Finding MLEs Using Optim

Mansi Birla, Pinki Sawaria, Arindam Fadikar

October 13, 2010

Abstract

R provides a function called *optim*() which optimizes functions of single parameter(may be vector valued) using 5 different optimization methods. We will be using *optim* to solve our problem with a brief description of the optimization methods used in *optim* and a basic comparison among them with respect to our problem.

1 Introduction to the problem

As a matter of fact the inheritance of blood groups in human being is controlled by three allelomorphic genes O, A, B of which O is recessive to A and B. Given that r, p, q are gene frequencies of O, A, B respectively in the population, then the expected probabilities of the six genotypes (four phenotypes) in random mating are as follows

Table 1:

Genotype	Phenotype	Probability
OO	O	r^2
AA	A	p^2
AO	A	$2pr$
BB	B	q^2
BO	B	$2qr$
AB	AB	$2pq$

Phenotype	Probability
O	r^2
A	$p^2 + 2pr$
B	$q^2 + 2qr$
AB	$2pq$

Now given the observed counts n_O, n_A, n_B, n_{AB}, we are interested to calculate the mles of p, q, r. Essentially this is an optimisation problem and we solve this problem using the `R` function *optim* ().

2 Obtained MLEs

2.1 Result

We have taken the values of n_O, n_A, n_B, n_{AB} from an example in LSI[1] and obtained the following results.

Table 2: MLEs of p, q and r using different optimisation method available in *optim*

Methods	p	q	r
Nelder-Mead	0.26449953	0.09316453	0.64233594
CG	0.26444187	0.09316946	0.64238867
BFGS	0.2644437	0.0931701	0.64238620
L-BFGS-B	0.26444282	0.09317189	0.64238529
SANN	0.26492582	0.09153084	0.64354334

So all the methods are giving more or less same result which is satisfactory. Point to be noted that the initial estimates of p, q, r is taken as 0.3, 0.3, 0.4 respectively for all the methods in the above case. But we may want to see whether these initial estimates have any other role to play or not. We may want to know how much time the methods are taking, or which method is relatively faster etc. Does it depend on the choice of initial estimate or not. We will try to figure out some of this things in the later subsection.

2.2 Procedure

To obtain the mles first we need the likelihood function. The distribution of N_O, N_A, N_B, N_{AB} is multinomial given by

$$P(N_O = n_O, N_A = n_A, N_B = n_B, N_{AB} = n_{AB}) = \frac{n!}{n_O! n_A! n_B! n_{AB}!}(r^2)^{n_O}(p^2 + 2pr)^{n_A}(q^2 + 2qr)^{n_B}(2pq)^{n_{AB}} \quad (1)$$

with $p, q, r \geq 0 \quad \text{and} \quad p + q + r = 1$.

[1] Linear Statistical Inference, C. R. Rao

and the loglikelihood function is given by

$$\log L(p,q,r) = k + n_O \log r + n_A \log(p^2 + 2pr) + n_B \log(q^2 + 2qr) + n_{AB} \log 2pq \quad (2)$$

where k is constant not involving any parameters.
Now we are to minimize (2) with respect to the constraints $0 < p, q, r < 1$ and $p + q + r = 1$.

2.3 Using R

optim is a function which takes arguments an objective function, that we want to optimize and an initial estimate of the parameter and some other optional arguments. By default *optim* minimizes a function.

```
optim(par, fn, gr = NULL, ...,
           method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN"),
           lower = -Inf, upper = Inf,
           control = list(), hessian = FALSE)
```

The mles have been obtained in two steps. First an R function has been constructed which takes arguments the values of n_O, n_A, n_B, n_{AB} and returns the negative of the loglikelihood function. And in the second step *optim* has been used to find the values of the parameters for which the negative of the log likelihood is minimized.

✖ NOTE : Lexical scoping is an important thing in `R`. In lexical scoping the values of the global free variables are defined by the bindings that were in effect at the time the function was created. We have used this lexical scoping to create a function which returns the likelihood function and we can pass the value of this function as an argument to the *optim* function.

2.4 Comparison

Here we will see some of the comparison among the methods. As we were talking about time taken by the methods, it is not very meaningful if we compare the actual time. The time gap may hardly be fraction of seconds. So instead what we can observe is the number of times *optim* iterates the function under each method and for different initial estimates. We make tables to see the comparison.

optim returns list of values of which one is 'counts' which gives the number of times *optim* called the function and its gradient while obtaining the optimum value. We extract these values to make comparison.

Table 3: No of counts while using Nelder-mead method

p and q	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
0.1	61	51	57	65	57	61	59	61
0.2	45	53	57	59	57	59	61	NA
0.3	47	55	63	59	59	63	NA	NA
0.4	51	53	59	67	63	NA	NA	NA
0.5	47	61	63	75	NA	NA	NA	NA
0.6	57	63	65	NA	NA	NA	NA	NA
0.7	59	63	NA	NA	NA	NA	NA	NA
0.8	63	NA	NA	NA	NA	NA	NA	NA

Table 4: No of counts while using BFGS method

p and q	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
0.1	56	48	39	38	23	57	51	78
0.2	24	39	40	60	40	56	75	NA
0.3	35	39	40	41	40	52	NA	NA
0.4	64	39	53	40	80	NA	NA	NA
0.5	49	39	50	59	NA	NA	NA	NA
0.6	39	39	40	NA	NA	NA	NA	NA
0.7	52	42	NA	NA	NA	NA	NA	NA
0.8	50	NA	NA	NA	NA	NA	NA	Na

It is apparently expected that if we start with the initial values of the parameters close to the actual mles then it will take less steps to reach the mle values. But from table 4 and figure 2 we can not see any evidence of our claim.

One possible explanation to this may be that sometimes luckily we reach at the optimal value in lesser step irrespective of where we start from. The optimization methods does not search for optimum values in ascending or descending order, e.g. first for $p = 0.1$ then for 0.2 etc. Instead they have different procedures which may end up in reaching the mles in lesser steps though starting with bad initial choices.

Thus these study might not be meaningful. So we move on to another comparison and this time among the methods. Figures are in the next page.

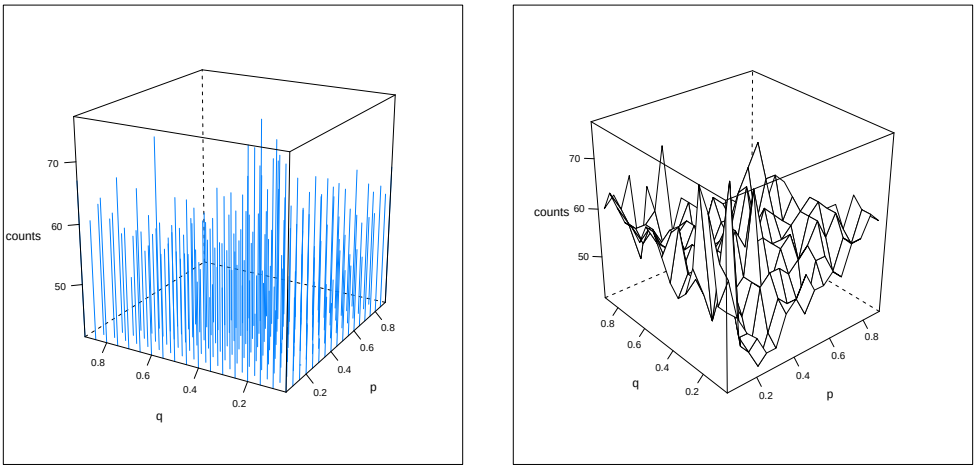

Figure 1: # of counts of function call for Nelder-Mead method

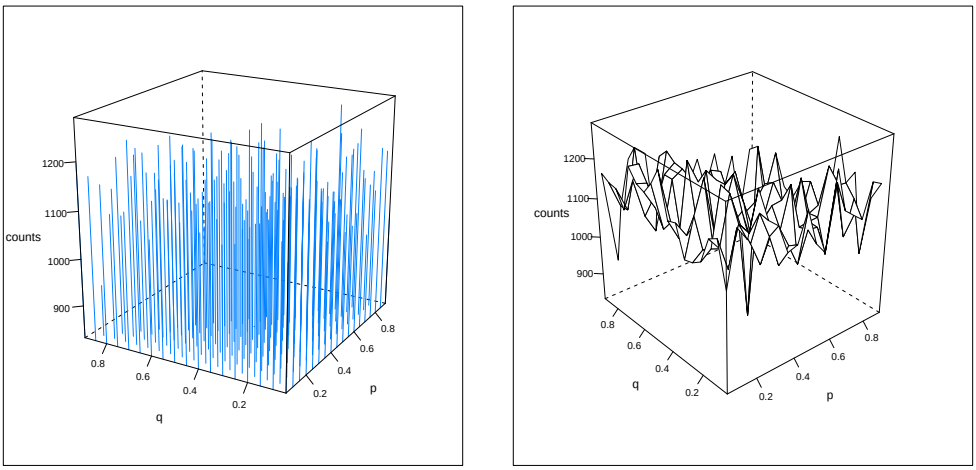

Figure 2: # of counts of function call for CG method

Comparison among the 3 Methods

Here we note the function calls for each of the 3 methods Nelder-Mead, CG and BFGS. For 4 different choices of p, q we observe the counts. 4 different choices is made in a way that one pair is near to the mles, one pair is worst in that sense, and two pairs are partially good as one of them in each pair is near to the mle value. Following table gives the result.

Table 5: comparative study

(p, q)	Nelder-mead	CG	BFGS
(0.26,0.1)	41	1232	24
(0.30,0.3)	63	1261	39
(0.30,0.6)	63	1159	42
(0.70,0.1)	59	1078	58

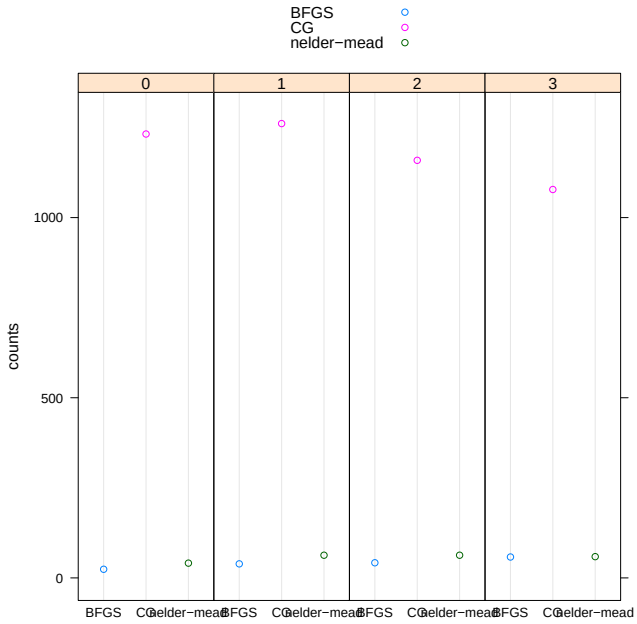

Figure 3:

0,1,2,3 represents the values of the pair (p, q) respectively as appeared in the table **??**.

Thus what we see that in general the *BFGS* method calls the objective function far more than *Nelder-Mead* and *CG* do. *SANN* method by default searches for a finer value of the parameter for 10000 times even if it reaches the minimum before. But unless we do optimize a function without using any programe or machine these differences hardly have an issue to be looked upon. But it may be handy if we have a complicated function. But at least for our purpose all methods are equally satisfactory and equally efficient.

3 Methods

As we have earlier mentioned that *optim* uses 5 different optimization methods, they are

i. Nelder-Mead method

ii. BFGS

iii. CG

iv. L-BFGS-B

v. SANN

we will be discussing briefly how these methods work to find an optimum of a function.

3.1 Nelder-Mead method

The *Nelder-Mead method* is a simplex method for finding a local minimum of a function of several variables. It's discovery is attributed to J. A. Nelder and R. Mead. For two variables, a simplex is a triangle, and the method is a pattern search that compares function values at the three vertices of a triangle. The worst vertex, where is largest, is rejected and replaced with a new vertex. A new triangle is formed and the search is continued. The process generates a sequence of triangles (which might have different shapes), for which the function values at the vertices get smaller and smaller. The size of the triangles is reduced and the coordinates of the minimum point are found.

The algorithm is stated using the term simplex (a generalized triangle in n dimensions) and will find the minimum of a function of n variables. It is effective and computationally compact.

3.2 BFGS

In numerical optimization, the *Broyden–Fletcher–Goldfarb–Shanno* (BFGS) method is a method for solving nonlinear optimization problems (which lack constraints). The BFGS method approximates Newton's method, that seeks a stationary point of a (twice continuously differentiable) function. For such problems, a necessary condition for optimality is that the gradient be zero. Newton's method and the BFGS methods need not converge unless the function has a quadratic Taylor expansion near an optimum. These methods use the first and second derivatives.

In *quasi-newton method* the assumption is made that the function can be locally approximated as a quadratic in the region around the optimum, and it uses first and second derivative to obtain the 'stationary' point.

The basic idea is to select a search direction so that it is perpendicular to the search direction of the previous iteration.

3.3 CG

The conjugate gradient method is an algorithm for the numerical solution of particular systems of linear equations, namely those whose matrix is symmetric and positive-definite.

In numerical optimization, the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method is a method for solving non-linear optimization problems(which lack constraints).The BFGS method is one of the most popular class of Quasi-Newton methods. The BFGS may or may not converge.The method uses first and second derivatives.

3.4 L-BFGS-B

L-BSGS algorithms is a member of the broad family of Quasi-Newton optimization methods. L-BSGS stands for Limited memory BFGS.L-BFGS-B is a limited-memory quasi-Newton code for bound-constrained optimization, i.e. for problems where the only constraints are of the form $l \leq x \leq u$.

3.5 SANN

Simulated Annealing (SA) is a generic probabilistic technique for the global optimization problem of applied mathematics, namely locating a good approximation to the global optimum of a given function in a large search space.The search space is discrete.

SA algorithm replaces the current solution by a random "nearby" solution,chosen with a probability that depends both on the difference between the corresponding function values and also on a global parameter T that is gradually decreased during the process. The dependency is such that the current solution changes almost randomly when T is large, but increasingly "downhill" as T goes to zero.This saves the method from becoming stuck at local optima.

4 Reference

(1) wikipedia.org

(2) Deepayan Sarkar