Simple Programming: Simulation

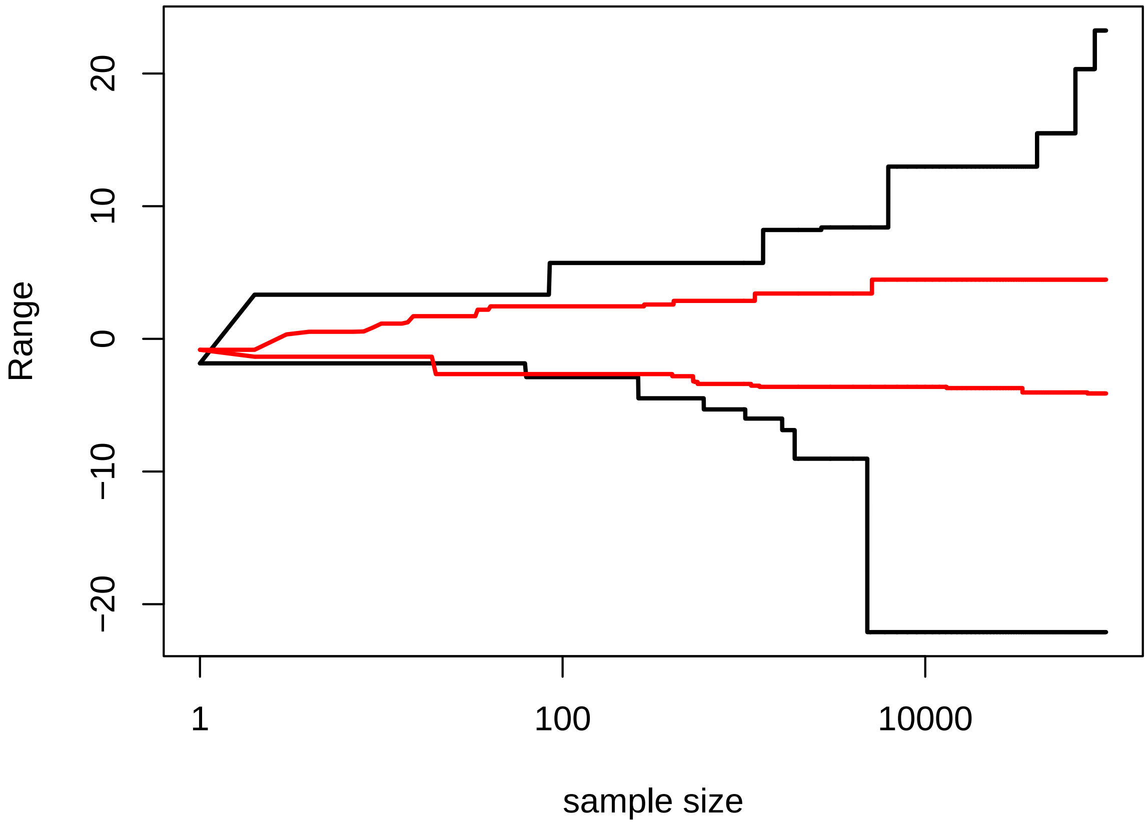
Thomas Lumley

Biostatistics

2004-10-14

Example

Range of t and Normal distributions



Steps

1. Generate 20,000 numbers from the t distribution
2. Find the maximum of the first 2,3,4,5,6,... (or perhaps 1,3,10,30,100,..)
3. Plot vs sample size
4. Find corresponding minimums
5. Uses `lines` to add then
6. Generate 20,000 numbers from Normal distribution

Steps

7. Find maximums, use `lines` to add them

8. Find minimums, use `lines` to add them.

Storing 20,000 numbers is no big deal, but if it had been 20,000,000 we would need to find some clever way to work divide the problem into chunks.

Generating data

rnorm(n) generates random numbers from Normal distribution

dnorm(x) computes Normal density (or log-density)

pnorm(x) computes Normal CDF (either tail; or log)

qnorm(p) computes Normal quantiles (either tail; or log)

There are similar sets of functions for many distributions (eg `gamma`, `weibull`, `pois`, `binom`, `t`, `F`,...)

Generating data

`rnorm` has two other parameters with defaults `mean=0` and `sd=1`.

We want 20,000 standard Normals

```
lotsofnormals <- rnorm(20000)
```

Finding maximums

A dumb algorithm

```
maximums<-numeric(20000)
for(i in 1:20000){
maximums[i]<-max(lotsofnormals[1:i])
}
```

On a smaller problem it wouldn't matter that this was dumb. On this example it takes 1.5 minutes, which is too long. [use `system.time()` to time things].

Finding maximums

If we have the maximum up to n, the maximum up to $n-1$ is either the same or is the nth number

```
maximums<-numeric(20000)
maximums[1]<-lotsofnormals[1]
for(i in 2:20000){
maximums[i]<-max(lotsofnormals[i], maximums[i-1])
}
```

This takes 1.8 seconds, which is Good Enough.

Finding maximums

Often there is an R function that will work faster than some sort of loop.

```
help.search("max")
```

finds a number of possible help pages, including

```
cumsum    Cumulative Sums, Products, and Extremes
```

and `help(cumsum)` reveals the `cummin` and `cummax` functions.

```
maximums <- cummax(lotsofnormals)
```

takes too little time to be reliably measured (about 0.01s)

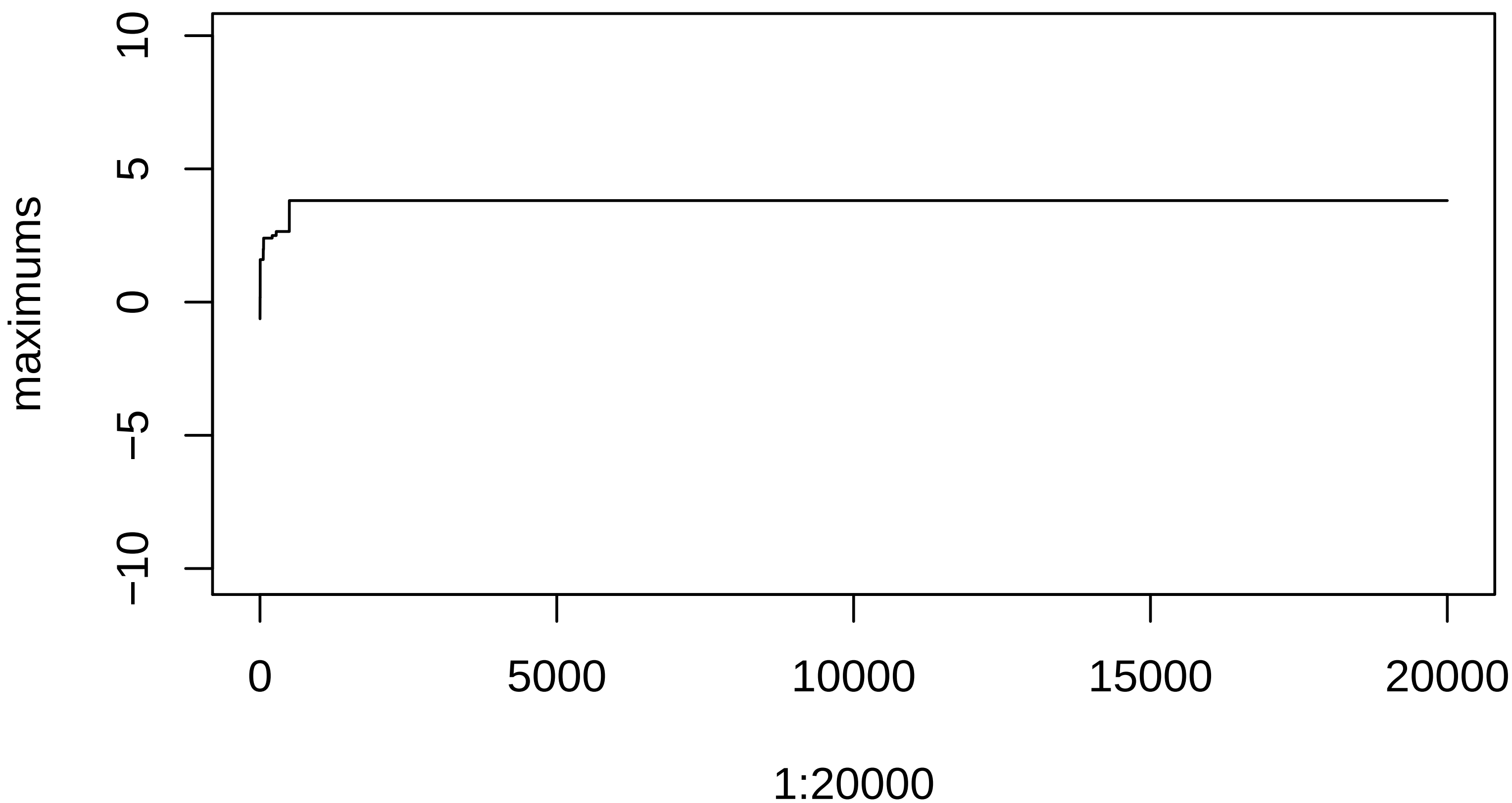
Plotting

We need to allocate enough space for the t-distribution and normal distribution maximum and minimum lines.

One approach is to compute all the lines first, then work out how much space is needed:

```
lotsofnormals <- rnorm(20000)
maximums <- cummax(lotsofnormals)
minimums <- cummin(lotsofnormals)
lotsoft <- rt(20000,df=7)
tmax <- cummax(lotsoft)
tmin <- cummin(lotsoft)
ylim<-c(min(tmin[20000], minimums[20000]),
        max(tmax[20000], maximums[20000]))
plot(1:20000, maximums, type="l",ylim=ylim)
```

Plotting

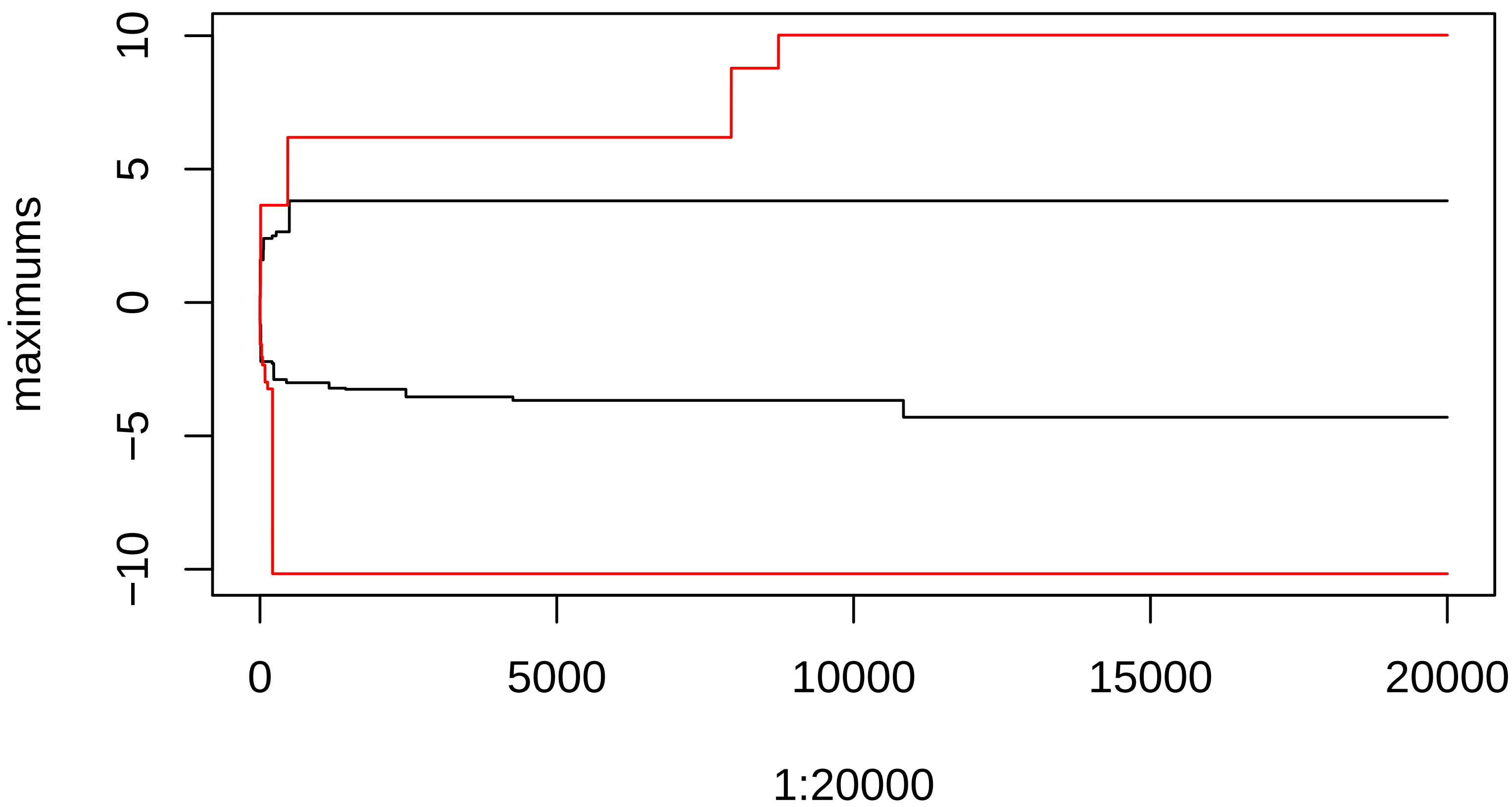


Plotting

Now add the other lines

```
lines(1:20000, minimums)
lines(1:20000, tmin, col="red")
lines(1:20000, tmax, col="red")
```

Plotting

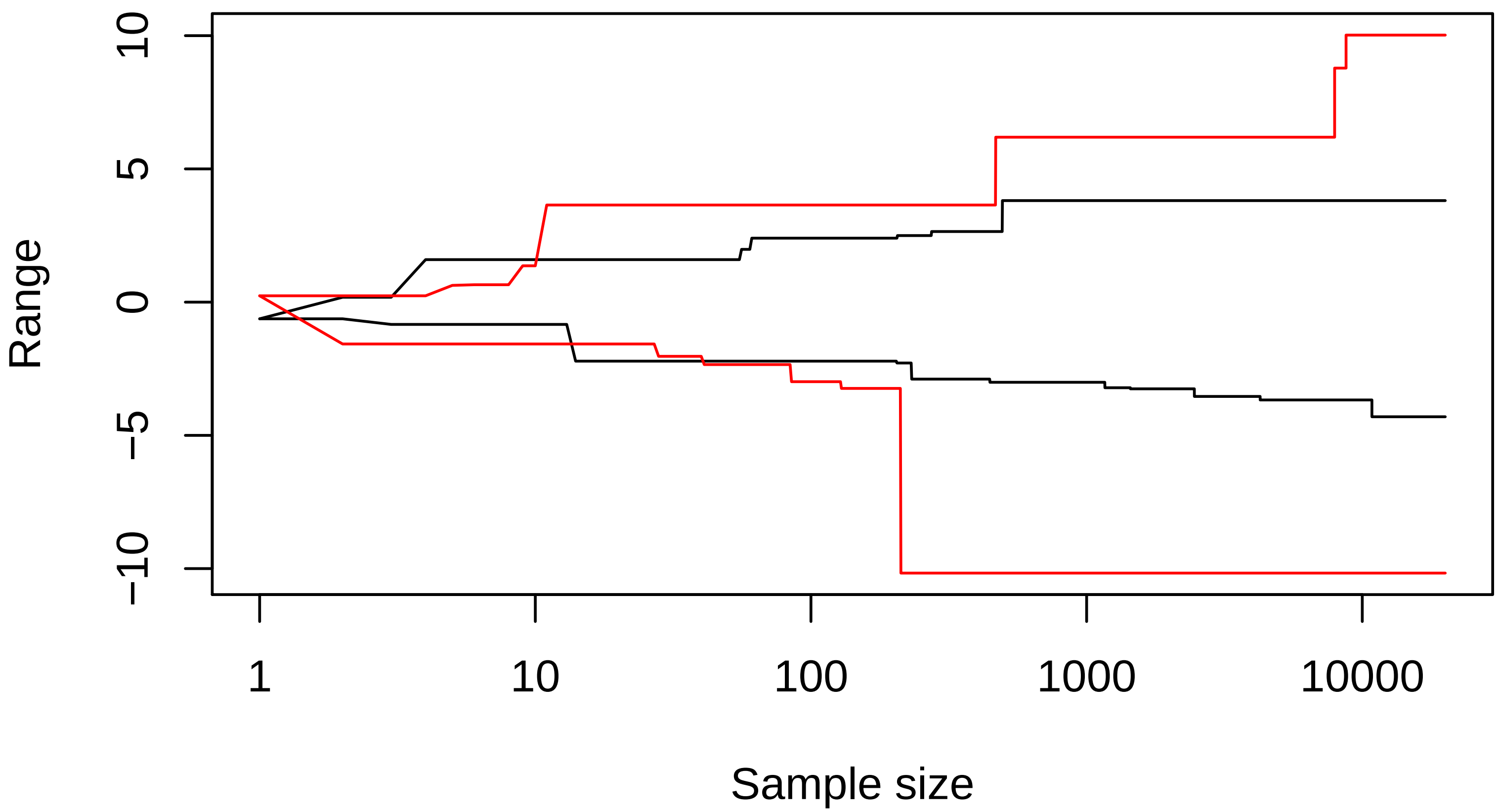


Details

We wanted a logarithmic scale on the x-axis, and we need better axis labels

```
lotsofnormals <- rnorm(20000)
maximums <- cummax(lotsofnormals)
minimums <- cummin(lotsofnormals)
lotsoft <- rt(20000,df=4)
tmax <- cummax(lotsoft)
tmin <- cummin(lotsoft)
ylim<-c(min(tmin[20000], minimums[20000]),
        max(tmax[20000], maximums[20000]))
plot(1:20000, maximums, type="l",ylim=ylim, log="x",
     xlab="Sample size",ylab="Range")
lines(1:20000, minimums)
lines(1:20000, tmin, col="red")
lines(1:20000, tmax, col="red")
```

Details



Reproducibility

When calculations are based on random numbers it is important to be able to reproduce them (as the graph indicated).

The random numbers are generated by applying some complicated algorithm to a seed value. Specifying a particular seed value will give reproducible results.

The exact format of a seed is complicated and varies by which generator you are using (the default generator uses 624 integers plus one number between 1 and 624), so the simpler `set.seed()` function is useful. It takes a single number as argument.

```
set.seed(42)
```

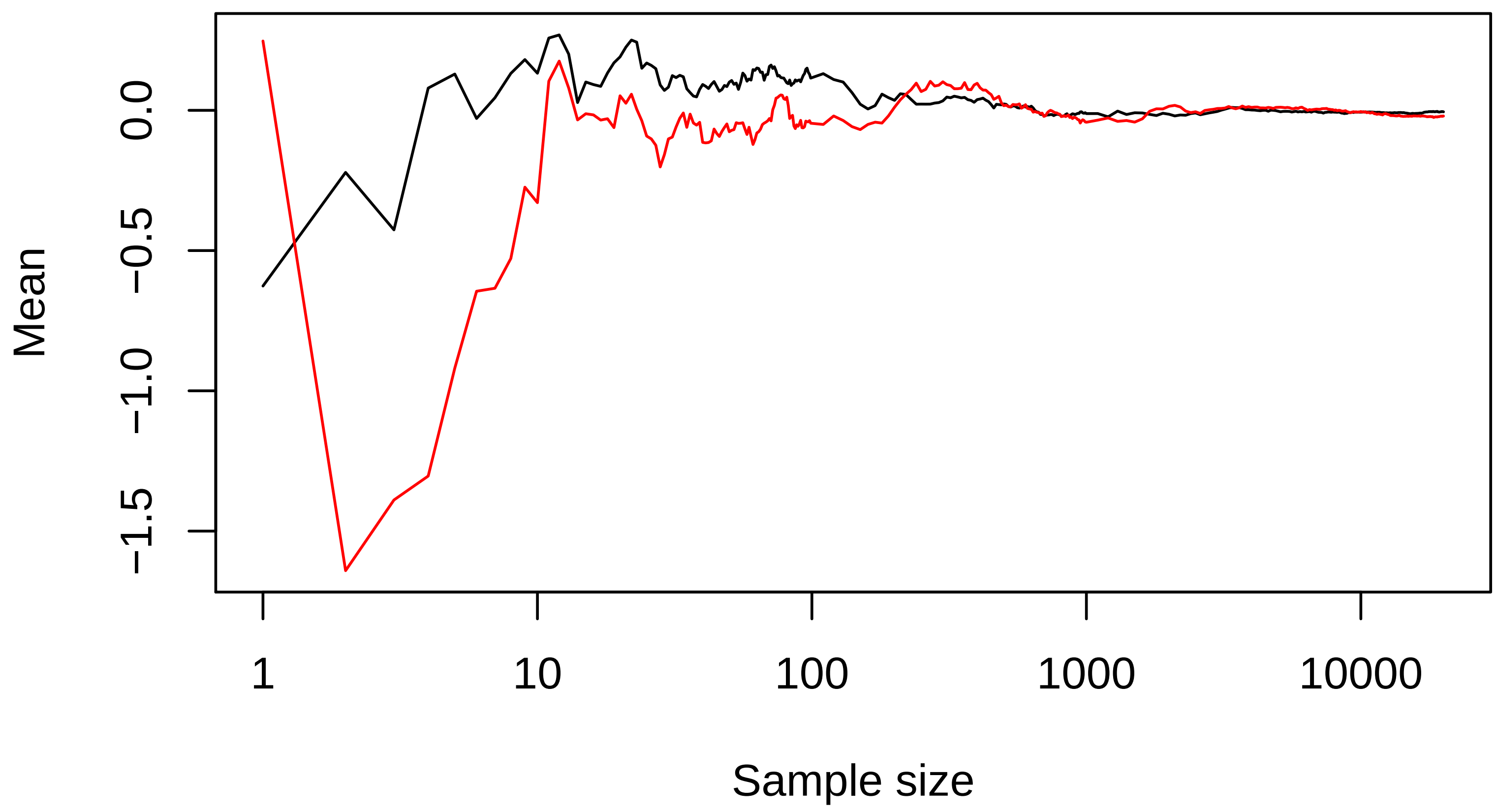
Do this before you generate any random numbers.

Means

We could do the same thing for the mean, to compare

```
set.seed(1)
lotsofnormals <- rnorm(20000)
means <- cumsum(lotsofnormals)/(1:20000)
lotsoft <- rt(20000,df=4)
tmean <- cumsum(lotsoft)/(1:20000)
ylim<-range(tmean,means)
plot(xs, means[xs], type="l",ylim=ylim, log="x",
     xlab="Sample size",ylab="Mean")
lines(xs, tmean[xs], col="red")
plot(1:20000, means, type="l",ylim=ylim, log="x",
     xlab="Sample size",ylab="Mean")
lines(1:20000, tmean, col="red")
```

Means



Means: Cauchy distribution

The Cauchy (or t_1) distribution has pdf

$$f(x) = \frac{1}{\pi(1+x^2)}$$

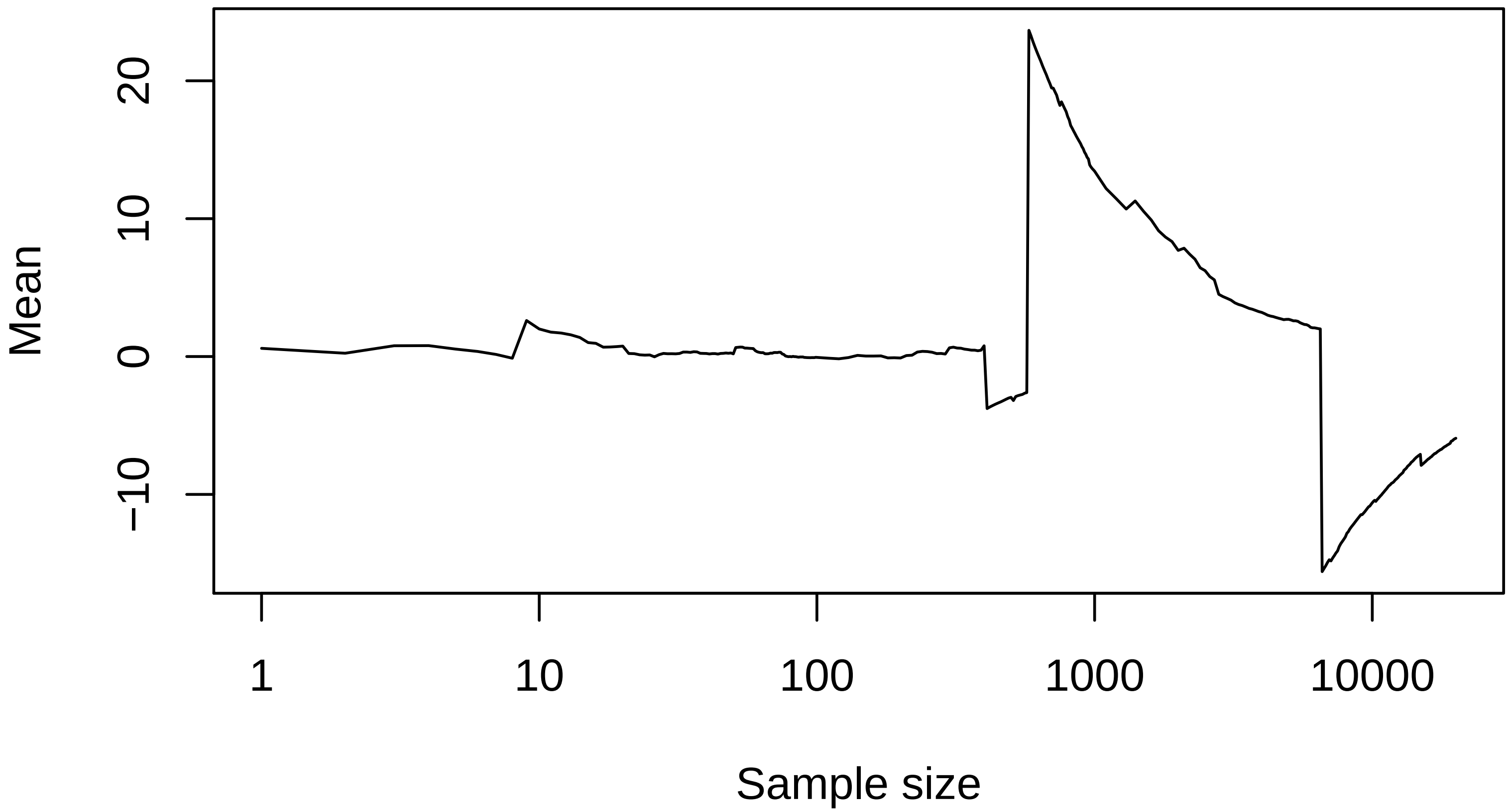
Since

$$\int_0^\infty x f(x)\, dx = \int_0^\infty \frac{x}{\pi(1+x^2)}\, dx$$

does not converge, the Cauchy distribution has no mean.

Samples from the Cauchy distribution are just sets of numbers, and always have a finite mean, so what happens to the sample average as $n \to \infty$ to stop it converging?

Means: Cauchy distribution

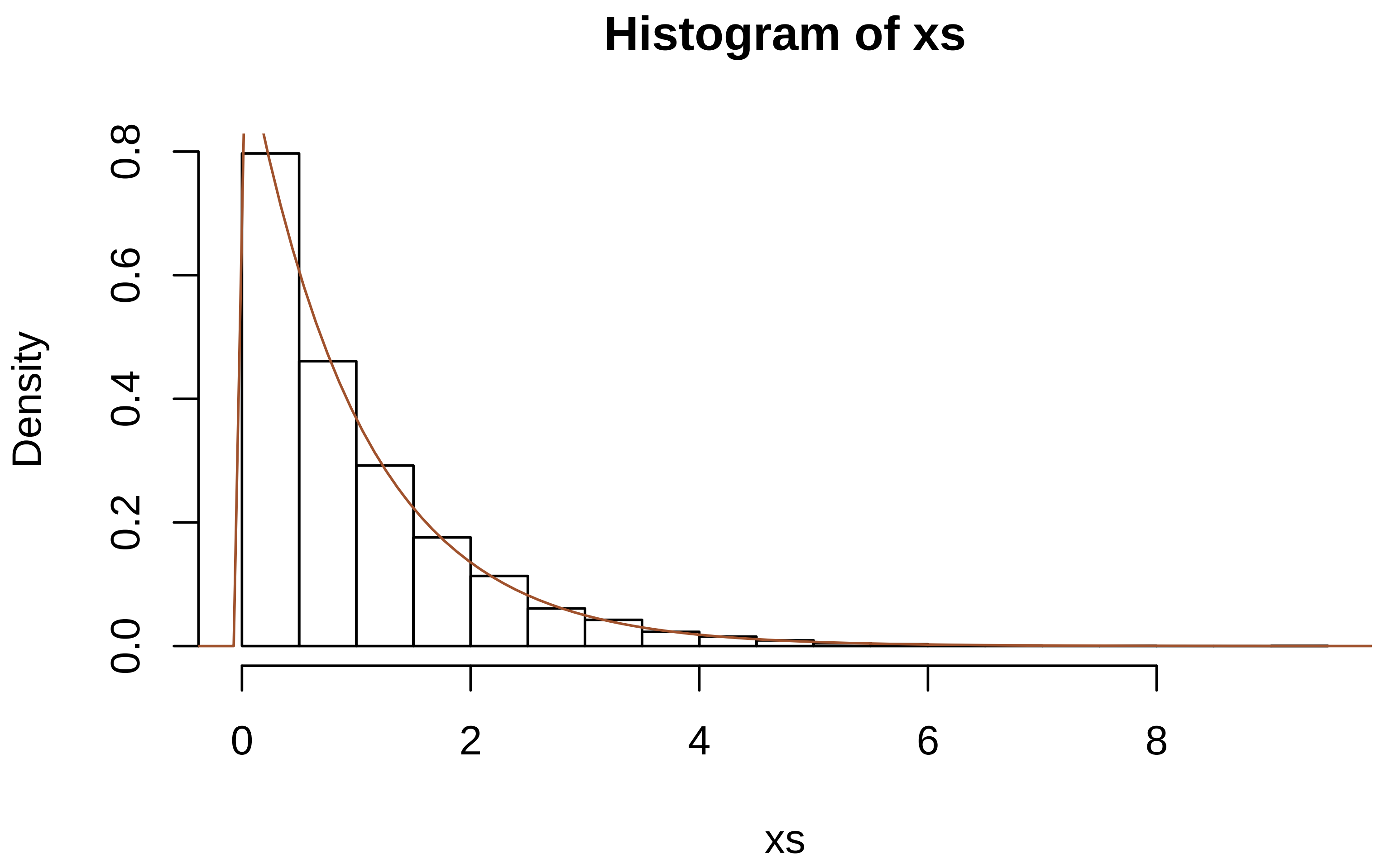


Central Limit Theorem

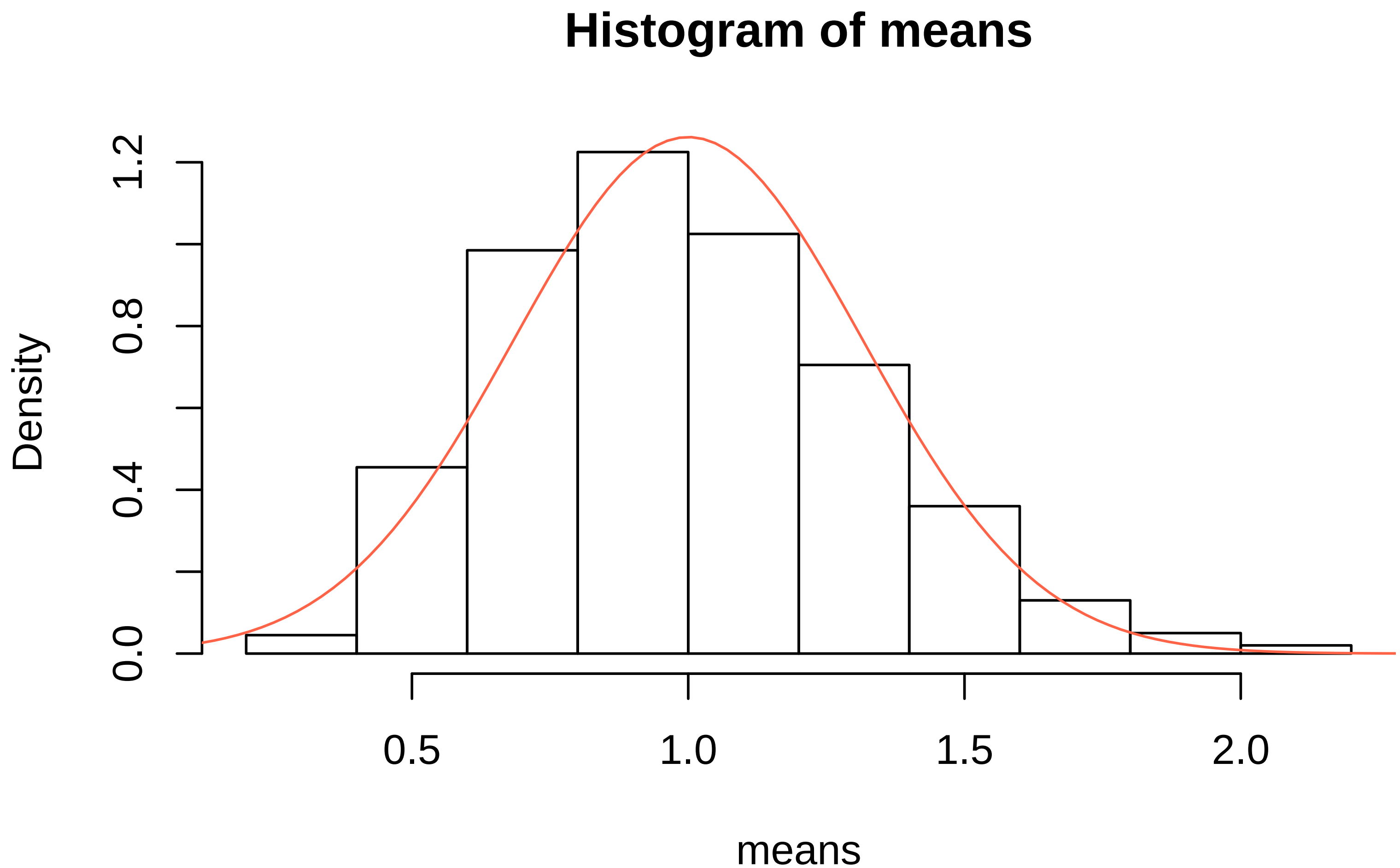
In large samples, means are approximately Normally distributed regardless of the distribution of the data (as long as the distribution has finite mean and variance)

```
xs <- matrix(rgamma(10*1000,1,1), nrow=10)
means <- colMeans(xs)
hist(xs, prob=TRUE)
curve(dgamma(x,1,1),add=TRUE,col="sienna")
hist(means, prob=TRUE)
curve(dnorm(x,mean=1,sd=1/sqrt(10)), add=TRUE,col="tomato")
qqnorm(means)
```

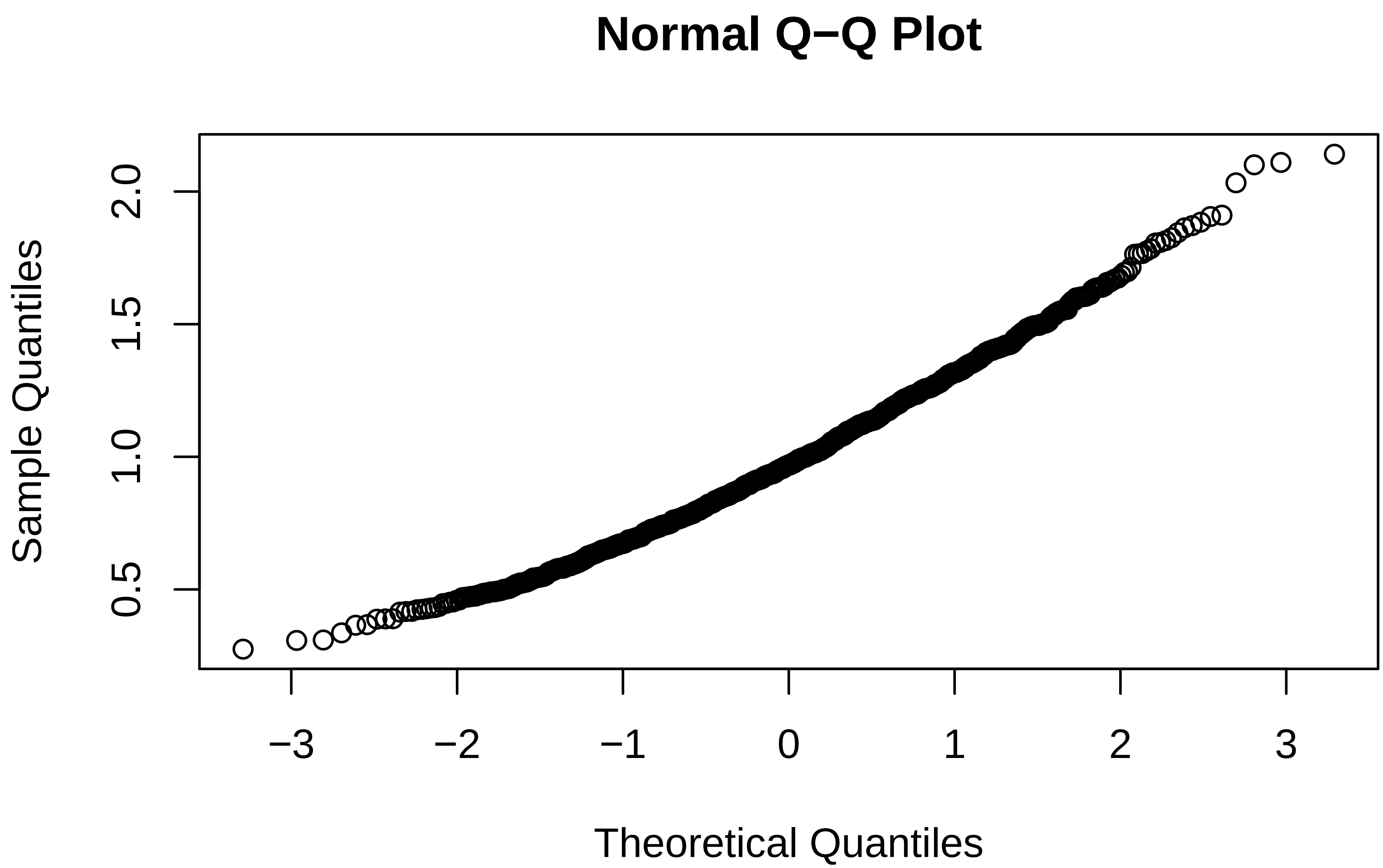
Central Limit Theorem



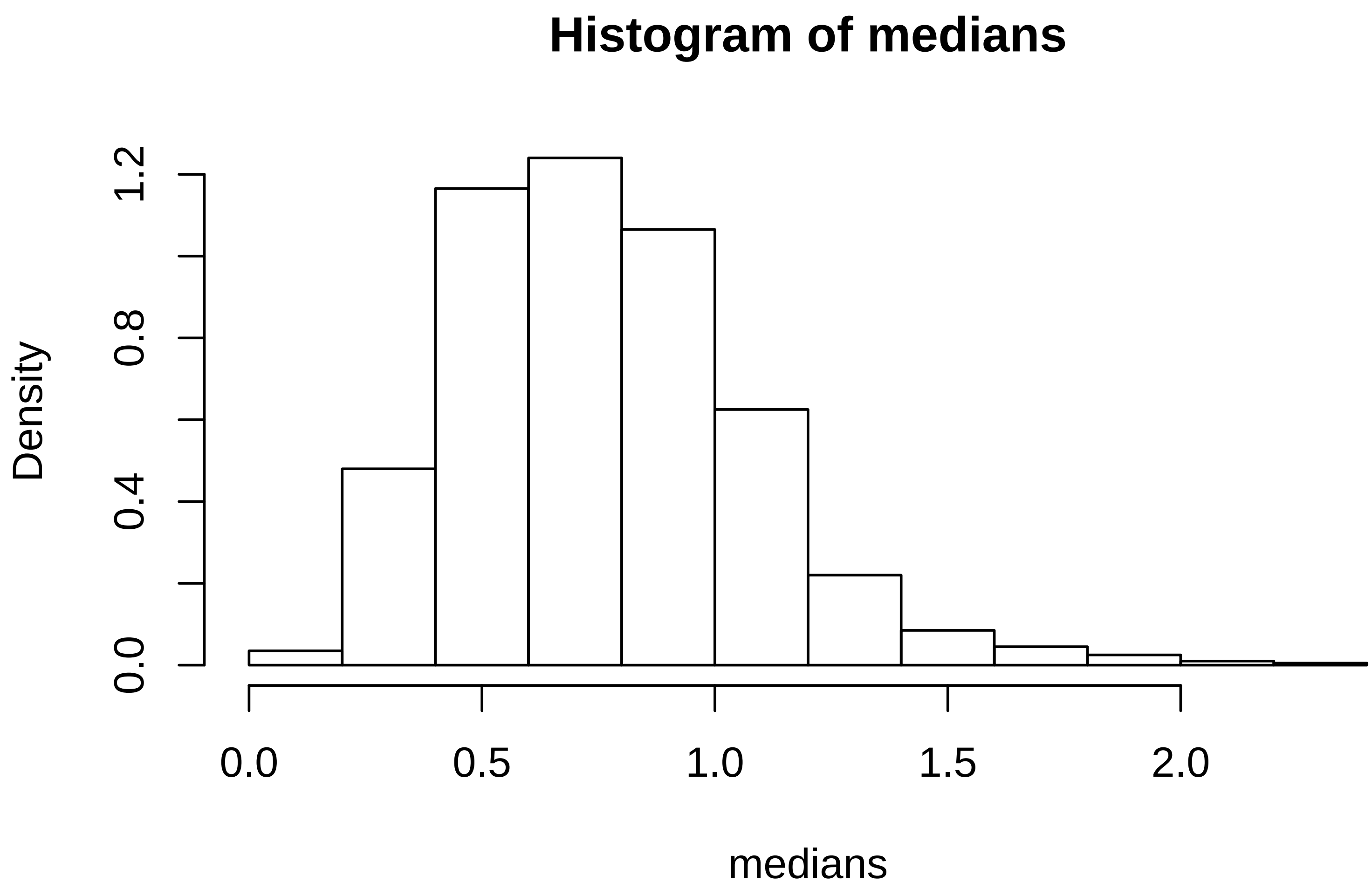
Central Limit Theorem



Central Limit Theorem

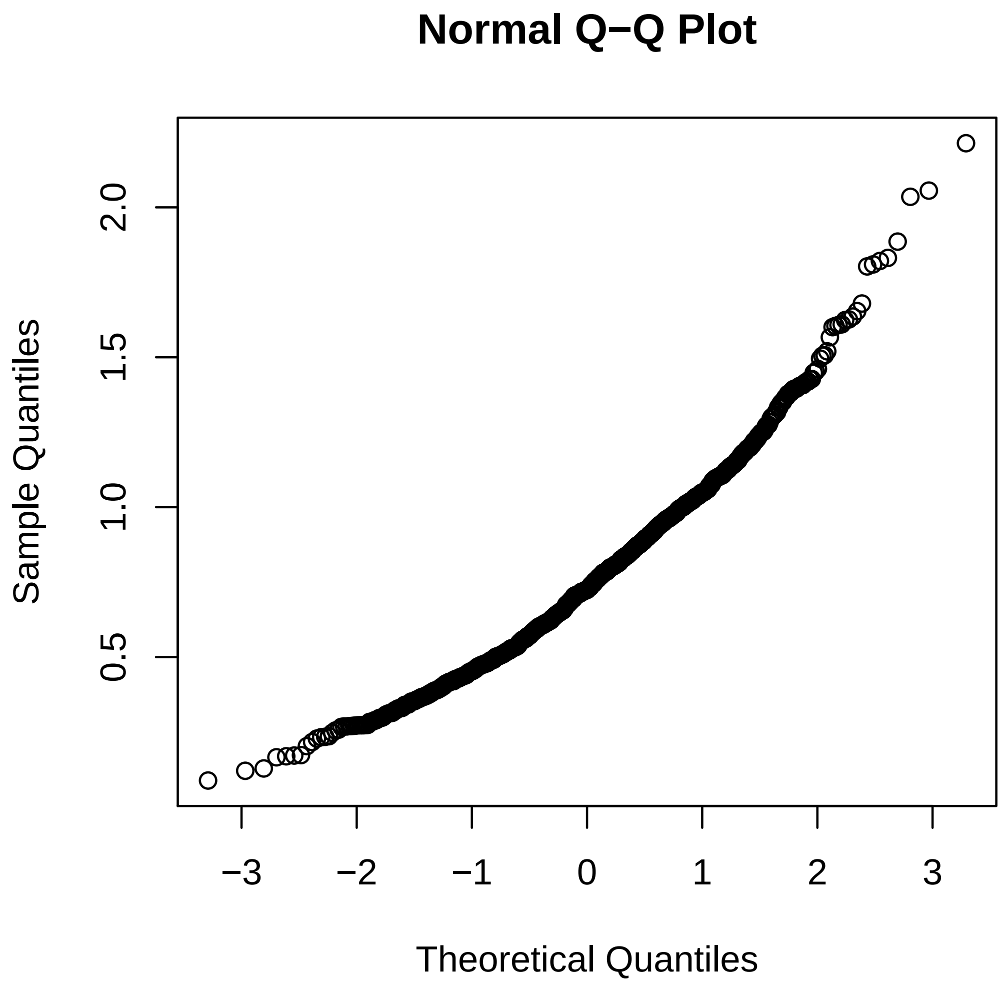


Medians



Histogram of medians
Density
0.0
0.4
0.8
1.2
0.0
0.5
1.0
1.5
2.0
medians

Medians



Normal Q−Q Plot
Sample Quantiles
0.5
1.0
1.5
2.0
−3
−2
−1
0
1
2
3
Theoretical Quantiles

Stopping rules

Suppose we conduct a randomized trial with three interim and one final analysis, and that we stop the trial as soon as the nominal p-value is less than 0.0183 (the 'Pocock' design). This means the Z statistic is greater than 2.359.

What does the distribution of the mean, the Z-statistic, the p-value look like?

For simplicity, assume that the sample mean of the observations for 1/4 of the trial is $N(m, 1)$, where m is the true mean.

```
m<-0 ## true mean

alt1<-rnorm(10000,mean=m)
time<-ifelse(abs(alt1)>2.359,1,2)
```

Stopping rules

```
alt2<-ifelse(time<=1,
             alt1,
             alt1+rnorm(10000,mean=m))
time<-ifelse(abs(alt2/sqrt(time))>2.359,time,3)

alt3<-ifelse(time<=2,
             alt2,
             alt2+rnorm(10000,mean=m))
time<-ifelse(abs(alt3/sqrt(time))>2.359,time,4)

alt4<-ifelse(time<=3,
             alt3,
             alt3+rnorm(10000,mean=m))
```

Stopping rules

Now we can analyse the results. With $m = 0$ we get

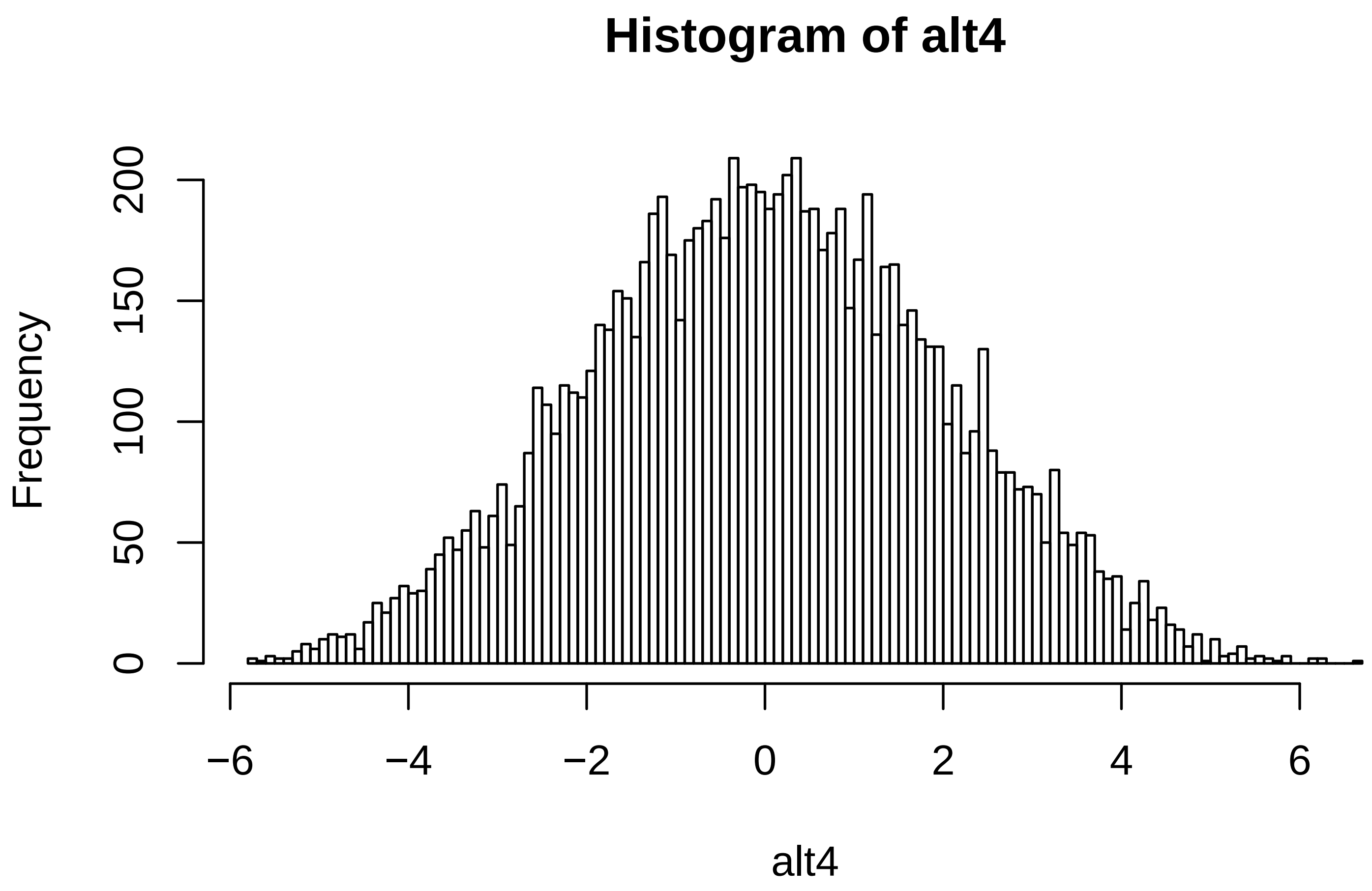
```
> mean( abs(alt4/sqrt(time)) > 2.359)
[1] 0.0516
```

so the trial has the correct Type I error rate.

The total looks like

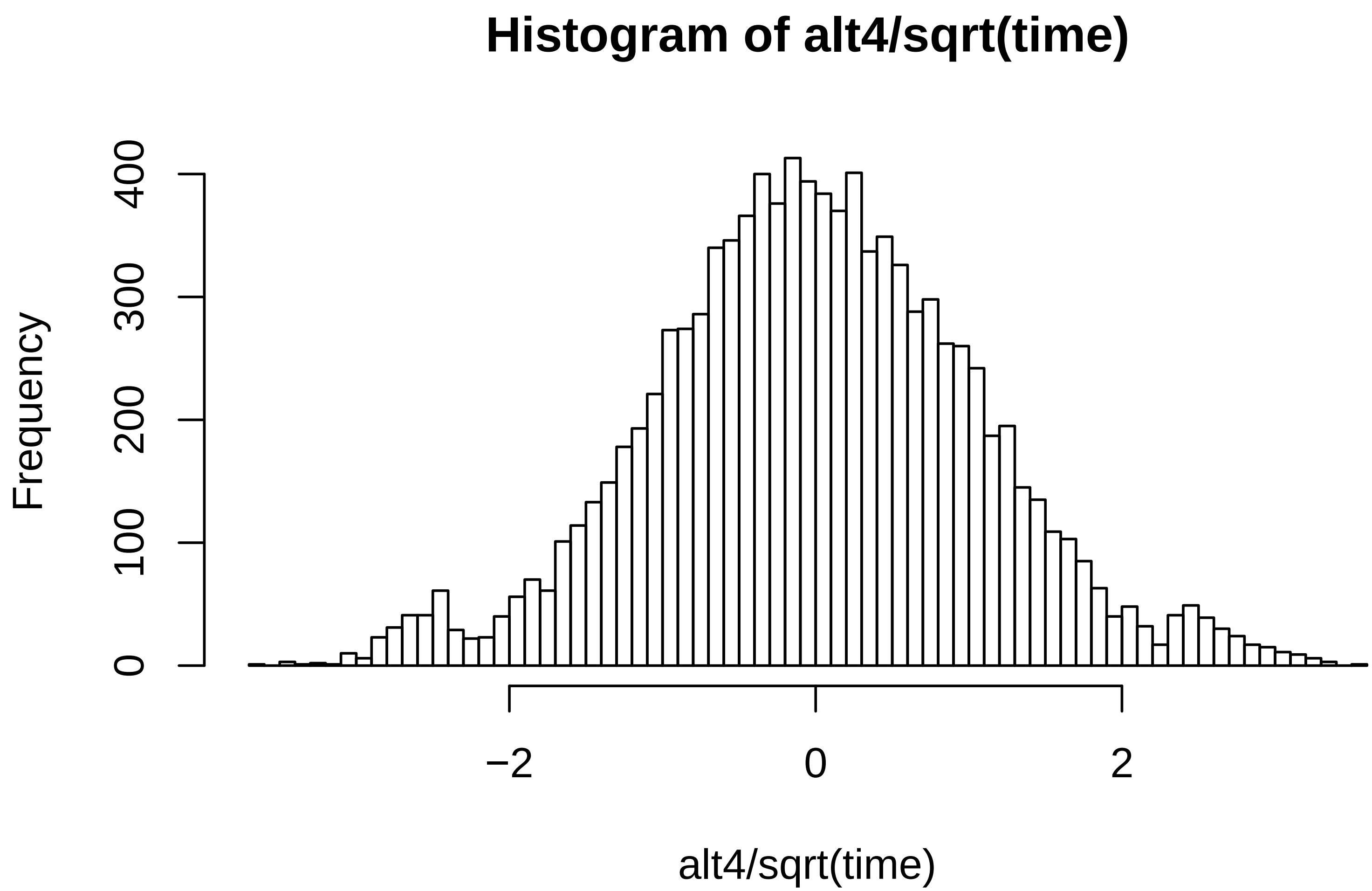
```
> pdf("~/TEACHING/b514/gseq-mean-0.pdf",height=4,width=6)
> hist(alt4,breaks=100)
> dev.off()
```

Stopping rules



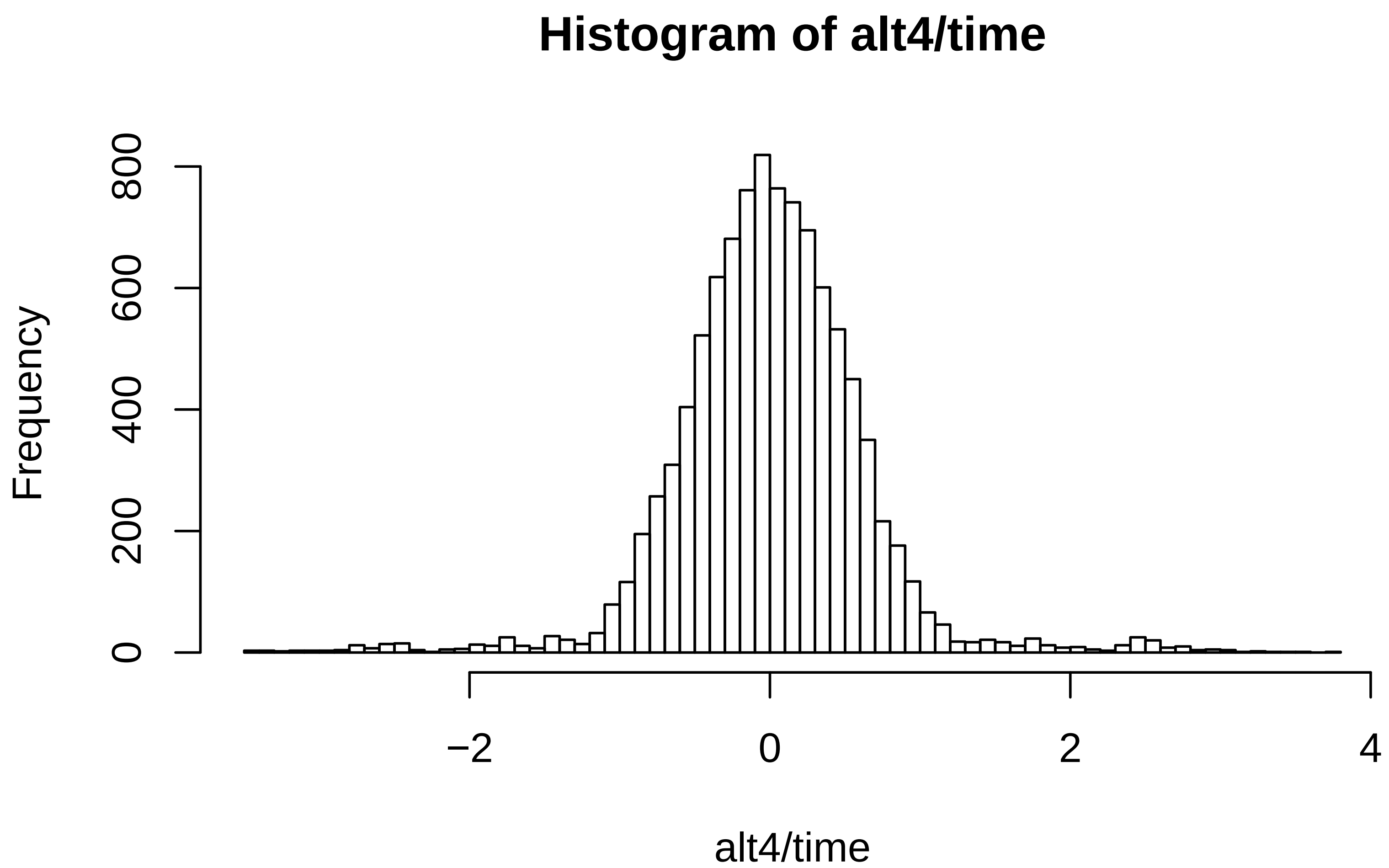
This is superficially Normal. The Z-statistic, dividing by the estimated standard error, looks less Normal:

Stopping rules



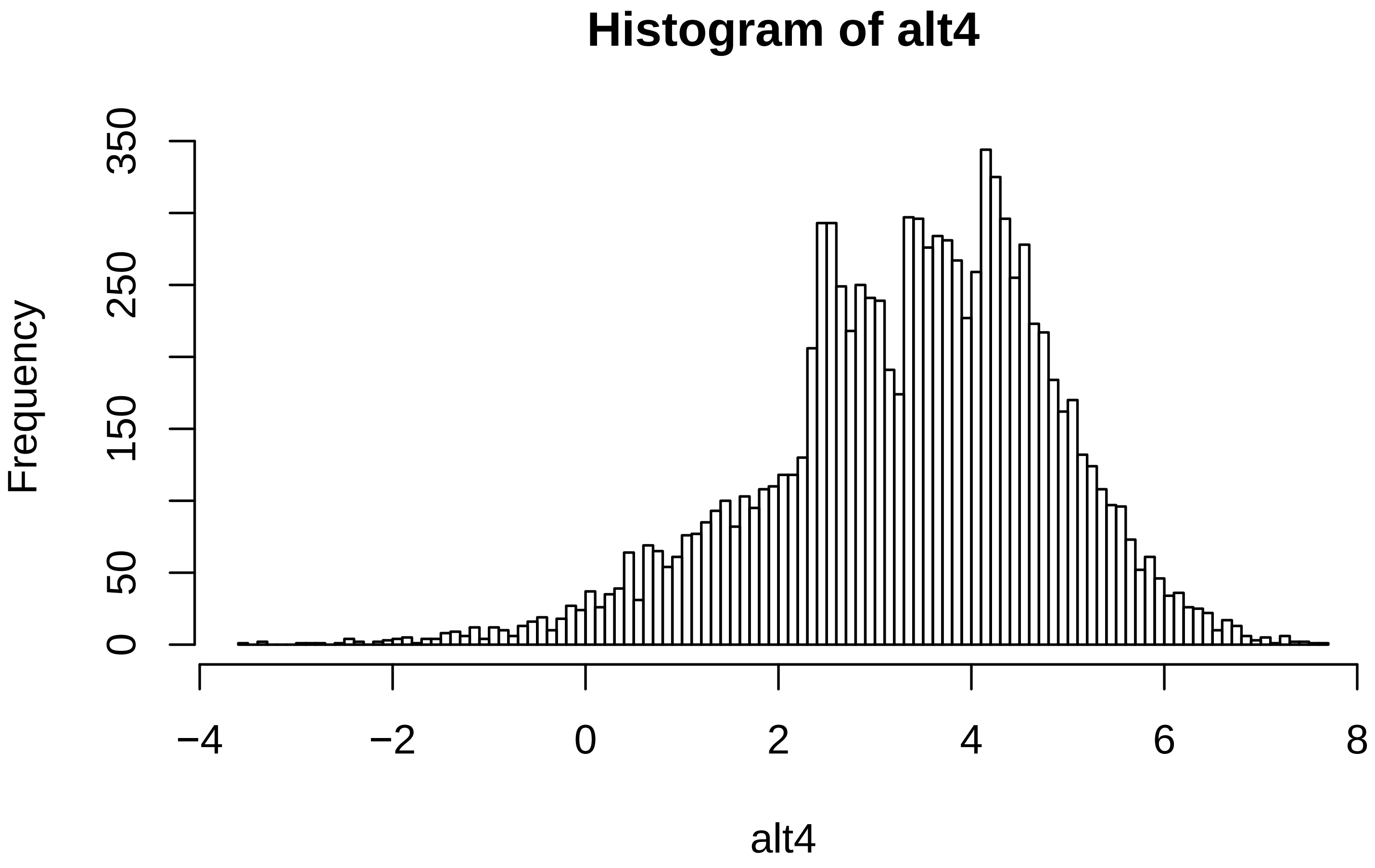
and the mean looks even less Normal.

Stopping rules

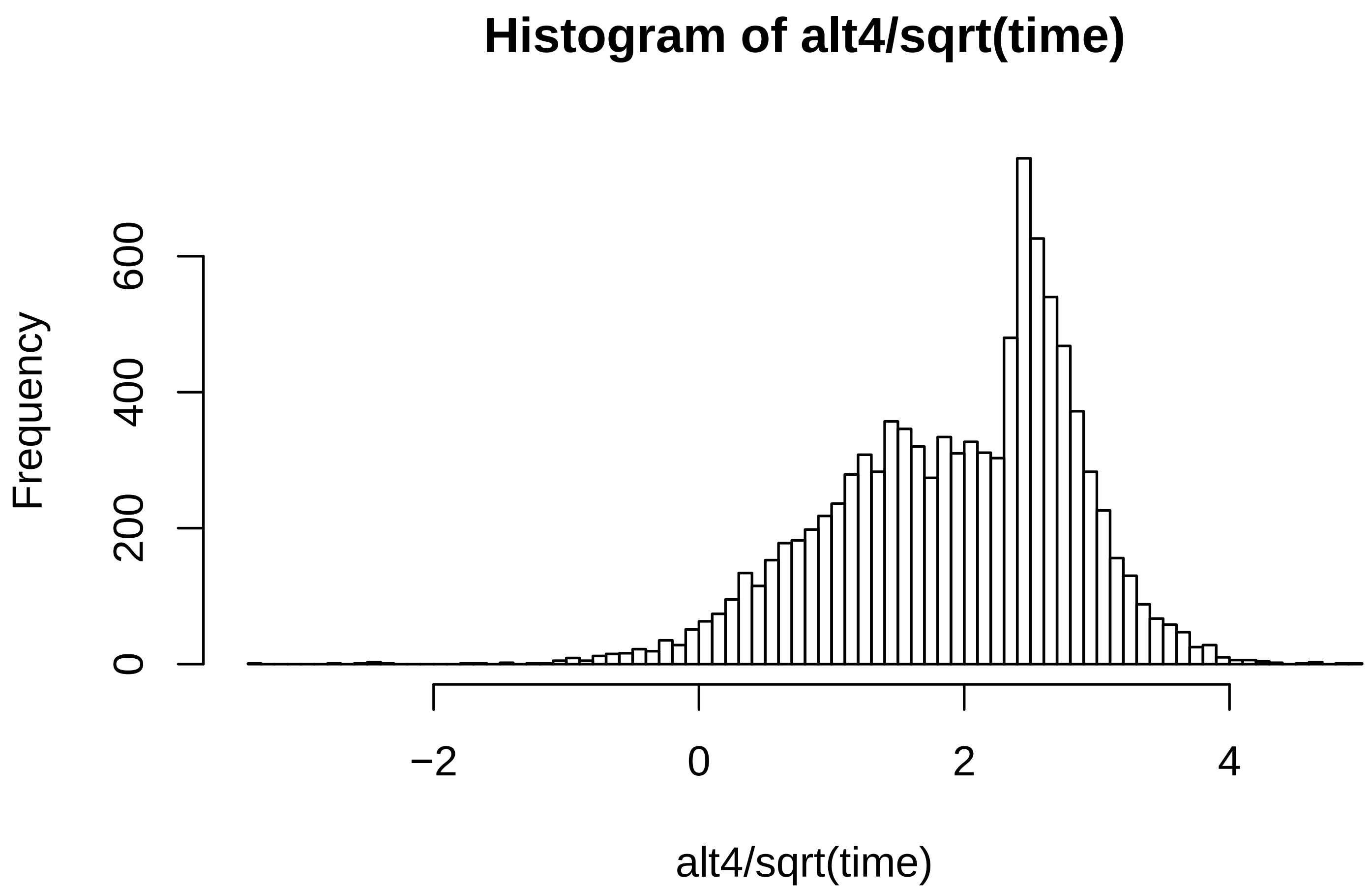


Stopping rules

At $m = 1$ the impact of sequential testing is much clearer



Stopping rules



Stopping rules