Maximum Likehood Estimator

# Introduction

maximum likelihood estimation is one common method for estimating paremater in a parametric model, just like method of moments. We assume be IID with *PDF* , define the likelihood function as

And the log likelihood function is defined by .

The *maximum likelihood estimator* MLE, denoted by , is the value of that maximizes

# Derivation

### Point mass at

Notice the probability mass function for point mass distribution is in and 0 elsewhere. In fact, have only two values:1 and 0, so does the which is the product of many s. We choose the MLE for point mass distribution to be mode among .

### Bernoulli

The probability function is for , the unknown parameter is .

where . Hence,

Take the derivation and set it equal to 0 gave us

### Binomial(N,p)

### Geometric(p)

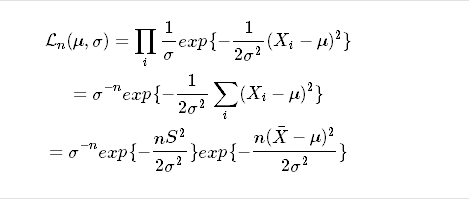
where

### Poisson Distribution()

### Uniform(a,b)

if all else To maximize, we need to minize , the boundary condition is to choose and

### Normal(, )



where ,

Solving eqations and we conclude that and

### Exponential()

### Gamma()

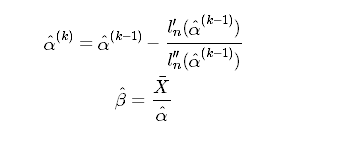
Solve and yield:

and

where

, is called Digamma Function However, no closed form exists for , only numerical solution exist.

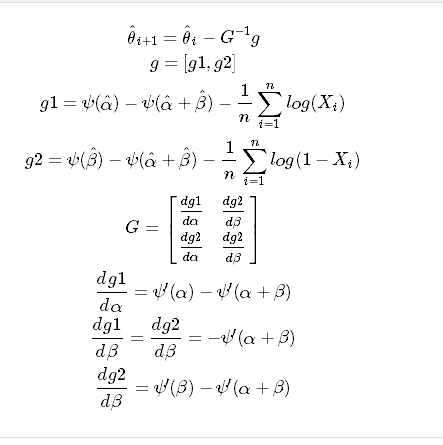
we calculate the numerical estimation using Newton-Raphson [Refernce](http://bioops.info/2015/01/gamma-mme-mle/)



### Beta()

solve and yield no analytical solution. However, numerical solution via Newton-Raphson mehod does exist. [For reference, check page 27 on this paper](https://scholarsarchive.byu.edu/cgi/viewcontent.cgi?article=2613&context=etd)

We set iteratively:



### Student t distribution

We cannot get closed-form solution.

### Chi-Squared Distriubtion

But [in fact](http://mathworld.wolfram.com/DigammaFunction.html), digamma function is not monotone. Therefore it doesn’t have an inverse function at all. So the formula above is not right! In other words, the student t distributions doesn’t have a closed-form expression for MLE.

# Program

MLE\_Beta<-function(x, iteration=100){  
 mu=mean(x)  
 s2<-var(x)  
 theta<-c(mu\*(-1+ mu\*(1-mu)/s2),  
 (1-mu)\*(-1+ mu\*(1-mu)/s2)  
 )  
 G<-matrix(0, ncol=2, nrow=2)  
 g<-matrix(0, nrow=2, ncol=1)  
 for(i in 1:iteration){  
 g[1,1]=digamma(theta[1])-digamma(theta[1] + theta[2])-sum(log(x))/length(x)  
 g[2,1]=digamma(theta[2])-digamma(theta[1] + theta[2])-sum(log(1-x))/length(x)  
 G[1,1]=trigamma(theta[1])-trigamma(theta[1]+theta[2])  
 G[1,2]=-trigamma(theta[1]+theta[2])  
 G[2,1]=-trigamma(theta[1]+theta[2])  
 G[2,2]=trigamma(theta[2])-trigamma(theta[1]+theta[2])  
 theta<-theta-solve(G) %\*% g   
 }  
 return( list(shape1=theta[1], shape2=theta[2] ) )  
}  
  
MLE\_Gamma<-function(x, iteratin=100){  
 n<-length(x)  
 mean\_x<-mean(x)  
 alpha<-n\*(mean\_x^2)/sum((x-mean\_x)^2)  
 beta<-sum((x-mean\_x)^2)/n/mean\_x  
 for(i in 1:iteratin){  
 #first derivative of alpha\_k-1  
 der1<-n\*log(alpha/mean\_x)-n\*digamma(alpha)+sum(log(x))  
 #second derivative of alpha\_k-1  
 der2<-n/alpha-n\*trigamma(alpha)  
 #calculate next alpha  
 alpha<-alpha-der1/der2  
 beta<-mean\_x/alpha  
 }  
  
 return(list(shape=alpha, scale=beta))  
}  
  
MLE<-function(x, type, bino=5){  
 if(type=="PointMass") { ux <- unique(x) ; return( ux[which.max(tabulate(match(x, ux)))]) }  
 if (type=="Bernoulli") return( list(size=1, prob=sum(x)/length(x)) )  
 if(type=="Binomial") return( list(size=bino, prob=mean(x)/bino) )  
 if(type=="Geometric") return( list(prob=length(x)/mean(x) ))  
 if(type=="Poisson") return( list( lambda=mean(x) ) )  
 if(type=="Uniform") return( list(min=min(x), max=max(x) ) )  
 if(type=="Normal" ) return( list(mean=mean(x), sd=sqrt( sum( (x-mean(x) )^2)/ length(x) )))  
 if(type=="Exponential") return( list(rate=mean(x) ) )  
 if(type=="Gamma") return(MLE\_Gamma(x))  
 if(type=="Beta") return(MLE\_Beta(x))  
 #if(type=="t") return(2\*(a1^2+a2)/(a1^2+a2-1))  
 #if(type=="chi") return(2\* )  
}

# Test

We combine parametric bootstrapping and KS-test to test the goodness of fit.

### Parametric Bootstrapping

In parametric bootstrapping, we sample from our “estimated” distribution instead of the empirical distribution

### Kolmogorov-Smirnov test

In statistics, the Kolmogorov-Smirnov test (K-S test or KS test) is a nonparametric test of the equality of continuous, one-dimensional probability distributions that can be used to compare a sample with a reference probability distribution (one-sample K-S test), or to compare two samples (two-sample K-S test). It is named after Andrey Kolmogorov and Nikolai Smirnov.

The Kolmogorov-Smirnov statistic quantifies a distance between the empirical distribution function of the sample and the cumulative distribution function of the reference distribution, or between the empirical distribution functions of two samples. The null distribution of this statistic is calculated under the null hypothesis that the sample is drawn from the reference distribution (in the one-sample case) or that the samples are drawn from the same distribution (in the two-sample case). In each case, the distributions considered under the null hypothesis are continuous distributions but are otherwise unrestricted.

The two-sample K-S test is one of the most useful and general nonparametric methods for comparing two samples, as it is sensitive to differences in both location and shape of the empirical cumulative distribution functions of the two samples.(<https://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Smirnov_test>)

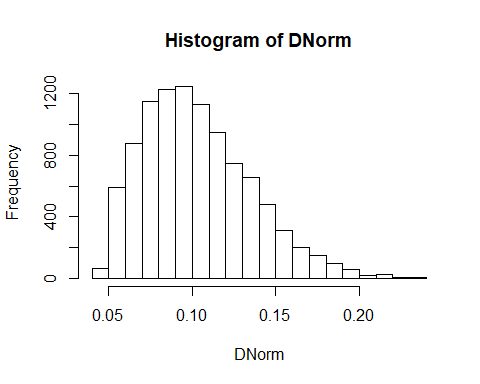
# Bootstrap and KS-test code

I created two *partial* functions for later use, which can create a parameters-partially-fed function.

partial <- function(f, ...) {  
 l <- list(...)  
 function(...) {  
 do.call(f, c(l, list(...)))  
 }  
}  
partial\_list<-function(f, L){  
 function(...){  
 do.call(f, c(L, list(...)))  
 }  
}  
  
  
BootKS<-function(x, n0 , nboot=10000, rvecFUN, Distritype, FUN)  
{  
Dvec<-NULL  
for(i in 1:nboot){  
 vec<-rvecFUN(n=n0)  
 theta<-MLE(vec, type = Distritype)  
 Dvec<-c(Dvec, unname(ks.test(x, partial\_list(FUN, theta))$statistic ))  
}  
return(Dvec)  
}

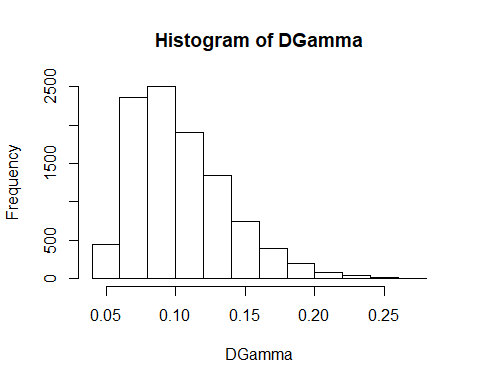
### Test with Normal Distribution

set.seed(553)  
x0<-rnorm(n=100, mean=, sd=6)  
theta0<-MLE(x0, type = "Normal")  
DNorm<-BootKS(x0, n0=50, rvecFUN = partial\_list(rnorm, theta0), Distritype = "Normal", FUN = pnorm)  
hist(DNorm)



### Test with Gamma Distribution

set.seed(239)  
x1<-rgamma(n=100, shape=5, scale=6)  
theta1<-MLE(x1, type = "Gamma")  
DGamma<-BootKS(x1, n0=50, rvecFUN = partial\_list(rgamma, theta1), Distritype = "Gamma", FUN=pgamma)  
hist(DGamma)



### Test with Beta Distribution

set.seed(461)  
x2<-rbeta(n=100, shape1=45, shape2 = 13)  
theta2<-MLE(x2, type="Beta")  
DBeta<-BootKS(x2,n0=50, rvecFUN = partial\_list(rbeta, theta2), Distritype = "Beta", FUN = pbeta)  
hist(DBeta)

