)
lines(data1, data1, type = 'l')
```



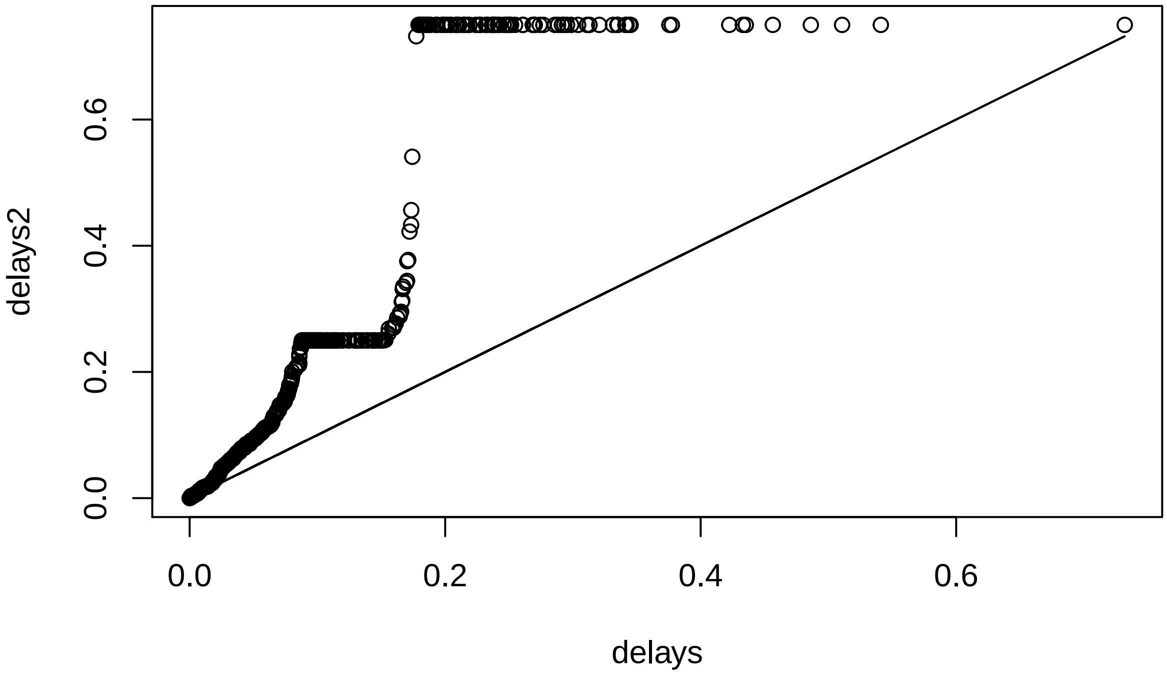

```
data1 <- rnorm(500)
data2 <- rexp(500)
qqplot(data1, data2)
lines(data1, data1, type = 'l')
```



Their relation is exponential.

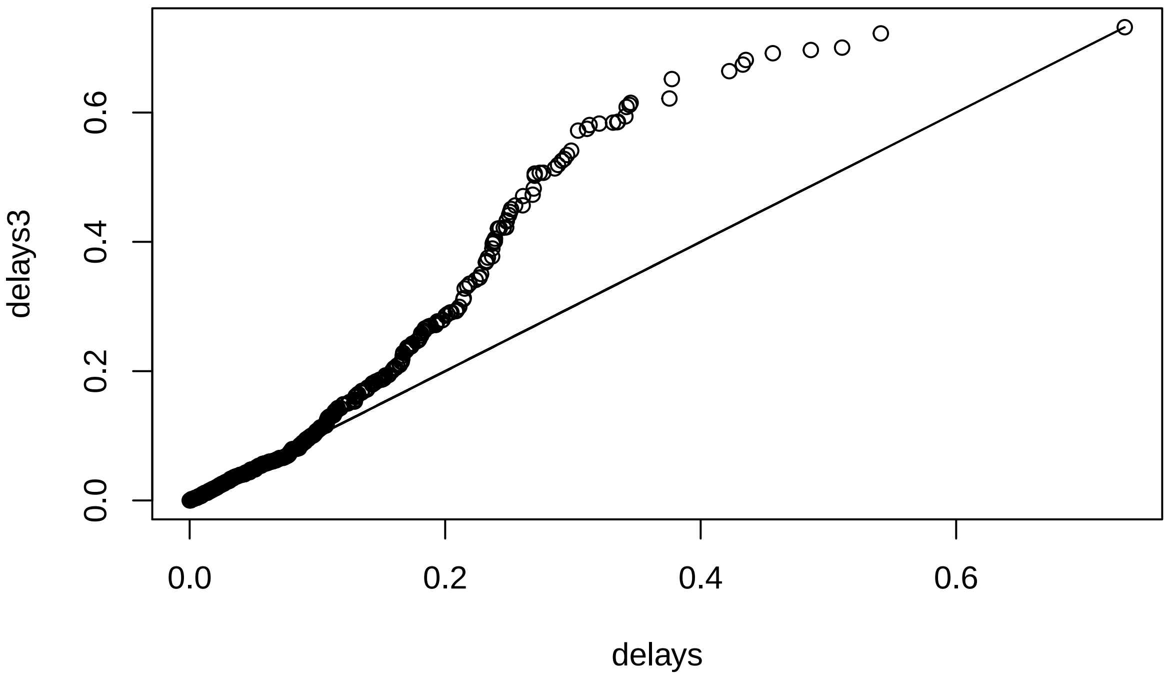
Step 6

```r
qqplot(delays, delays2)
lines(delays, delays, type = 'l')
```



Step 7

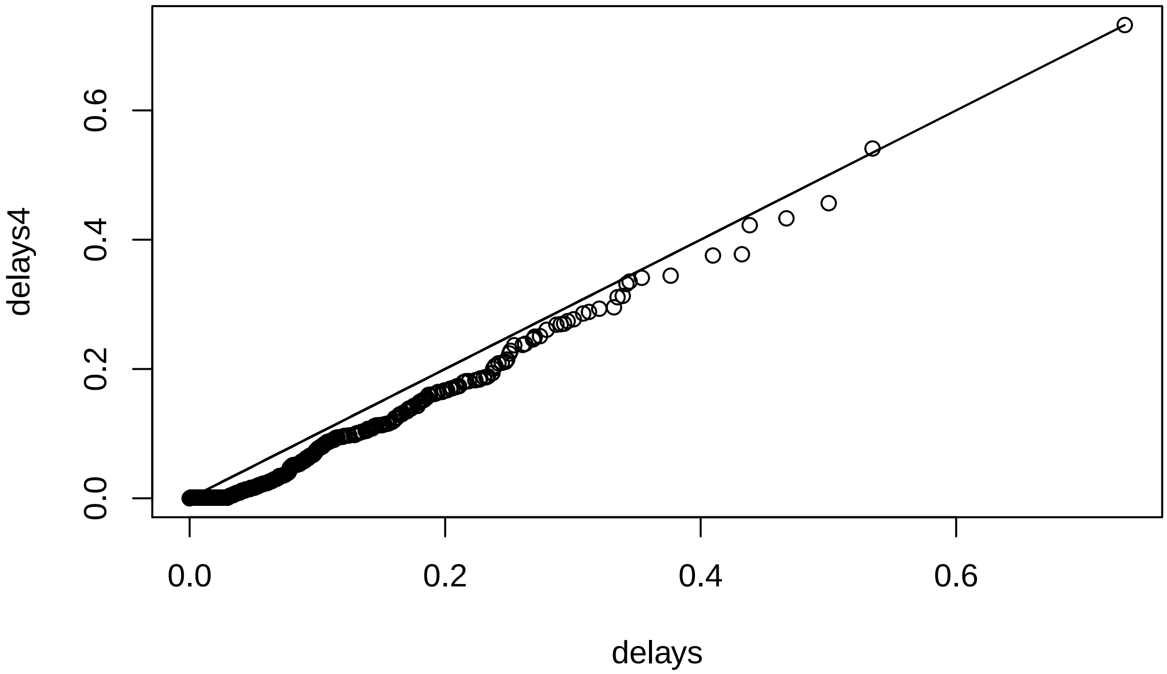
```r
qqplot(delays, delays3)
lines(delays, delays, type = 'l')
```



Step 8

```
Traffic_data_orig <- read.csv("Traffic_data_orig.csv", header=TRUE)
message <- "this is a secret message"
raw <- charToRaw(message)
time = Traffic_data_orig$Time
num = as.integer(rawToBits(raw))
delays = numeric(length(time) - 1)
for (i in (1:(length(time) - 1))) {
  delays[i] = time[i+1] - time[i]
}
m = median(delays)
max = max(delays)
min = min(delays)

index = 1
bitlen = length(raw)*8
encrpt <- numeric(length(raw)*8)
for (i in (0:(length(raw)-1))) {
  for (j in 1:8) {
    if (num[i * 8 + j] == 0) {
      encrpt[index] = sample(delays[which(delays >= min && delays <= m)])[1]
    }
    else {
      encrpt[index] = sample(delays[which(delays >= m && delays <= max)])[1]
    }
    index = index + 1
    j = j - 1
  }
}
```

```r
delays4 = delays
for (i in (1:bitlen)) {
  delays4[i] = encrpt[i]
}
qqplot(delays, delays4)
lines(delays, delays, type = 'l')
```



Implementation

Implementation 1

```r
generateMessage <- function (len) {
  message <- numeric(len)
  for (i in (1:len)){
    message[i] = sample(c(0,1), 1)
  }
  return(message)
}

generateTime <- function(ipd, len) {
  time <- numeric(len)
  time[1] = 0
  for (i in (2:len)){
    time[i] = time[i-1] + ipd[i-1]
  }
  return(time)
}

generateProb <- function (mlen, bufferNum) {
```

```r
  bufferSize = 20
  currbuffer = bufferNum
  message <- generateMessage(mlen) # Generate the random bit pattern
  ipdSource <- rexp(100)
  ipdSend <- rexp(100)
  ipdEncrypt <- ipdSend
  ipdTime <- generateTime(ipdSource, 101)
  currTime = ipdTime[bufferNum]
  min = min(ipdSend)
  max = max(ipdSend)
  med = median(ipdSend)
  underflow = 0
  overflow = 0
  currbuffer = 2

  index = bufferNum + 1

    for (i in (1:mlen)) {
      # Generate a delay
      if (message[i] == 0) {
        delay = runif(1, min, med)
        ipdEncrypt[i] = delay
      } else {
        delay = runif(1, med, max)
        ipdEncrypt[i] = delay
      }
      currTime = currTime + delay # update time
      # Update the state of the buffer depending on the number of arrivals during that time.
      if (currTime <= ipdTime[index]) {
        currbuffer = currbuffer - 1
      } else {
        currbuffer = currbuffer + 1
      }
      index = index + 1
      if (currbuffer > bufferSize){
        overflow = 1
        break
      }
      if (currbuffer < 1){
        underflow = 1
        break
      }
    }
  return(c(underflow,overflow))
}
probsU <- numeric(1000)
probsO <- numeric(1000)
for (t in (1:1000)) {
  m <- c(16,32)
  bufferNum <- c(2,6,10,14,18)
  underflow = 0
  overflow = 0
  count = 0
```

```r
  for (i in (1:2)) {
      for (j in (1:5)) {
        output <- generateProb(m[i], bufferNum[j])
        underflow = underflow + output[1]
        overflow = overflow + output[2]
        count = count + 1
      }
  }
  probsU[t] = underflow/count
  probsO[t] = overflow/count
}
c(mean(probsU), mean(probsO))
```

```
## [1] 0.3667 0.3021
```

Implementation 2

```r
generateProb <- function (mlen, bufferNum) {
  bufferSize = 20
  currbuffer = bufferNum
  message <- generateMessage(mlen) # Generate the random bit pattern
  ipdSource <- runif(100,0,1)
  ipdSend <- runif(100,0,1)
  ipdEncrypt <- ipdSend
  ipdTime <- generateTime(ipdSource, 101)
  currTime = ipdTime[bufferNum]
  min = min(ipdSend)
  max = max(ipdSend)
  med = median(ipdSend)
  underflow = 0
  overflow = 0
  currbuffer = 2

  index = bufferNum + 1

    for (i in (1:mlen)) {
      # Generate a delay
      if (message[i] == 0) {
        delay = runif(1, min, med)
        ipdEncrypt[i] = delay
      } else {
        delay = runif(1, med, max)
        ipdEncrypt[i] = delay
      }
      currTime = currTime + delay # update time
      # Update the state of the buffer depending on the number of arrivals during that time.
      if (currTime <= ipdTime[index]) {
        currbuffer = currbuffer - 1
      } else {
        currbuffer = currbuffer + 1
      }
      index = index + 1
```

```r
      if (currbuffer > bufferSize){
        overflow = 1
        break
      }
      if (currbuffer < 1){
        underflow = 1
        break
      }
    }
  return(c(underflow,overflow))
}
probsU <- numeric(1000)
probsO <- numeric(1000)
for (t in (1:1000)) {
  m <- c(16,32)
  bufferNum <- c(2,6,10,14,18)
  underflow = 0
  overflow = 0
  count = 0
  for (i in (1:2)) {
      for (j in (1:5)) {
        output <- generateProb(m[i], bufferNum[j])
        underflow = underflow + output[1]
        overflow = overflow + output[2]
        count = count + 1
      }
  }
  probsU[t] = underflow/count
  probsO[t] = overflow/count
}
c(mean(probsU), mean(probsO))
```

```
## [1] 0.6470 0.1329
```