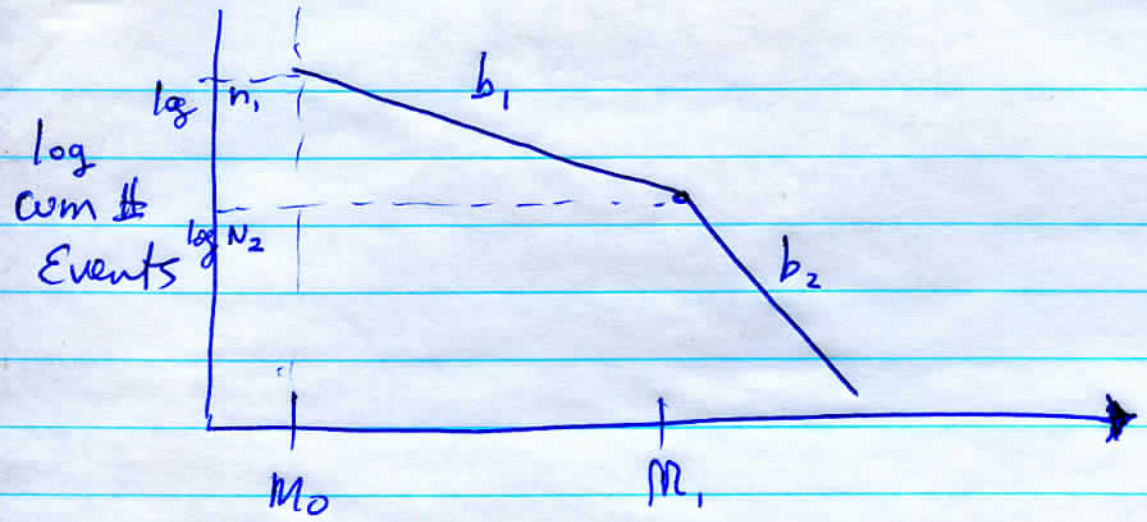
MRN 047

10/23/95

For Enrique Diep



for fixed m_1, maximum likelihood method gives these formulas for b_1, b_2:

$$b_1 = \frac{1}{\bar{m}_1 - m_0 + \frac{N_2}{N_1}(m_1 - m_0)}$$

$$b_2 = \frac{1}{\bar{m}_2 - m_1}$$

to determine best m_1, evaluate Likelihood L (see attached) for suite of m_1's, and select one with greatest liklihood.

$$P(m) = \exp\{a - (m<m_1)\, b_1 (m - m_0) - (m \geq m_1)\{b_1(m_1 - m_0) + b_2(m - m_1)\}\}$$

$$P(m) = \begin{cases} \exp\{a - b_1(m - m_0)\} & m < m_1 \\ \exp\{a - b_1(m_1 - m_0) - b_2(m - m_1)\} & m \geq m_1 \end{cases}$$

$$p = \frac{\partial P}{\partial m} = \begin{cases} b_1 \exp\{a - b_1(m - m_0)\} & m < m_1 \\ b_2 \exp\{a - b_1(m_1 - m_0) - b_2(m - m_1)\} & m \geq m_1 \end{cases}$$

$L =$

$$\ln L(p) = \sum_{<} \ln b_1 + \sum_{<} a - \sum_{<} b_1 (m_i - m_0) + \sum_{>} \ln b_2 + \sum_{>} a - \sum_{>} b_1 (m_1 - m_0) - \sum_{>} b_2 (m_i - m_1)$$

$$= N_< \ln b_1 + N_> \ln b_2 + Na - b_1 \sum m_i + b_1 N_< m_0 - N_> b_1 (m_1 - m_0) - \sum_{>} b_2 m_i + b_2 m_1 N_>$$

fixed m_1 :

$$\frac{\partial L}{\partial b_2} = 0 = \frac{N_>}{b_2} - \left(\sum_{>} m_i - N_> m_1\right) \qquad \frac{1}{b_2} = \frac{1}{N_>} \sum_{>} m_i - m_1$$

$$\frac{\partial L}{\partial b_1} = 0 = \frac{N_<}{b_1} - \sum_{<} m_i + N_< m_0 - N_> (m_1 - m_0)$$

$$\frac{1}{b_1} = \frac{1}{N_<} \sum_{<} m_i - m_0 + \frac{N_>}{N_<} (m_1 - m_0)$$

```
/* searchb, by William Menke, October 1995
   calculates extimate of two bvalues and
   corner magnitude in modified Gutenburg-Richter
   distribution */

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

#define LOG10 (2.3025851)

float m[10000];
double b1[100];
double b2[100];
double L[100];

main(argc,argv)
int argc;
char *argv[];
{
        long i, j, n;
        long n_quakes;
        long n_quakes1;
        long n_quakes2;
        double mean;
        double mean1;
        double mean2;
        long t;

        double m0;
        double m1;
        double m2;
        double mm;
        char s[100];

        double Lmax;
        long jmax;

        if( argc != 5 ) {
                fprintf(stderr,"Usage: searchb m0 m1 m2 n < catalog\n");
                fprintf(stderr,"       m0: minimum magnitude\n");
                fprintf(stderr,"       m1: minimum corner magnitude\n");
                fprintf(stderr,"       m2: maximum corner magnitude\n");
                fprintf(stderr,"       n:  number of magnitudes in search\n");
                exit(-1);
                }
        if( sscanf(argv[1],"%le",&m0) != 1 ) {
                fprintf(stderr,"cant read m0: <%s>\n", argv[1] );
                exit(-1);
                }
        if( sscanf(argv[2],"%le",&m1) != 1 ) {
                fprintf(stderr,"cant read m1: <%s>\n", argv[2] );
                exit(-1);
                }
        if( sscanf(argv[3],"%le",&m2) != 1 ) {
                fprintf(stderr,"cant read m2: <%s>\n", argv[3] );
                exit(-1);
                }
        if( sscanf(argv[4],"%ld",&n) != 1 ) {
                fprintf(stderr,"cant read n: <%s>\n", argv[4] );
                exit(-1);
                }
        if( n>=100 ) {
                fprintf(stderr,"sorry: n must be < 100\n");
                exit(-1);
```

```
        }

if( scanf("%s %s",s,s) != 2 ) {
        fprintf(stderr,"cant read past header line\n");
        exit(-1);
        }

n_quakes=0;
while( scanf("%ld%e",&t,&m[n_quakes]) == 2 ) {
        n_quakes++;
        }

printf("mc\tb1\tb2\n");

for( j=0; j<=n; j++ ) {
mm = m1 + ((double)j)*(m2-m1)/((double)n);

n_quakes1=0;
n_quakes2=0;
mean1=0.0;
mean2=0.0;
for( i=0; i<n_quakes; i++ ) {
        if( m[i]<mm ) {
                mean1 += m[i];
                n_quakes1++;
                }
        else {
                mean2 += m[i];
                n_quakes2++;
                }
        }
mean1 /= (double) n_quakes1;
mean2 /= (double) n_quakes2;
b2[j] = 1.0/(mean2-mm);
b1[j] = 1.0/(mean1-m0+ ((double)n_quakes2/(double)n_quakes1)*(mm-m0) );
L[j] = n_quakes1 * log(b1[j])
     + n_quakes2 * log(b2[j])
     - b1[j] * n_quakes1 * mean1
     + b1[j] * n_quakes1 * m0
     - b1[j] * n_quakes2 * (mm-m0)
     - b2[j] * n_quakes2 * mean2
     + b2[j] * n_quakes2 * mm;

}

jmax = 0;
Lmax = L[jmax];
for(j=0;j<=n;j++) {
        if( L[j] >= Lmax ) {
                Lmax=L[j];
                jmax=j;
                }
        }

mm = m1 + ((double)jmax)*(m2-m1)/((double)n);
printf("%f\t%f\t%f\n", mm, b1[jmax]/LOG10, b2[jmax]/LOG10 );

exit(0);
}
```