THE TOBIT MODEL: AN EXAMPLE OF MAXIMUM LIKELIHOOD ESTIMATION WITH SAS/IML®

Charlie Hallahan
USDA/Economic Research Service

Introduction

The censored normal regression model is also known as the tobit model. Tobit models can be estimated with maximum likelihood estimation, a general method for obtaining parameter estimates and performing statistical inference on the estimates. Although the Tobit model can be estimated with Proc Lifereg, this paper will illustrate how the extensive library of optimization routines callable from the matrix programming language SAS/IML are available to solve nonstandard estimation problems. A summary of MLE and the optimization algorithms will be followed by a Tobit example.

Maximum Likelihood Estimation

Let Y be a random variable with probability density function (pdf) $f(y;\underline{\theta})$- or probability mass function (pf) if Y is discrete - where $\underline{\theta}$ is a p-element vector of parameters and let $\underline{Y} = [Y_1, \ldots, Y_N]'$ be a random vector where each Y_i is distributed as Y. Given a random sample $\underline{y} = [y_1, \ldots, y_N]'$ drawn from Y, the likelihood function is defined as $L(\underline{\theta};\underline{y}) = \prod_{i=1}^{N} f(\underline{y}_i;\underline{\theta})$ and $l(\underline{\theta};\underline{y}) = \log(L(\underline{\theta};\underline{y}))$ is the log-likelihood. The maximum likelihood estimate (MLE) of $\underline{\theta}$ is that value of $\underline{\theta}$, say $\hat{\underline{\theta}}$, that maximizes $l(\underline{\theta};\underline{y})$. Under fairly general conditions the MLE has the desirable properties of being consistent, asymptotically efficient, asymptotically normal, invariant and computable. By invariance is meant that if $\underline{\phi} = G(\underline{\theta})$ then $\hat{\underline{\phi}} = G(\hat{\underline{\theta}})$. References for maximum likelihood estimation include Eliason, Davidson and MacKinnon (Chapter 8), and Cramer.

When the MLE $\hat{\underline{\theta}}$ lies in the interior of the parameter space, then $\hat{\underline{\theta}}$ satisfies the likelihood equations or first-order conditions:

$$g(\hat{\underline{\theta}};\underline{y}) = \underline{0}$$

where $g(\underline{\theta};\underline{y})$ is the gradient vector or score vector,

$$g_j(\underline{\theta};\underline{y}) = \frac{\partial l(\underline{\theta};\underline{y})}{\partial \theta_j} \quad j=1,\cdots,p$$

For inference on $\hat{\underline{\theta}}$, an estimate of $V(\hat{\underline{\theta}})$, the covariance of $\hat{\underline{\theta}}$, is needed. Let $H(\hat{\theta};\underline{y}) = \frac{\partial^2 l(\hat{\underline{\theta}};\underline{y})}{\partial \underline{\theta}\, \partial \underline{\theta}'}$ be the Hessian matrix of $l(\underline{\theta};\underline{y})$ evaluated at $\hat{\underline{\theta}}$. Then

$\hat{V}(\hat{\underline{\theta}}) = [-H(\hat{\underline{\theta}};\underline{y})]^{-1}$ is

a consistent estimator of $V(\hat{\underline{\theta}})$. An alternative estimator of $V(\hat{\underline{\theta}})$ involving only first derivatives is given by using a matrix C, called the contributions to the gradient matrix by Davidson and MacKinnon. For N observations and p parameters, C is a pxN matrix with

$C_{ij}(\underline{\theta}) = \frac{\partial l_j}{\partial \theta_i}$ where $l_j = l(\underline{\theta};y_j)$ is the log-likelihood function evaluated at the j^{th} observation of y. Letting $\underline{C}_j$ be the j^{th} column of C, then $\hat{V}(\hat{\underline{\theta}}) = [\sum_{j=1}^{N} \underline{C}_j(\hat{\underline{\theta}})\underline{C}_j(\hat{\underline{\theta}})']^{-1}$.

The procedure then to carry out maximum likelihood estimation is to specify a likelihood function, use nonlinear optimization to maximize the likelihood function and, finally, calculate the standard errors of the estimates. The above discussion shows that a matrix language is a natural tool for this last step. The inclusion of many optimization routines in SAS/IML provides IML with all the pieces necessary to carry out the complete MLE procedure.

Nonlinear Optimization

IML offers a number of nonlinear optimization routines that can be applied to obtain a MLE. The available routines are:

NLPCG - Congugate Gradient
NLPDD - Double Dogleg
NLPNMS - Nelder-Mead Simplex
NLPNRA - Newton-Raphson
NLPNRR - Newton-Raphson Ridge
NLPQN - (Dual) Quasi-Newton
NLPQUA - Quadratic Optimization
NLPTR - Trust Region

and for nonlinear least squares problems:

NLPLM - Levenberg-Marquardt
NLPHQN - Hybrid Quasi-Newton

Two specialized routines are:

NLPFDD - Approximate Derivatives by Finite Differences
NLPFEA - Feasible Point Subject to Constraints

For documentation see Chapter 4 of SAS/IML Software - Changes and Enhancements through Release 6.11. Each method has its own requirements for derivative information and the kinds of constraints it can handle. For example, Newton-Raphson requires first- and second-order derivatives and handles boundary and linear constraints; quasi-Newton needs only first-order derivatives and can handle boundary, linear and nonlinear constraints; Nelder-Mead Simplex method needs no derivatives and can also handle boundary, linear and nonlinear constraints.

For those methods requiring first- and/or second-order derivatives, if the user does not supply these derivatives then they are approximated by finite difference formulas. The routine NLPFDD is helpful in checking user-supplied analytic derivatives with finite difference approximations.

The objective function for least squares problems can be written as $f(\underline{\theta}) = \frac{1}{2}(f_1^2(\underline{\theta}) + \ldots + f_m^2(\underline{\theta}))$. When using the two special least squares routines NLPLM or NLPHQN, the functions $f_i(\underline{\theta})$ are specified. For the other algorithms the nonlinear function $f(\underline{\theta})$ is passed as an argument and the special least squares structure is not taken advantage of.

When nonlinear constraints are not imposed on the parameters the optimization algorithms are feasible point methods. Given a feasible point $\underline{\theta}^{(k)}$, a search direction $\underline{s}^{(k)}$ is found and the next feasible point $\underline{\theta}^{(k+1)}$ determined. If an initial feasible point is not supplied, the optimzation routines call NLPFEA to find one.

The NLP procedure in SAS/OR can also do nonlinear optimization (see Hartmann and Ashton for a comparison of IML with NLP). An advantage of IML is the flexibility of the matrix language in formulating likelihood functions and calculating standard errors. NLP is faster and can compute analytic first- and second-order derivatives with a special compiler.

The syntax for calling the various optimization routines is essentially the same. The Quasi-Newton routine NLPQN will be used for illustrative purposes. Since NLPQN uses first derivatives, it is better that the user provide analytic derivatives instead of relying on finite difference approximations.

The first step is to write an IML function module defining the function to be maximized, in our case the log-likelihood. The second example presented below in more detail is a probit model. Call its log-likelihood function LL and let theta be the (row) vector of arguments for LL, i.e., the parameters of the underlying distribution. For use with the optimization routines the function module LL can only have the distribution parameters passed as arguments when LL is called. Other quantities needed to evaluate LL, such as the observed data, can be passed to LL via the global option. In this example the data consists of three variables, dose, n and response. A skeleton of the module LL is:

```
start LL(theta) global(dose, n, response);
        . (body of function)
        f = ... ;
        return(f);
finish LL;
```

The second module needed for this example defines the gradient of LL. Since the parameter vector theta has two components in this case, the gradient is a 1x2 row vector. A skeleton of the gradient module is:

```
start GRAD(theta) global(dose, n, response);
        nparm = ncol(theta);
        g = j(nparm,1,0);
        g[1] = ...;
        g[2] = ...;
        return(g);
finish GRAD;
```

For standard errors the user can define modules to calculate either the pxp Hessian matrix or pxN contributions to the gradient matrix CG defined above. Skeletons of these modules are:

```
start HESS(theta) global(dose, n, response);
```

```
        nparm = ncol(theta);
        H = j(nparm,nparm,0);
        H[1,1] = ...;
                .
        H[nparm,nparm] = ...;
        return(H);
finish HESS;

start CG(theta) global(dose, n, response);
        nparm = ncol(theta);
        nobs = nrow(dose);
        CG = j(nparm,nobs,0);
        CG[1,1] = ...;
                .
        CG[nparm,nobs] = ...;
        return(CG);
finish CG;
```

In most cases there will be a pattern to the matrix CG and it can be defined in a loop.

Once modules for the function and its derivatives have been defined, the IML function NLPFDD is useful to compare the user-defined analytic derivatives with finite difference approximations produced by NLPFDD. The syntax for NLPFDD is:

```
CALL NLPFDD(f,g,h,"fun",x0<,par,"grd">);
```

The argument "fun" refers to the user-defined module for the function to be optimized. In our case, we're calling this function LL. If "fun" returns a scalar, then NLPFDD returns the function f, gradient g and Hessian h evaluated at the point x0. Also, for a scalar "fun" the optional argument "grd" refers to a user-defined module (which in our example is called GRAD) which calculates analytic derivatives. NLPFDD then reports on any discrepancies between the analytic and approximate derivatives. The user could do this directly as follows:

```
theta = (0.5 0.3);
grad1 = GRAD(theta);
call NLPFDD(f,g,h,"LL",theta);
print `For theta = ` theta ` analytic gradient = `
        grad1 ` approximate gradient = ` g;
```

If "fun" returns a column vector of m function values, then NLPFDD assumes a least-squares function is specified and calculates the vector f, Jacobian matrix J and cross-product matrix J`J evaluated at x0. The now required argument par has three components: par[1] specifies m, the number of values returned by "fun" and the other components control details of the approximation and precision.

The syntax for Quasi-Newton Optimization is:

```
CALL NLPQN(rc,xr,"fun",x0<,...>);
```

<,...> refers to a list of 8 optional, positional arguments. These arguments also have an alternative keyword-type specification that avoids counting commas. For example, to pass the name of the gradient module to NLPQN:

```
CALL NLPQN(rc,xr,"LL",theta,,,,,,"GRAD");
```

or

```
CALL NLPQN(rc,xr,"LL",theta) grd="GRAD";
```

The argument rc represents a return code ranging from -10 to +10 indicating the status at termination of the optimization problem. A return code rc > 0 indicates successful termination. A summary of the values of rc is on p. 161 of the SAS/IML reference.

The row vector xr is the optimal solution when rc > 0. A variety of user-controlled options are available with the other arguments; for example, specifying constraints, termination criteria and algorithmic-specific variations. Through the "ptit" argument the user can provide an IML module to control the iteration history and termination criteria.

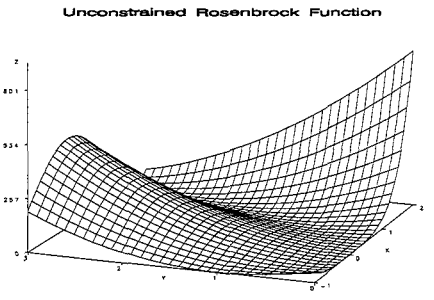
An Illustrative Example

Before presenting the Tobit model a simple example with the Rosenbrock function will be given to illustrate the mechanics of using the IML optimization routines. It is discussed in the SAS/IML Software reference.

The Rosenbrock function is defined as

$$f(x) = \frac{1}{2}[100(x_2 - x_1^2)^2 + (1 - x_1)^2]$$
$$= \frac{1}{2}[f_1^2(x) + f_2^2(x)], \quad x=(x_1,x_2)$$

The minimum occurs at $x^* = (1,1)$. A plot of f(x) is given below.



The IML modules defining the function, gradient and Hessian are listed below. Since the Trust Region Method is used, second derivatives are needed.

```
* define function;
start F_ROSEN(x);
  y1 = 10. * (x[2] - x[1]*x[1]);
  y2 = 1. - x[1];
  f = .5 * (y1*y1 + y2*y2);
  return(f);
finish F_ROSEN;
* define gradient;
start G_ROSEN(x);
  g = j(1,2,0.);
  g[1] = -200. * x[1] * (x[2] - x[1]*x[1]) - (1. - x[1]);
  g[2] = 100. * (x[2] - x[1]*x[1]);
  return(g);
finish G_ROSEN;
* define Hessian matrix;
start H_ROSEN(x);
  h = j(2,2,0.);
  h[1,1] = -200. * (x[2] - 3.*x[1]*x[1]) + 1.;
  h[2,2] = 100;
  h[2,1] = -200. * x[1];
  h[1,2] = h[2,1];
  return(h);
finish H_ROSEN;
* define starting values;
x = {2. 2.};
* define options;
* 1st arg defines type of problem: = 0 means
     'minimization problem';
* 2nd arg controls output: = 2 means iteration history +
     initial & final parameter estimates;
optn = {0 2};
* syntax for NLPTR: call
  nlptr(rc,xr,"fun",x0,<opt,blc,tc,par,"ptit","grd","hes">);
call nlptr(rc,xres,"F_ROSEN",x,optn) grd="G_ROSEN"
     hes="H_ROSEN";
reset noname;
print 'Solution = ' xres;
```

Various combinations of starting values and specifying the gradient and/or Hessian were tried. The results are below.

Start	Solution	Gradient	Hessian
(2.0 2.0)	(1.0000 1.0000)	Y	Y
(2.0 2.0)	(1.0000 1.0000)	Y	N
(2.0 2.0)	(0.9999 0.9999)	N	N
(-2.0 -2.0)	(0.9999 0.9999)	Y	N
(-20.0 20.0)	(1.0000 1.0000)	Y	N

The Tobit Model

The tobit model is also known as a censored regression model (see Breen & Greene for thorough discussions of the tobit model). A censored model is similiar to a truncated model. In a censored model some sample values are reported at a limit value instead of at actual values, and in a truncated model only non-limit values are reported, i.e., the truncated sample is drawn from a subset of the target population. For example, in a studying household incomes for a population of households, a minimum value of household income of $10,000 may be set and all values below $10,000 are reported at the limit value of $10,000. If the sample consists of observations both above and at the limit then the sample is censored. If only observations above the limit are in the sample, then the sample is truncated.

The tobit model can be described in terms of a latent variable y^*. Suppose $y^* = \underline{\beta}'\underline{x} + \epsilon$ where $\epsilon \sim N(0, \sigma^2)$ and the observed variable y satisfies :

$$y = y^* \quad \text{if } y^* > a$$
$$y = a \quad \text{if } y^* \le a$$

for some limit value a. Truncation from above can be similarly treated. y is called a censored normal variate. It can be shown that

$$E[y/x] = \Phi(\alpha)\, a + (1 - \Phi(\alpha))\,(\mu + \sigma\lambda(\alpha))$$

where $\alpha = (a - \mu)/\sigma$, $\lambda(\alpha) = \phi(\alpha)/(1 - \Phi(\alpha))$, $\mu = \underline{\beta}'\underline{x}$ and ϕ and Φ are the standard normal density and distribution functions respectively (see Greene, p. 692). $\lambda(\alpha)$ is called the inverse Mills ratio. Therefore, the marginal effects are

$$\partial\, E[y^* / \underline{x}\,] / \partial\, \underline{x} = \underline{\beta}$$

and

$$\partial\, E[y / \underline{x}\,] / \partial\, \underline{x} = \underline{\beta}\, \Phi((\underline{\beta}'\underline{x} - a)/\sigma).$$

Note that the marginal effect on $E[y^* / \underline{x}]$ is the usual formula for a linear model, but that the marginal effect on the mean of the censored variable y is a positive multiple of $\underline{\beta}$. The marginal effect can be easily calculated in IML.

In deriving the log likelihood function for the censored regression model, we can assume that the limit value a = 0 (otherwise, define z = y - a, z is censored at 0).

$$\ln L = -\tfrac{1}{2} \Sigma_1 (\ln(2\pi) + \ln(\sigma^2) + (y_i - \underline{\beta}'\underline{x}_i)^2 / \sigma^2) + \Sigma_0 \ln(1 - \Phi(\underline{\beta}'\underline{x}_i/\sigma))$$

where the first sum Σ_1 is over the non-censored observations and the second sum Σ_0 is over the censored observations.

While not required, the first derivatives of ln L with respect to the parameters β and σ can be carefully worked out and , even more carefully, coded into an IML module to calculate the gradient vector. Fortunately, the IML module NLPFDD is available to catch the inevitable errors made on the first ten attempts.

The example on page 1024 of the SAS/STAT User's Guide uses PROC LIFEREG to estimate a tobit model.

```
title 'TOBIT.SAS: MLE using IML of Tobit Model for
Durable Goods Expenditures';
data test;
   input durable age lqty @@;
   if durable = 0 then lower = .;
      else lower = durable;
   limit = lower;
   label durable = 'Durable Goods Purchase'
         age = 'Age in Years'
         lqty = 'Liquidity Ratio Times 1000';
cards;
0.0 57.7 236 0.0 59.8 216 10.4 46.8 207  0.0 39.9 219
0.7 50.9 283 0.0 44.3 284  0.0 58.0 249  0.0 33.4 240
0.0 48.5 207 3.7 45.1 221  0.0 58.9 246  3.5 48.1 266
0.0 41.7 220 0.0 51.7 275  0.0 40.0 277  6.1 46.1 214
0.0 47.7 238 3.0 50.0 269  1.5 34.1 231  0.0 53.1 251;
proc lifereg;
   model (lower,durable) = age lqty / d=normal
 nolog itprint corrb covb;
run;
```

Some of the output is:

```
        Log Likelihood for NORMAL -28.92596097
          Last Evaluation of the Gradient

INTERCPT         AGE         LQTY        SCALE
3.6226E-10  1.8215E-8  8.69292E-8 -1.49385E-9

Variable  DF Estimate  Std Err ChiSquare
INTERCPT  1 15.2771208 16.03272  0.907964
AGE       1 -0.1340075 0.218931  0.374664
LQTY      1 -0.0451356 0.058269  0.600026
SCALE     1 5.56935051 1.728145
```

```
Estimated Covariance Matrix

          INTERCPT     AGE      LQTY     SCALE
INTERCPT  257.0481  -1.7205  -0.7251   1.36720
AGE        -1.7205   0.0479  -0.0019  -0.07174
LQTY       -0.7251  -0.0019   0.0033  -0.00053
SCALE       1.3672  -0.0717  -0.0005   2.98648
```

A macro using SAS/IML to estimate the tobit model is now listed:

```
%macro tobit(dsn=_last_,y=,x=,cutoff=0,theta=0);
   proc iml;
     reset noname;
     use &dsn;
     read all var {&x} into x;
     read all var {&y} into y;
     yx = y || x;
     * a = lower truncation point;
     a = &cutoff;
     * define var indicating censored obs;
     limit = y;
     limit[loc(y=a)] = .;
     n = nrow(y);
     n0 = ncol(loc(limit=.));
     print n0 " observations are censored at
       &cutoff out of " n " obs";

     * define log likelihood function;
     start LL(theta) global(yx, limit, a);
        * split data into censored and uncensored obs;
        yx1 = yx[loc((limit ^= .)),];
        n1 = nrow(yx1);
        k = ncol(yx1);
        y1 = yx1[,1];
        x1 = j(n1,1,1) || yx1[,2:k];
        yx0 = yx[loc((limit = .)),];
        n0 = nrow(yx0);
        k = ncol(yx0);
        y0 = yx0[,1];
        x0 = j(n0,1,1) || yx0[,2:k];
        * separate param vector into beta and sigma;
        k = ncol(theta);
        k1 = k-1;
        beta = theta[1:k1] ;
        sigma = theta[k];
        sigma2 = sigma*sigma;
        alpha = (a - x0*beta)/sigma;
        n = nrow(y);
        pi = arcos(-1);
        chk = probnorm(alpha);
        zero = loc(chk<=0);
        if nrow(zero) >= 1 then chk[zero] = 0.1e8;
        f = -(n1/2) *(log(2*pi) + log(sigma2))
            - ssq(y1 - x1*beta)/(2*sigma2) +
              (log(chk))[+];
        return(f);
     finish LL ;

     * define gradient;
     start GRAD(theta) global(yx, limit, a);
        * split data into censored and uncensored obs;
        yx1 = yx[loc((limit ^= .)),];
        n1 = nrow(yx1);
        k = ncol(yx1);
        y1 = yx1[,1];
        x1 = j(n1,1,1) || yx1[,2:k];
        yx0 = yx[loc((limit = .)),];
```

```
      n0 = nrow(yx0);
      k = ncol(yx0);
      y0 = yx0[,1];
      x0 = j(n0,1,1) || yx0[,2:k];
      * define gradient vector;
      k = ncol(theta);
      g = j(1,k,0);
      * separate parms into beta and sigma;
      k1 = k-1;
      beta = theta[1:k1] ;
      sigma = theta[k];
      sigma2 = sigma*sigma;
      n1 = nrow(y1);
      pi = arcos(-1);
      alpha = (a - x0*beta)/sigma;
      phi =
          1/(sqrt(2*pi))*exp(-alpha#alpha/2);
      lambda = phi/probnorm(alpha);
      temp1 = (y1 - x1*beta)/sigma2;
      temp2 = lambda/sigma;
      g[1,1:k1] = temp1`*x1 - temp2`*x0;
      g[1,k] = -n1/sigma + ((y1 - x1*beta)#(y1 -
               x1*beta)/(sigma2*sigma))[+] -
               ((alpha#lambda)/sigma)[+];
       return(g);
   finish GRAD ;

   optn={1 2};
   * define starting values for theta:
     either user-provided or OLS;
   theta0 = &theta;
   if theta0 = 0 then
      do;
         k = ncol(yx) + 1;
         n = nrow(y);
         theta0 = j(1,k,0);
         xx = j(n,1,1) || x;
         beta = inv(xx`*xx)*xx`*y;
         e = y - xx*beta;
         s = sqrt(ssq(e)/(n-k));
         theta0[1:(k-1)] = beta`;
         theta0[k] = s;
         print 'OLS starting values for theta = '
               theta0;
      end;
    else print 'User-supplied starting values for
         theta = ' theta0;
   call nlpnrr(rc,theta,'LL',theta0,optn)
        grd='GRAD';
   call nlpfdd(f,g,h,'LL',theta);
   var = inv(-h);
   sd = sqrt(vecdiag(var));
   print 'Hessian = ' h '  Covariance matrix = ' var
         ' Standard errors = ' sd;
   run;
%mend;

%tobit(dsn=test,y=durable,x=age lqty);
```

Some of the output is:

```
13  observations are censored at 0 out of    20  obs
OLS starting values for theta =  11.135579 -0.027717
-0.034506 2.7914862


 Parameter          Estimate     Gradient
  1   X1          15.277121   -1.086E-10
  2   X2          -0.134008   -5.527E-9
  3   X3          -0.045136    -2.8239E-8
  4   X4           5.569351     4.3235E-10
   Value of Objective Function = -28.92596097
    Standard errors =  16.067133
                       0.219058
                       0.0584599
                       1.7280038
```

Comparison of the LIFEREG and macro output shows agreement for the parameter estimates and standard errors. The advantage of IML is that quantities of interest, such as the marginal effects, can be computed with just an additional line of code.

Conclusions

SAS/IML provides all the tools necessary to perform maximum likelihood estimation and inference. IML's flexibility places all aspects of the MLE procedure under user control. To illustrate the process a Tobit model was presented and estimated with IML.

References

Breen,R. (1996), *REGRESSION MODELS Censored, Sample Selected, or Truncated Data*, SAGE Publications

Cramer,J.S. (1986), *Econometric applications of Maximum Likelihood methods*, Cambridge University Press

Davidson,R. and J. MacKinnon, (1993), *Estimation and inference in econometrics*, Oxford University Press

Eliason,S.R. (1993), *Maximum Likelihood Estimation*, SAGE Publications

Greene,W., (1993), *Econometric Analysis*, Macmillan

Hartmann,W. and J. Ashton, (1995), *Maximum Likelihood Estimation with PROC NLP and SAS/IML Software*, Proceedings of the 20th SAS Users Group International Conference, 311-317

SAS/IML Software: Changes and Enhancements through Release 6.11, (1995), SAS Institute

SAS/STAT User's Guide, Version 6, Fourth Edition, Volume 2 (1990), SAS Institute

The author can be contacted at:

Charles Hallahan
USDA/ERS/ISD, RM 212
1301 New York Ave, NW
Washington, DC 20005

hallahan@econ.ag.gov
202-501-6928
202-219-0112 (fax)