A Modified Marquardt-Levenberg Parameter Estimation Routine for Matlab.

Andreas Fahlman[1,2]✉

[1]Naval Medical Research Center
Diving and Environmental Physiology Department
8901 Wisconsin Ave
Bethesda, MD 20889-5607

[2]Department of Biology
Carleton University
Ottawa, Ontario, Canada
K1S 5B6.

Keywords: Nonlinear parameter estimation, Matlab program, Least squares, Maximum likelihood

ABSTRACT

A non-linear parameter estimation routine was written for the Matlab language. The program was used the method of least squares for parameter estimation, and a modification was made to allow estimation based on the method of maximum likelihood.

TABLE OF CONTENTS

List of Tables

List of Appendices

ACKNOWLEDGEMENTS

This work was supported by the Naval Medical Research and Development Command Work Unit No.61153N MR04101.00D-1103. The opinions and assertions contained herein are the private ones of the author and are not to be construed as official or reflecting the views of the Navy Department or the naval service at large.

INTRODUCTION

Programs for non-linear parameter estimation exist for a variety of computer languages, many of which are not compatible with current PCs, and most programs are not user friendly. To make a program that can be run on a PC, by users unfamiliar with computer programming, a modified non-linear parameter estimation program was written for Matlab. The program allows parameter estimation to be made by the method of least squares, and a modification allows estimation based on the method of maximum likelihood (1).

The non-linear parameter estimation method is based on the approach by Marquardt (5), with a modification allowing maximum likelihood estimation (1). Briefly, it can be shown that if a parameter Lambda is chosen to be large enough, the parameters (β) will always converge at the value giving the best fit by the least squares criterion (5). The smaller the value of Lambda, the faster the program will reach convergence (5). For an initial Lambda and a set of starting β values, the program will calculate a new set of β values. Next, the program compares the sum of squared errors (SSE) or the log-likelihood (LL) for the current set of parameters and compares it with the SSE or LL for the old β values. If the new set of β values reduces the deviation (reduces the SSE or increases the LL), a new set of β values are computed by first reducing the Lambda by a factor of 10. Conversely, if the deviation is increased, the new set of β values are computed by first increasing the Lambda by a factor of 10. This iterative procedure continues until the program has found a new set of β values that does not change the deviance by more than a set value from the old β values, the convergence criterion (conv). When this is achieved, the program computes the result for the converged set of parameters using propagation of error formulas (4). For mathematical justification for the routines, the interested reader is referred to the references (1, 3, 5).

THE PROGRAM

The parameter estimation routine, called "Marquardt", was written in the Matlab language (Matlab Student Edition Version 5). The program is detailed in Appendix A, and the routines required to run the program are: marquardt, mod1-mod5, E, get_data, get_function, new_array, binary and a function in the form AB_Homer1. The program uses functions instead of "goto" statements, the advantage being a more modular program that is easier to read and modify (6).

RUNNING THE PROGRAM

Save the data in a text file, where dependent (y) and independent (x) variables are ordered in columns, starting with the dependent variable as shown in Table 1 for the data set in Example 1.

Before running the program, the number of independent variables (N2), convergence criterion (CONV), number of iterations (T1), and Lambda (L) need to be specified during initialization in Marquardt. Once this is done, the program is run by typing "marquardt" at the "edu>" prompt in Matlab.

A diary file is created, named with the date and time and with extension "dry". The diary file saves the information printed to the screen in a file that may be opened with any program that is able to open text files.

Next, the user is asked to enter the data file and the function file name. The function file contains the number of parameters to be used. Currently, the function files are named with the prefix "AB_", but this can be changed in the function "Get_Function". After the name of the function has been specified, the user is asked to enter starting values for the parameter estimation. Once the starting values have been entered, the parameter search begins, with the log-likelihood or standard error for each iteration printed to the screen and saved to the diary file. At convergence, the final result, including the parameter estimates, the standard error of the parameter estimates, the coefficients of variation, and the variance-covariance matrix, is printed to the screen and saved in the diary file.

Changes in the program can be made for the number of iterations (T1), the convergence criterion (CONV), the lambda (L), and the number of independent variables (N2). This is done by changing the initilization of any of these in the "marquardt" routine.

The program uses the function "binary" to determine if the dependent variable is binary (1 or 0) or continuous. Therefore, no change is necessary when dealing with different data sets. The procedure used is seen in the output of the result, where continuous data output a SSE, and binary data a LL.

Addition of "mod8" computes the 95% confidence regions of the parameter estimate. Currently, "mod8" is only able to do this for estimation of one independent variable. After the estimation is completed, "mod8" plots the result. The approximation for the standard error of the parameters is computed using the formulas for propagation of error (1, 4).

EXAMPLES

Example 1, Continuous variables

For the estimation of continuous variables, the data set presented in Appendix C of Bailey and Homer (1977) will be used to present the non-linear least square estimation procedure. These data are presented in Table 1, and Appendix B shows the fit using the non-linear equation

$$f(x) = \beta_1 \bullet X_1^{(X2*\beta2)} \qquad [1]$$

to the continuous data.

Example 2, Binary variables

Many times in medical or biological research, one is faced with data that are binary, i.e. response or no response, death or no death etc. For these problems, one can use a probabilistic formulation. The unknown parameter becomes the probability (P) of a specific outcome or response. Accordingly, the probability of no-response is then 1-P.

In Appendix C, the binary data presented in the Table 2 is fitted to the dose response function,

$$P(x) = X_1 \bullet (\beta_1 \bullet X_1)^{-1} \qquad [2]$$

and

$$P(x) = X_1^{\beta2} \bullet (\beta_1^{\beta2} + \beta_1 \bullet X_1)^{-1} \qquad [3]$$

using the maximum likelihood technique. In these cases, the outcome (Y) is defined as either response (1) or no-response (0). The likelihood of an event for the n^{th} observation is:

$$L(n) = P^{Y(n)} \bullet (1\text{-}P)^{(1-Y(n))}$$

That is, the response (1) occurs with a probability P, and the no-response with a probability 1-P.

The likelihood for the n independent observations is the product of their outcomes:

$$LL = \sum_{i=1}^{n} L(i)$$

The estimation routine adjusts the parameters, and consequently P, to maximize the LL which is defined as the best fit to the data.

REFERENCES

1. Bailey, R.C. and Homer, L.D., Iterative parameter estimation, NMRI 76- . Naval Medical Research Institute, Bethesda, MD, 1976, p. 16.

2. Bellelli, A. and di Prisco, G., "Thermodynamic and stereochemical modelling of vertebrate haemoglobin." In: Oxygen transport in biological systems : modelling of pathways from environment to cell, Egginton, S. and Ross, H.F. (eds.) Cambridge University Press, Cambridge (U.K.), New York, p. 298, 1992.

3. Homer, L.D. and Bailey, R.C., An analogy permitting maximum likelihood estimation by a simple modification of general least squares algorithm. NMRI 77- , Naval Medical Research Institute, Bethesda, MD, pp. 1-8, 1977.

4. Ku, H.H., "Notes on the use of propagation of error formulas." Journal of Research, Vol. 70, pp. 263-273, 1966.

5. Marquardt, D.W., "An algorithm for least-squares estimation of nonlinear parameters." Journal of Soc. Indust. Applied Mathematics, Vol. 11, pp. 431-441, 1963.

6. Waite, M., Prata, S. The Waite Group's new C Primer Plus. H.W. Sams, Carmel, IN, 1993.

7. Lillo, R.S., Parker E.C., and Porter W.R. "Decompression comparison of Helium and Hydrogen in rats." Journal of Applied Physiology, 82:892-901.

8. Dromsky, D.M., Toner C.B., Survanshi S.S. , Fahlman A., Parker E.C., and Weathersby P. "The Natural History of Severe Decompression Sickness after Rapid Ascent from Air Saturation in a Porcine Model." Journal of Applied Physiology, 89:791-798.

Table 1. Data format for a file. The first row shown should be omitted, i.e. only the actual data should be presented in the form below.

Reproduced from Bailey and Homer (1)

Y	*X1*	*X2*
1	0.0001	0
0	0.0001	2
4	1	1
2	1	2
8	2	1
2	2	0

Equation 1 is used to fit the data

Table 2. Binary data for use in fitting with Equation 2 and Equation 3.

Y	*X*
0	0.6
0	0.6
0	0.6
1	1.04
1	1.04
1	1.04
1	1.44
1	1.44
1	1.44
0	2
1	2
1	2
1	2.75
1	2.75

APPENDIX A

Each routine is separated by "%%%%%%%", and its name is written within the string of %-signs. In the program, comments begin with a "%"-sign, which stops Matlab from reading the remainder of that line. Some comments have been left in for clarification. To use "mod8", erase the %-sign in front of these in "marquardt", save the file and run the program again.

```
%%%%%%%%%%%%%%%BEGIN MARQUARDT%%%%%%%%%%%%%%
function marquardt()
clear all;
   format long;
   timearray = clock;
   month = int2str(timearray(2));
   if length(month)<2
      month = ['0' month];
   end
   day = int2str(timearray(3));
   if length(day)<2
      day = ['0' day];
   end
   hour = int2str(timearray(4));
   if length(hour)<2
      hour = ['0' hour];
   end
   minute = int2str(timearray(5));
   if length(minute)<2
      minute = ['0' minute];
   end
   diaryfile = [month day '-' hour minute '.dry'];
   diary(diaryfile);
   disp(['Open diary file: ' diaryfile]);

   % read data from text file
   % input file has records in the following format {Y,X(1), %(2),...X(k)
   %only  one independent variable is allowed
   %We've defined arrays and vectors that will change and which needs
   %to be accesed through the whole program as global variables, while
   %the other variables that must not be changed are passed to each %function
   global A B G Z FUNCNAME %Matrices to be used
   global SO N BETA N2 L CONV T1 T2%global variables
      %%%%%%%%%%%%%%%%%%%%%%INITILIZATION%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   SO=0;    %initializing the standard error
   %ll=0;%If the function is using log-likelihood ll==0 least %squares
   %ll==1
   T1 =20;%input('Enter the number of iterations? ');
   N2 = 1;%input('Enter the number of independent variables? ');
   L = 1.0;%input('Enter the starting Lambda? ');
   CONV = 0.001;%input('Enter the convergence criterion? ');
   T2 = 0;%actual number of iterations

   s=GET_DATA;%gets the data array s
   w = new_array(s,N2);  %the data array
   ydat=w(:,1);          %ydat gets Y-data
   for i=1:N2
      xdat(:,i)=w(:,i+1);      %xdat gets x data
```

```
   end
   N=length(ydat);
   b_num=GET_FUNCTION;%gets the function name from the user
   ll=binary(ydat);%binary determines if ll=0 or 1
   A=zeros(b_num,b_num);%VARIANCE-COVARIANCE matrix init. to zeros
   G=zeros(b_num,1);%gradient vector initialized
   mod1(b_num,ydat,xdat,ll);%begin parameter search mod1-mod5
   %to use these
   %u=mod8(b_num,ydat,xdat,N-b_num,ll);%get confidence region and %return
   conf. region data
   diary off; %turns off the diary file
   return
```

%%%%%%%%%%%%%%%END MARQUARDT%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%BEGIN MOD1%%%%%%%%%%%%%%%

```
function mod1(b_num,ydat,xdat,ll)
global BETA G SO A N T2
global Z %Z is used in mod 6 to avoid recomputation of p for the SSE case, in
the LL case it is not used since we need to recompute new partials,  since
creation of an array of dummy x-values p needs to be computed
p=zeros(N,b_num); %partial derivative in module 2 is  size= [N,b_num]
e1=E(ydat,xdat,ll); %function call to get error vector
if ll==0%ll=test statement to see if Log-like or least squares
   SO=sum(-e1);%for log-likelihood
else
   SO=sum(e1.^2);%for least squares
end
T2=T2+1;
for i=1:b_num
      BETA(i)=BETA(i)*1.001;%BETA gets changed temporarily
      e2(:,i)=E(ydat,xdat,ll); %e2 is the new error term to be compared %to
e1
      BETA(i)=BETA(i)/1.001;%Beta gets changed back
      s(:,i)=(e1-e2(:,i));  %s=temp array to hold subtracted values for %each
X Y pair, to dec. # calc
end
for i=1:b_num
   p(:,i)=s(:,i)/(BETA(i)*0.001);%creating the partial derivative for %each y
and beta
end
Z=p;%needed if SSE in mod 6
for i=1:b_num
   if ll==0
      G(i)=sum(-p(:,i));%gradient vector for log-likelihood
   else
      G(i)=sum((p(:,i).*e1));%gradient vector determining the next
            %array multiplication is used and not matrix multiplication
   end
end
for i=1:b_num
   for j=1:b_num
      A(i,j)=sum(p(:,i).*p(:,j));%variance-covariance vector
    end
end
mod2(b_num,ydat,xdat,ll);
return
```

%%%%%%%%%%%%%%%END MOD1%%%%%%%%%%%%%%%

```
%%%%%%%%%%%%%%%BEGIN MOD2%%%%%%%%%%%%%
function mod2(b_num,ydat,xdat,ll)
global L BETA G A
for i=1:b_num
   for j=1:b_num
      q(i,j)=A(i,j)/(sqrt((A(i,i)*A(j,j))));
   end
   G(i)=G(i)/sqrt(A(i,i));
end
mod3(b_num,ydat,xdat,q,ll);
return
%%%%%%%%%%%%%%%END MOD2%%%%%%%%%%%%%
%%%%%%%%%%%%%%%BEGIN MOD3%%%%%%%%%%%%%%%
function mod3_3(b_num,ydat,xdat,q,ll)
global L BETA G A
p=zeros(1,b_num);
for i=1:b_num
   q(i,i)=q(i,i)*(1+L); %adds the gradient vector to q
end
c=inv(q);                %the inverse of q
for i=1:b_num
   for j=1:b_num
      p(i)=p(i)+c(i,j)*G(j);%estimate new partials
   end
   p(i)=p(i)/sqrt(A(i,i));
end
mod4(b_num,ydat,xdat,q,p,ll);
return
%%%%%%%%%%%%%%%END MOD3%%%%%%%%%%%%%
%%%%%%%%%%%%%%%BEGIN MOD4%%%%%%%%%%%%%
function mod4(b_num,ydat,xdat,q,p,ll)%q only needed for recursive call %to
mod3
global L BETA G SO A CONV T1 T2
if T1<0
   disp(['print results']);
   mod5(b_num,ydat,xdat,ll);
elseif T1<=T2%you're out of iterations
   tt2=num2str(T2);
   disp(['Iteration number: ' , tt2]);
   T1=T1-1;
   L=0;
elseif abs(p./BETA)< CONV %if any is larger than conv do another %interation
      disp(['CONVERGENCE']);
      T1=-1;
      L=0;
      mod1(b_num,ydat,xdat,ll);
else
   BETA=BETA+p;%modify parameters by adding their partial derivatives
   e=E(ydat,xdat,ll);%E should return the error for each X Y pair,
   if ll==0
      s1=sum(-e);
      ss1=num2str(-s1);
      disp(['Log-likelihood for next B= ' , ss1]);
   else
      s1=sum(e.^2);%summing up the squared errors
      ss1=num2str(s1); %changed to positive from negative
      disp(['SSE for next B= ' , ss1]);
```

```
   end
   if s1>SO
      L=L*10;
      BETA=BETA-p;%the estimate was worse and L is incremented by then,
      %BETA is modified back to its
      %original value and module 3 is called again, this is not counted %as
      an iteration
      mod3(b_num,ydat,xdat,q,ll);%Recursive call to mod3
   else
      L=L/10;
      mod1(b_num,ydat,xdat,ll);%call mod1 again and start over from %scratch,
      i.e new iteration
   end
end
return
%%%%%%%%%%%%%%%END MOD4%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%BEGIN MOD5%%%%%%%%%%%%%%%
function mod5_5(b_num,ydat,xdat,ll)%not needed t1 t2 l
global BETA SO N A
df = N-b_num;%the degrees of freedom
if ll==0
   V=1;%for the log-like case
   so=num2str(-SO);
   disp(['Final log-likelihood=   ',so]);
else
   V=SO./df;%summing up the squared errors, i.e. variance
   V1=sqrt(V);%stdev
   v=num2str(V);
   v1=num2str(V1);
   so=num2str(SO);
   disp(['Variance=   ',v]);
   disp(['Std.dev=    ',v1]);
   disp(['Final SSE=  ',so]);
end
C=inv(A);
A=V*C;        %Var-covar matrix
for i=1:b_num
   D(i)=sqrt(A(i,i));%D=std. error of parameter
end
D2=D./BETA; %coefficient of variation
b=num2str(BETA);
d=num2str(D);
d2=num2str(D2);
disp(['Parameters=                  ',b]);
disp(['Std. Error of Parameters=    ',d]);
disp(['Coeff of var=                ',d2])
disp(['Var-Covar matrix= ']);
disp(A)
return
%%%%%%%%%%%%%%%END MOD5%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%BEGIN GET_FUNCTION%%%%%%%%%%%%%%%
function y=GET_FUNCTION
global FUNCNAME npars BETA%FUNCNAME is global and used by other %functions to
call
%the chosen function, contains the anme of the function and its %destination
% select model
[funcfilename, pathname] = ...
```

```
uigetfile('AB_*.m','Select model function file');
FUNCNAME = strtok(funcfilename, '.');
%THE BELOW COMMAND IS HOW TO EVALUATE THE FUNCTION I.E. CALL FUNCNAME
%string = ['u=' funcname '(ydat,W,BETA);'];%creates a string to be used %in
eval
%eval(string);  %the driver to call the function
Z = [1e-10 1e-10 1e-10 1e-10 1e-10 1e-10 1e-10 1e-10 1e-10 1e-10 ];
string = ['dummy=' FUNCNAME '(0,0,Z);'];
eval(string);   % dummy call to funcc to get npars
clear Z;

for i = 1:npars%get intial betas from user
   num_param = int2str(i);
   ask_value = ['Enter initial value for beta(' num_param '): '];
   BETA(i) = input(ask_value);
end
y=npars;
return
%%%%%%%%%%%%%%%ENDGET_FUNCTION %%%%%%%%%%%%%%
%%%%%%%%%%%%%%%BEGIN GET_DATA%%%%%%%%%%%%%%
function y=GET_DATA
disp('Please provide the name of for the array of outcome data.');
[filename,pathname]=uigetfile('*.txt','Outcome Data File Name');%gives %the
name of file and path
disp(['Data file name: ' filename]);%displays the file name chosen
fullfilename=[pathname filename];
%s=dlmread(fullfilename,',');%reads the data array
in = fopen(fullfilename,'rt');%fopen(filename,permission), rt=rad and %write
a txt file
s = fscanf(in,'%f');%reads the entire file into an array, file needs to %have
the y and x variables column wise
%i.e.Y, X(1), X(2), X(3), ...X(N2) etc
y=s;%returns s, i.e. the whole data array
fclose(in); %closes data file
return
%%%%%%%%%%%%%%%END GET_DATA%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%BEGIN E %%%%%%%%%%%%%%
function y=E(ydat,xdat,ll)
global FUNCNAME
global BETA
str = ['v=' FUNCNAME '(ydat,xdat,BETA);'];%creates a string to be used %in
eval
eval(str);  %the driver to call the function
for i=1:length(ydat)%to be used when v cannot be 0
   if v(i)==0
      v(i)=0.000001;
   elseif v(i)==1
      v(i)==0.999999;
   end
end
if ll==0 %if LL=compute log-likelihood for each observation
   m=ydat.*log(v)+(1.0-ydat).*log(1.0-v);
else %if SSE return the error estimate
   m=ydat-v;
end
y=m;%returns LL or SSE also called F in all NMRI programs
%%%%%%%%%%%%%%%END E%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%BEGIN NEW_ARRAY%%%%%%%%%%%%%%
function y= new_array(W,N2)
v=0; %coounter in the for loop
col=N2+1; %col gets total number of columns
row=length(W)/col;%this is the number of data points in each column, %i.e.
rows
for i=1:row
   for k=1:col
      v=v+1;
      temp(i,k)=W(v);
   end
end
y=temp;%returns the reshaped matrix with Y in the first column and then %the
x'es
return
%%%%%%%%%%%%%%%END NEW_ARRAY%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%BEGIN MOD8%%%%%%%%%%%%%%
function y=mod8(b_num,ydat,xdat,df,ll)
global A N Z BETA%Z is a matrix containing the partial derivative p() from
mod1
T=1/(df);
T=1.96+T*(2.3724+T*(2.8227+T*(2.5561+T*1.5897)));
if df<=1.1
   T=12.706
end
xmin = min(xdat);               %gets minimum x
xmax = max(xdat);               %gets max x
xrange = xmax - xmin;           %gets the range of values

%+++++++++++++++++++++++++++BEGIN IF ELSE++++++++++++++++++
if ll==0%if LL=set all outcome to 1 and reestimate partials based on this
   newx = linspace(xmin,xmax,100+1)';%creates a data array of values at each
who
   V1=zeros(length(newx),1);%initializing
   YHAT=zeros(length(newx),1);%initializing
   yones=ones(length(newx),1);%creates an array with all ones for use in
making conf. region
   p=zeros(length(newx),b_num); %partial derivative in module 2 is by size=
[N,b_num]
   e1=E(yones,newx,ll); %function call to get error vector or LL also F in
the NMRI programs
   for i=1:b_num
      BETA(i)=BETA(i)*1.001;%BETA gets changed temporarily
      e2(:,i)=E(yones,newx,ll); %e2 is the new error term to be compared to
E1
      BETA(i)=BETA(i)/1.001;%Beta gets changed back
      s(:,i)=(e1-e2(:,i));  %s=temp array to hold subtracted values for each
X Y pair, to dec. # calc
   end
   for i=1:b_num
      p(:,i)=s(:,i)/(BETA(i)*0.001);%creating the partial derivative for each
y and beta
   end
   for q=1:length(newx)
      for i=1:b_num
         for j=1:b_num
            V1(q)=V1(q)+(p(q,i).*p(q,j).*A(i,j));%variance-covariance vector
```

```
            end
         end
      end
      SEYHAT=sqrt(abs(V1)); %the seyhat of the estimate

      YHAT=exp(e1);     %for LL estimation
      err_maxmin=T*SEYHAT;%for LL estimation
      YMIN=max(0,exp(e1-err_maxmin));%The lower end of the 95%CL region
      YMAX=min(1,exp(e1+err_maxmin));%The upper end of the 95%CL region

else %if the estimate is continuous variables and uses SSE estimation
   V1=zeros(length(xdat),1);%initializing
   YHAT=zeros(length(xdat),1);%initializing
   e1=E(ydat,xdat,ll); %function call to get error vector or LL also F in the
NMRI programs
   for q=1:length(xdat)
      for i=1:b_num
         for j=1:b_num
           V1(q)=V1(q)+(Z(q,i).*Z(q,j).*A(i,j));%variance-covariance vector,
uses Z vector from mod1
         end
      end
   end
   SEYHAT=sqrt(abs(V1)); %the seyhat of the estimate
   YHAT=ydat-e1;  %as defined in Bailey and Homer
   YMAX=YHAT+T*SEYHAT; %for SSE estimation
   YMIN=YHAT-T*SEYHAT;
   newx=xdat;
end
%+++++++++++++++++END IF ELSE STATEMENT+++++++++++++++++++++++++

plot(newx,YHAT,'b:p',newx,YMIN,'c-',newx,YMAX,'c-')
disp(['X   YHAT,         SEYHAT,    YMAX,     YMIN'])
u=[newx, YHAT, SEYHAT, YMAX, YMIN];
u=num2str(u);
disp(u)
y=u;%return data to marquardt
return
%%%%%%%%%%%%%%%END MOD8%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%BEGIN BINARY%%%%%%%%%%%%%%%
function y=binary(bin)
for i=1:length(bin)
   if (bin(i)==1) | (bin(i)==0)%if outcome 1 or zero, i.e. binary
      y=0;
   else
      y=1; %if each data not binary it use SSE and set ll=1
      return %if not 1 and 0 return immediatley no sense to continue
   end
end
return
%%%%%%%%%%%%%%%END BINARY%%%%%%%%%%%%%%
```

Variables in Use:

A(N,N)=holds information for variance-covariance matrix
G(N)=gradient vector
FUNCNAME = string for name of function
SO = sum-squared errors
N =number of observations
N1 = number of parameters
N2 = number of independent variables (x)
b_num = number of parameters
BETA = array of parameters
L = Lambda
CONV =convergence criterion
T1 = maximum number of iterations before stopping
T2 = iteration counter
V = variance of observation
V1 = standard deviation
ll = holds the type of dependent variables, binary or continuous
p(N, b_num) = partial derivatives
xdat = independent variable array
ydat = dependent variable array
e = error
e1 = error
e2 = error
Z (N, b_num) = temporary global storage of partial derivative
q = intermediate storage
q, i, j = counter variables
df = degrees of freedom

APPENDIX B

To run the data in Table 1 using the non-linear least squares method, Eq. 1 will be used to fit the data. First the data are saved as a text file and a Matlab m.file is made that defines the equation. In this case, the function file is called AB_Homer1:

```
function y=AB_Homer1(ydat,xdat,BETA)
global npars
if BETA(1)==1e-10 & BETA(2)==1e-10
   npars = 2;
   return
else
   y =  BETA(1)*(xdat(:,1).^(BETA(2)*xdat(:,2)));
end
return
```

The function can be called any name, beginning with "AB_". The "AB_" can be changed by modification of the "Get_Function" m.file. Next, "N2" in the "marquardt" routine is set to 2 for two independent variables, convergence (CONV) to 0.001, Lambda (L) to 1, and the program run with a maximum of 20 iterations (T1). Starting values for the parameters are 2 and 2 for β1 and β2, respectively. The output of the result is shown below and is equivalent to the result presented by Homer and Bailey (1).

```
Enter initial value for beta(1): 2
Enter initial value for beta(2): 2
SSE for next B= 4.9452
SSE for next B= 4.7942
SSE for next B= 4.7503
SSE for next B= 4.75
CONVERGENCE
print results
Variance=   1.1875
Std.dev=    1.0897
Final SSE=  4.75
Parameters=                2.2499     1.8301
Std. Error of Parameters=   0.54486     0.4006
Coeff of var=              0.24217    0.21889
Var-Covar matrix=
   0.296875  -0.190239
  -0.190239   0.160477
```

APPENDIX C

The maximum likelihood estimation procedure is used to fit the data in Table 2. The outcome of the data is binary, and in this case a response=1, and no-response= 0. The data is fit to the dose response functions Eq. 2 and Eq. 3.

Example 1

For Eq. 2, the parameter (β1) is the dose at which 50% of the outcome is 1. The function file for this equation is called AB_Homer2 :

```
function y=AB_Homer2(ydat,xdat,BETA)
global npars
if BETA(1)==1e-10 & BETA(2)==1e-10
   npars = 1;
   return
else
   y =  xdat./(BETA(1)+xdat);
end
```

For this estimation, the "N2" in "marquardt" needs to be set to 1, while all other variables remain as in the example above. The output of the results for a starting value of the parameter of 0.45 is shown below:

```
Enter initial value for beta(1): 0.45
Log-likelihood for next B= -6.8292
Log-likelihood for next B= -6.827
Log-likelihood for next B= -6.827
Log-likelihood for next B= -6.827
CONVERGENCE
print results
Final log-likelihood=   -6.827
Parameters=             0.49645
Std. Error of Parameters=   0.33669
Coeff of var=           0.67819
Var-Covar matrix=
  0.113360
```

Example 2

Equation 3, also known as the Hill Function, is commonly used to describe biological phenomena such as the O_2-dissociation curve (2), and has been used to describe the probability in decompression sickness (7,8). For our purpose, we use it to describe a dose-response relationship with only two outcomes. Again, the first parameter (β_1) is the dose at which 50% of the outcome is 1, and the second parameter (β_2) is the slope of the sigmoidal dose response curve. Compared to AB_Homer2, the function file for this equation only requires the following changes:

```
npars = 1;
```

and

```
y =  xdat./(BETA(1)+xdat);
```

The “marquardt” remains the same as for the first example using maximum likelihood above.

The printout of the result is shown below using the following starting parameters β_1 =0.8613 and β_{12} =3.5338, and with Lambda=1.0, conv=0.001, and T1=20:

Enter initial value for beta(1): 0.8613
Enter initial value for beta(2): 3.5338
Log-likelihood for next B= -5.5912
Log-likelihood for next B= -5.5904
Log-likelihood for next B= -5.5902
Log-likelihood for next B= -5.5906
Log-likelihood for next B= -5.5905
Log-likelihood for next B= -5.5902
Log-likelihood for next B= -5.5902
Log-likelihood for next B= -5.59
Log-likelihood for next B= -5.5904
Log-likelihood for next B= -5.5901
Log-likelihood for next B= -5.59
Log-likelihood for next B= -5.59
Log-likelihood for next B= -5.59
Log-likelihood for next B= -5.59
CONVERGENCE
print results
Final log-likelihood=   -5.59
Parameters=   0.86235   3.4309
Std. Error of Parameters=   0.34462   2.005
Coeff of var=   0.39962   0.58439
Var-Covar matrix=
0.11875   0.55232
0.55232   4.02002