# A PRIMER OF MAXIMUM LIKELIHOOD PROGRAMMING IN STATA

**Marco R. Steenbergen**[*]

2012 (Revised Edition)

**Abstract**

Stata provides a powerful and user-friendly platform for programming your own likelihood functions. These notes discuss and illustrate some of Stata's programming facilities. In addition, procedures for performing likelihood ratio and Wald tests are shown.

[*]Correspondence: Chair of Political Methodology, Department of Political Science (IPZ), University of Zurich, Zurich Ch-8050, Switzerland. Email: `steenbergen@ipz.uzh.ch`. The present document is a completely revised and updated version of my earlier text, *Maximum Likelihood Programming in Stata*, which has been circulating on the Internet for some time now.

## Introduction

Maximum likelihood-based methods are now so common that most statistical software packages have "canned" routines for many of these methods. Thus, it is rare that you will have to program a maximum likelihood estimator yourself. However, should this need arise (for example, because you are developing a new method or want to modify an existing one), then Stata offers a user-friendly and flexible programming language for maximum likelihood estimation (MLE).

This document describes the basic syntax elements that allow you to write and execute MLE routines in Stata.[1] The goal is to get you off the ground and running while working with Stata. There is no intention to provide a comprehensive overview of Stata's capabilities for programming and evaluating likelihood functions; for this, the reader is referred to the Stata documentation and the excellent book by Gould, Pitblado and Poi (2010).

## Features

Stata has many nice features, including: (1) quick convergence (under most circumstances) using the Newton-Raphson algorithm or a variety of quasi-Newton algorithms; (2) a conservative approach to declaring convergence, which leads to more trustworthy estimates; (3) simplifying features that allow for the implementation of MLE with a minimum of calculus; (4) robust variance estimation; (5) Wald and likelihood ratio test procedures; (6) a search routine that chooses improved starting values; (7) estimation under linear constraints; and (8) a variety of post-estimation commands. These features make Stata one of the most powerful and user-friendly MLE programming languages to work with.

## Syntactic Structure

Programming and executing MLE routines in Stata requires a specific sequence of commands. These may be part of an ado file, or they can be entered interactively. The following shows the sequence of commands and explains their

[1] The syntax shown here was tested in versions 8-12 of Stata.

meaning. Optional commands are indicated by an asterisk.

1. Program instructions: The program specifies the parameters and log-likelihood function. This is done in general terms, so that the commands can be used in any application where they are relevant. (The program may be kept in a separate `ado` file.)

2. `ml model`:[2] This command specifies the model that is to be estimated (i.e., dependent variable and predictors), as well as the MLE program that should be run and the way in which it should be run. This command is application-specific: it specifies the model in terms of the particular set of variables that is loaded into memory.

3. `ml check`*: This command checks the program syntax for mistakes. While optional, it is extremely useful for debugging MLE routines. Beginning programmers are advised to use this command.

4. `ml search`*: This optional command causes Stata to search for better starting values for the numerical optimization algorithm.

5. `ml init`*: An alternative command for specifying starting values.

6. `ml maximize`: This command starts the execution of the estimation commands and generates the output.

7. `ml graph`*: This is an optional command that produces a graph showing the iteration path of the numerical optimization algorithm. I recommend using this command so that one can monitor convergence.

8. `ml display`*: This optional command displays the results and is useful in case one has transformed one or more of the parameters. A discussion of this command appears in the Output section.

## Program Instructions

In most cases, writing an MLE program requires only a couple of lines of syntax. This is the case, at least, if (1) the log-likelihood function meets the linear

[2]In this document, I indicate Stata commands in `print type`.

form restriction—i.e., the observations are (conditionally) independent—and (2) Stata derives the first and second (partial) derivatives numerically (these derivatives are needed for numerical optimization). For the remainder of these notes, it is assumed that these conditions are met, i.e., we operate in `lf` mode. In most cases, this will be true (for the remaining modes see Gould, Pitblado and Poi, 2010).

The program is started by entering

```
program define name
```

where *name* is any name up to 32 characters in length. (It is preferable to choose a descriptive name.) The user may abbreviate `program define` to `pr de`. To end the program, one should type

```
end
```

In between these keywords, the user has to declare the parameters and the log-likelihood function. First, the log-likelihood function and the estimated parameters have to be labeled. This is done through the command `args` (which is an abbreviation for the computer term "arguments"). Next, the log-likelihood function has to be defined; this is done using the `quietly replace` command.[3] In addition to these specifications, it is often useful to declare the program version, especially if you are planning to make changes to the program over time. An example can illustrate the various programming commands.

**Example 1** To show the use of these commands, consider the Poisson distribution

$$f(y|\mu) = \frac{\mu^y}{y!}e^{-\mu}$$

Here $\mu$ is the parameter that we want to estimate. For a sample $n$ independent observations, this distribution produces the following

[3] "Replace" indicates that the user is substituting a new expression. "Quietly" implies that Stata does not echo this substitution—i.e., it is not displayed on the screen or in the output.

log-likelihood function:

$$\ell = \ln\mu\sum_i y_i - n\mu - \sum_i \ln y_i!$$

To program this function, we could use the following syntax

```
program define poisson
   version 12.1
   args lnf mu
   quietly replace ‘lnf’ = ln(‘mu’)*$ML_y1 - ///
   ‘mu’ - lnfact($ML_y1)
end
```

Let us analyze what this program does. In the first line we define the program, calling it `poisson`. In the second line, we show the version of the program (`version 12.1`). Here, I use the convention of declaring the version of Stata for which the program was written before the period (version 12) and the program version after the period (version 1). The third line provides a name for the log-likelihood function (`lnf`) and its one parameter (`mu`). The fourth and fifth line specify the log-likelihood function.[4] The last line ends the program.

The action, of course, is in the fourth and fifth lines. These lines are based on the arguments specified in `args`. Because we are referring to arguments, they should be placed in apostrophes.[5] The fourth line also contains the variable `$ML_y1`, which is the internal label for the (first) left-hand side (or dependent) variable. Stata will replace this with the appropriate variable from the data set after the `ml model` command has been issued. By not referring to a specific variable and using a generic placeholder, the program can be used for any data set. Finally, the fourth and fifth lines specify the likelihood function.[6]

[4]The symbol /// at the end of the fourth line is the normal Stata syntax continuation separator.

[5]While it does not appear this way, the first apostrophe is actually backward leaning.

[6]The last term in this function, `lnfact($ML_y1)`, stands for $\ln y!$. It could also have been omitted since this term does not contain the parameter of interest and is hence not a part of the kernel of the log-likelihood function.

If the fourth and fifth lines show the log-likelihood function, one may wonder what has happened to the summation signs. Actually, the function we have specified in the program gives the log-likelihood for a single observation. As long as the linear form restriction on the log-likelihood is met, this is all one has to specify. Stata knows that it should evaluate the function for every observation and sum the results. The fact that, under the linear form restrictions, one only has to specify the log-likelihood function for one observation greatly simplifies programming. This works because the linear form restriction allows one to decompose the log-likelihood as a linear function

$$\ell \;=\; \sum_{i=1}^{N} \ell_i$$

This requires, however, if the observations are (conditionally) independent. If this is not the case, then programming becomes considerably more complex. This situation falls outside the scope of these notes but is described in considerable detail in Gould, Pitblado and Poi (2010). Note that `lf` is the preferred mode in Stata because of its speed.

**Example 2(a)** Consider the normal distribution

$$f(y|mu, \sigma^2) \;=\; \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2}\frac{(y-\mu)^2}{\sigma^2}\right\}$$

The kernel of the log-likelihood for a single observation is

$$\ell_i \;=\; -.5\ln\sigma^2 - .5\frac{(y_i-\mu)^2}{\sigma^2}$$

We can program the log-likelihood for this observation using the following syntax:

```
program define normal
   version 12.1
   args lnf mu sigma2
   quietly replace ‘lnf’ = .5*ln(‘sigma2’) - ///
```

```
        (1/(2*‘sigma2))*($ML_y1 -‘mu’)^2
    end
```

**Example 2(b)** The normal likelihood function can also be written differently. Noting that,

$$f(y) \quad = \quad \frac{1}{\sigma}\phi(z),$$

where $\phi(.)$ is the standard normal probability density function and $z = (y-\mu)/\sigma$, the log-likelihood for a particular unit may be written as

$$\ell_i \quad = \quad \ln\sigma + \ln\phi(z_i)$$

This formulation of the log-likelihood function produces the following syntax:

```
    program define normal
        version 12.2
        args lnf mu sigma
        tempvar z
        quietly {
        gen double ‘z’=($ML_y1 - ‘mu’)/‘sigma’
        replace ‘lnf’=ln(normd(‘z’)) - ln(‘sigma’)
        }
    end
```

Here `normd` is the Stata command for the standard normal density. Note that the present program estimates $\sigma$ rather than $\sigma^2$.

The syntax in Example 2(b) illustrates a practice that can help to simplify programming (or, in any case, make programs more readable): the creation of temporary variables. Since the standard normal density takes $z$ and not $y$ as its argument, the program went ahead and created it as a temporary variable (`tempvar`). Temporary variables are variables that are not stored in the data set; they reside entirely within a program and disappear as soon as the program has done its work. The temporary variable is explicitly generated in the sixth line of the program, using double precision.[7] The creation of the variable is nested within the `quietly` command, which means that the user will never see that `z` is being created; the whole operation occurs in the background.

So far, the programs shown here have dealt with one left-hand side variable. It is, however, relatively straightforward to write programs with multiple left-hand side variables. Example 3 illustrates this. It also shows how clever transformations can help to avoid common convergence problems.

**Example 3** Consider the bivariate normal distribution

$$f(y_1, y_2) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left\{-\frac{z_1^2 + z_2^2 - 2\rho z_1 z_2}{2(1-\rho^2)}\right\}$$

where $z_1 = (y_1 - \mu_1)/\sigma_1$ and $z_2 = (y_2 - \mu_2)/\sigma_2$. Assuming that the data consist of $n$ independent draws from this distribution, the kernel of the log-likelihood function for a single observation is given by

$$\begin{aligned} \ell &= -.5\ln\sigma_1^2 - .5\ln\sigma_2^2 - .5\ln(1-\rho^2) - \\ &\quad \frac{1}{2(1-\rho^2)}\left\{z_{i1}^2 + z_{i2}^2 - 2\rho z_{i1} z_{i2}\right\} \end{aligned}$$

The following program declares this log-likelihood function for a single observation.

```
program define bivnorm
   version 12.1
   args lnf mu1 mu2 lnv1 lnv2 z
   tempvar v1 v2 z1 z2 r t
   quietly {
   gen double ‘v1’=exp(‘lnv1’)
```

[7]It is essential to use double precision in order to avoid inaccurate estimates.

```
        gen double ‘v2’=exp(‘lnv2’)
        gen double ‘z1’=($ML_y1-‘mu1’)/sqrt(‘v1’)
        gen double ‘z2’=($ML_y2-‘mu2’)/sqrt(‘v2’)
        gen double ‘r’=(exp(2*‘z’)-1)/(exp(2*‘z’)+1)
        gen double ‘t’=1-‘r’^2
        replace ‘lnf’=-.5*ln(‘v1’) - ///
        .5*ln(‘v2’) - .5*ln(‘t’) -(1/(2*‘t’))* ///
        (‘z1’^2 + z2’^2 - 2*‘r’*‘z1’*‘z2’)
        }
    end
```

A couple of things may seem strange about this syntax. First, the log-likelihood function has five parameters: $\mu_1$, $\mu_2$, $\sigma_1^2$, $\sigma_2^2$, and $\rho$. But if one inspects the args command, only two of these parameters appear—$\mu_1$ and $\mu_2$. The other parameters declared in args are not listed in the log-likelihood function.

The second thing that may seem odd are the gen commands for v1, v2, and r. These commands actually create $\sigma_1^2$, $\sigma_2^2$, and $\rho$. But why would not one declare these directly in the args command? Why take the roundabout way of estimating lnv1, lnv2, and z and only then construct the quantities of real interest? The answer lies in avoiding estimates at the boundary of parameter space. For parameters like $\mu$, the parameter space stretches from negative to positive infinity and is, as such, unbounded. This is different for parameters like $\sigma^2$, which cannot be negative and are thus bounded from below. The bounds on $\rho$ are even more restrictive: this parameter is bounded between -1 and 1. When parameters are restricted in this manner, numerical optimization can become complicated when one approaches the boundaries. The reason is that if one pushes past those boundaries, the log-likelihood function may no longer be defined. You can see this for yourself when you imagine what would happen if, for example, $\rho$ was pushed outside of its boundary, for example to a value of 1.1.[8] To avoid these problems, it is good programming practice to

[8]Truth be told, Stata is actually quite good at avoiding these problems on its own. Still, the transformation of parameters to unbounded quantities can be extremely helpful.

make all estimated parameters unbounded. In Example 3, `lnv1`, `lnv2`, and `z` can take on any value on the real number line. By feeding `lnv1` and `lnv2` into an exponential function, the end result is a non-negative statistic, just as is required for the variance. The parameter `z` is parsed through the inverse Fisher z-transformation, thus yielding a correlation.[9]

## Ml Model

To apply a program to a data set, you need to issue the `ml model` command. This command also controls the method of maximization that is used, but it will be assumed here that this method is `lf`—i.e., the linear form restrictions hold and the derivatives are obtained numerically. The syntax for `ml model` is:

`ml model lf` *name equations* [`if`] , `technique(`*algorithm*`)`

[`options`] [`weight`]

Here `ml model` may be abbreviated to `ml mod`, *name* is the name of the MLE program (e.g., `poisson`), and *equations* specifies the model that should be estimated through the program. A subset of the data may be selected through the `if` statement. It is also possible to specify various options, to perform a weighted estimation, and to select an algorithm, as will be discussed below.

### Equations

To perform MLE, Stata needs to know the model that you want to estimate. That is, it needs to know the dependent and, if relevant, the predictor variables. These variables are declared by specifying one or more equations. The user

[9]The Fisher z-transformation of the correlation coefficient is as follows:

$$z = \tanh^{-1} \rho = .5 \ln \left( \frac{\rho + 1}{1 - \rho} \right)$$

where $-\infty < z < \infty$ and $\tanh^{-1}$ is the inverse hyperbolic tangent (sometimes also referred to as arctanh). To reverse the operation, one computes

$$\rho = \frac{e^{2z} - 1}{e^{2z} + 1}$$

This is the operation specified in the gen command for `r` in Example 3.

specifies these equationsin the `ml model` command, placing each equation in parentheses.

The general rule in Stata is that a separate equation is specified for each parameter declared in `args`. For example, if we wanted to estimate the parameters of the normal distribution, we would need an equation for the mean and an equation for the variance. In a linear regression model, we would need an equation for the conditional mean of $y$ (i.e., $E[y_i|\boldsymbol{x'}_i]$) and for the variance (the latter model would include only a constant, unless we specify a heteroskedastic regression model). In a simultaneous equations model, there would be as many equations as endogenous variables, plus additional equations to specify the covariance structure. For the Poisson distribution, the `ml model` command would look something like:

```
ml model lf poisson (y=)
```

Note that nothing is specified after the equality sign because there are no predictors.[10] For the normal distribution, it would look something like

```
ml model lf normal (y=) (y=)
```

When multiple parameters are estimated, as in a normal distribution, it is often useful to label the equations. These labels will then be repeated in the output and will make it easier to read it. For example, one could issue the following command:

```
ml model lf normal (mean:  y=) (var:  y=)
```

This syntax clarifies that the first estimate is that of the mean and the second estimate is that of the variance.[11]

## Additional Examples

The following are some additional examples for specifying the `ml model` command.

[10] A constant is included by default.

[11] It is assumed here that the parameters have been declared in this order in the `args` part of the syntax.

**Example 4(a)** The syntax for the normal distribution could also have been written as

```
ml model lf normal (y=) ()
```

That is, one does not need to refer to a dependent variable in the variance equation.

**Example 4(b)** Yet another way to estimate the parameters of the normal distribution is the following:

```
ml model lf normal (y=) /sigma
```

The instruction `/sigma` will cause Stata to output an estimate of $\sigma$.

**Example 5** Now imagine that the normal density describes the conditional distribution of `Y` given two predictors, `X` and `Z`. We assume that the conditional variance of `Y` is constant and given by $\sigma^2$. We also assume that the conditional mean of `Y` is given by $\beta_0 + \beta_1 X + \beta_2 Z$. In other words, we are estimating the classical linear regression model. The following command will yield estimates for this model:

```
ml model lf normal (Y=X Z) (Y=)
```

The first equation calls for the estimation of the conditional mean of `Y`, which is a function of the predictors `X` and `Z`. The second equation pertains to the estimation of $\sigma$, which is constant so that no predictors are specified.

Example 5 shows that estimation of the classical regression model requires no additional programming compared to estimating the mean and variance of a normal density. This minimizes the burden of programming and gives MLE routines a great deal of portability. This is an important benefit of Stata.

**Example 6** For the bivariate normal distribution, there are two dependent variables. Imagine a data set in which these dependent variables are labeled `left` and `right`, respectively. Then the parameters of the bivariate normal can be estimated in the following manner:

```
ml model lf bivnorm (left=) (right=) () () ()
```

Given the order of the arguments in the `bivnorm` program, the first expression in parentheses corresponds to the mean of `left`, the second expression to the mean of `right`, the third expression to the log-variance of `left`, the fourth expression to the log-variance of `right`, and the last expression to the Fisher z-transformation of the correlation between `left` and `right`

## Technique

As of version 8, Stata allows users to choose from a variety of algorithms. The default is the Newton-Raphson (`nr`) algorithm. One can, however, also choose one of the quasi-Newton algorithms, including `bhhh` for Bendt-Hall-Hall-Hausman, `dfp` for Davidson-Fletcher-Powell, or `bfgs` for Broyden-Fletcher-Goldfarb-Shanno.

## Options

There are three options that can be specified with `ml model`. Two of these produce robust variance estimates, also known as Huber-White or sandwich estimates (see Greene, 2008; White, 1980):

(1) `robust` generates heteroskedasticity-corrected standard errors. (This may be abbreviated as `rob`.)
(2) `cluster(`*varname*`)` generates cluster-corrected standard errors, where *varname* is the name of the clustering variable. (This may be abbreviated as `cl`.)

Note that users should be aware that these options result in pseudo maximum likelihood estimation.

A third option is `svy`, which is used with survey data. These should have been previously declared using the `svyset` command. Here, too, sandwich variance estimates are computed. As before, the estimation maximizes the log-pseudo-likelihood

### Weights

It is possible to add a variety of weights in `lf` mode. We maximize

$$\ell = \sum_i w_i \ln f(y_i|\boldsymbol{\theta})$$

where $w_i$ is the weight associated with the $i$th sample unit. Stata accepts a variety of weights, including the `fweight`, which is a frequency weight that can be used to replicate an observation. Another weight is the `aweight`, which is the analytical weight and gives greater weight to observations that are more precise (e.g., group means with smaller within-group variance). To accommodate sampling designs, the probability weight `pweight` is useful; this is based on sampling fractions. A last type of weight is the idiosyncratic weight `iweight`, which is useful mostly to accomplish certain programming steps. The syntax for each of these weights is identical:

```
[weight=wname]
```

Here *wname* is the name of the weight variable. The square brackets are obligatory.

## Ml Check

It is useful to check an MLE program for errors. In Stata, you can do this by issuing the command `ml check`. This command evaluates whether the program can compute the log-likelihood function and its first and second derivatives. If there is a problem with the log-likelihood function, or with its derivatives, `ml` `check` will let the user know. Stata will not be able to estimate the model before these problems are fixed.

## Ml Search

Numeric algorithms require an initial guess of the parameter estimates to begin the iterations. These initial guesses are the so-called *starting values*. In Stata, the user has two options: (1) use a default procedure for starting values or (2) do a more extensive search.

The default procedure in Stata is to set the initial values to 0. If the log-likelihood function cannot be evaluated for this choice of starting values, then Stata uses a pseudo-random number generator to obtain the starting values. (It will regenerate numbers until the log-likelihood function can be evaluated.) This procedure is a quick-and-dirty way to start the Newton-Raphson or any other algorithm.

Through `ml search` (which may be abbreviated as `ml sea`) the selection of starting values can be improved. The `ml search` command searches for starting values based on equations. A nice feature here is that the user can specify boundaries on the starting values. For example, before estimating the Poisson distribution, we could specify

```
ml search 1 3
```

This causes the search command to pick starting values for $\mu$ that lie between 1 and 3. If the ML estimate lies within these bounds, beginning the iterations there can speed up estimation considerably. This is not so important for the example of the Poisson distribution, which is quite simple and does not even require numerical methods, but can come in handy for complex estimation problems.

## Ml Init

An alternative way to specify starting values is through the `ml init` command. This command does not ask Stata to search for starting values in a specific range. Rather, it provides a set of fixed values that Stata should use in lieu of its defaults. The syntax can take on several forms, as the following example illustrates for the normal distribution.

**Example 7** Imagine we want to set the starting values for the normal distribution to $\mu = 0$ and $\sigma^2 = 1$. The following three specifications of the `ml init` command accomplish this task.

```
ml init 0 1, copy
ml init eq1:_cons=0 eq2:_cons=1
ml init /eq1=0 /eq2=1
```

If the equations are labeled, then one should use the labels in lieu of `eq1`, `eq2`, etc.

One of the very nice aspects of the `ml init` command is that it allows us to use the estimates from other estimation routines to generate starting values for maximum likelihood estimation. For example, one could use estimates from a linear probability model to "seed" a logit estimation routine. The following illustrates using OLS estimates as starting values for a normal linear regression model estimated using MLE.[12]

**Example 8** We estimate a regression model with `Y` being a function of `X` and `Z`. We first use OLS to obtain starting values. We then use these in maximum likelihood estimation.

```
regress Y X Z
mat b=e(b)
mat v=e(rmse)*e(rmse)
ml model lf normal (Y=X Z) (Y=)
ml init b v, copy
ml max
```

The second command line tells Stata to take the OLS estimates, which are stored in the internal object `e(b)`, and to place them

[12]Of course, the OLS estimates of the regression coefficients are perfect starting values, since the ML estimates are identical to them.

into the vector `b`. The third line takes the root mean squared error, which is the square root of an unbiased estimator of $\sigma^2$ and is stored in the internal object `e(rmse)`. It then squares this and stores it into the matrix `v`. The fifth line uses the matrices `b` and `v` as starting values.

## Ml Maximize

None of the commands discussed so far actually causes Stata to generate a table of parameter estimates. To do this, the MLE program has to be executed and this is done through the `ml maximize` command. You simply type

```
ml maximize [, options]
```

(which may be abbreviated to `ml max`) and the estimation process commences—at least, when the program is correct and there are no "funky" things in the data.

It is possible to add several options to the `ml maximize` command that control the output and convergence. In general, I advise against tinkering with these features, but for the sake of completeness I will list the most important options.

1. `nolog` suppresses the iteration log. This reduces the length of the output. Since the iteration log contains important information about convergence, one should not suppress it too quickly.

2. `iterate(#)` sets the maximum number of iterations (#). The default value is 16000. Since this default is very large—it could take hours to reach this limit—there is usually no reason to change it.

3. `ltolerance(#)` sets the tolerance, which controls the point at which Stata cuts off the iterations. Specifically, convergence is declared when

$$\left|\frac{\ell_j - \ell_{j-1}}{\ell_{j-1}}\right| \leq ltolerance$$

where $\ell_j$ is the log-likelihood function for the set of estimates generated in the $j$th iteration and $\ell_{j-1}$ is the log-likelihood function for the set of estimates in the previous iteration. The default of `ltolerance` is 1E-7. By increasing this value, convergence can be speeded. However, this is a risky approach because the resulting estimates may not be true maximum likelihood estimates. Hence, the decision to change `ltolerance` should not be made lightly.

4. `difficult` forces Stata to put in extra effort to estimate difficult log-likelihood functions and is especially useful in conjunction with the Newton-Raphson algorithm. Complex log-likelihoods tend to have many ridges, flat areas, and saddle points so that they are not concave. This makes it impossible to compute the direction vector, which is used to update estimates. The `difficult` option prompts Stata to determine if the direction vector exists. If not, then the program supplements the Newton-Raphson algorithm with a variation on the steepest ascent method to obtain new estimates (see Gould, Pitblado and Poi, 2010). Specifying `difficult` increases estimation time, so I do not suggest using it by default. However, if Stata generates many warnings about non-concavity, especially on later iterations, it may be worthwhile to repeat estimation using this option.

## Monitoring Convergence and MI Graph

Even when the program produces output, it is useful to check the convergence of the Newton-Raphson algorithm. One should pay attention to three pieces of information. First, the algorithm should converge relatively quickly. A dozen or so iterations would generally not worry us, especially not for complex estimation problems. But if the algorithm requires a large number of iterations (e.g., in the hundreds), then this could indicate a serious problem.

Second, one should pay attention to warning messages about the concavity of the log-likelihood function. The message `not concave`, which follows the value of the log-likelihood function in the iteration log, indicates that the log-likelihood function is essentially flat at a particular iteration. This means that Stata cannot establish a direction vector to guide new parameter estimates. if this warning message occurs early in the iteration process, it can be safely ignored. However, if this message appears on the last iteration, then this indicates a serious problem. One cannot trust the parameter estimates in this case and should consider re-estimating the model with the `difficult` option (see above). This option may also be useful when one receives the `backed up` error message. This is related to the step size in algorithms and appears when an increase in step size does not improve the log-likelihood. The `baked up` warning, too, signals problems with convergence and should concern us when it occurs on the final iteration.

Third, one should monitor the convergence path. If we place the iterations on the horizontal axis and the corresponding values of $\ell$ on the vertical axis, then we should ideally see a concave function. That is, initially the changes in the log-likelihood function should be large between iterations. However, nearing the end of the iterations, these changes should be relatively tiny. Small perturbations of this pattern need not worry us. However, if the iteration function is convex (or a straight line), then we should be worried. This could indicate a bug in the program or an ill-behaved log-likelihood function.

Plotting the log-likelihood against the iterations in Stata is easy. All one has to do is to issue the command

```
ml graph
```

(which may be abbreviated as `ml gr`) after running `ml maximize`. I recommend that you always create this plot, as it reveals a lot about convergence.

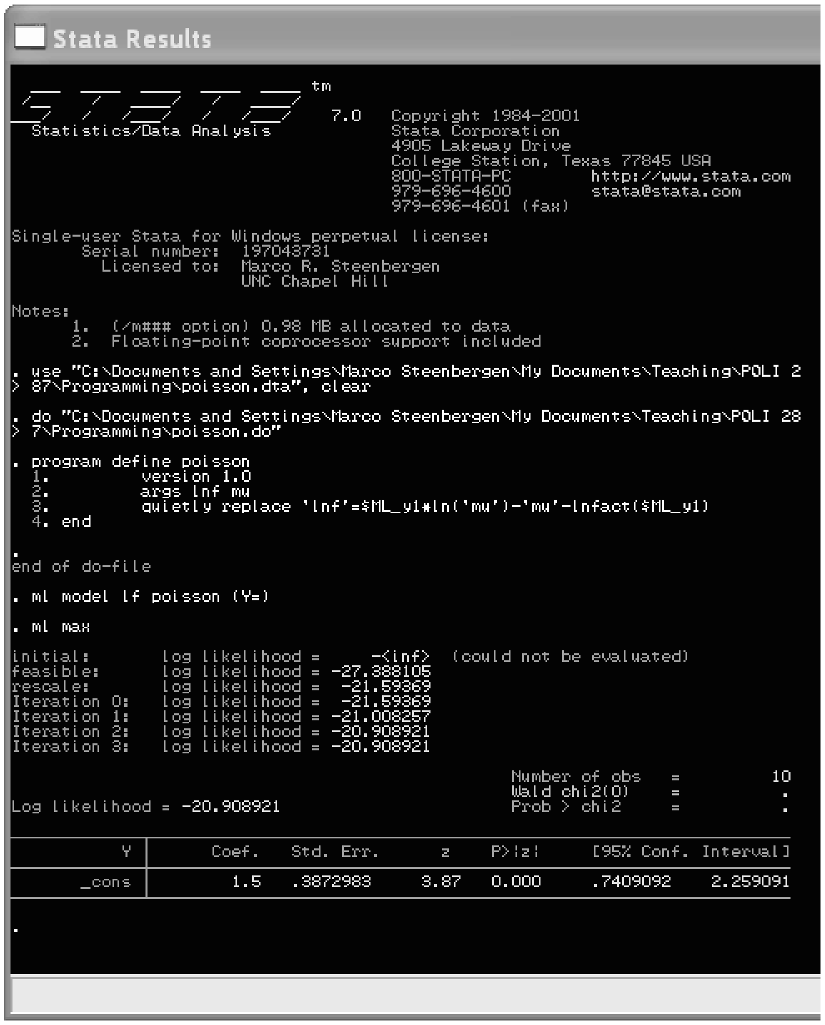
## Output

### Basic Output

After running a MLE program, Stata will produce the following output.

1. An iteration log, showing the iterations and the value of the log-likelihood at each iteration. (This log will not be shown if you specified `nolog` as an option for the `ml maximize` command.)

2. The final value of the log-likelihood function. (This is always shown, even if you have specified the `nolog` option.)



```
                                          tm
                                             7.0   Copyright 1984-2001
  Statistics/Data Analysis                         Stata Corporation
                                                   4905 Lakeway Drive
                                                   College Station, Texas 77845 USA
                                                   800-STATA-PC        http://www.stata.com
                                                   979-696-4600        stata@stata.com
                                                   979-696-4601 (fax)

Single-user Stata for Windows perpetual license:
       Serial number:  197043731
         Licensed to:  Marco R. Steenbergen
                       UNC Chapel Hill

Notes:
      1.  (/m### option) 0.98 MB allocated to data
      2.  Floating-point coprocessor support included

. use "C:\Documents and Settings\Marco Steenbergen\My Documents\Teaching\POLI 2
> 87\Programming\poisson.dta", clear

. do "C:\Documents and Settings\Marco Steenbergen\My Documents\Teaching\POLI 28
> 7\Programming\poisson.do"

. program define poisson
  1.          version 1.0
  2.          args lnf mu
  3.          quietly replace 'lnf'=$ML_y1*ln('mu')-'mu'-lnfact($ML_y1)
  4. end

.
end of do-file

. ml model lf poisson (Y=)

. ml max

initial:       log likelihood =     -<inf>  (could not be evaluated)
feasible:      log likelihood = -27.388105
rescale:       log likelihood =  -21.59369
Iteration 0:   log likelihood =  -21.59369
Iteration 1:   log likelihood = -21.008257
Iteration 2:   log likelihood = -20.908921
Iteration 3:   log likelihood = -20.908921

                                                  Number of obs   =         10
                                                  Wald chi2(0)    =          .
Log likelihood = -20.908921                       Prob > chi2     =          .

------------------------------------------------------------------------------
           Y |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       _cons |        1.5   .3872983     3.87   0.000     .7409092    2.259091
------------------------------------------------------------------------------

.
```

**Figure 1: Stata MLE Commands and Output**

3. The number of observations on which the estimation is based.

4. The Wald chi-square test and its (asymptotic) p-value.

5. For each equation, the parameter estimates, their estimated standard errors, the test statistics and their p-value, and the upper and lower bounds of the 95% confidence intervals. If the option robust, cluster, or svy was specified on the ml model command, then sandwiched standard errors are reported. The test statistic is the ratio of the estimate to its standard error.

Figure 1 shows the output generated by the "poisson" poisson program that we created earlier (in Example 1) as applied to some artificial data. Following the ml max command we first see the iteration history (or iteration log). It took the program 3 iterations to find the ML estimate for $\mu$. Notice that the initial iteration produced an error message because Stata started by setting $\hat{\mu} = 0$ and for this value $\ln(\mu)$, which is part of the log-likelihood function, is not defined. On the final iteration, the log-likelihood function was -20.908921; this value is repeated underneath the iteration log. To the right of the (final) log-likelihood we find the number of observations and the Wald chi-squared and its associated $p$-value (see below). The bottom portion of the output shows the estimate (1.5), the estimated standard error (.3872988), the $z$-test statistic (3.87) and its associated $p$-value (0.000), and the lower and upper bounds of the 95% confidence interval (.7409092 and 2.259091, respectively).

The optional ml graph command produces the output in Figure 2 (Stata will show this output in a separate window). The horizontal axis of this graph shows the iteration number and the vertical axis, labeled ln L0 gives the value of the log-likelihood function at that iteration. The iteration path shown in Figure 2 is precisely what we would like to see: as the iterations progress, changes in the log-likelihood function become ever smaller. (In fact, this example shows no change from the 2nd to 3rd iteration because there is a closed form solution for the ML estimator.)

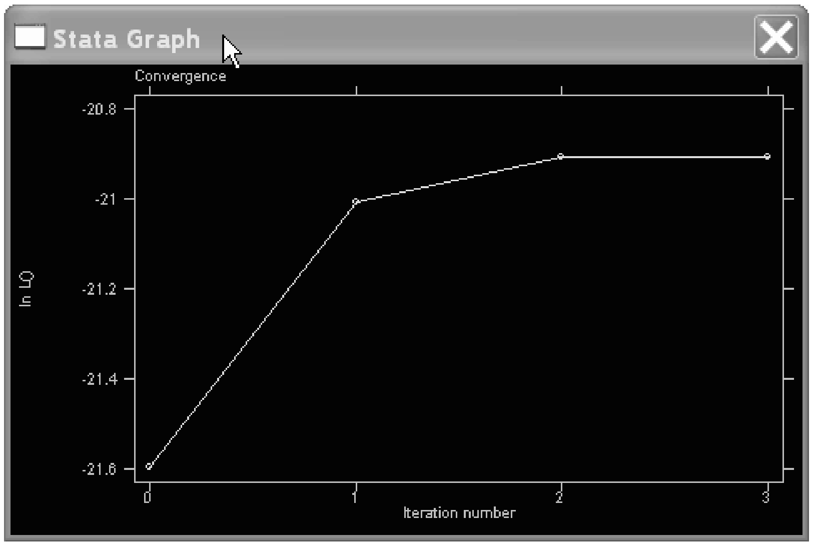


**Figure 2: Graph Produced by** `ml graph`

## Augmented Output

The `ml display` command also produces screen output but has an interesting option to display reparameterizations of results. This comes in handy when the estimated parameters are actually transformations of the parameters of interest.

Consider, for example, the bivariate normal distribution from examples 3 and 6. If we run `ml max`, we will obtain estimates of $\mu_1$ and $\mu_2$, as desired. However, we will not obtain estimates of $\sigma_1^2$, $\sigma_2^2$, and $\rho$ since these are not the parameters that were specified in the `args` of the `bivnorm` program. Instead, we will obtain estimates of $\ln \sigma_1^2$, $\ln \sigma_2^2$, and the Fisher z-transformation of $\rho$ because these were specified in the `args` portion of the `bivnorm` program. What to do if we want to actually obtain the estimates of $\sigma_1^2$, $\sigma_2^2$, and $\rho$? One option is to use the `diparm` option in `ml display`.[13] For a number of standard transformations, this shows the estimates in the metric of interest. Some of these standard transformations include: `exp` if the estimated quantity is the natural logarithm of the quantity of interest and `tanh` if the estimated quantity

[13]In recent versions of Stata, `diparm` can also be specified as part of the `ml model` command. See Gould, Pitblado and Poi (2010) for details.

is the Fisher z-transformation.[14] The following example shows the use of the `diparm` option with the bivariate normal distribution.

**Example 9** To display estimates of $\sigma_1^2$, $\sigma_2^2$, and $\rho$ in lieu of $\ln \sigma_1^2$, $\ln \sigma_2^2$, and the Fisher z-transformation of $\rho$, the following command may be used:

```
ml display, diparm(eq3,exp prob) diparm(eq4,exp
prob) diparm(eq5,tanh prob)
```

This causes Stata to output estimates of $\sigma_1^2$, $\sigma_2^2$, and $\rho$ at the bottom of the estimation results. Stata also shows the standard errors and 95 percent confidence interval of these estimates and, if desired, the Wald test statistic and its associated $p$-value (this is what the `prob` option requests).

## Constrained Optimization

Stata allows maximum likelihood estimation of models with linear constraints. Constraints are specified as an option with the `ml model` command. More specifically, the `constraint` option invokes a constraint that was defined previously. For example, consider the linear regression model $y_i = \beta_0 + \beta_1 x_i + \beta_2 z_i + \epsilon_i$ with $\epsilon_i \sim N(0, \sigma^2)$. Now imagine that we have a priori reason to believe that $\beta_2 = -\beta_1$ (or put differently, $\beta_1 + \beta_2 = 0$). We could build this constraint into the estimation by issuing the following commands:

```
constraint 1 Z=-X
ml model lf normal (Y=X Z) (Y=), constraint(1)
ml max
```

The program will now find estimates that satisfy the constraint, aliased as the number 1. (Whether these estimates are true ML estimates depends, of course, on the validity of the constraint.)

[14]For a complete listing of transformations type `help _diparm` in the Stata command window.

As a second example, consider the normal distribution with the restriction that the variance is equal to 1. This can be estimated using the following syntax:

```
constraint 1 [eq2]_cons=1
ml model lf normal (Y=) (Y=), constraint(1)
ml max
```

Here the constraint is defined in terms of the constant of the second equation, which corresponds to the variance.

## Hypothesis Testing

### Wald Test

When using the `ml maximize` command, Stata by default reports a Wald test statistic and its p-value. This statistic compares the fit of a model including predictors to the fit of a model excluding *all* of those predictors. The first model is the unconstrained model, while the second model is the constrained model.[15]

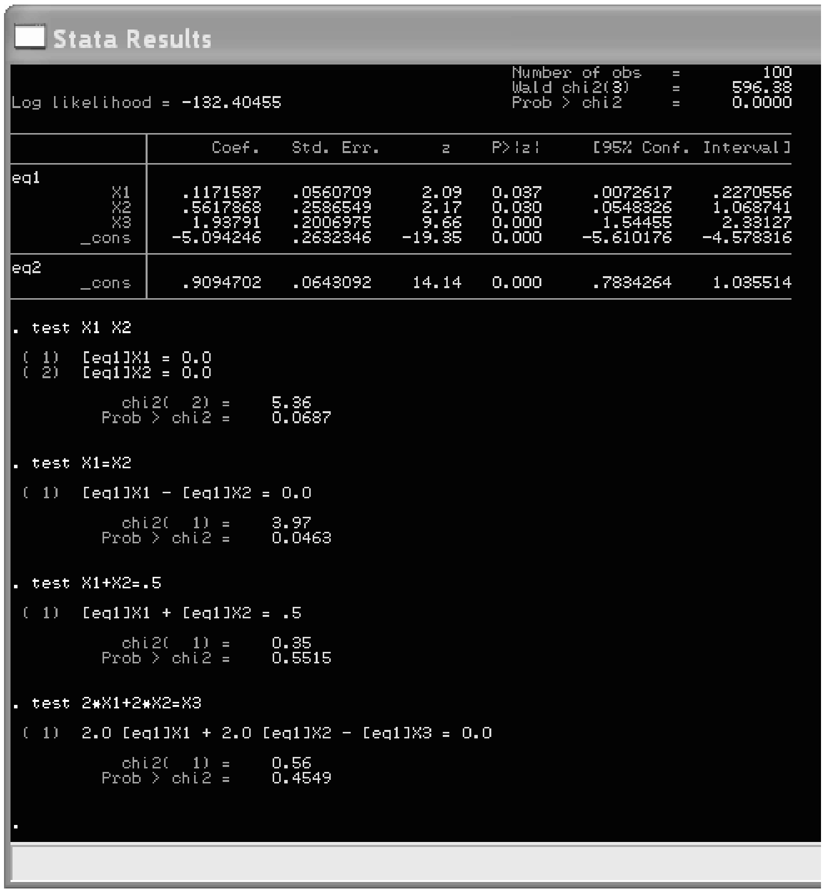
Sometimes we may want to exclude only a subset of the predictors. In this case, we can use the `test` command to obtain the relevant Wald test statistic. The test command is very easy to use. After running `ml max`, you simply type in `test` followed by the variables you want to exclude from the model. Stata will then show the relevant Wald test statistic and its $p$-value.

The `test` command can be used to test any hypothesis that involves a linear combination of the parameters. Figure 3 shows a number of examples pertaining to the classical linear regression model with normally distributed errors (see Example 5).

### Likelihood Ratio Test

The Wald test does not actually estimate the constrained model, but evaluates its fit based on the difference between the parameter estimate and its constrained value, as well as the curvature of the log-likelihood function (as measured by the

[15]When the model contains no predictors, Stata reports a period for the Wald test statistic and its p-value.



```
                                                  Number of obs   =        100
                                                  Wald chi2(3)    =     596.38
Log likelihood = -132.40455                       Prob > chi2     =     0.0000

------------------------------------------------------------------------------
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
eq1          |
          X1 |   .1171587   .0560709     2.09   0.037     .0072617    .2270556
          X2 |   .5617868   .2586549     2.17   0.030     .0548326    1.068741
          X3 |    1.93791   .2006975     9.66   0.000      1.54455     2.33127
       _cons |  -5.094246   .2632346   -19.35   0.000    -5.610176   -4.578316
-------------+----------------------------------------------------------------
eq2          |
       _cons |   .9094702   .0643092    14.14   0.000     .7834264    1.035514
------------------------------------------------------------------------------

. test X1 X2

 ( 1)  [eq1]X1 = 0.0
 ( 2)  [eq1]X2 = 0.0

           chi2(  2) =    5.36
         Prob > chi2 =    0.0687

. test X1=X2

 ( 1)  [eq1]X1 - [eq1]X2 = 0.0

           chi2(  1) =    3.97
         Prob > chi2 =    0.0463

. test X1+X2=.5

 ( 1)  [eq1]X1 + [eq1]X2 = .5

           chi2(  1) =    0.35
         Prob > chi2 =    0.5515

. test 2*X1+2*X2=X3

 ( 1)  2.0 [eq1]X1 + 2.0 [eq1]X2 - [eq1]X3 = 0.0

           chi2(  1) =    0.56
         Prob > chi2 =    0.4549

.
```

**Figure 3: Wald Test Examples**

second derivative). A more precise (but asymptotically equivalent) approach to testing is to explicitly estimate the constrained model and to perform a likelihood ratio test. The Stata command for doing this is `lrtest`. This test compares the values of the log-likelihood functions for the constrained and unconstrained models and computes the p-value of the resulting likelihood ratio test statistic.

To use `lrtest`, a specific sequence of estimation commands is typically followed.

(1) Estimate the unconstrained model.
(2) Save the statistics associated with the unconstrained model:

`lrtest, saving` *name*

where *name* is an arbitrary name of no more than 4 characters.
(3) Estimate the constrained model.
(4) Type `lrtest` to perform the likelihood ratio test.

For example, consider the regression model shown in Figure 3. The following sequence of commands allows one to perform the likelihood ratio test for the null hypothesis that $\beta_1 = \beta_2 = 0$.

```
ml model lf normal (Y=X1 X2 X3) (Y=)

ml max

lrtest, saving(0)

ml model lf normal (Y=X3) (Y=)

ml max

lrtest
```

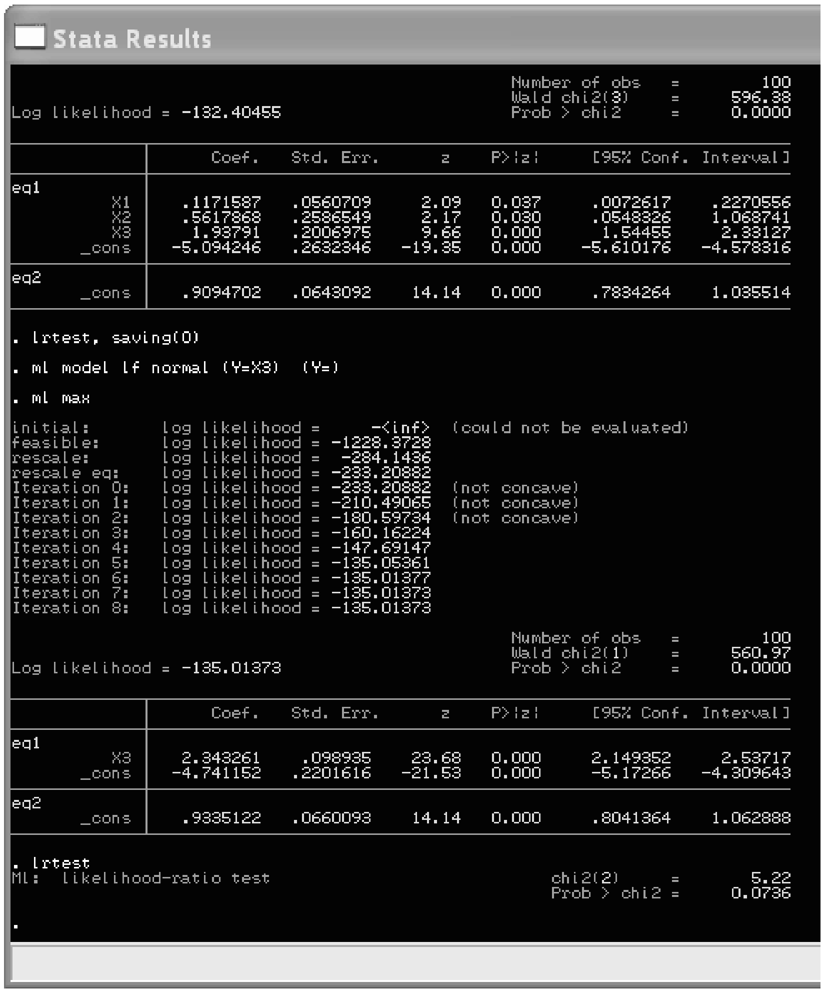
The results are shown in Figure 4.

The same results could also have been obtained using the following command sequence:

```
ml model lf normal (Y=X1 X2 X3) (Y=)

ml max

est sto un
```



```
Stata Results

                                                  Number of obs   =        100
                                                  Wald chi2(3)    =     596.38
Log likelihood = -132.40455                       Prob > chi2     =     0.0000

------------------------------------------------------------------------------
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
eq1          |
          X1 |   .1171587   .0560709     2.09   0.037     .0072617    .2270556
          X2 |   .5617868   .2586549     2.17   0.030     .0548326    1.068741
          X3 |    1.93791   .2006975     9.66   0.000      1.54455     2.33127
       _cons |  -5.094246   .2632346   -19.35   0.000    -5.610176   -4.578316
-------------+----------------------------------------------------------------
eq2          |
       _cons |   .9094702   .0643092    14.14   0.000     .7834264    1.035514
------------------------------------------------------------------------------

. lrtest, saving(0)

. ml model lf normal (Y=X3)  (Y=)

. ml max

initial:       log likelihood =     -<inf>  (could not be evaluated)
feasible:      log likelihood = -1228.3728
rescale:       log likelihood =  -284.1436
rescale eq:    log likelihood = -233.20882
Iteration 0:   log likelihood = -233.20882  (not concave)
Iteration 1:   log likelihood = -210.49065  (not concave)
Iteration 2:   log likelihood = -180.59734  (not concave)
Iteration 3:   log likelihood = -160.16224
Iteration 4:   log likelihood = -147.69147
Iteration 5:   log likelihood = -135.05361
Iteration 6:   log likelihood = -135.01377
Iteration 7:   log likelihood = -135.01373
Iteration 8:   log likelihood = -135.01373

                                                  Number of obs   =        100
                                                  Wald chi2(1)    =     560.97
Log likelihood = -135.01373                       Prob > chi2     =     0.0000

------------------------------------------------------------------------------
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
eq1          |
          X3 |   2.343261    .098935    23.68   0.000     2.149352     2.53717
       _cons |  -4.741152   .2201616   -21.53   0.000     -5.17266   -4.309643
-------------+----------------------------------------------------------------
eq2          |
       _cons |   .9335122   .0660093    14.14   0.000     .8041364    1.062888
------------------------------------------------------------------------------

. lrtest
Ml:  likelihood-ratio test                            chi2(2)     =       5.22
                                                      Prob > chi2 =     0.0736

.
```

**Figure 4: Likelihood Ratio Test Examples**

```
ml model lf normal (Y=X3) (Y=)

ml max

est sto re

lrtest un re
```

Here, `est sto` causes Stata to store estimation results for later use. The assumption is that the model stored in `re` is a subset of the model stored in `un`.

## General Programming Commands

MLE is a part of Stata's programming language. The following general programming commands will prove quite useful.

1. `program dir`: This command shows a list of programs (including `ado` files) that are currently in memory. If a program is already in memory, you cannot define it again.

2. `program drop`: This command drops a program from memory. Once this command is issued, Stata can no longer execute the affected program(s).

## References

Gould, William, Jeffrey S. Pitblado and Brian Poi. 2010. *Maximum Likelihood Estimation with Stata*. 4th ed. College Station, TX: Stata Press.

Greene, William H. 2008. *Econometric Analysis*. 6th ed. Upper Saddle River, N.J.: Prentice Hall.

White, Halbert. 1980. "A Heteroskedasticity-Consistent Covariance Matrix Estimator and a Direct Test for Heteroskedasticity." *Econometrica* 48(4):817–838.