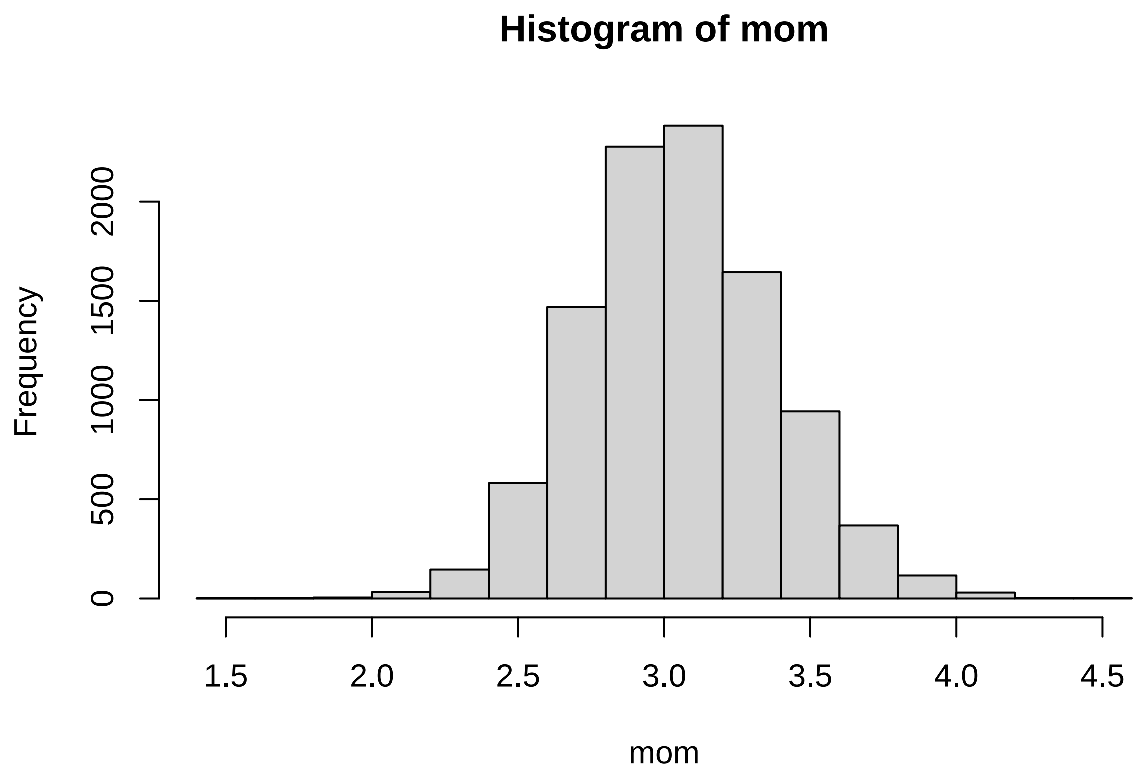
Utilizing Search Algorithms to Reduce Differences in Predicted and Actual Values

2025-11-10

1.

a.

```r
set.seed(1230)
n = 100
theta = 3
simnum = 10000

mypareto <- function(theta, n){
  u <- runif(n)
  return((1-u)^(-1/theta)-1)
}

mom <- numeric(simnum)

for(i in 1:simnum){
  x <- mypareto(theta, n)
  mom[i] <- 1/mean(x) + 1
}
hist(mom)
```



```
mean(mom)
```

```
## [1] 3.050374
```

```
var(mom)
```

```
## [1] 0.1073769
```

```
meansquaredestimate <- mean((mom - theta)^2)
meansquaredestimate
```

```
## [1] 0.1099038
```

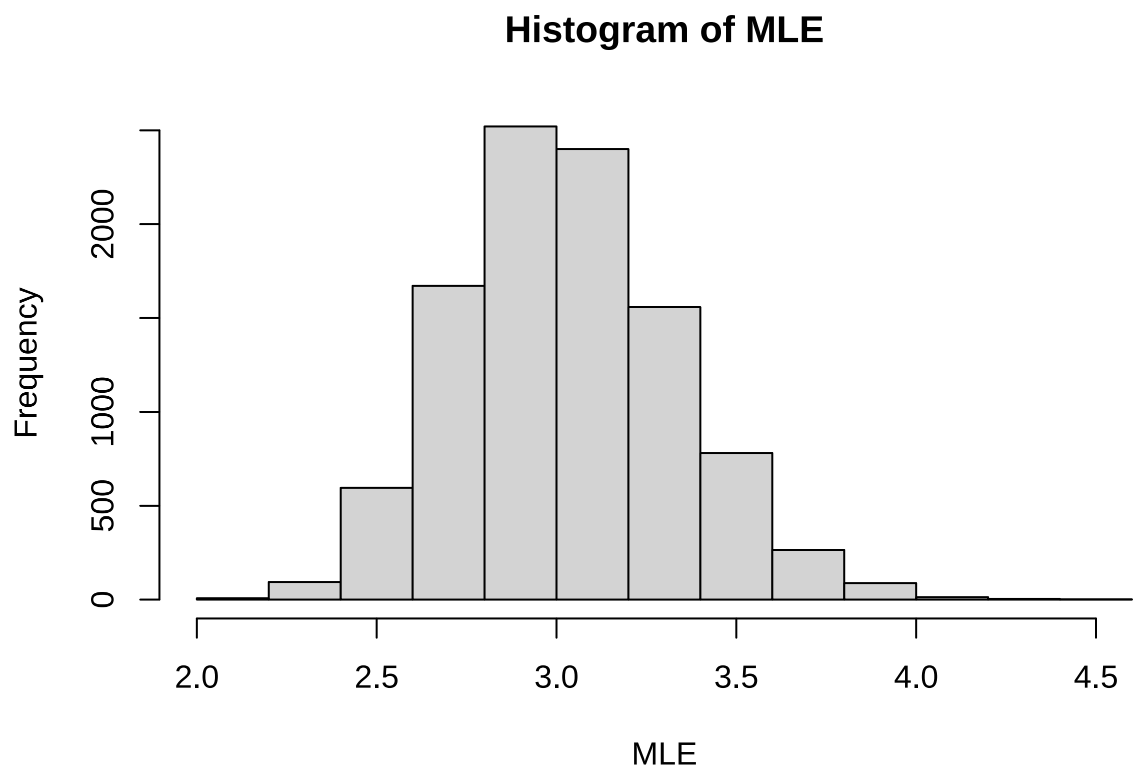
b. pareto probability function -> (1-u)^(-1/theta)-1

converting to log likelihood: ln((1-u)^(-1/theta)-1)

Differentiating with respect to theta: df/dtheta = -1/theta * ln(1-u) = (1/theta^2) * ln(1-u)

c.

```r
set.seed(1230)
n = 100
theta = 3
simnum = 10000

mypareto <- function(theta, n){
  u <- runif(n)
  return((1-u)^(-1/theta)-1)
}

MLE <- numeric(simnum)

for(i in 1:simnum){
  x <- mypareto(theta, n)
  MLE[i] <- n/sum(log(1+x))
}
hist(MLE)
```



```r
mean(MLE)
```

```
## [1] 3.025355
```

```r
var(MLE)
```

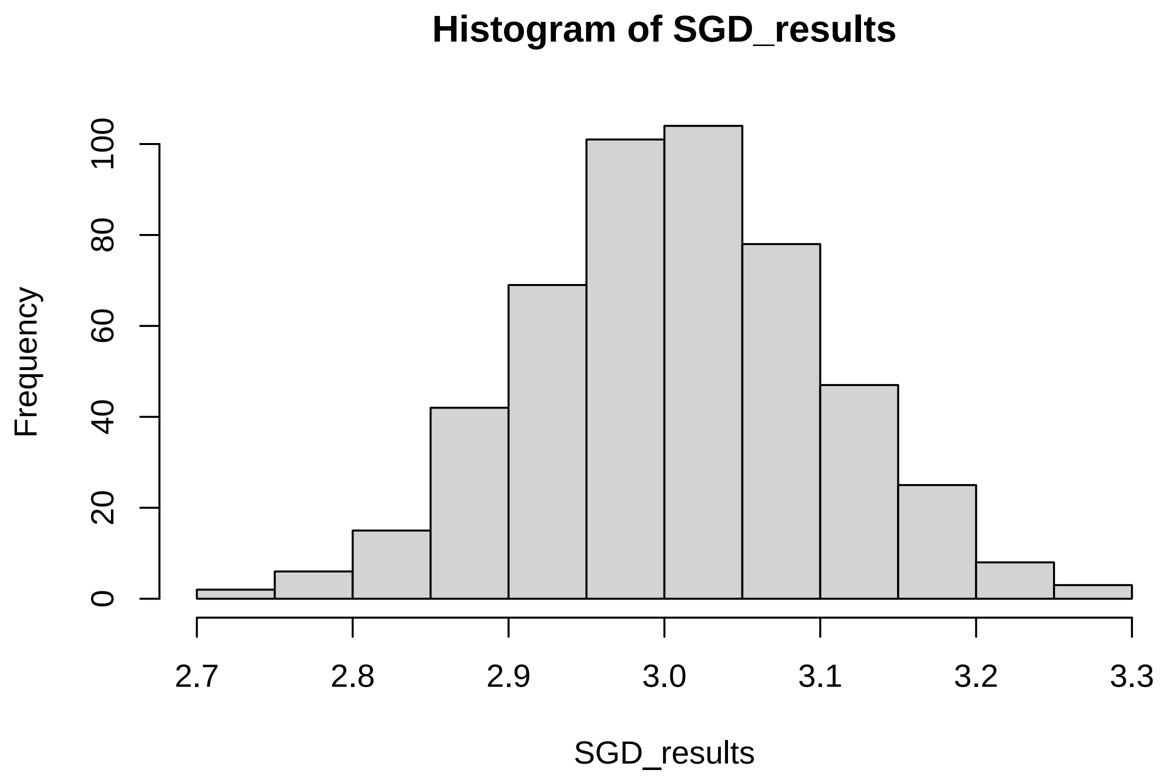
```
## [1] 0.09214051
```

```r
meansquaredestimate <- mean((MLE - theta)^2)
meansquaredestimate
```

```
## [1] 0.09277416
```

d.

```r
stochastic_gradient_descent <- function(x, learning_rate = 0.01, n_epochs = 100) {

  theta <- 1
  n <- length(x)

  theta_hist <- numeric(n_epochs)
  for (epoch in 1:n_epochs) {
    indices <- sample(1:n)
  for (i in indices) {
    grad_theta <- (1/theta) - log(x[i] + 1)
    theta <- theta + learning_rate * grad_theta

  if(theta <= 0) theta<- 1e-6
}
  theta_hist[epoch] <- theta
  }
  return(list(
  theta = theta,theta_hist = theta_hist))

sgd_result <- stochastic_gradient_descent(x, learning_rate = 0.001, n_epochs = 100)
}

set.seed(1230)
n = 1000
theta = 3
simnum = 500

mypareto <- function(theta, n){
  u <- runif(n)
  return((1-u)^(-1/theta)-1)
}

SGD_results <- numeric(simnum)

for(i in 1:simnum){
  x <- mypareto(theta, n)
  SGD <- stochastic_gradient_descent(x, learning_rate = 0.001, n_epochs = 1000)
  SGD_results[i] <- SGD$theta
}
hist(SGD_results)
```



```
mean(SGD_results)
```

```
## [1] 3.006865
```

```
var(SGD_results)
```

```
## [1] 0.00894835
```

```
meansquaredestimate <- mean((SGD_results - theta)^2)
meansquaredestimate
```

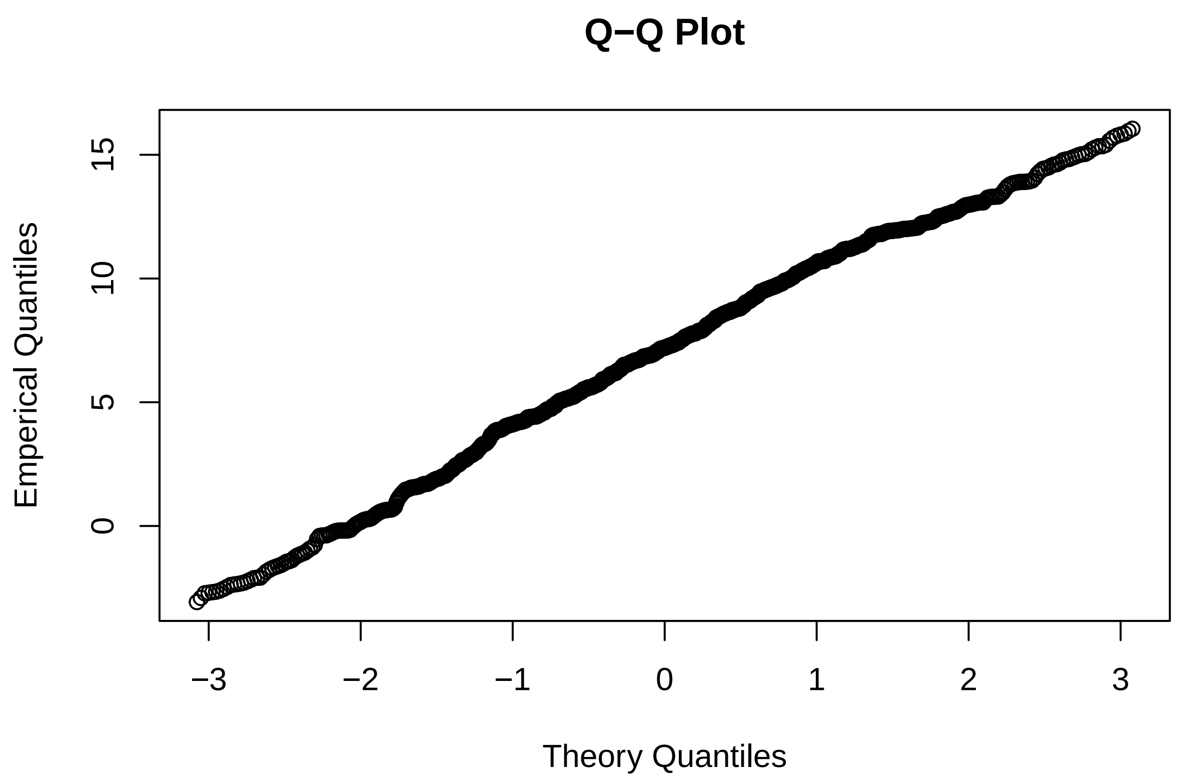
```
## [1] 0.008977575
```

e. Each simulation had mean squared errors that were very close to their variances. I guess it makes sense since both variance and means square error are different ways to quantify error. The histograms for all of them all seem normal too.

2.

a.

```r
set.seed(3701)
n <- 1000
data = read.csv("OlisCauchy.csv")

x <- data$x

quantiles <- seq(0.1,0.9, length.out = n)
empquant <- quantile(x, quantiles)
theoryquant <- qcauchy(quantiles)

plot(theoryquant, empquant,
     main = "Q-Q Plot",
     xlab = "Theory Quantiles",
     ylab = "Emperical Quantiles")
```



b.

```r
phi <- median(x)
realphi <- phi
phi
```

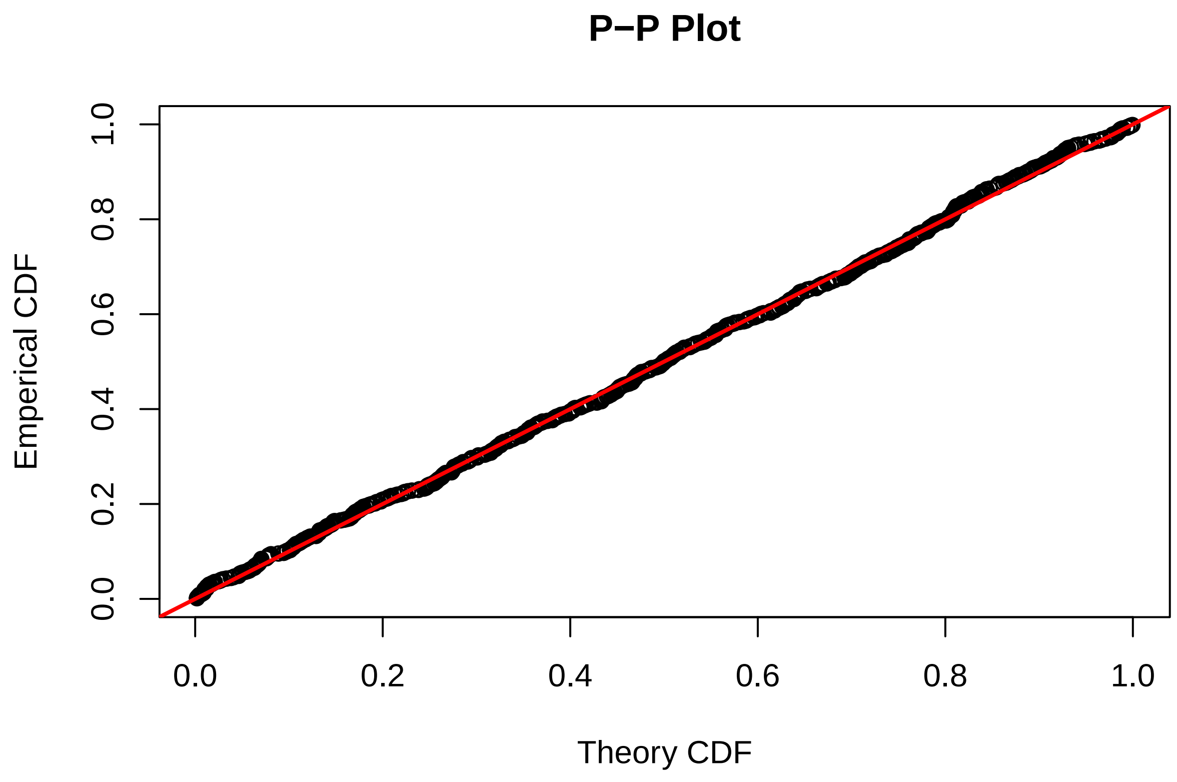
```
## [1] 7.200047
```

```r
gamma <- (quantile(x,0.75) - quantile(x, 0.25))/2
realgamma <- gamma
gamma
```

```
##      75%
## 3.263619
```

c.

```r
set.seed(3701)
n <- length(x)

empcdf <- (1:n)/(n+1)
theorycdf <- pcauchy(sort(x), phi, gamma)

plot(theorycdf, empcdf,
     main = "P-P Plot",
     xlab = "Theory CDF",
     ylab = "Emperical CDF")
abline(0, 1, col = "red", lwd = 2)
```



d.

```r
loglikelihood <- function(x, phi){
  -n*log(pi) + n*log(gamma) - sum(log(gamma^2 + (x - phi)^2))
}
tol<-10^(-6)
high<-1
low<--1
epsilon<-10^(-7)
```

```r
check_func<-function(phi){
  loglikelihood(x, phi + epsilon) - loglikelihood(x, phi - epsilon) >0
}
while(high-low>tol){
  mid<-(high+low)/2
  if(check_func(mid)){
    high<-mid
  }
  else{
    low<-mid
  }
}
final_phi <- (high + low)/2
final_phi
```

```
## [1] -0.9999995
```

e.

```r
gradient_descent <- function(x, learning_rate = 0.01, n_iterations = 1000) {

  phi <- median(x)
  gamma <- (quantile(x,0.75) - quantile(x, 0.25))/2
  n <- length(x)

  phi_hist <- numeric(n_iterations)
  gamma_hist <- numeric(n_iterations)

  for (i in 1:n_iterations) {

    gradient_phi <- sum(2*(x-realphi)/(gamma^2 + (x-phi)^2))
    gradient_gamma <- n/gamma - sum((2*gamma)/(gamma^2 + (x-phi)^2))

    phi <- phi - learning_rate * gradient_phi
    gamma <- gamma - learning_rate * gradient_gamma

    phi_hist[i] <- phi
    gamma_hist[i] <- gamma
  }
  return (list(phi = phi, gamma = gamma))

}
gradient_descent(x, learning_rate = 0.001, n_iterations =1000)
```

```
## $phi
## [1] 7.002108
##
## $gamma
##      75%
## 25.49646
```