# Clarify: Software for Interpreting and Presenting Statistical Results

Michael Tomz Jason Wittenberg Gary King[1]

June 1, 2001

[1]Tomz: Department of Political Science, Stanford University, Encina Hall, Stanford, CA 94305-6044, email tomz@stanford.edu; Wittenberg: Department of Political Science, University of Wisconsin, Madison, 1050 Bascom Mall, 221 North Hall, Madison, WI, 53706, email witty@polisci.wisc.edu; King: Center for Basic Research in the Social Sciences, 34 Kirkland Street, Harvard University, Cambridge MA 02138, email King@Harvard.Edu, website http://GKing.Harvard.Edu. 

# Contents

# 1 Introduction

Clarify is a program that uses Monte Carlo simulation to convert the raw output of statistical procedures into results that are of direct interest to researchers, without changing statistical assumptions or requiring new statistical models. The program, designed for use with the Stata statistics package, offers a convenient way to implement the techniques described in:

> Gary King, Michael Tomz, and Jason Wittenberg (2000). "Making the Most of Statistical Analyses: Improving Interpretation and Presentation." *American Journal of Political Science* 44, no. 2 (April 2000): 347−61.

We recommend that you read this paper before using the software.

Clarify 2.0 simulates quantities of interest for the most commonly used statistical models, including linear regression, binary logit, binary probit, ordered logit, ordered probit, multinomial logit, Poisson regression, negative binomial regression, weibull regression, seemingly unrelated regression equations, and the additive logistic normal model for compositional data.

# 2 Software requirements

Clarify works in conjunction with Stata Statistical Software, produced by Stata Corporation. Clarify will run on any platform (Windows, Unix, or Macintosh) where Stata is already installed. You must have Stata 6.0 or later to run Clarify. To obtain a copy of Stata or learn more about the software, visit `http://www.stata.com` or send email to stata@stata.com.

# 3 How to Install Clarify For the First Time

If you do not have a previous version of Clarify on your personal computer or your network, there are two ways to install the software.

## 3.1 Installing on Computers that are Connected to the Internet

To install Clarify 2.0 for use with a personal copy of Stata, launch Stata and then type:

```
net from http://gking.harvard.edu/clarify/
net install clarify
```

To install Clarify for use with a networked copy of Stata, launch Stata and then type:

```
net from http://gking.harvard.edu/clarify/
net set ado SITE
net install clarify
```

In either case, the following files will be installed on your computer: estsimp.ado, estsimp.hlp, setx.ado, setx.hlp, simqi.ado, simqi.hlp, sumqi.ado, sumqi.hlp, tlogit.ado, tlogit.hlp. Note that these files will be installed onto your adopath, the path where Stata searches for the files it needs. If you ever want to remove these files, simply type `ado uninstall clarify`

### 3.2 Installing on Computers that are Not Connected to the Internet

Download `clarify.zip` from `http://gking.harvard.edu` and copy that file to a floppy disk. Then insert the floppy into the disk drive of the machine that is not connected to the internet. Copy `clarify.zip` into a temporary directory or folder on the hard disk, and use a utility such as `pkunzip` (available for the PC at `http://www.pkware.com`) or StuffIt (available for the MacIntosh at `http://www.aladdinsys.com/expander/index.html`) to extract the files into your temporary directory. Finally, launch Stata and type the following commands from within Stata:

```
net from <temporary path designator>
net install clarify
```

Here, replace <temporary path designator> with the path to the temporary directory or folder where you extracted the contents of `clarify.zip`. Thus, for example, if the archive were copied and opened into the c:\temp folder on a Windows machine, the appropriate installation commands would be:

```
net from c:\temp
net install clarify
```

On a Macintosh, if you copied `clarify.zip` into a temporary folder called TEMPFOLDER, you could install the program by typing:

```
net from :TEMPFOLDER
net install clarify
```

# 4 How to update Clarify

There are two ways to update Clarify. The choice depends on which version you currently have on your computer or network.

## 4.1 Updating from Version 1.2x or earlier

If you are using a version of Clarify released before June 2001, manually delete the following files from your Stata adopath, working directory, or network: estsimp.ado, estsimp.hlp, setx.ado, setx.hlp, simqi.ado, simqi.hlp, sumqi.ado, sumqi.hlp. Once you have done this, you may install Clarify according to the directions in Section 3.

## 4.2 Updating from Version 2.0 or later

Once you've installed Version 2.0 on your computer, it should be easy to update the program as new releases become available. If you have a personal copy of Stata, launch Stata and then type:

```
net from http://gking.harvard.edu/clarify/
net install clarify, replace
```

To update the copy on a network, launch Stata and then type:

```
net from http://gking.harvard.edu/clarify/
net set ado SITE
net install clarify, replace
```

# 5 What Clarify Does

Clarify uses stochastic simulation techniques to help researchers interpret and present their statistical results. It uses whatever statistical model you have chosen and as such changes no statistical assumptions. As a first step, the program draws simulations of the main and ancillary parameters ($\tilde{\gamma}$) from their asymptotic sampling distribution, in most cases a multivariate normal with mean equal to the vector of parameter estimates ($\hat{\gamma}$) and variance equal to the variance-covariance matrix of estimates $\hat{\mathrm{V}}(\hat{\gamma})$.[1] Thus,

$$\tilde{\gamma} \sim \mathrm{N}\left(\hat{\gamma}, \hat{\mathrm{V}}(\hat{\gamma})\right)$$

[1] There are two exceptions, the variance parameters for `sureg`, drawn from an inverse Wishart distribution; and the variance for `regress`, drawn from an inverse $\chi^2$ distribution. For details, see section 10.

By default the program draws $M = 1000$ sets of simulated parameters, which should be sufficient for most applications.

Next, Clarify converts the simulated parameters into substantively interesting quantities, such as predicted values, expected values, or first differences. To achieve this objective, the user need only choose real or hypothetical values for the explanatory variables (the $X$'s) and indicate which quantities should be calculated, conditional on those $X$'s. The program allows researchers to calculate virtually any quantity that would shed light on a particular problem, and provides a number of Stata procedures to do this easily.

Clarify 2.0 simulates quantities of interest for the most commonly used statistical models, including linear regression, binary logit, binary probit, ordered logit, ordered probit, multinomial logit, Poisson regression, negative binomial regression, weibull regression, seemingly unrelated regression equations, and compositional data.

# 6 What's new in Clarify 2.0?

Clarify 2.0 includes a number of enhancements over previous versions, including:

- Support for more models, including weibull regression, seemingly unrelated regression equations, and the additive logistic normal model for compositional data.
- The ability to apply standard transformations – such as natural logs and exponents – to dependent variables, estimate a model, and then reverse those transformations when interpreting the results.
- Clarify is Amelia-compatible: if you use multiple imputation to correct for problems with missing values, Clarify will analyze all the multiply imputed data sets and appropriately combine the results and compute your quantity of interest automatically. (For information on the software program Amelia, see `http://GKing.Harvard.Edu`).
- The option to generate antithetical simulations, which guarantees that the mean of the simulated parameters equals the vector of point estimates, $\hat{\gamma}$, and reduces Monte Carlo variance.
- More powerful commands for setting the values of the explanatory variables (the $X$'s), using either single or multiply-imputed datasets to compute descriptive statistics.
- The ability to re-display the previous point estimates by entering the `estsimp` command without any arguments.
- The option in the `estsimp` command to drop previously simulated parameters.

For automatic notifications (via email) of updates to Clarify, see the webpage `http://GKing.Harvard.Edu/netmind.shtml`.

# 7 A Simple Example

Clarify is based on three simple commands:

| | |
|---|---|
| `estsimp` | (estimates the model and simulates its parameters) |
| `setx` | (sets values of Xs before simulating quantities of interest) |
| `simqi` | (simulates quantities of interest) |

In general, the commands should be run in that order, although it is often useful to call `setx` and `simqi` many times after running `estsimp`. Clarify also contains two minor commands: `sumqi` and `tlogit`. `sumqi` assists in summarizing quantities of interest that have been saved to the dataset. `tlogit` applies the logistic transformation to one or more variables. For instructions on using these commands, consult the on-line help file. All Clarify commands can be run interactively at the Stata command line or in batch mode by using a Stata do-file.

Each command contains many options, which we describe in Section 8 of this manual. Here, we offer a simple example designed to show how Clarify can be used. With a dataset loaded into memory, type the following 3 commands:

```
estsimp logit Y X1 X2    /* Estimate a logit and simulate its parameters */
setx mean                /* Set X's to their means.  */
simqi                    /* Report Pr(Y=1) conditional on the X's */
```

Each of these commands performs a particular operation, which we summarize here. A more detailed discussion appears in Section 8.

**estsimp logit Y X1 X2** Clarify works by capturing and interpreting the statistical results that Stata produces when estimating a particular model. To use Clarify, insert the word `estsimp` at the beginning of an estimation command that you would normally run in Stata. In this example, the built-in Stata command is `logit`, the binary dependent variable is $Y$, and the explanatory variables are $X1$, $X2$, and a constant. Unless the user specifies otherwise, `estsimp` will save the simulated parameters as new variables bearing the names $b1$, $b2$, and $b3$, which will hold simulations of the coefficients on $X1$, $X2$, as well as the constant term. If any of the variables $b1$ through $b3$ already exist in the dataset, Clarify will ask the user to delete those variables or choose different names for the simulations.

**setx mean** The `setx` command allows the user to choose a real or hypothetical value for each explanatory variable before computing quantities of interest. In this example, the command `setx mean` sets each $X$ equal to its average value.

**simqi** The `simqi` command computes and reports quantities of interest and associated measures of uncertainty. Used without specifying any options (many options are possible – see Section

8.3), `simqi` will compute intelligent default quantities that are appropriate to the model being estimated. In the case of logit, `simqi` will report the probability that $Y = 1$.

# 8 The Main Commands

This section provides more detailed information about each of the main commands in Clarify: `estsimp`, `setx`, and `simqi`. Much of the information also appears in on-line help files, which can be viewed by searching the Stata help menu or entering `help` *command name* at the Stata prompt.

## 8.1 estsimp

**Format:**

```
estsimp modelname depvar [indepvars] [weight] [if exp] [in range]
    [, sims(m) genname(newvar) antisim mi(file1 file2 ... filek) iout dropsims]
```

**Description:**

`estsimp` estimates a variety of statistical models and generates $M$ simulations of each parameter. Currently supported models include regress, logit, probit, ologit, oprobit, mlogit, poisson, nbreg. weibull, sureg, and the additive normal model for compositional data. The simulations are stored in new variables bearing the names $newvar1, newvar2, \ldots, newvark$, where $k$ is the number of parameters. Each variable has $M$ observations corresponding to the $M$ simulations. `estsimp` labels the simulated variables and lists their names on the screen, so you can verify what was simulated. The estsimp command accepts nearly all options that are typically available for the supported models. It also accepts several special options that are described below.

**Options:**

**sims(M)** specifies the number of simulations, $M$, which must be a positive integer. The default is 1000 simulations. If you choose a large number of simulations, you may need to allocate more memory to Stata. See [R] memory in the Stata reference manual for more details about memory allocation.

**genname(newvar)** specifies a stub-name for the newly generated variables. If no stub is given, Stata will generate the variables $b1, b2, \ldots, bk$, otherwise it will generate $newvar1$, $newvar2$, ... , $newvark$, provided that the variables do not exist in memory already.

**antisim** instructs `estsimp` to use antithetical simulations, in which numbers are drawn in pairs from the uniform[0,1] distribution, with the second draw being the complement of the first.

The antithetical draws are then used to obtain simulations from a multivariate normal distribution. This procedure ensures that the mean of the simulations for a particular parameter is equal to the point estimate of that parameter.

**mi(filelist)** allows `estsimp` to analyze multiply-imputed datasets: files in which missing values have been multiply imputed, such as created by Amelia. Enter the name for each imputed dataset you want to use, such as `mi(file1 file2 file3)`. Alternatively, you can enter a common stub name for all imputed datasets, such as `mi(file)`. In this case, `estsimp` assumes that you want to use all files in the working directory that are part of the uninterrupted sequence file1, file2, file3... `estsimp` will estimate the parameters for each dataset and use the estimates to generate simulations, which will reflect not only estimation uncertainty but also the uncertainty arising from the imputation process. Note: if the data in memory have been changed, you cannot specify the `mi()` option until you clear the memory or save the altered dataset.

**iout** instructs `estsimp` to print intermediate output (a table of parameter estimates) for each imputed dataset that it analyzes. By default, `estsimp` suppresses the intermediate output and displays only the final estimates produced by combining the results from each imputed dataset.

**dropsims** drops the simulated parameters from the *previous* call to `estsimp`.

**Examples:**

To estimate a linear regression of $y$ on $x1$, $x2$, $x3$, and a constant term; simulate 1000 sets of parameter estimates; and then save the simulations as $b1$, $b2$, ..., $bk$, type:

```
. estsimp regress y x1 x2 x3
```

In this example, Stata will create five new variables. The variables $b1$, $b2$ and $b3$ will contain simulated coefficients for $x1$, $x2$ and $x3$; $b4$ will hold simulations of the constant term; and $b5$ will contain simulated values for sigma squared, the mean squared error of the regression.

To simulate 500 sets of parameters from a logit regression and save the results as variables beginning with the letter "s", type:

```
. estsimp logit y x1 x2 x3, sims(500) genname(s)
```

Since the logit model contains no ancillary parameters, this command will generate four new variables: $s1$, $s2$, $s3$, and $s4$. Variables $s1$-$s3$ are simulated coefficients for $x1$, $x2$ and $x3$, and the final variable, $s4$, is the simulated constant term.

To simulate 1000 sets of parameters from an ordered probit regression in which the dependent variable can assume three values (low, medium, and high), type:

```
. estsimp oprobit y x1 x2 x3
```

The ordered probit model does not contain a constant term, but it does have ancillary parameters called cut-points. Thus, the estsimp command listed above will generate five new variables. The variables $b1$, $b2$ and $b3$ will hold simulated coefficients for $x1$ $x2$ and $x3$. Variables $b4$ and $b5$ will contain simulations for the two cutpoints (cut1 and cut2).

To obtain antithetical variates, simply use the antisim option, as in

```
. estsimp oprobit y x1 x2 x3, antisim
```

Suppose that we have three imputed datasets, called imp1.dta, imp2.dta, and imp3.dta. We could analyze all three datasets and combine the results by issuing the following command:

```
. estsimp oprobit y x1 x2 x3, mi(imp1 imp2 imp3)
```

The resulting simulations of the main and ancillary parameters would reflect both estimation uncertainty and the variability associated with the multiple imputations.

To view the intermediate output from each ordered probit estimation, add the iout option to the previous command, as in

```
. estsimp oprobit y x1 x2 x3, mi(imp1 imp2 imp3) iout
```

## 8.2 setx

**Format:**

```
setx

setx function [weight] [if exp] [in range] [, noinher nocwdel]

setx varname1 function1 varname2 function2 ...
              [weight] [if exp] [in range] [, noinher nocwdel]

setx (varname1 varname2) function1 (varname3 varname4) function2 ...
              [weight] [if exp] [in range] [, noinher nocwdel]

where
```

`function = mean|median|min|max|p#|math|#|'macro'|varname[#]`

**Description:**

After simulating parameters from the last estimation (see Section 8.1), use `setx` to set values for the explanatory variables (the $X$'s), change values that have already been set, or list the values that have been chosen. The main value types are

| | |
|---|---|
| `mean` | arithmetic mean |
| `median` | median |
| `min` | minimum |
| `max` | maximum |
| `p#` | #th percentile |
| `math` | a mathematical expression, such as 5*5 or sqrt(23) |
| `#` | a numeric value, such as 5 |
| `'macro'` | the contents of a local macro |
| `[#]` | the value in the #th observation of the dataset |

If you used multiply imputed datasets at the estimation stage, `setx` will use those same imputed datasets to calculate values for the explanatory variables. For instance, `setx x1 mean` would calculate the mean of x1 across all the imputed datasets.

When using `setx` or any other Stata command to calculate summary statistics such as means, medians, minimums, maximums, and percentiles, it is important to define the sample. At the estimation stage, Stata automatically disregards observations that do not satisfy the "if", "in", and "weight" conditions specified by the user. It also ignores observations with missing values on one or more variables. Before setting a particular variable equal to its mean or any other summary statistic, users must decide whether to calculate the statistic based only on observations that were used during the estimation stage, or to include other observations in the calculation.

By default, `setx` inherits the if-in-weight conditions from estsimp and disregards (casewise-deletes) any observation with missing values on the dependent or explanatory variables. You can specify different if-in-weight conditions by including them in the `setx` command line, and you can disregard all inherited conditions by using the `noinher` and `nocwdel` options described below.

The `setx` command is also used by another statistical package called `relogit`, which is also available from `http://gking.harvard.edu`. If you are running `relogit` with the `wc()` or `pc()` options, indicating that the data were selected on the dependent variable, `setx` will correct the selection bias when calculating summary statistics. For this reason, means and percentiles produced by `setx` may differ from means and percentiles of the (biased) sample. When the proportion of 1's in the population is known only to fall within a range, such as pc(.2 .3), `setx` will calculate bounds on

the values of the explanatory variables. The result will be two $X$-vectors, the first assuming that the true proportion of 1's is at its lower bound, and the second conditional on the true proportion being at its upper bound. The program will pass these vectors to `relogitq` and use them to calculate bounds on quantities of interest. To set each explanatory variable at a single value that falls midway between its upper and lower bounds, use the `nobound` option that is described below.

`setx` relies upon three globals: the matrix mrt_xc and the macros mrt_vt and mrt_seto. If you change the values of these globals, the program may not work properly.

`setx` accepts aweights and fweights. It also accepts the special options listed below.

Notes: (1) Setx will not accept spaces in mathematical expressions unless you enclose the expression in parentheses. `setx x4 ln(20)` is a valid command, but `setx x4 ln( 20 )` is a syntax error. Similarly, `setx x4 5*5` and `setx x4 (5 * 5)` are valid, but `setx x4 5 * 5` is not. (2) You may use square brackets (`[]`) when referring to observation numbers, e.g. `setx x [15]`, but do not use square brackets in mathematical expressions, or you may get unexpected results. We recommend that users check the values they have set with `setx` by entering `setx` without any arguments. (3) `setx` relies on three globals: the matrix `mrt_xc` and the macros `mrt_vt` and `mrt_seto`. If you change the values of these globals, the program may not work properly.

**Options:**

**noinher** causes `setx` to ignore all if-in-weight conditions that are inherited from `estsimp`. The user can specify new if-in-weight conditions by typing them as part of the `setx` command.

**nocwdel** forces `setx` to calculate summary statistics based on all valid observations for a given variable, even if the observations contain missing values for the other variables. If nocwdel is not specified, `setx` will casewise-delete observations with missing values.

**nobound** This option is available only after `relogit`, and only when the true proportion of 1's is assumed to fall within a specified range. Suppose the user typed pc(.2 .4) with `relogit` and then entered `setx x1 mean`. By default, `setx` would set the variable x1 equal to two values: the mean of x1, assuming that the true proportion of ones is only 0.2, and the mean of x1, allowing that the true proportion is as high as 0.4. Both values for x1 will be passed to `relogitq` and used to calculate bounds on quantities of interest. The `nobound` option overrides this procedure by setting each x to a single value: the midpoint of its upper and lower bound. Thus, the command `setx x1 mean`, `nobound` would set x1 equal to the following expression: $[(mean(x1)|\tau = 0.2) + (mean(x1)|\tau = 0.4)]/2$, where $\tau$ represents the presumed proportion of 1's in the population.

**keepmrt** is a programmer's option that instructs `setx` to return the matrix r(mrt_xc) without changing the globals mrt_xc, mrt_vt, and mrt_seto. If you don't understand what this means you should not use this option.

**Examples:**

To list values that have already been set:

```
. setx
```

To set each explanatory variable at its mean:

```
. setx mean
```

To set each explanatory variable at its median, based on a sample in which $x3 > 12$ and all inherited conditions are ignored and casewise deletion is suppressed:

```
. setx median if x3>12, noinh nocw
```

To set each explanatory variable to the value contained in the 15th observation of the dataset

```
. setx [15]
```

The command can also set each variable separately. For instance, to set x1 at its mean, x2 at its median, x3 at its minimum, x4 at its maximum, x5 at its 25th percentile, x6 at ln(20), x7 at 2.5, and x8 equal to a local macro called myval, type:

```
. setx x1 mean x2 median x3 min x4 max x5 p25 x6 ln(20) x7 2.5 x8 ‘myval’
```

To change the value of x3 from its previously chosen value to 5*5

```
. setx x3 5*5
```

To set all variables except x10 at their means, and fix $x10$ at its 25th percentile, call `setx` twice: once to set all variables at their means, and a second time to change the value of $x10$ to its 25th percentile.

```
. setx mean
. setx x10 p25
```

`setx` can also set values for groups of variables. To set x1 and x2 to their means, x3 to its median, and x4 and x5 to their 25th percentiles, type:

```
. setx (x1 x2) mean x3 median (x4 x5) p25
```

## 8.3 simqi

**Format:**

```
simqi [, pv genpv(newvar)
         ev genev(newvar)
         pr prval(value1 value2...) genpr(newvar1 newvar2...)
         fd(existing option) changex(var1 val1 val2 [& var2 val1 val2] )
          msims(#) tfunc(function) level(#) listx ]
```

**Description:**

After simulating parameters from the last estimation (see Section 8.1) and setting values for the explanatory variables (see Section 8.2), use `simqi` to simulate various quantities of interest, including predicted values, expected values, and first differences.

Predicted values contain two forms of uncertainty: "fundamental" uncertainty arising from sheer randomness in the world, and "estimation" uncertainty caused by not having an infinite number of observations. More technically, predicted values are random draws of the dependent variable from the stochastic component of the statistical model, given a random draw from the posterior distribution of the unknown parameters.

If there were no estimation uncertainty, the expected value would be a single number representing the mean of the distribution of predicted values. But estimates are never certain, so the the expected value must be a distribution rather than a point. To obtain this distribution, we average-away the fundamental variability, leaving only estimation uncertainty. For this reason, expected values have a smaller variance than predicted values, even though the point estimate should be roughly the same in both cases. `simqi` calculates two kinds of expected values: the expected value of $Y$, and the probability that $Y$ takes on a particular value. For models in which these two quantities are equal, `simqi` avoids redundancy by reporting only the probabilities.

Note: simulated expected values are equivalent to simulated probabilities for all the discrete choice models that `simqi` supports (`logit, probit, ologit, oprobit, mlogit`). In these models, the expected value of $Y$ is a vector, with each element indicating the probability that $Y = j$. Consider an ordered probit with outcomes 1, 2, 3. The expected value is [Pr($Y = 1$), Pr($Y = 2$), Pr($Y = 3$)], the mean of a multinomial distribution that generates the dependent variable.

A first difference is the difference between two expected values. To simulate first differences use the fd "wrapper", which is described below.

`simqi` can generate predicted values, expected values and first differences for all the models that it supports. By default, however, it will only report the quantities of interest that appear in the

table below. To view other quantities of interest or save the simulated quantities as new variables that can be analyzed and graphed, use one of `simqi`'s options.

| Statistical Model | Quantities displayed by default |
|---|---|
| `regress` | E(Y) |
| `logit` | Pr(Y=1) |
| `probit` | Pr(Y=1) |
| `ologit` | Pr(Y=j) for all j |
| `oprobit` | Pr(Y=j) for all j |
| `mlogit` | Pr(Y=j) for all j |
| `poisson` | E(Y) |
| `nbreg` | E(Y) |
| `sureg` | $E(Y_j)$ for all equations j |
| `weibull` | E(Y) |

**Options:**

**pv** displays a summary of the predicted values that simqi generated via simulation.

**genpv(newvar)** saves the predicted values as a new variable in the current dataset. Each "observation" of newvar represents one simulated predicted value.

**pr** displays a summary of the probabilities that `simqi` generated via simulation.

**prval(value1 value2 ...)** instructs `simqi` to evaluate the probability that the dependent variable takes-on each of the listed values

**genpr(newvar1 newvar2 ...)** saves the simulated probabilities as new variables in the current dataset. Each new "observation" represents one simulated probability. If both the prval() option and the genpr() option are used, `simqi` will save Pr(Y=value1) in *newvar*1, Pr(Y=value2) in *newvar*2, etc. If the prval() option is not specified, genpr() will save the probabilities in the order that they appear on the screen.

**ev** displays a summary of expected values that `simqi` generated via simulation. This option is not available for discrete choice models, where it is redundant with pr.

**genev(newvar)** saves the expected values in a new variable called newvar. Each observation of newvar represents one simulated expected value. This option is not available for discrete choice models, where it is redundant with genpr().

**fd(existing option)** is a `wrapper` that makes it easy to simulate first differences. Simply wrap the fd() wrapper around an existing option and specify the changex() option.

**changex(var1 val1 val2)** specifies how the explanatory variables (the $x$'s) should change when evaluating a first difference. changex uses the same basic syntax as `setx`, except that each explanatory variable has two values: a starting value and an ending value. For instance, fd(ev) changex(x1 .2 .8) instructs `simqi` to simulate a change in the expected value of $Y$ caused by increasing $x1$ from its starting value, 0.2, to its ending value, to 0.8.

**level(#)** specifies the confidence level, in percent, for confidence intervals. The default is level(95) or the value set by `set level`. For more information on the `set level` command, see the on-line help for level.

**msims(#)** sets the number of simulations to be used when calculating expected values. The number must be a positive integer. By default, the value of msims is set at 1000. `simqi` disregards the `msims` option whenever the expected value is parametrically defined.

**listx** instructs `simqi` to list the x-values that were used to produce the quantities of interest. These values were set using the `setx` command.

**tfunc(function)** allows the user to specify a transformation function for transforming the dependent variable. This option is only available for `regress` and `sureg`. The currently supported functions are

| Function | Transformation (for all variables j) |
|---|---|
| `squared` | $y_j \longrightarrow y_j * y_j$ |
| `sqrt` | $y_j \longrightarrow \sqrt{(y_j)}$ |
| `exp` | $y_j \longrightarrow e^{y_j}$ |
| `ln` | $y_j \longrightarrow \ln(y_j)$ |
| `logiti` | $y_j \longrightarrow e^{y_j}/(1+\sum_j e^{y_j})$ |

**Basic Examples:**

To display the default quantities of interest for the last estimated model, type:

```
. simqi
```

For a summary of the simulated expected values, type:

```
. simqi, ev
```

For a summary of the simulated probabilities, Pr(Y=j), for all j categories of the dependent variable, type:

```
. simqi, pr
```

To display only a summary of Pr(Y=1), the probability that the dependent variable takes on a value of 1, type:

```
. simqi, prval(1)
```

To generate first differences, use the fd() wrapper and the changex() option. For instance, the following command will simulate the change in the expected value of Y caused by increasing $x4$ from 3 to 7, while holding other explanatory variables at their means

```
. setx mean
. simqi, fd(ev) changex(x4 3 7)
```

To simulate the change in the simulated probabilities, Pr(Y=j), for all j categories of the dependent variable, given an increase in $x4$ from its minimum to its mean, type:

```
. setx mean
. simqi, fd(pr) changex(x4 min mean)
```

If you are only interested in the change in Pr(Y=1) caused by raising x4 from its 20th to its 80th percentile when other variables are held at their mean, type:

```
. setx mean
. simqi, fd(prval(1)) changex(x4 p20 p80)
```

**More Intricate Examples:**

To display not only the simulated expected values but also the x-values used to produce them, we would type:

```
. simqi, ev listx
```

`simqi` displays 95% confidence intervals by default, but we could modify the previous example to give a 90% confidence interval for the expected value:

```
. simqi, ev listx level(90)
```

To save the simulated expected values in a new variable called *predval*, type:

```
. simqi, genev(predval)
```

To simulate Pr(Y=0), Pr(Y=3), and Pr(Y=4), and then save the simulated probabilities as variables called *simpr*0, *simpr*3 and *simpr*4, type:

```
. simqi, prval(0 3 4) genpr(simpr0 simpr3 simpr4)
```

The changex option can be arbitrarily complicated. Suppose that we want to simulate the change in Pr(Y=1) caused by simultaneously increasing $x1$ from .2 to .8 and $x2$ from ln(7) to ln(10). The following lines will produce the quantities we seek:

```
. setx mean
. simqi, fd(prval(1)) changex(x1 .2 .8 x2 ln(7) ln(10))
```

We could augment the previous example by requesting a second first difference, caused by increasing $x3$ from its median to its 90th percentile. Simply separate the two changex requests with an ampersand.

```
. setx mean
. simqi, fd(prval(1)) changex(x1 .2 .8 x2 ln(7) ln(10) & x3 median p90)
```

Likewise, the fd() option can be as intricate as we would like. For instance, suppose that we have run a poisson regression. We want to see what happens to Pr(Y=2), Pr(Y=3), and the expected count when we increase $x1$ from its minimum to its maximum. To obtain our quantities of interest, we would type:

```
. setx mean
. simqi, fd(prval(2 3)) fd(ev) changex(x1 min max)
```

`simqi` allows us to save any simulated variable for subsequent analysis. To find the mean, standard deviation, and a confidence interval around any quantity of interest that has been saved in memory, use the `sumqi` command. To graph the simulations, use `graph` or `kdensity`.

The `tfunc()` option reverses common transformations that users have applied to the dependent variable. Suppose that you have taken the log of the dependent variable before running `estsimp regress`. The command `simqi` would provide quantities of interest on the logged scale. If you wanted to reverse the transformation, thereby recovering the original scale, you could type

```
.  simqi, tfunc(exp)
```

# 9 Frequently Asked Questions

**Why does Clarify give slightly different results each time?** Clarify uses random simulation to create quantities of interest and associated measures of uncertainty. Slight discrepancies are a result of taking a finite number of simulations and using a different random number seed. If you require more precision, increase the number of simulations drawn (see Section 8, Sub-section 8.1). If *exactly* the same numerical results are required, set the random number seed with the Stata command `set seed` before beginning the analysis.

**Is it ok if some of my explanatory variables are statistically insignificant?** Yes. Clarify computes quantities of interest based on *all* estimated coefficients, regardless of their level of statistical significance. This is not problematic because the true quantities of interest are usually the predicted values, expected values, and first differences, not the coefficients themselves. It is usually better to focus on the confidence intervals Clarify reports for each quantity it computes than the standard errors of coefficients.

**How do I know how large to make $M$?** In our experience $M = 1000$ is sufficient for most analyses. However, one check on the adequacy of $M$ is to verify that the means of the simulated parameters are equal to the estimated parameters within the desired degree of precision. If they are not, increase $M$ until you achieve the desired precision. Be aware that in larger models increasing $M$ may add to the computer time and memory required for simulation.

**How can I set the X's equal to the actual values in my dataset?** You can use the `setx` command to set the x's at any value you'd like, including the actual values that appear in the dataset. For instance, `setx [93]` will set all the x's equal to the values that appear in the 93rd observation of your dataset. The sequence of commands

```
setx mean                /* sets all x's to their means */
setx x1 x1[7]            /* resets x1 to value in 7th observation */
```

will set all the x's equal to their means, and then set x1 equal to the value of x1 that appears in the 7th observation of the dataset. If you wanted to get results for each x1 in your dataset, you could write a little loop, such as:

```
setx mean                /* set all x's to their mean levels */
local i 1                /* create a counter that runs from 1 to */
while `i' <= _N {        /*   _N, where _N is the # of observations */
    setx x1 x1[`i']      /* set x1 to the value in the ith obs */
```

```
        simqi                /* simulate quantity of interest */
        local i = `i' + 1    /* repeat for other observations */
    }
```

**How can I set the X's when I have interaction terms?** `setx` will work with interacted variables. Suppose the independent variables in your dataset are X1 and X2 and you have created an interaction term $X1X2 = X1 \times X2$. First you would run `estsimp` with the interacted variable, e.g.:

```
estsimp regress y x1x2
```

If you wish to set X1X2 to its mean, then you can use `setx` in the normal way:

```
setx x1x2 mean
```

However, if you want to set X1X2 to the product of the means of X1 and X2 (rather than the mean of the product), then you have two choices. First, you could set the values by hand, e.g.,

```
setx x1x2 10*12
```

where 10 is the mean of x1 and 12 is the mean of x2.

However, an even better method is the following sequence of commands:

```
summarize x1, meanonly        /* Compute the mean of x1 */
local mx1=`r(mean)'           /* Save the mean in a local macro */
summarize x2, meanonly        /* Compute the mean of x2 */
local mx2=`r(mean)'           /* Save the mean in a local macro */
setx x1x2 `mx1'*`mx2'         /* Setx to mean(x1)*mean(x2) */
```

**How can I use Clarify to analyze compositional data?**

The procedure involves four basic steps:

1. Run `tlogit` to transform the vote shares (or other compositional data) into log ratios

2. Run `estsimp sureg` to estimate a seemingly unrelated regression and simulate the parameters
3. Run `setx` to choose real or hypothetical values for the explanatory variables ($X$'s)
4. Run `simqi` with the `tfunc()` option to simulate the distribution of votes, conditional on the simulated parameters and chosen $X$'s.

Suppose that we are studying a political system with 500 electoral districts. Each observation or row in the dataset pertains to one of those districts. In this example, we have three political parties that each garner a percentage of the vote. Their vote shares, collected in variables v1, v2, and v3, sum to 100 percent.

First, we select party 3 as our reference party and transform the vote shares of the other two parties into log ratios with respect to party 3. Thus, $y1 = ln(v1/v3)$ and $y2 = ln(v2/v3)$. The appropriate syntax in Clarify is `tlogit v1 y1 v2 y2, base(y3) percent`, which will create two new variables: y1 and y2, which are the log ratios for v1 and v2 with respect to the base variable v3.

Second, use the `estsimp` command to run a seemingly unrelated regression model with the log ratios y1 and y2 as our dependent variables. The syntax is `estsimp sureg (y1 x1 x2) (y2 x3 x4)`. Each equation is enclosed in parentheses. Thus, the first equation states that the log ratio y1 is a linear function of the explanatory variables x1 and x2. The program will automatically add a constant term, as well, unless the user asks that it be suppressed. Likewise, the second equation states that y2 is a linear function of x3, x4, and a constant. The `estsimp` command will estimate the model and simulate the parameters. By default, `estsimp` will draw 1000 values for each parameter. In this example, the program would draw 1000 sets of betas (each set has six elements: three betas for equation 1 and three for equation two); the program would also generate 1000 simulations of $\Sigma$, a 2x2 matrix that governs the relationship between the errors of the two equations. Clarify will store these simulations in memory for subsequent use.

Third, use the `setx` command to choose some hypothetical or real values for our explanatory variables. For instance, type `setx (x1 x2) mean x3 15 x4 p20` to set variables x1 and x2 at their respective means, x3 equal to the number 15, and x4 equal to its twentieth percentile.

Finally, use the `simqi` command to simulate quantities of interest, such as the predicted distribution of votes. The command is `simqi, pv tfunc(logiti)`, where `tfunc(logiti)` tells the program to apply the inverse logistic function to transform the log ratios into shares of the total vote.

**How did you generate the graphs in your paper?** Clarify does not automatically produce graphs. In order to produce a graph, such as Figure 1 from King, Tomz, and Wittenberg (2000), you will need to use Stata's graphics commands. The sequence of commands used to generate Figure 1 is:

```
generate plo = .
generate phi = .
generate ageaxis = _n + 17 in 1/78
setx educate 12 white 1 income mean
local a = 18
while `a' <= 95 {
   setx age `a' agesqrd (`a'^2)/100
   simqi, prval(1) genpr(pi)
   _pctile pi, p(2.5,97.5)
   replace plo = r(r1) if ageaxis==`a'
   replace phi = r(r2) if ageaxis==`a'
   drop pi
   local a = `a' + 1
}
sort ageaxis
graph plo phi ageaxis, s(ii) c(||)
```

**How can I create a log of all the commands and output in a Clarify session?** See [R] log in the Stata reference manual or consult the on-line help for the `log` command.

**How do I cite this program?** If you use this software, please cite

> Michael Tomz, Jason Wittenberg, and Gary King (2001). CLARIFY: Software for Interpreting and Presenting Statistical Results. Version 2.0 Cambridge, MA: Harvard University, June 1. `http://gking.harvard.edu`

and

> Gary King, Michael Tomz, and Jason Wittenberg (2000). "Making the Most of Statistical Analyses: Improving Interpretation and Presentation." *American Journal of Political Science* 44, no. 2 (April 2000): 347-61.

**Can I share Clarify with others?** Clarify is (C) Copyright, 1999−2001, Michael Tomz, Jason Wittenberg and Gary King, All Rights Reserved. You may copy and distribute this program provided the copy is identical to the original and you do not charge for it. To request an exception, please contact Michael Tomz, `tomz@stanford.edu`.

We recommend that you distribute the current version of this program, which is available from `http://GKing.Harvard.Edu`.

**What if I find a bug?** First, get the most recent version (from `http://gking.harvard.edu`) and try to replicate the problem. If the problem persists, copy down *exactly* what you see on the screen when the program crashes, and email it along with the command you used to generate the error, to `tomz@stanford.edu`, `witty@polisci.wisc.edu`, or `king@harvard.edu`.

You may also send comments to Michael Tomz, Department of Political Science, Encina Hall, Stanford University, Stanford, CA 94305-6044.

# 10 Formulae - A Peek Under the Hood

This section is intended for advanced users who want details about the algorithms that Clarify uses to simulate parameters, set values for the explanatory variables, and compute quantities of interest. We welcome any suggestions for improvement.

## 10.1 Algorithms for `estsimp`

Recall that the `estsimp` command performs two functions: it estimates the main and ancillary parameters ($\gamma$) of the statistical model, and it draws simulations of those parameters from their asymptotic sampling distribution.

Typically, the sampling distribution is multivariate normal with mean equal to the point-estimates of the parameters ($\hat{\gamma}$) and variance equal to the variance-covariance matrix of estimates $\hat{\mathrm{V}}(\hat{\gamma})$. The current version of Clarify contains two exceptions to this rule.

In the case of linear regression, the effect coefficients ($\beta$s) are drawn from a multivariate normal, but simulations of the homoskedastic variance $\sigma^2$ are obtained in a separate step from a scaled inverse $\chi^2$ distribution with $\nu = n - k$ degrees of freedom, where $n$ is the number of observations in the dataset and $k$ is the number of explanatory variables, including the constant term (Gelman, et al., 1995, p. 237). The two-step procedure is legitimate because the effect coefficients and the variance parameter are orthogonal in a linear regression; the procedure is desirable because $\sigma^2$ is strictly positive, and therefore more appropriately drawn from its exact posterior than from a normal distribution. To obtain simulations of $\sigma^2$, the program draws $c$ from a $\chi^2$ with $\nu$ degrees of freedom, and then calculates $\tilde{\sigma}^2 = \nu\sigma^2/c$. The resulting draws have an expected value of $\left(\frac{\nu}{\nu-2}\right)\hat{\sigma}^2$, which approaches $\hat{\sigma}^2$ as $\nu \to \infty$.

Likewise, the effect coefficients ($\beta$s) of a seemingly unrelated regression are drawn from the multivariate normal, but simulations of the variance matrix $\Sigma$ are obtained in a separate step. Here, the appropriate posterior distribution is the inverse Wishart (Gelman, et al., 1995, p. 481) with $\nu$ degrees of freedom and dimension $p$, where $p$ is the number of equations in the seemingly unrelated regression model. In cases where the number of explanatory varables varies from one equation to

the next, Clarify calculates $n-k$ for each equation and sets $\nu$ equal to the mean of those values. To obtain simulations of $\Sigma$, the program draws from a Wishart with scale factor $(\nu\hat{\Sigma})^{-1}$ and inverts the draws. The algorithm for drawing from the Wishart relies on Bartlett's decomposition, which is concisely summarized in Johnson (1987, p. 204) and Ripley (1987, pp. 99-100). `estsimp` produces draws that have an expected value of $\frac{\nu}{\nu-p-1}\hat{\Sigma}$, which approaches $\hat{\Sigma}$ as $\nu$ goes to infinity. In small samples this procedure is conservative, since $\nu > \nu - p - 1$, implying that $E(\tilde{\Sigma}) > \hat{\Sigma}$.

For all models, simulations of the main and ancillary parameters are random. This means that, in any given run of `estsimp`, the average value of $\tilde{\gamma}$ may be slightly smaller or larger than the point estimate $\hat{\gamma}$, though the approximation becomes more precise with a higher number of simulations. Users can force the mean of the simulated parameters to equal the vector of point estimates by requesting antithetical simulations (Stern 1997, pp. 2028-29). The `antisim` option instructs the program to draw random numbers in pairs from the uniform[0,1] distribution, with the second draw being the complement of the first. For instance, if the first draw is 0.3 then the complementary draw is 0.7. The draws are, therefore, exactly balanced around the mean of the uniform distribution. These anthithetical simulations are then used to obtain antithetical or balanced draws from the multivariate normal.

When users are analyzing a single dataset, Clarify estimates a single vector $\hat{\gamma}$ with variance $\hat{V}(\hat{\gamma})$ and draws all $M$ simulations based on those estimates. The table that appears on the screen gives the exact point estimates and standard errors, instead of reporting the means and standard deviations of the simulations.

The procedure is somewhat more complicated when the researcher employs the `mi` option to analyze several imputed datasets. In this case, `estsimp` repeats the following algorithm $I$ times, where $I$ is the number of completed datasets: estimate the parameters and their variance-covariance matrix conditional on the information in dataset $i$ ($i = 1, 2, \ldots, I$), and then draw $M/I$ sets of parameters from their sampling distribution. By repeating this algorithm $I$ times, the program generates $M$ sets of simulated parameters. The output table gives the analytical point-estimate, standard error, and $t$-statistic for each parameter, instead of reporting the means and standard deviations of the simulations. Specifically, the multiple-imputation point estimate for parameter $q$ is $\bar{q} = \frac{1}{I}\sum_{i=1}^{I}\hat{q}_i$ and the variance associated with $\bar{q}$ is a weighted combination of the within-imputation and between-imputation variances: $V(\bar{q}) = \bar{w} + (1 + I^{-1})b$, where $\bar{w} = \frac{1}{I}\sum_{i=1}^{I} V(\hat{q}_i)$ and $b = \frac{1}{I-1}\sum_{i=1}^{I}(\hat{q}_i - \bar{q})^2$. The ratio of $\bar{q}$ (the parameter estimate) to $V(\bar{q})^{1/2}$ (its standard error) forms a $t$-statistic with degrees of freedom $\nu = (I-1)[1 + \frac{\bar{u}}{(1+I^{-1})b}]^2$. For more information about these procedures, see King, et al. (2001) and Schafer (1997, pp. 109-110).

## 10.2 Algorithms for `setx`

`setx` allows the user to choose real or hypothetical values for the explanatory variables (the $X$s). The program employs standard formulae for the mean, the minimum, the maximum, percentiles,

and other descriptive statistics. If the user is analyzing several imputed datasets, `setx` will calculate the average statistic across the datasets. For instance, the command `setx x1 mean` will calculate the mean of x1 in each dataset, and then set x1 equal to the average of those means. Similarly, the command `setx x1 x1[3]` will obtain the value of x1 in the third row of each imputed dataset, and then set x1 equal to the average of those values.

### 10.3 Algorithms for `simqi`

`simqi` simulates quantities of interest based on the parameters that were generated by `estsimp` and the $x$-values that were chosen with `setx`. The program obtains simulations of the dependent variable and uses them to calculate expected values, probabilities, first differences, and other quantities of interest. This procedure works in all cases but involves some approximation error, which users can make arbitrarily small by choosing a sufficient number of simulations. In many cases, though, shortcuts exist that can curtail both computation time and approximation error. `simqi` employs such shortcuts whenever possible. Here, we sketch the algorithms for each model that Clarify supports.

`regress`: The exact algorithm in `simqi` depends on whether the user has transformed the dependent variable (e.g., taken the log of $y$) prior to estimation. If no such transformation has occurred, the program generates one predicted value according to the formula $\tilde{y} = X_c\tilde{\beta} + \tilde{\epsilon}$, where $\tilde{\beta}$ is a vector of simulated effect coefficients and $\tilde{\epsilon}$ is one draw from $N(0, \tilde{\sigma}^2)$. Likewise, the program simulates one expected value as $\tilde{E}(y) = X_c\tilde{\beta}$. The algorithm becomes a bit more complicated if the user transformed the dependent variable prior to estimation, and would like to reverse the transformation when interpreting the results. Let $f$ represent a function, as identified by the `tfunc()` option, that reverses the transformation. If $f$ has been specified, the program simulates one predicted value according to the formula $f(X_c\tilde{\beta} + \tilde{\epsilon})$. For an expected value, the program draws $m$ values of $\tilde{\epsilon}_d$ $(d = 1, 2, \ldots, m)$ from $N(0, \tilde{\sigma}^2)$ and then computes $(1/m)\sum_{d=1}^{m} f(X_c\tilde{\beta} + \tilde{\epsilon}_d)$, which is the average of $m$ predicted values.

`logit`: The formula for $\tilde{\pi}$, the simulated probability that the dependent variable $y$ takes on a value of 1, is $1/(1 + e^{-X_c\tilde{\beta}})$. To obtain one simulation of $y$, the program draws a number from the Bernoulli distribution with parameter $\tilde{\pi}$.

`probit`: The formula for $\tilde{\pi}$, the simulated probability that the dependent variable $y$ takes on a value of 1, is $\Phi(X_c\tilde{\beta})$ where $\Phi$ is the c.d.f. of the standard normal distribution. To obtain one simulation of $y$, the program draws a number from the Bernoulli distribution with parameter $\tilde{\pi}$.

`ologit`: The exact formula depends on the number of categories in the dependent variable. Suppose there are three categories. Let $\tilde{\beta}$ represent one simulated vector of effect coefficients and let $\tilde{\tau}_{lo}$ and $\tilde{\tau}_{hi}$ stand for draws of the cutpoints. To obtain one simulation of the probabilities for each category $(y = 0, y = 1, y = 2)$, the program calculates: $\tilde{\pi}_0 \equiv \tilde{Pr}(y = 0) = \frac{1}{1+e^{(X_c\tilde{\beta} - \tilde{\tau}_{lo})}}$, $\tilde{\pi}_1 \equiv \tilde{Pr}(y = 1) =$

$\frac{1}{1+e^{(X_c\tilde{\beta}-\tilde{\tau}_{hi})}} - \frac{1}{1+e^{(X_c\tilde{\beta}-\tilde{\tau}_{lo})}}$, and $\tilde{\pi}_2 \equiv \tilde{Pr}(y=2) = 1 - \frac{1}{1+e^{(X_c\tilde{\beta}-\tilde{\tau}_{hi})}}$. With these results, the program can draw a predicted value, $\tilde{y}$, from a multinomial distribution with parameters $\tilde{\pi}_0$, $\tilde{\pi}_1$, $\tilde{\pi}_2$, and $n = 1$.

`oprobit`: The exact formula depends on the number of categories in the dependent variable. Suppose there are three categories. Let $\tilde{\beta}$ represent one simulated vector of effect coefficients and let $\tilde{\tau}_{lo}$ and $\tilde{\tau}_{hi}$ stand for draws of the cutpoints. To obtain one simulation of the probabilities for each category $(y = 0, y = 1, y = 2)$, the program calculates $\tilde{\pi}_0 \equiv \tilde{Pr}(y = 0) = \Phi(\tilde{\tau}_{lo} - X_c\tilde{\beta})$, $\tilde{\pi}_1 \equiv \tilde{Pr}(y = 1) = \Phi(\tilde{\tau}_{hi} - X_c\tilde{\beta}) - \Phi(\tilde{\tau}_{lo} - X_c\tilde{\beta})$, and $\tilde{\pi}_2 \equiv \tilde{Pr}(y = 2) = \Phi(X_c\tilde{\beta} - \tilde{\tau}_{hi})$. With these results, the program can draw a predicted value, $\tilde{y}$, from a multinomial distribution with parameters $\tilde{\pi}_0$, $\tilde{\pi}_1$, $\tilde{\pi}_2$, and $n = 1$.

`mlogit`: The probability equation for the $K$ nominal outcomes of the multinomial logit is $\tilde{\pi}_j \equiv \tilde{Pr}(y = j) = \frac{e^{X_c\tilde{\beta}_j}}{\sum_{k=1}^{K} e^{X_c\tilde{\beta}_k}}$, where one of the $J$ outcomes is the base category, such that the effect coefficients $\tilde{\beta}$ for that category are set to zero. With these results, the program can draw a predicted value, $\tilde{y}$, from a multinomial distribution with parameters equal to the $\tilde{\pi}$s and $n = 1$.

`poisson`: The formula for the expected value $\tilde{\mu}$ is $e^{X_c\tilde{\beta}}$, and the probability that the dependent variable takes on the integer value $j$ is $\tilde{Pr}(y = j) = \frac{e^{-\tilde{\mu}}\tilde{\mu}^j}{j!}$. To obtain one predicted value, the program draws $\tilde{y}$ from a Poisson distribution with parameter $\tilde{\mu}$. The Poisson simulator is adapted from Press, et al. (1992), pp. 293-95.

`nbreg`: The formula for the expected value $\tilde{\mu}$ is $e^{X_c\tilde{\beta}}$, just as in the Poisson regression model (Long 1997, pp. 230-33). The probability that the dependent value takes on the integer value $j$ can be simulated as $\tilde{Pr}(y = j) = \frac{\Gamma(j+\tilde{\alpha}^{-1})}{j!\Gamma(\tilde{\alpha}^{-1})} \left(\frac{\tilde{\alpha}^{-1}}{\tilde{\alpha}^{-1}+\tilde{\mu}}\right)^{\tilde{\alpha}^{-1}} \left(\frac{\tilde{\mu}}{\tilde{\alpha}^{-1}+\tilde{\mu}}\right)^{j}$. `simqi` obtains $\tilde{\alpha}$, the "overdispersion" parameter, by drawing simulations of $ln(\alpha)$ and the other parameters from the multivariate normal distribution and then calculating $e^{ln(\alpha)}$. To obtain a predicted value $\tilde{y}$, the program draws one number from a poisson distribution with mean $e^{X_c\tilde{\beta}+\tilde{\epsilon}}$, where $e^{\tilde{\epsilon}}$ is simulated from a gamma distribution with shape parameter $\tilde{\alpha}^{-1}$ and scale parameter $\tilde{\alpha}$. When $\tilde{\alpha}^{-1} < 1$, the gamma simulator is based on the algorithm developed by Ahrens and Dieter, as described in Ripley (1987, p. 88). For other values of $\tilde{\alpha}^{-1}$, the gamma simulator is based on the procedure by Best, as described in Devroye (1986, p. 410).

`sureg`: As with `regress`, the algorithm for interpreting the results of a `sureg` depends on whether the user transformed the dependent variable. If the user estimated the model without transforming the dependent variable, the program generates one predicted value for equation $k$ according to the formula $\tilde{y}_k = X_c\tilde{\beta}_k + \tilde{\epsilon}_k$, where $\tilde{\beta}_k$ is a vector of simulated effect coefficients for equation $k$ and $\tilde{\epsilon}_k$ is a simulated disturbance term for that equation. Disturbances for all equations are drawn simultaneously from a multivariate normal distribution with mean 0 and variance matrix $\tilde{\Sigma}$, as obtained from the inverse Wishart. Likewise, the program simulates one expected value for equation $k$ as $\tilde{E}(y_k) = X_c\tilde{\beta}_k$. If the user has transformed the dependent variable, let $f$ represent

the function that reverses the transformation. The program simulates one predicted value for equation $k$ according to the formula $f(X_c\tilde{\beta}_k + \tilde{\epsilon}_k)$. For an expected value, the program draws $m$ sets of disturbance terms from $N(0, \tilde{\Sigma})$ and indexes them as $\tilde{\epsilon}_{k,d}$, where $k$ marks the equation and $d = 1, 2, \ldots, m$. Then, for each equation $k$ the program computes $(1/m) \sum_{d=1}^{m} f(X_c\tilde{\beta}_k + \tilde{\epsilon}_{k,d})$, which is the average of $m$ predicted values.

`weibull`: The algorithm depends on which metric, proportional hazard (PH) or accelerated failure-time (AFT) metric, was used at the `estsimp` stage. The expected value is defined as $\tilde{\lambda}^{1/\tilde{p}}\Gamma\,(1 + 1/\tilde{p})$. In the AFT metric, $\tilde{\lambda} = e^{-X_c\tilde{\beta}\tilde{p}}$; in the PH metric, $\tilde{\lambda} = e^{X_c\tilde{\beta}}$. The program obtains simulations of the ancillary shape parameter $p$ drawing by $ln(p)$ and the other parameters from a multivariate normal distribution and then calculating $e^{ln(p)}$. To obtain a predicted value, the program draws one number from the Weibull distribution with parameters $\tilde{\lambda}$ and $\tilde{p}$.

# 11 References


Devroye, Luc (1986). *Non-Uniform Random Variate Generation*. New York: Springer-Verlag.

King, Gary, Michael Tomz, and Jason Wittenberg (2000). "Making the Most of Statistical Analyses: Improving Interpretation and Presentation." *American Journal of Political Science* 44, no. 2 (April 2000): 347-61.

King, Gary, James Honaker, Anne Joseph, and Kenneth Scheve (2001). "Analyzing Incomplete Political Science Data: An Alternative Algorithm for Multiple Imputation." *American Political Science Review* 95, no. 1 (March 2001): 49-69.

Johnson, Mark E. (1987). *Multivariate Statistical Simulation*. New York: John Wiley & Sons.

Long, J. Scott (1997). *Regression Models for Categorical and Limited Dependent Variables*. Thousand Oaks, CA: SAGE Publications.

Press, William H., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery (1992). *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. New York: Cambridge University Press.

Ripley, Brian D. (1987). *Stochastic Simulation*. New York: John Wiley & Sons.

Schafer, J.L. (1997). *Analysis of Incomplete Multivariate Data*. New York: Chapman & Hall.

Stern, Steven (1997). "Simulation-Based Estimation." *Journal of Economic Literature* 35, no. 4 (December): 2006-39.


# 12 Acknowledgements


We gratefully acknowledge comments and suggestions by Nick Cox, William Gould, Andrew D. Martin, and Ken Scheve. We also wish to thank the many users of Clarify who have provided numerous suggestions for making the software more flexible and user-friendly. Parts of this program were inspired by J. Scott Long, "CATDEV: Stata Modules for Interpretation of Categorical Dependent Variables" (Indiana University, April 16, 1998). The multiple imputation procedure in `estsimp` extends upon the `miest` command written by Ken Scheve (Harvard University, February 6, 1999).