Chapter 2

The Bootstrap

2.1 Introducing The Bootstrap

Recall the regression model that related the genotype x to the phenotype y. The parameters of the model are α, which we call the genotypic covariate, the baseline level of the homozygous wildtype μ, the frequency of the mutation in the population p, and the conditional variance of the phenotype σ^2. In this section we would like to consider the estimation of the genotypic covariate.

Initially, let us assume that α, the quantity of interest, and μ, a nuisance parameter, are unknown but that the quantities p and σ^2 are known. The maximum likelihood estimate of parameter of interest is obtained by solving the normal equation $\dot{\ell} = 0$. Examining the equation in our case we get that

$$\hat{\alpha} = \frac{\bar{x\bar{y}} - \bar{x}\bar{y}}{\bar{x\bar{x}} - \bar{x}^2}.$$

Large sample theory proposes to use the appropriate diagonal element of the Fisher information matrix as an approximation of the variance of this estimate. Hence

$$\hat{\text{var}}(\hat{\alpha}) = \frac{\sigma^2}{n(\bar{x\bar{x}} - \bar{x}^2)}.$$

Let us examine in an example the validity of this approximation. Assume the true parameter values are as given before and let us simulate estimates of the variance and compare them to the actual variance. The latter is also obtained via simulations:

```
> rm(list=ls())
>
> n <- 10; mu <- 15; al <- 1.5; sig <- 3; p <- 0.3
> n.sim <- 10^5
> al.s <- vector(length = n.sim)
> for(i in 1:length(al.s))
+ {
+     x.s <- rbinom(n,2,p)
+     y.s <- rnorm(n,mu+al*x.s,sig)
+     v.x <- var(x.s)
+     if (v.x > 0) al.s[i] <- cov(x.s,y.s)/v.x else
```

```
+         al.s[i] <- 0
+ }
> true.var <- var(al.s)
> true.var
[1] 2.846964
```

Let us look at an example:

```
> x <- rbinom(n,2,p)
> y <- rnorm(n,mu+al*x,sig)
> x
 [1] 1 1 0 0 0 0 0 0 1 1
> y
 [1] 15.48461 11.00769 19.03690 19.27874 12.51359 19.73240 17.39038 14.10033
 [9] 13.29940 17.28394
> al.hat <- cov(x,y)/var(x)
> al.hat
[1] -2.739816
> est.var.app <- sig^2/((n-1)*var(x))
> est.var.app
[1] 3.75
> true.var
[1] 2.846964
```

The bootstrap is a simulation-based statistical procedure. The essential component in this procedure, and the source for its name, is the fact that the current sample is the basis for the simulation algorithm. In a way, one may think of the bootstrap as an attempt to mimic the simulation we carried out to obtain the actual variance of the estimate but without the benefit we had of "knowing" what the values of the unknown parameters are.

As a matter of fact the name "bootstrap" may appear in the title of different algorithms that may be applied in different settings. Hence, the name is more a description of a theme than a description of a specific algorithm. We will demonstrate two distinct bootstrap algorithms that may fit the inference problem that is dealt with here.

Assume that we are using the estimator $\hat{\alpha}$ but are not satisfied with the standard assessment of the variance of the estimate which is proposed by the large sample theory. Had we known the actual values of the parameters we would have been able to simulate artificial samples and use them to compute the variance. Unfortunately, these parameters are unknown to us. As a remedy, one may use *estimates* of the parameters, namely $\hat{\alpha}$ and $\hat{\mu}$ as if they were the true population parameters values for the simulation of the samples. The resulting estimate of the variance is called the *parametric bootstrap estimate of the variance*

Let us demonstrate the application of the parametric bootstrap for estimating the variance of the estimate $\hat{\alpha}$:

```
> alpha.est <- function(x,y,var.x,sig)
+ {
+     v.x <- var(x)
```

```
+     if(v.x > 0) al.hat <- cov(x,y)/var(x) else al.hat <- 0
+     return(al.hat)
+ }
```

This is a function that computes the statistic. It will be used in the bootstrapping algorithm:

```
> p.boot.alpha <- function(x,y,B=200,sig,p)
+ {
+     alpha.boot <- vector(length=B)
+     n <- length(x)
+     if (var(x) > 0)
+     {
+         al.hat <- cov(x,y)/var(x)
+         mu.hat <- mean(y) - al.hat*mean(x)
+         for(b in 1:B)
+         {
+             x.s <- rbinom(n,2,p)
+             y.s <- rnorm(n,mu.hat+al.hat*x.s,sig)
+             alpha.boot[b] <- alpha.est(x.s,y.s,var.x,sig)
+         }
+     alpha.boot[is.na(alpha.boot)] <- 0
+     v <- var(alpha.boot)
+     } else v <- 0
+     return(v)
+ }
```

Let us apply it to the example:

```
> est.var.p.boot <- p.boot.alpha(x,y,sig=sig,p=p)
> est.var.p.boot
[1] 3.005303
> true.var
[1] 2.846964
```

An alternative bootstrap approach is the *non-parametric bootstrap* In this approach, which is the better known application of the bootstrap method, one does not base the sampling of the artificial samples on an assumed model for the marginal distribution of the observations (x, y), but uses instead a non-parametric description of the joint distribution, the empirical distribution for example. The practical consequence of using the empirical distribution is that an artificial sample is formed by by sampling with replacement n elements from the original sequence $\{(x_1, y_1), \ldots, (x_n, y_n)$.

Let us demonstrate the application of the non-parametric approach on our data:

```
> np.boot.alpha <- function(x,y,B=200,sig,p)
+ {
+     alpha.boot <- vector(length=B)
+     n <- length(x)
```

```
+     if (var(x) > 0)
+     {
+         index <- 1:n
+         for(b in 1:B)
+         {
+             i.s <- sample(index,rep=TRUE)
+             x.s <- x[i.s]
+             y.s <- y[i.s]
+             alpha.boot[b] <- alpha.est(x.s,y.s,var.x,sig)
+         }
+     v <- var(alpha.boot)
+     } else v <- 0
+     return(v)
+ }
> est.var.np.boot <- np.boot.alpha(x,y,sig=sig,p=p)
> est.var.np.boot
[1] 3.32312
> true.var
[1] 2.846964
```

Observe that both the parametric and non-parametric approach assume the basic model of independence of observations, namely that the per-sample observations (x_i, y_i) are independent between the sample members. The two approaches differ in the way they treat the distribution of a single pair in a parametric way, in one case, and in a non-parametric in the other.

It should be realized that there are more than two ways to produce artificial samples in an attempt to mimic the simulation computational method. Examples of other non-parametric re-sampling methods can be found in the paper by Efron and Tibshirani.

2.2 Estimating the location of the QTL

Let us turn now to a more complex problem. Up to this point we have looked at a single locus with the aim of relating it to the phenotype, either with the binomial modeling in the case of affected half-sibs or using regression modeling in the case of a quantitative trait. In reality, the location of genetic factor, also known as the *quantitative trait locus* (QTL) is unknown. Hence, a-priory we do not know which location should be tested. Therefore, the common practice it to examine many locations at once and try to infer from the data the presence, and if so, the location of the QTL.

In order to fix ideas let us imagine an experiment that aims at the mapping of QTLs that contribute to some quantitative trait in mice. A popular design for an experiment is the backcross design. This design initiates with two inbred strains, one showing an elevated level of the trait compared to the other.

Inbred strains are created by repeated brother-sister mating, which results in the reduction, and eventually the practical elimination of any genetic variability within the strain. Consequently, any mouse of the given strain is homozygous at each genetic locus and any pair of mice from the strain are genetically identical.

However, mice from distinct strains may differ genetically from each other at many loci.

Observe that there may by phenotypic variability within each strain but it cannot be attributed to genetic factors. On the other, some of the phenotypic difference between strains may be due to their genetic differences. The goal is to identify these factors.

In order to carry out the task one identifies a collection of markers, scattered across the genome. A marker is a locus in which the two strains differ. Standard maps of markers involve a few hundreds of such mapped polymorphic loci. Notice, that QTLs can be any of hundreds of thousands of polymorphic sites. Hence, we may not assume that any of the markers is itself a QTL.

The first step in the creation of the backcross is the formation of F_1 mice by crossing a mouse from one strain with a mouse from the other strain. Backcross mice are formed by crossing an F_1 mouse with a mouse from one of the inbred strains. Denote one of the inbred strain by capital letters and the other by small letters. Hence the genotype at an anonymous locus is MM for the first strain and mm for the second. The genotype of F_1 must be Mm. If the F_1 is crossed back with the mm mouse then the genotype may either be mm or it may be Mm, depending on which of the two copies of the F_1 mouse was passed to the offspring.

Concentrate on a given chromosome. Assume that one identifies markers at loci $t_1, t_2, \ldots, t_m$ over that chromosome and creates a sample of backcross mice. An (asymptotically) sufficient statistic is the vector $(\hat{\alpha}_1, \ldots, \hat{\alpha}_m)$ of estimated regression coefficients of the phenotype over each of the markers. Notice that if there is no QTL on this chromosome then the mean of each component in the vector is 0, and the variance is approximately $2\sigma^2/n$, where n is the number of mice. After dividing by the standard variation one obtains a standardized vector $(Z_1, \ldots, Z_m)$, the components of which are standard normal.

The components of the vector are not independent. The dependence structure results mainly from the process of recombinations. This process involves crossovers of genetic material between the two copies during the formation of a gamete. Consider the gamete that originated from the F_1 parent of a given backcross mouse and look at two loci. We say that a recombination took place if the allele at the first locus is from one (grand-parental) source and the allele at the other locus is from a different source. The probability of a recombination is denoted by θ. A popular model for the recombination rate, the *Haldane Model*, produces

$$\theta = 0.5 - 0.5 \times e^{-\beta|t_i - t_j|} \ ,$$

where $|t_i - t_j|$ is the distance between the two markers and β is a constant that depends on the units of distance and on the design of the experiment. If the distance is measured in centiMorgans (cM) then for the backcross design $\beta = 0.02$. For the affected sib-pairs (ASP), on the other hand one gets that $\beta = 0.04$. It turns out that

$$\mathrm{cov}(Z_i, Z_j) = e^{-\beta|t_i - t_j|} \ .$$

The vector $(Z_1, \ldots, Z_m)$ can be used in order to test the null hypothesis of the absence of a QTL on the chromosome. The null is rejected if any of the components is significantly different than zero. However, our focus now is not in testing but in estimation. Consequently, we will assume that at least one QTL is present and

our goal is to locate it. For simplicity, let us assume the presence of a single QTL at a locus unknown to the applied statistician and with an unknown effect. The presence of this QTL is reflected in the distribution of the vector $(Z_1, \ldots, Z_m)$. We will consider a large sample and the resulting normal approximation. Based on large sample theory we get that for local alternatives the distribution of the vector of standardized statistics is multi-normal with a non-zero mean vector and with the covariance structure as under the null hypothesis.

Specifically in our example, the source of the signal is the QTL and this signal is reflected in the markers as a result of the correlation between the genotype of the QTL and the genotypes at the markers. In order to quantify the signal imagine a statistic, parallel to the statistics obtained at the markers, is computed based on the genotype of the QTL. The mean of this imaginary statistic is non-zero and may be denoted by μ. The parameter μ results from the effect on the phenotype of the alleles of the QTL and is a function of the relative contribution of the QTL to the phenotypic variance. The (asymptotic) correlation between the imaginary statistic and the statistics at the markers is a function of the pairwise distances and obeys the same formula as the covariance between the markers themselves. Moreover, the expectation of the statistic at the markers is equal to the correlation between the marker and the QTL times the expectation at the QTL. Hence, if the QTL is located at q, then:

$$\mathbb{E}(Z_i) = \mu\sigma_q = \mu e^{-\beta|t_i - q|}\ , \quad \text{for } i = 1, 2, \ldots, m.$$

The unknown parameters in this asymptotic presentation of the statistics problem of QTL mapping are two: the location of the QTL q and the effect of the QTL μ. We will estimate these parameters using the MLE approach. Unfortunately, the likelihood function in this setting is not a smooth function of the parameters. Hence, one may not use the standard asymptotic theory of large samples in order to obtain an approximation of the variance of the estimates. Instead, we will apply the parametric bootstrap to produce an assessment of the sampling properties of the estimates based on the observed data. In order to speed up the computations we will simulate directly the sufficient vector of statistics $(Z_1, \ldots, Z_m)$ based on its asymptotic normal distribution.

Let us demonstrate the simulation of the sufficient statistics with an example. Assume a QTL, located 10 cM from the beginning of the chromosome, that produces a mean of 6:

```
> qtl <-  10
> mu <-  6
> b <-  0.04
```

Notice that we are assuming the correlation rate that fits the ASP experiment. The resulting vector of means of the markers' statistics is:

```
> mark <-  seq(0,160,by=40)
> Mu <-  mu*exp(-b*abs(mark-qtl))
> Mu
[1] 4.02192028 1.80716527 0.36486038 0.07366404 0.01487251
```

Observe that the markers are placed at $0, 40, 80, 120$, and 160 and that the QTL is located between the firs two markers. Notice that the covariance matrix of the markers and its inverse are given by:

```
> Sig <-  outer(mark,mark,function(x,y) exp(-b*abs(x-y)))
> round(Sig,5)
        [,1]    [,2]    [,3]    [,4]    [,5]
[1,] 1.00000 0.20190 0.04076 0.00823 0.00166
[2,] 0.20190 1.00000 0.20190 0.04076 0.00823
[3,] 0.04076 0.20190 1.00000 0.20190 0.04076
[4,] 0.00823 0.04076 0.20190 1.00000 0.20190
[5,] 0.00166 0.00823 0.04076 0.20190 1.00000
> W <-  solve(Sig)
> round(W,5)
         [,1]     [,2]     [,3]     [,4]     [,5]
[1,]  1.04249 -0.21048  0.00000  0.00000  0.00000
[2,] -0.21048  1.08499 -0.21048  0.00000  0.00000
[3,]  0.00000 -0.21048  1.08499 -0.21048  0.00000
[4,]  0.00000  0.00000 -0.21048  1.08499 -0.21048
[5,]  0.00000  0.00000  0.00000 -0.21048  1.04249
```

The function `outer` takes as input two vectors and a binary function and produces a matrix. The components of the matrix are the outputs of the application of the function to a pair of components, one from each vector. The default binary function is "`*`", in which case the function produces the standard outer product of the two vectors. However, the default function may be replaced by any other binary function, either by its name – hence using "`"-"`" would produce the pairwise differences – or by its definition as in the example.

The function `solve` solves a system of linear equations if its arguments is an invertible matrix and a vector or, if the argument is only the matrix, it compute the inverse of the matrix. Observe the particular form of the inverse of the correlation matrix, which has non-zero elements only on the main-, super-, and sub-diagonals and the matrix. (This is a consequence of the fact that the entries of the correlation matrix take the form $\rho^{|i-j|}$, for some ρ.)

Let us try to simulate a vector of observed sufficient statistics with the function `rmvnorm`:

```
> Z <-  rmvnorm(7,Mu,Sig)
Error: could not find function "rmvnorm"
```

The reason that we have failed is that given function is not part of the standard R distribution and also is not loaded automatically at the initiation of the session. Therefore, we need first to install on our computer the library of functions that contains this specific one – an operation that we do once – and upload the library – an operation we do for each session that requires the function. The package `mvtnorm` is the name of the library that includes the function `rmvnorm`. It can be installed directly from `CRAN` using the option `packages` in the R GUI toolbar. After the installation on can load the library and get:

```
> library(mvtnorm)
> Z <-  rmvnorm(7,Mu,Sig)
> Z
         [,1]        [,2]       [,3]       [,4]        [,5]
[1,] 4.146552  0.26869404  0.7084614  0.2741802 -0.92907689
```

```
[2,] 1.713306  1.62798543  1.2676685 -0.3750106  0.03495147
[3,] 3.020231 -0.02227084  0.4362036 -2.1767059 -2.23372021
[4,] 4.962628  2.14258340  0.4810787  0.1951507 -0.67051660
[5,] 4.541066  2.24139000 -0.2948301 -0.2132515  0.24400656
[6,] 3.211752  1.63506256  1.6919032  1.0679673  0.89321538
[7,] 4.769529  2.15444741  0.5249833 -0.7202958  0.55261256
```

The input to the function `rmvnorm` was an integer, that gives the requested number of independent copies, a mean vector and a covariance matrix. The output is a matrix, where each row represents an independent copy and each column represents a component of the multi-normal vector. As a result of the crossing experiment for gene mapping the applied statistician gets to observe a single row of observations which he or she can use for making inference. In the simulation, however, we can produce independent copies of the result of such experiment.

The estimation of the unknown parameters is based on the maximization of the log-likelihood of a multi-normal vector. Observe that if z is such a vector with mean vector $\mu\sigma_q$ covariance matrix given by Σ then the log-likelihood is

$$\ell(q,\mu;z) = \mu(\sigma_q)'Wz - (\mu^2/2)(\sigma_q)'W\sigma_q - 0.5z'Wz + 0.5\log(|W|) - 0.5d\log(2\pi)\ ,$$

where $W = \Sigma^{-1}$ is the inverse of the covariance matrix, $|W|$ is its determinant and d is the dimension of the vector. Notice that only the first two components are a function of the unknown quantities q and μ. Hence, we may ignore the other terms when looking for the maximizers.

The maximization of the target function with respect to the pair of parameters may be carried out sequentially by first maximizing the function with respect to mu for each fixed q and then maximizing the outcome with respect to q. The advantage of this approach is the availability of an analytic expression for the maximization with respect to μ:

$$\frac{\partial}{\partial\mu}\ell(q,\mu;z) = (\sigma_q)'Wz - \mu(\sigma_q)'W\sigma_q = 0 \quad\Longrightarrow\quad \hat{\mu}(q) = \frac{(\sigma_q)'Wz}{(\sigma_q)'W(\sigma_q)}$$

and therefore

$$\ell(q,\hat{\mu}(q);z) = \frac{[(\sigma_q)'Wz]^2}{2(\sigma_q)'W(\sigma_q)}\ , \tag{2.1}$$

which still needs to be maximized with respect to q.

Further simplification may be obtained by noticing that vector that results from multiplying σ_q with the matrix W may have non-zero values only fro the two components that are associated with the two markers adjacent to the QTL. Let us demonstrate this fact for the given QTL:

```
> sig <- exp(-b*abs(mark-qtl))
        [,1]    [,2] [,3] [,4] [,5]
[1,] 0.63541 0.17291    0    0    0
```

The consequence of this property is that in order to locate the local maxima of the log-likelihood with respect to the location of the QTL between a pair of adjacent markers we can consider the sub-problem in which only these two markers are observed and maximize the resulting log-likelihood. For example, consider the computation of the profile log-likelihood (2.1) for a tentative QTL location at $t = 3$:

```
> Z1 <-  Z[1,1:2]
> Z1
[1] 4.1465524 0.2686940
> t1 <-  40
> r <-  exp(-b*t1)
> V1 <-  matrix(c(1,r,r,1),2,2)
> W1 <-  solve(V1)
> W1
           [,1]      [,2]
[1,]  1.042494 -0.210476
[2,] -0.210476  1.042494
> t <-  3
> sig <-  exp(-b*c(t,t1-t))
> sig
[1] 0.8869204 0.2276377
> (sig%*%W1%*%Z1)^2/(sig%*%W1%*%sig)
         [,1]
[1,] 16.87304
```

Let us write a function that compute the profile log-likelihood of the location:

```
>  ll <-  function(t,t1,Z1,W1,b)
+ {
+     sig <-  exp(-b*c(t,t1-t))
+     return((sig%*%W1%*%Z1)^2/(sig%*%W1%*%sig))
+ }
> ll(t,t1,Z1,W1,b)
         [,1]
[1,] 21.36515
```

The function *optimize* is a general linear maximizer and it can be used in order to maximize the profile log-likelihood with respect to location of the QTL:

```
> optimize(ll,c(0,t1),maximum=TRUE, t1=t1,Z1=Z1,W1=W1,b=b)
$maximum
[1] 4.526025e-05

$objective
         [,1]
[1,] 17.19389
```

Notice that the function located a maximizing location at the origin. The input of the function `optimize` is the name (or definition) of the function to be optimized, the interval over which optimization should be carried on, an argument named `maximum`, with a default value of `FALSE`, which indicates whether minimization or maximization is requested, and other arguments that may be used in order to control the optimization algorithm. On top of these arguments a value for any argument that is used by the target function may be passed on using the structure `argument.name = argument.value`. The output of the function is a list with the maximizer and with maximal evaluation of the objective function. In general, one can think of a

list as a vector with components that can be any object. Hence, in the output of the current application the first component, that is given the name `maximum`, is a vector of length one, and the second component, titled `objective`, is a matrix.

We pack this function `optimize` in a form that will be more convenient for the application of the bootstrap:

```
> opt.Z <-  function(Z1,t1,W1,b)
+     return(unlist(optimize(ll,c(0,t1),maximum=TRUE,
+     t1=t1,Z1=Z1,W1=W1,b=b)))
```

Finally, we are in a position to write a function that compute the MLE estimates of the location and the genetic effect. The input to this function are the observations, the marker positions and the parameter β and the output is a list with the estimated location of the QTL, the estimated parameter μ, and the maximum of the target function. We would like to be able to apply this function for the case of a single vector of observations, but also for a matrix composed of rows of independent observations. The second situation is appropriate for the application of the bootstrap, in which the maximization should be carried out in parallel for each of the rows of simulated observations:

```
> mle.t <-  function(Z,mark,b)
+ {
+     if(is.vector(Z))
+     {
+         Z <-  matrix(Z,nrow=1,ncol=length(Z))
+         isvector <-  TRUE
+     } else isvector <- FALSE
+     Q <-  L <-  M <-  NULL
+     for(i in 2:length(mark))
+     {
+         t1 <-  mark[i]-mark[i-1]
+         r <-  exp(-b*t1)
+         V1 <-  matrix(c(1,r,r,1),2,2)
+         W1 <-  solve(V1)
+         temp <-  apply(Z[,c(i-1,i),drop=FALSE],1,opt.Z,t1=t1,W1=W1,b=b)
+         Q <-  cbind(Q,temp[1,])
+         L <-  cbind(L,temp[2,])
+         sig <-  exp(-b*cbind(temp[1,],t1-temp[1,]))
+         M1 <-  apply(sig%*%W1*Z[,c(i-1,i),drop=FALSE],1,sum)
+         M1 <-  M1/apply(sig%*%W1*sig,1,sum)
+         M <-  cbind(M,M1)
+     }
+     interval <-  apply(L,1,which.max)
+     QTL <-  NCP <-  vector(length=nrow(Z))
+     for (i in 1:nrow(Z))
+     {
+         QTL[i] <-  Q[i,interval[i]]+mark[interval[i]]
+         NCP[i] <-  M[i,interval[i]]
+     }
```

```
+     LLIK <-  apply(L,1,max)/2
+     names(LLIK) <-  NULL
+     if (isvector)
+     {
+         QTL <- QTL[1];NCP <- NCP[1];LLIK <- LLIK[1]
+     }
+     return(list(hat.qtl=QTL,hat.ncp=NCP,max.log.lik=LLIK))
+ }
```

As a demonstration, let us apply this function to the entire matrix `Z`

```
> mle.t(Z,mark,b)
$hat.qtl
[1] 4.526025e-05 1.936148e+01 5.038375e-05 9.500960e+00 1.117419e+01
[6] 1.156080e+01 1.006609e+01

$hat.ncp
[1] 4.146559 3.716878 3.020236 7.257053 7.100245 5.100041 7.134136

$max.log.lik
[1]  8.596947  2.324491  4.560895 12.992020 11.225152  5.665068 12.114199
```

and to its first component:

```
> mle.t(Z[1,],mark,b)
$hat.qtl
[1] 4.526025e-05

$hat.ncp
[1] 4.146559

$max.log.lik
[1] 8.596947
```

Consider the application of the bootstrap algorithm for the assessment of the variance of the of the MLE of the location. We will assume a chromosome of typical length of 160 cM for the ASP design. Markers are located in spacings of 10 cM apart. We will consider, first, a QTL located 90 cM from the telomer with a genetic effect of 6:

```
> qtl <-  90
> mu <-  6
> b <-  0.04
> mark <-  seq(0,160,by=10)
> Mu <-  mu*exp(-b*abs(mark-qtl))
> Sig <-  outer(mark,mark,function(x,y) exp(-b*abs(x-y)))
```

Let us see first what is the true distribution of the MLE:

```
> sim.mle <- mle.t(rmvnorm(1000,Mu,Sig),mark,b)
> mean(sim.mle$hat.qtl)
[1] 90.19342
> sd(sim.mle$hat.qtl)
[1] 2.300008
> plot(density(sim.mle$hat.qtl))
```

Notice that the distribution of the MLE is not normal, in contradiction to the expected behavior under the standard theory of large samples.

The bootstrap approach does not depend on the validity of the assumptions of large sample theory. Let us apply it on the observed data:

```
> obs <- rmvnorm(1,Mu,Sig)
> est <-  mle.t(obs,mark,b)
> est
$hat.qtl
[1] 92.1713

$hat.ncp
[1] 5.654128

$max.log.lik
[1] 13.83031
```

Notice that the MLE algorithm produced an estimate of the unknown parameters, which can be used as a basis for the parametric bootstrap simulation:

```
> hat.Mu <-  est$hat.ncp*exp(-b*abs(mark-est$hat.qtl))
> est.boot <-  mle.t(rmvnorm(1000,hat.Mu,Sig),mark,b)
> mean(est.boot$hat.qtl)
[1] 91.57936
> sd(est.boot$hat.qtl)
[1] 3.379363
> sd(sim.mle$hat.qtl)
[1] 2.300008
```

The bootstrap produced the estimate standard deviation 3.38, which is slightly larger than the true standard deviation of 2.30.

Consider a second example in which the QTL is not located exactly at a marker:

```
> qtl <-  95
> Mu <-  mu*exp(-b*abs(mark-qtl))
> Sig <-  outer(mark,mark,function(x,y) exp(-b*abs(x-y)))
> sim.mle <- mle.t(rmvnorm(1000,Mu,Sig),mark,b)
> mean(sim.mle$hat.qtl)
[1] 95.08576
> sd(sim.mle$hat.qtl)
[1] 3.751068
> plot(density(sim.mle$hat.qtl))
```

```
> obs <- rmvnorm(1,Mu,Sig)
> est <-  mle.t(obs,mark,b)
> est
$hat.qtl
[1] 95.20342

$hat.ncp
[1] 8.500297

$max.log.lik
[1] 29.00849

> hat.Mu <-  est$hat.ncp*exp(-b*abs(mark-est$hat.qtl))
> est.boot <-  mle.t(rmvnorm(1000,hat.Mu,Sig),mark,b)
> mean(est.boot$hat.qtl)
[1] 95.15049
> sd(est.boot$hat.qtl)
[1] 2.070334
> sd(sim.mle$hat.qtl)
[1] 3.751068
```

This time the bootstrap produced a slight underestimation of the true standard deviation.

2.3 Estimating the statistical properties of the Bootstrap

In order to assess the statistical properties of the bootstrap procedure for estimating the MLE of the location of the QTL we should conduct a simulation study in which we simulate artificial data, apply the bootstrap to the data, and iterate several times in order to obtain a distribution of the bootstrap estimate. Unfortunately, the function `mle.t` for the computation of the MLE of the location of the QTL and of its genetic effect is too slow for our current purpose. The main reason for that is the need to apply the general optimization function `optimize`. Let examine the prospects of developing an alternative optimization function which exploits the characteristics of this special problem in order to scale up the speed of computation. Let us look at an example:

```
> library(mvtnorm)
> qtl <- 90
> mu <- 6
> b <- 0.02
> Delta <- 40
> mark <- seq(0,160,by=Delta)
> Mu <- mu*exp(-b*abs(mark-qtl))
> Sig <- outer(mark,mark,function(x,y) exp(-b*abs(x-y)))
> Z <- rmvnorm(7,Mu,Sig)
```

The function `ll` was written in order to compute the profile function of the location, after the maximization with respect to the genetic effect:

```
> ll <-  function(t,t1,Z1,W1,b)
+ {
+     sig <-  exp(-b*c(t,t1-t))
+     return((sig%*%W1%*%Z1)^2/(sig%*%W1%*%sig))
+ }
```

Let us use this function in order to plot the profile based on the data of the first observation, i.e. the first row of the matrix `Z`:

```
> t1 <-  40
> r <-  exp(-b*t1)
> V1 <-  matrix(c(1,r,r,1),2,2)
> W1 <-  solve(V1)
> t <-  0:40
> l.hat <- rep(t,4)
> for (j in 1:4) for(i in 1:length(t))
+ l.hat[i+(j-1)*length(t)] <- ll(t[i],t1,Z[1,j:(j+1)],W1,b)
> plot(l.hat,type="l")
```

Examining the plot one may note that between markers the function is essentially a quadratic function. Quadratic functions have explicit maximizers, which may either be an internal point or the location of either bracketing markers. The function `mle.2` is a substitute for the previously used `mle.t`. In this version we use analytical expression of obtaining the between-markers local maximizers. Moreover, the function is written so that computation is conducted in parallel for all rows of the data matrix `Z`, exploiting the efficient parallel computation abilities of `R`.

```
> mle.2 <- function(Z,Delta,b)
+ {
+ # Change a vector to a matrix
+     if(is.vector(Z))
+     {
+         Z <- matrix(Z,ncol=length(Z),nrow=1)
+         isvector <- TRUE
+     } else isvector <- FALSE
+ # calculate W1
+     r <- exp(-b*Delta)
+     V1 <- matrix(c(1,r,r,1),2,2)
+     W1 <- solve(V1)
+ # Calculate tentative qtl in each interval
+     Z0 <- Z[,-ncol(Z),drop=FALSE]
+     Z1 <- Z[,-1,drop=FALSE]
+     Q <- log(pmax(Z1/Z0,0))/(2*b)
+     Q <- pmin(Q,Delta/2)
+     Q <- pmax(Q,-Delta/2)
+     Q <- Delta/2+ Q
+ # Calculate log-likelihood
```

```
+     sig0 <- exp(-b*Q); sig1 = exp(-b*(Delta-Q))
+     term1 <- sig0*(W1[1,1]*Z0+W1[1,2]*Z1)
+     term2 <- sig1*(W1[2,1]*Z0+W1[2,2]*Z1)
+     term3 <- sig0^2*W1[1,1]+2*sig0*sig1*W1[1,2]+sig1^2*W1[2,2]
+     L <- (term1+term2)^2/term3
+     M <- (term1+term2)/term3
+ # Find the max
+     LLIK <- apply(L,1,max)/2
+     names(LLIK) <- NULL
+     I <- apply(L,1,which.max)
+     QTL <- NCP <- vector(length=nrow(Z))
+     for (i in 1:nrow(Z))
+     {
+         QTL[i] <- Q[i,I[i]]+Delta*(I[i]-1)
+         NCP[i] <- M[i,I[i]]
+     }
+     if (isvector)
+     {
+         QTL<-QTL[1];NCP<-NCP[1];LLIK<-LLIK[1]
+     }
+     return(list(hat.qtl=QTL,hat.ncp=NCP,max.log.lik=LLIK))
+ }


> mle.t(Z,mark,b)
$hat.qtl
[1] 94.20679 70.95575 72.43281 92.89604 95.75981 83.12447 86.91135

$hat.ncp
[1] 8.717983 6.266412 5.690262 8.087221 6.197648 4.341192 7.110072

$max.log.lik
[1] 24.716946 14.331089 12.339417 21.773280 12.220130  8.353205 19.663286

> mle.2(Z,Delta,b)
$hat.qtl
[1] 94.20679 70.95575 72.43280 92.89603 95.75981 83.12445 86.91135

$hat.ncp
[1] 8.717983 6.266412 5.690263 8.087221 6.197648 4.341191 7.110072

$max.log.lik
[1] 24.716946 14.331089 12.339417 21.773280 12.220130  8.353205 19.663286

> mle.2(Z[1,],Delta,b)
$hat.qtl
[1] 94.2068

$hat.ncp
```

```
[1] 8.717983

$max.log.lik
[1] 24.71695
```

Examining the application of both functions on the same dataset we can see that they produce essentially the same output. Notice that the new function, like the old one, may be applied to both matrices and vectors.

Let us examine first the distribution of the MLE of the location, this time based on a much larger number of iterations:

```
> qtl <- 90
> mu <- 6
> b <- 0.04
> Delta <- 10
> mark <- seq(0,160,by=Delta)
> Mu <- mu*exp(-b*abs(mark-qtl))
> Sig <- outer(mark,mark,function(x,y) exp(-b*abs(x-y)))
> est.sim <- mle.2(rmvnorm(10^5,Mu,Sig),Delta,b)
> plot(density(est.sim$hat.qtl,from=80,to=100))
```

The non-normal distribution of the MLE is clearly evident.

Let us turn now to the investigation of the statistical properties of the bootstrap algorithm. To warm up, let us start by applying it once to a vector of simulated data as we did before:

```
> obs <- rmvnorm(1,Mu,Sig)
> est <- mle.2(obs,Delta,b)
> est
$hat.qtl
[1] 90.58005

$hat.ncp
[1] 5.699034

$max.log.lik
[1] 15.53167

> hat.Mu <- est$hat.ncp*exp(-b*abs(mark-est$hat.qtl))
> est.boot <- mle.2(rmvnorm(10^3,hat.Mu,Sig),Delta,b)
> sd(est.boot$hat.qtl)
[1] 2.607806
> sd(est.sim$hat.qtl)
[1] 2.357489
```

In principle, now all we have to do is iterate the above by sampling new observations, computing and storing the bootstrap estimate of the standard deviations. However, we would like to exploit the simulations in order to examine another application of the bootstrap algorithm: the construction of confidence intervals.

Notice that the bootstrap produces a sample of the estimate, which is supposed to mimic its real sampling distribution. A natural proposal for a confidence interval of confidence probability $1-\alpha$ is to take the as a lower confidence limit the $\alpha/2$ quantile of that distribution and use the $1-\alpha/2$ quantile as the upper confidence limit. Consider as an example the construction of a confidence interval for the location of the QTL given the generated bootstrap samples for the given observations:

```
> hat.qtl <- sort(est.boot$hat.qtl)
> LCL <- hat.qtl[round(0.05*length(hat.qtl))]
> UCL <- hat.qtl[round(0.95*length(hat.qtl))]
> LCL
[1] 86.42792
> UCL
[1] 94.09775
```

Notice, that the produced confidence interval contains the actual QTL, which is located at 90. A question we may ask is what is the actual confidence limit of this procedure for constructing confidence intervals?

Let us now run the simulation that will assess both the statistical properties of the bootstrap estimate of the standard deviation and the properties of the bootstrap generated confidence interval:

```
> n.rep <- 10^3
> SD <- LCL <- UCL <- rep(-1,n.rep)
> for(i in 1:n.rep)
+ {
+     obs <- rmvnorm(1,Mu,Sig)
+     est <- mle.2(obs,Delta,b)
+     hat.Mu <- est$hat.ncp*exp(-b*abs(mark-est$hat.qtl))
+     est.boot <- mle.2(rmvnorm(1000,hat.Mu,Sig),Delta,b)
+     SD[i] <- sd(est.boot$hat.qtl)
+     hat.qtl <- sort(est.boot$hat.qtl)
+     LCL[i] <- hat.qtl[round(0.05*length(hat.qtl))]
+     UCL[i] <- hat.qtl[round(0.95*length(hat.qtl))]
+ }
> mean(SD)
[1] 3.361597
> sd(est.sim$hat.qtl)
[1] 2.357489
> sd(SD)
[1] 2.21148
```

Examining the the properties of the estimate of the standard deviations one may conclude that it is biased upwards by about 1 cM. The standard deviation of the estimate is about 2 cM. Considering now the confidence interval:

```
> mean((LCL <= qtl) & (UCL >= qtl))
[1] 0.985
> mean(UCL-LCL)
[1] 8.635271
```

We may see that the confidence interval that the bootstarp is proposing is too wide and it covers the true location with probability 0.985, instead of the requested 0.95.