Determination of Maximum Likelihood: Iterative Approaches

PROJECT REPORT

Aurindam Dhar, Arijit Dutta, and Kaustav Nandy

Indian Statistical Institute

Abstract. In this project, we have paid our attention to determine the Maximum Lilelihood estimators in those cases where no explicit expression of the same is found. We have computed MLE's using iterative procedures for a given problem.

1 Introduction

The problem of Statistical Inference involves the idea of acquiring some knowledge about the general nature of the phenomenon under study on the basis of a particular set of observations. It usually takes one of the two forms, viz. Estimation and Hypotheses-testing. In many spheres of life, Statistical Estimation provides us with some tools to infer about the "unknown". One such tool is Maximum Likelihood estimation. During the course of this project, we have focussed only on that kind of Maximum Likelihood estimation procedures which involves no explicit expression for the estimates.

2 Discussion on the problem

2.1 Problem in hand

- Every human being can be classified into one of the four blood groups O, A, B and AB.
- The inheritance of these blood groups is controlled by three alleomorphic genes O, A and B.
- Among them O is recessive to A and B.
- Suppose p, q and r are the gene frequencies of A, B and O respectively and $p+q+r=1$.
- We have to find the MLE's of p, q and r.

2.2 Genotype and Phenotype Probabilities

Table 1. Expected Probability of Genotypes

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

Table 2. Expected Probabilities of Phenotypes

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

2.3 Likelihood & Likelihood Equations

- Suppose n_O, n_A, n_B and n_{AB} are the observed counts of the persons with blood groups O, A, B and AB respectively.
- Total counts $= n_O + n_A + n_B + n_{AB} = n$, say.
- Clearly, $(n_O, n_A, n_B, n_{AB}) \sim$ Multinomial$(n, \alpha_1, \alpha_2, \alpha_3, \alpha_4)$, where $\alpha_1 = r^2,\ \alpha_2 = p^2+2pr,$ $\alpha_3 = q^2+2qr,\ \alpha_4 = 2pq$ and $\sum_{i=1}^{4}\alpha_i = 1$.
- The likelihood of (p,q,r) is given by:

$$L(p,q,r) = \frac{n!}{n_O!n_A!n_B!n_{AB}!}\alpha_1^{n_O}\alpha_2^{n_A}\alpha_3^{n_B}\alpha_4^{n_{AB}}$$

- The *log-likelihood* function can be written as

$$\begin{aligned}\log L(p,q) = k + 2n_O\log(1-p-q) + \ n_A\left[\log p + \log(2-p-2q)\right] + \ n_B\left[\log q + \log(2-q-2p)\right] \\ + n_{AB}\left(\log 2 + \log p + \log q\right)\end{aligned}$$

- The likelihood Equations can be obtained from the following:

$$\left.\begin{aligned}\tfrac{\partial \log L(p,q)}{\partial p} = 0 \ \Rightarrow \ \phi(p,q) = 0 \\ \tfrac{\partial \log L(p,q)}{\partial q} = 0 \ \Rightarrow \psi(p,q) = 0\end{aligned}\right\} \text{say } \ldots\ldots(*)$$

- By computation, we have,

$$\phi(p,q) = -\,n_O\left(\tfrac{2}{1-p-q}\right) + n_A\left(\tfrac{1}{p} - \tfrac{1}{2-p-2q}\right) - \ n_B\left(\tfrac{2}{2-q-2p}\right) + n_{AB}\left(\tfrac{1}{p}\right)$$

$$\psi(p,q) = -\,n_O\left(\tfrac{2}{1-p-q}\right) + n_B\left(\tfrac{1}{q} - \tfrac{1}{2-q-2p}\right) - \ n_A\left(\tfrac{2}{2-p-2q}\right) + n_{AB}\left(\tfrac{1}{q}\right)$$

3 Iterative Approach

3.1 Numerical Solution to the Likelihood Equations

- Clearly, we don't have any explicit form of p and q if we solve the set of equations $(*)$.
- That's why we go for solving $(*)$ numerically.
- We have considered the following two numerical methods:
  1. Newton-Raphson Method
  2. Fisher's Scoring Method

3.2 Newton-Raphson procedure

- Consider the case of two equations in two unknowns. Let the given equations be:

$$\begin{aligned} \phi(p,q) &= 0 \\ \psi(p,q) &= 0 \end{aligned} \tag{1}$$

- Suppose p_0, q_0 are approximate values of a pair of roots and h, k are the corrections.
- Then,

$$p = p_0 + h$$

$$q = q_0 + k$$

- From (1) we get,

$$\begin{aligned} \phi(p_0+h, q_0+k) &= 0 \\ \psi(p_0+h, q_0+k) &= 0 \end{aligned} \tag{2}$$

- Using *Taylor's theorem* and neglecting higher order terms, we get:

$$\phi(p_0+h, q_0+k) = \phi(p_0,q_0) + h\Big(\frac{\partial\phi}{\partial p}\Big)_0 + k\Big(\frac{\partial\phi}{\partial q}\Big)_0$$

$$\psi(p_0+h, q_0+k) = \psi(p_0,q_0) + h\Big(\frac{\partial\psi}{\partial p}\Big)_0 + k\Big(\frac{\partial\psi}{\partial q}\Big)_0$$

- Now the equations become:

$$\begin{aligned} \phi(p_0,q_0) + h\Big(\frac{\partial\phi}{\partial p}\Big)_0 + k\Big(\frac{\partial\phi}{\partial q}\Big)_0 &= 0 \\ \psi(p_0,q_0) + h\Big(\frac{\partial\psi}{\partial p}\Big)_0 + k\Big(\frac{\partial\psi}{\partial q}\Big)_0 &= 0 \end{aligned} \tag{3}$$

- Applying Cramer's rule we get the solution for h and k as:

$$h = \frac{1}{D}\begin{vmatrix} -\phi(p_0,q_0) & \left(\frac{\partial\phi}{\partial q}\right)_0 \\ -\psi(p_0,q_0) & \left(\frac{\partial\psi}{\partial q}\right)_0 \end{vmatrix} \text{ and } k = \frac{1}{D}\begin{vmatrix} \left(\frac{\partial\phi}{\partial p}\right)_0 & -\phi(p_0,q_0) \\ \left(\frac{\partial\psi}{\partial p}\right)_0 & -\psi(p_0,q_0) \end{vmatrix}, \quad \text{where } D = \begin{vmatrix} \left(\frac{\partial\phi}{\partial p}\right)_0 & \left(\frac{\partial\phi}{\partial q}\right)_0 \\ \left(\frac{\partial\psi}{\partial p}\right)_0 & \left(\frac{\partial\psi}{\partial q}\right)_0 \end{vmatrix}$$

- In this way, we get new estimates $p_1 = p_0 + h$ and $q_1 = q_0 + k$.
- We now follow the similar steps using p_1 and q_1 as our initial values.
- This process is repeated until we reach at the desired level of accuracy.

3.3 Fisher's approach

- Suppose $X_1, X_2, \ldots, X_n$ are *i.i.d.* observations from some distribution $f(x, \underset{\sim}{\theta})$, where $\underset{\sim}{\theta} = (\theta_1, \theta_2, \ldots, \theta_k)'$
- Suppose $L(\underset{\sim}{\theta}|\underset{\sim}{x})$ is the likelihood of $\underset{\sim}{X}$. Let $l = \log L$.
- To find MLE of $\underset{\sim}{\theta}$
- Define the following:
  - ❑ **Score Function:**

$$V(\underset{\sim}{\theta}) = \Big(\frac{\partial l}{\partial \theta_1}, \frac{\partial l}{\partial \theta_2}, \ldots, \frac{\partial l}{\partial \theta_k}\Big)'$$

  - ❑ **Information Matrix:**

$$I(\underset{\sim}{\theta}) = \begin{bmatrix} \frac{\partial^2 l}{\partial \theta_1^2} & \frac{\partial^2 l}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 l}{\partial \theta_1 \partial \theta_k} \\ \frac{\partial^2 l}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 l}{\partial \theta_2^2} & \cdots & \frac{\partial^2 l}{\partial \theta_2 \partial \theta_k} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 l}{\partial \theta_k \partial \theta_1} & \frac{\partial^2 l}{\partial \theta_2 \partial \theta_k} & \cdots & \frac{\partial^2 l}{\partial \theta_k^2} \end{bmatrix}$$

- Suppose $\underset{\sim}{\theta}^{(t)}$ is the estimate of $\underset{\sim}{\theta}$ at t^{th} stage.
- Then the estimate at $(\mathrm{t}+1)^{th}$ stage is given by:

$$\underset{\sim}{\theta}^{(t+1)} = \underset{\sim}{\theta}^{(t)} - I^{-1}(\underset{\sim}{\theta}^{(t)}) V(\underset{\sim}{\theta}^{(t)})$$

4 Convergence of iterations

- "Convergence" of an iterative procedure indicates whether the process approaches to a particular value as the no. of iterations is increased.
- A "good" iterative process should rapidly converge for a given level of accuracy.
- Here both Newton-Raphson and Fisher's Scoring methods are not so "good" in the sense that each of them may not converge if we do not choose the initial estimates appropriately.
- An argument for the non-convergence can be put in the following way.
- **Newton-Raphson:**
  Non-convergence in Newton-Raphson method arises if the matrix D becomes close to singular at some stage; i.e. if $|D| \simeq 0$, then it may lead to non-convergence.
- **Fisher's Scoring:**
  In Fisher's Scoring method if the information matrix $I(\underset{\sim}{\theta})$ becomes close to singular at some stage in the iteration process, then it may result in non-convergence.

5 Results, Analysis & Remarks

5.1 Two Datasets

- We have considered two sets of data as follows.
  1. For the first data, $n_O = 5$, $n_A = 1$, $n_B = 2$, $n_{AB} = 1$.
  2. In the second case, we have $n_O = 176$, $n_A = 182$, $n_B = 60$, $n_{AB} = 17$.
- We have written a C and C++ code to find the MLE's and obtained the following.

5.2 Analysis & Remarks

- We have noticed that for the first set of data each of the iteration methods converges and the MLE's obtained after some iteration steps are:

Parameters	**MLE for the 1st dataset**	
	`Newton-Raphson`	`Fisher's Scoring`
p	0.115308	0.115308
q	0.179187	0.179187
r	0.705505	0.705505

Parameters	MLE for the 2nd dataset	
	Newton-Raphson	Fisher's Scoring
p	0.264444	0.264444
q	0.093168	0.093168
r	0.642387	0.642387

☞ **Note :** Observe that both the methods are giving the same result.

– This is because we are solving the same set of equations in two different but equivalent ways, viz. by Cramer's rule and by matrix inversion.

6 Programming Codes

The following are the codes we have used.

C Codes

6.1 Newton Raphson Method:

```
/*Newton Raphson Method*/


#include <stdio.h>
#include <stdlib.h>
#include <math.h>

float score(float n_o, float n_a, float n_b,
    float n_ab, float p, float q);
float info11(float n_o, float n_a, float n_b,
     float n_ab, float p, float q);
float info21(float n_o, float n_a, float n_b,
     float n_ab, float p, float q);



int main(void)
{
  int i;
  float  n_a, n_b, n_ab, n_o, acc, p, q,
  a, b , c, s1, s2,
  h, k, dm;
  /*Data Entry*/
  printf("Enter the Frequencies...\n\n");
  printf("Enter the number of persons with Phenotype O: ");
  scanf("%f",&n_o);
  printf("Enter the number of persons with Phenotype A: ");
  scanf("%f",&n_a);
  printf("Enter the number of persons with Phenotype AB: ");
  scanf("%f",&n_b);
  printf("Enter the number of persons with Phenotype AB: ");
  scanf("%f",&n_ab);

  /*Initialization*/

  p = pow((n_a + n_o)/(n_a + n_b + n_o + n_ab),0.5)
    - pow(n_o/(n_a + n_b + n_o + n_ab),0.5);
  q = pow((n_b + n_o)/(n_a + n_b + n_o + n_ab),0.5)
    - pow(n_o/(n_a + n_b + n_o + n_ab),0.5);
```

```
  for(i = 1; i <= 10; i++){
    a = info11(n_o, n_a, n_b, n_ab, p, q);
    b = info11(n_o, n_b, n_a, n_ab, q, p);
    c = info21(n_o, n_a, n_b, n_ab, p, q);
    s1 = score(n_o, n_a, n_b, n_ab, p, q);
    s2 = score(n_o, n_b, n_a, n_ab, q, p);
    dm = a * b - c * c;
    h = -s1 * b + s2 * c;
    k =  s1 * c - s2 * a;
    p = p + h/dm;
    q = q + k/dm;
    printf("%f, %f\n",p,q);
  }
  return 0;
}




float score(float n_o, float n_a, float n_b,
    float n_ab, float p, float q)
{

  return (-2 * n_o/(1 - p - q) + n_a * (1/p - 1/(2 - p - 2 * q))
  - n_b * (2/(2 - q - 2 * p)) + n_ab/p);

}


float info11(float n_o, float n_a, float n_b,
     float n_ab, float p, float q)
{
return (-2* n_o/pow((1 - p -q),2.0) - n_a * (1/(p *p) + 1/pow((2 - p - 2 * q),2.0))
- n_b * 2/pow((2 - q - 2 * p),2.0) - n_ab/(p*p));
}


float info21(float n_o, float n_a, float n_b,
     float n_ab, float p, float q)
{
  return (-2 * n_o/pow((1 - p -q),2.0) - n_a * 2/pow((2 - p - 2 * q),2)
  - n_b * 4/pow((2 - q - 2 * p),2.0));
}
```

6.2 Fisher's Scoring Method:

```
/*Fisher's Scoring Method*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>


float score(float n_o, float n_a, float n_b,
    float n_ab, float p, float q);
float info11(float n_o, float n_a, float n_b,
     float n_ab, float p, float q);
float info21(float n_o, float n_a, float n_b,
     float n_ab, float p, float q);

int main(void)
{
  int i;
  float n_a, n_b, n_ab, n_o, acc, p, q,
    a, b , c, s1, s2;

  /*Data Entry*/
  printf("Enter the Frequencies...\n\n");
  printf("Enter the number of persons with Phenotype O: ");
  scanf("%f",&n_o);
  printf("Enter the number of persons with Phenotype A: ");
  scanf("%f",&n_a);
  printf("Enter the number of persons with Phenotype AB: ");
  scanf("%f",&n_b);
  printf("Enter the number of persons with Phenotype AB: ");
  scanf("%f",&n_ab);

  /*Initialization*/
   p = pow((n_a + n_o)/(n_a + n_b + n_o + n_ab),0.5)
     - pow(n_o/(n_a + n_b + n_o + n_ab),0.5);
   q = pow((n_b + n_o)/(n_a + n_b + n_o + n_ab),0.5)
     - pow(n_o/(n_a + n_b + n_o + n_ab),0.5);


  /* /\*Accuracy*\/ */
  /* printf("\n\n Enter the Level of accuracy: "); */
  /* scanf("%f",&acc); */

  for(i = 1; i <= 10; i++){
    a = info11(n_o, n_a, n_b, n_ab, p, q);
    b = info11(n_o, n_b, n_a, n_ab, q, p);
    c = info21(n_o, n_a, n_b, n_ab, p, q);
    s1 = score(n_o, n_a, n_b, n_ab, p, q);
    s2 = score(n_o, n_b, n_a, n_ab, q, p);
    p = p - 1/(a * b - c * c) * (b * s1 - c * s2);
    q = q - 1/(a * b - c * c) * (a * s2 - c * s1);
    printf("%f, %f\n",p,q);
  }
}

//Information Matrix and Score Function...

float score(float n_o, float n_a, float n_b,
```

```
    float n_ab, float p, float q)
{

  return (-2 * n_o/(1 - p - q) + n_a * (1/p - 1/(2 - p - 2 * q))
  - n_b * (2/(2 - q - 2 * p)) + n_ab/p);

}


float info11(float n_o, float n_a, float n_b,
     float n_ab, float p, float q)
{
return (-2* n_o/pow((1 - p -q),2.0) - n_a * (1/(p *p) +
       1/pow((2 - p - 2 * q),2.0))
- n_b * 2/pow((2 - q - 2 * p),2.0) - n_ab/(p*p));
}


float info21(float n_o, float n_a, float n_b,
     float n_ab, float p, float q)
{
  return (-2 * n_o/pow((1 - p -q),2.0) - n_a * 2/pow((2 - p - 2 * q),2)
  - n_b * 4/pow((2 - q - 2 * p),2.0));
}
```

C++ Codes

6.3 Newton Raphson Method (Header File):

```
#ifndef NEWTON_H
#define NEWTON_H

#include <cstdio>

class newton
{
 private:
 double no, na, nb, nab, p, q,
    a, b, c, s1, s2;
  double *info;
  double *scvec;

 public:
  newton(int n_o,
 int n_a,
 int n_b,
 int n_ab
 );
  ~newton();

  void summary();
  void result();
  void score();
  void infos();
  double p1();
  double q1();
};

#endif
```

6.4 Main File

```
//Newton Raphson Method

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cmath>

using namespace std;

#include "newton.h"

newton::newton(int n_o,
       int n_a,
       int n_b,
       int n_ab)
{
  no = n_o;
  na = n_a;
  nb = n_b;
  nab = n_ab;
  p = 0.3;
  q = 0.4;
  a = 0;
  b = 0;
  c = 0;
  info = (double*) malloc(3 * sizeof(double));
  scvec = (double*) malloc(2 * sizeof(double));
  s1 = 0;
  s2 = 0;
}


newton::~newton()
{
  cout << "Calling destructor." << endl;
  free(info);
  free(scvec);
}

void newton::summary()
{
  cout <<"p= " << p << "q= " << q << endl;
}


void newton::result()
{
  infos();
  score();
    a = info[0];
    b = info[1];
    c = info[2];

    s1 = scvec[0];
    s2 = scvec[1];

    p = p + (-s1 * b + s2 * c)/ (a * b - c * c);
```

```
    q = q + (s1 * c - s2 * a)/ (a * b - c * c);
}

void newton::score()
{

  scvec[0] = (-2 * no/(1 - p - q) + na * (1/p - 1/(2 - p - 2 * q))
      - nb * (2/(2 - q - 2 * p)) + nab/p);
  scvec[1] = (-2 * no/(1 - p - q) + nb * (1/q - 1/(2 - q - 2 * p))
      - na * (2/(2 - p - 2 * q)) + nab/q);
  return ;
}


void newton::infos()
{
info[0] = (-2* no/pow((1 - p - q),2.0) - na * (1/(p *p)
 + 1/pow((2 - p - 2 * q),2.0))
 - nb * 2/pow((2 - q - 2 * p),2.0) - nab/(p*p));

info[1] = (-2* no/pow((1 - p - q),2.0) - nb * (1/(q * q)
  + 1/pow((2 - q - 2 * p),2.0))
          - na * 2/pow((2 - p - 2 * q),2.0) - nab/(q*q));

info[2] = (-2 * no/pow((1 - p - q),2.0)
   - na * 2/pow((2 - p - 2 * q),2)
          - nb * 4/pow((2 - q - 2 * p),2.0));
}


int main()
{
  int no, na, nb, nab;
  newton *f;

  cout << "Enter the Number of persons with blood group O: ";
  cin >> no;
  cout <<"Enter the Number of persons with blood group A: ";
  cin >> na;
  cout << "Enter the Number of persons with blood group B: ";
  cin >> nb;
  cout << "Enter the Number of persons with blood group AB: ";
  cin >> nab;

  f = new newton(no, na, nb, nab);

  for (int i = 0; i < 10; i++) {
    f->result();
    f->summary();
  }
  delete f;
}
```

6.5 Fisher's Scoring Method (Header File):

```
#ifndef FISHER_H
#define FISHER_H
```

```
#include <cstdio>

class fisher
{
 private:
  double no, na, nb, nab, p, q,
    a, b, c, s1, s2;
  double *info;
  double *scvec;

 public:
  fisher(int n_o,
 int n_a,
 int n_b,
 int n_ab
 );
   ~fisher();

  void summary();
  double result();
  void score();
  void infos();
  double p1();
  double q1();
};


#endif
```

6.6 Main File

```
//Fisher's Scoring Method

#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <cmath>

using namespace std;

#include "fisher.h"

fisher::fisher(int n_o,
       int n_a,
       int n_b,
       int n_ab)

//Initializations...

{
  no = n_o;
  na = n_a;
  nb = n_b;
  nab = n_ab;
  p = pow((na + no)/(na + nb + no + nab),0.5)
    - pow(no/(na + nb + no + nab),0.5);
  q = pow((nb + no)/(na + nb + no + nab),0.5)
```

```
    - pow(no/(na + nb + no + nab),0.5);
  a = 0;
  b = 0;
  c = 0;
  info = (double*) malloc(3 * sizeof(double));
  scvec = (double*) malloc(2 * sizeof(double));
  s1 = 0;
  s2 = 0;
}

//Destructor Function...

fisher::~fisher()
{
  //  cout << "Calling destructor." << endl;
  free(info);
  free(scvec);
}


//Iterative p and q values...

void fisher::summary()
{
  cout <<"p="<< p << "q="<< q << endl;
}

//Doing the Iteration...

double fisher::result()
{
  infos();
  score();
    a = info[0];
    b = info[1];
    c = info[2];
    s1 = scvec[0];
    s2 = scvec[1];
    p = p - 1/(a * b - c * c) * (b * s1 - c * s2);
    q = q - 1/(a * b - c * c) * (a * s2 - c * s1);
}

//Calculating Score Function...

void fisher::score()
{

  scvec[0] = (-2 * no/(1 - p - q) + na * (1/p - 1/(2 - p - 2 * q))
      - nb * (2/(2 - q - 2 * p)) + nab/p);
  scvec[1] = (-2 * no/(1 - p - q) + nb * (1/q - 1/(2 - q - 2 * p))
      - na * (2/(2 - p - 2 * q)) + nab/q);
  return ;
}

//Calculating Information Matrix

void fisher::infos()
{
```

```
info[0] = (-2* no/pow((1 - p - q),2.0)
   - na * (1/(p *p) + 1/pow((2 - p - 2 * q),2.0))
   - nb * 2/pow((2 - q - 2 * p),2.0) - nab/(p*p));

info[1] = (-2* no/pow((1 - p - q),2.0)
   - nb * (1/(q * q) + 1/pow((2 - q - 2 * p),2.0))
   - na * 2/pow((2 - p - 2 * q),2.0) - nab/(q*q));

info[2] = (-2 * no/pow((1 - p - q),2.0)
   - na * 2/pow((2 - p - 2 * q),2)
   - nb * 4/pow((2 - q - 2 * p),2.0));
}

//Main Function...

int main()
{
  fisher *f;
  int no, na, nb, nab;

  cout << "Enter the Number of persons with blood group O: ";
  cin >> no;
  cout <<"Enter the Number of persons with blood group A: ";
  cin >> na;
  cout << "Enter the Number of persons with blood group B: ";
  cin >> nb;
  cout << "Enter the Number of persons with blood group AB: ";
  cin >> nab;

  f = new fisher(no, na, nb, nab);

  for (int i = 0; i < 10; i++) {
    f->result();
    f->summary();
  }
  delete f;
}
```

7 Acknowledgement & References

– Prof. Deepayan Sarkar, ISI, Delhi.

References :

(1) Linear Statistical Inference by *C.R.Rao*

(2) Numerical Mathematical Analysis by *Scarborough*