powersim

–

Stata module for simulation-based power analysis for linear and generalized linear models

–

Tutorial

version 1.0

Joerg Luedicke

September 3, 2013

Contents

Introduction

The Stata package `powersim` exploits the flexibility of a simulation-based approach to the analysis of statistical power by providing a facility for automated power simulations in the context of linear and generalized linear regression models. The package supports a wide variety of uni- and multivariate covariate distributions and all family and link choices that are implemented in Stata's `glm` command (as of version 13). The package mainly serves two purposes: First, it provides access to simulation-based power analyses for researchers without much experience in simulation studies. Second, it provides a convenient simulation engine for more advanced users who can easily complement the automated data generation with their own code for creating more complex synthetic datasets. Beyond that, `powersim` can be used as a convenient tool for simple Monte Carlo experiments in a regression modeling context.

Statistical power in the context of a null hypothesis significance testing framework is thought of as the probability of correctly rejecting the null-hypothesis when it is indeed false. Power is thus the complement of the Type-II error rate β, the rate at which a null hypothesis is not rejected (at a given level of α) when it is false. If we only allow for the distinct events of either rejecting or not rejecting a null-hypothesis and the null-hypothesis can be either true or false, we have the following four conditional probabilities:

	H_0 true	H_0 false
fail to reject H_0	$1-\alpha$	β
reject H_0	α	$1-\beta$

where α is the Type-I error probability (detecting an effect when there is none) and β is the probability of committing a Type-II error, the failure to detect an effect if there indeed is one in the larger population. Thus $1-\beta$ is the probability of correctly rejecting a false null-hypothesis, and this is the quantity that `powersim` estimates via simulation.

Basic functionality

The primary purpose of `powersim` is to compute statistical power with respect to specified hypotheses in a point null hypothesis significance testing framework. However, `powersim` can also be used for quickly generating synthetic datasets or for performing simple Monte Carlo experiments. In any case, `powersim` requires the specification of a data generating process. A *data generating model* can be specified by either using command options or by providing an existing do-file. If command options are used, users need to specify a distributional family, a link function, covariates (with specified distributions; up to three covariates can be specified), the true parameter values, and optionally up to three interaction effects. Normally distributed covariates can be drawn from multivariate normal distributions by specifying Pearson correlation coefficients. Based on these inputs, `powersim` automatically creates a do-file which is used for data generation

throughout the simulations. Depending on a user's purposes, it happens that the specification of a data generating process is not directly possible via command options. In this case users can feed an *existing* do-file to `powersim` which needs to contain all specifications of the data generating model (including the link function), except for the distributional family. Also note that this do-file must contain the placeholder `_bp` for the effect of interest. `_bp` will be filled in by the effect sizes (population parameters) that are specified via the command option `b()`. If users want to provide an existing do-file it is recommended to create one with `powersim` and then use that file as a template.

It further needs to be decided whether the predictor data be generated inside or outside the actual simulation loops. If the predictor data itself is regarded as random then it should be put inside the simulations such that new data be generated for each Monte Carlo replication. If, on the other hand, predictor data is regarded as fixed (e.g., an experimental design variable) the data may be generated outside the simulation loops so that the only stochastic component of the simulations is the error variance as defined by the choice of a distributional family.

Users also need to specify an *analysis model* for which either Stata's `regress` or `glm` commands may be used. `powersim` collects the coefficient of interest and its standard error from the saved results of these commands. The effect of interest must be specified using the `position()` option. Using the `force` option allows users to use model commands other than `regress` or `glm`. Please see example 5 below for a list of conditions that need to be met in order for other commands to work with `powersim`. By default, significance tests and 95% confidence interval coverages are based on the normal distribution for any models other than `regress` (for which Student's t distribution is assumed). This default behavior can be changed, however.

Note that `powersim` **replaces the data in memory** with a dataset that contains results from each Monte Carlo replication for further analysis. Also note that a `scalar` with the name `_bp` is defined by the program which would override an existing `scalar` that happened to have the same name. In case users specify multivariate normal data, Stata matrices with the names `__M`, `__SD`, and `__C` are created which would replace any existing matrices with these names.

Finally, a note on random number generation: official Stata RNGs are used for all distributional families, except for the inverse Gaussian distribution. Random deviates from the inverse Gaussian distribution are drawn using a Mata translation of the user-written `rndivgx` command (Hilbe and Linde-Zwirble (1995)). Random deviates from the negative binomial distribution are drawn using (continuous) Poisson-gamma mixture distributions. The following table provides an overview over `powersim`'s (essential) usage of the different RNGs in generating the outcome variable `y`; `xb` is the linear predictor defined in the generated or provided do-file:

family [#]	generation of outcome variable y
`gaussian #`	`y = rnormal(xb,#)`
`igaussian #`	`y = rig(xb,#)`
`binomial #`	`y = rbinomial(#,xb)`
`poisson`	`y = rpoisson(xb)`
`nbinomial #`	`y = rpoisson(xb*rgamma(1/#, #))`
`gamma #`	`y = rgamma(xb,#)`

Specific usage and features of `powersim` are explained below by going through a number of worked examples. Each example highlights different features but example 1 and 2 cover the most basic ones. Not all options are shown in the examples, type `help powersim` in Stata for more details.

For a brief general introduction to power analysis, see chapter 20 in Gelman and Hill (2007). For a Stata related introduction to simulation-based power analysis, see Feiveson (2002). For a Stata related introduction to generalized linear models, see Hardin and Hilbe (2012), which also includes a chapter on data fabrication and simulation.

Example 1: Differences in means

Suppose we would be interested in calculating power for testing differences in means between two equally sized groups, assuming a linear model with Gaussian errors. We could do that using the analytical approach with Stata's `power` command. With the following code we can request a plot of power as a function of sample size, by effect size. We are interested in power calculations for sample sizes ranging from N=10 to N=100, at intervals of 10, and for three different effect sizes 0.4, 0.5, and 0.6, measured at standard deviation scale:

```
. power twomeans 0 (0.4 0.5 0.6), n(10(10)100) ///
> graph(ylabel(0(.1)1) title("") subtitle("")  ///
> xval recast(line))
```

The resulting graph is shown in Figure 1.

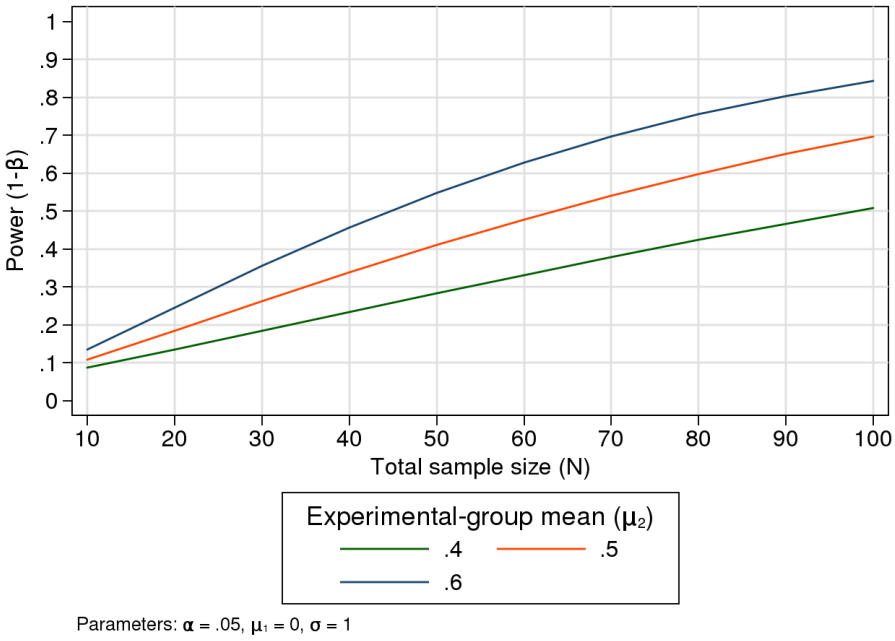

Figure 1: Analytically derived power curves for a mean comparison test

Now we can replicate these results using simulations. In the `powersim` code below, we specify effect sizes as a *numlist* (option `b()`; the `position()` option is required for the specification of the matrix position of b in matrix `e(b)` of the analysis model; this is to tell `powersim` where to find the coefficient of interest in the analysis model), the desired α-level, sample sizes, and the number of Monte Carlo replications. We also define our data generating model by choosing a distributional family, a link function, and a covariate. Regarding the family, the 1 after `gaussian` is the value of the standard deviation of the Gaussian residuals. We would not need to specify it here since 1 is the default, but did so for explicity. The specification of our covariate involves four inputs: first, an arbitrary variable name, the second input is the placeholder `_bp` with which we tell `powersim` that this is the variable for which effect we intend to simulate the statistical power and that the specified effect sizes will be plugged in for this variable. The third input specifies the distribution of the covariate which in this case consists of fixed design blocks of equal size. Finally, the last input specifies the number of blocks where the number can range between two and four. Our data generating model can thus be expressed as

$$y = \_bp * x1 + \epsilon \qquad , \quad \epsilon \sim \mathcal{N}(0,1) \tag{1}$$

The `dofile()` option at the very bottom is required and we have to specify a filename for the do-file that `powersim` automatically generates. Finally, we can specify our analysis model after the colon by using Stata's `regress` command:

```
. powersim , ///
> b(0.4 0.5 0.6) ///
> pos(1) ///
> alpha(0.05) ///
> sample(10(10)100) ///
> nreps(10000) ///
> family(gaussian 1) ///
> link(identity) ///
> cov1(x1 _bp block 2) ///
> dofile(example1_dofile, replace) : reg y x1
```

(output omitted)

Power analysis simulations

```
Effect sizes b:       .4 .5 .6
H0:                   b = 0
Sample sizes:         10 20 30 40 50 60 70 80 90 100
alpha:                .05
N of replications*:   10000
do-file used for data generation: example1_dofile
Model command:                    reg y x1
```

Power by sample and effect sizes:

Sample size	Specified effect size		
	.4	.5	.6
10	0.091	0.107	0.138
20	0.130	0.185	0.246
30	0.189	0.259	0.352
40	0.235	0.340	0.458
50	0.287	0.408	0.543
60	0.331	0.481	0.627
70	0.376	0.534	0.698
80	0.418	0.603	0.756
90	0.465	0.648	0.799
100	0.508	0.693	0.843

```
Total N of requested  MC replications: 300000
Total N of successful MC replications: 300000
* per sample and effect size combination
```

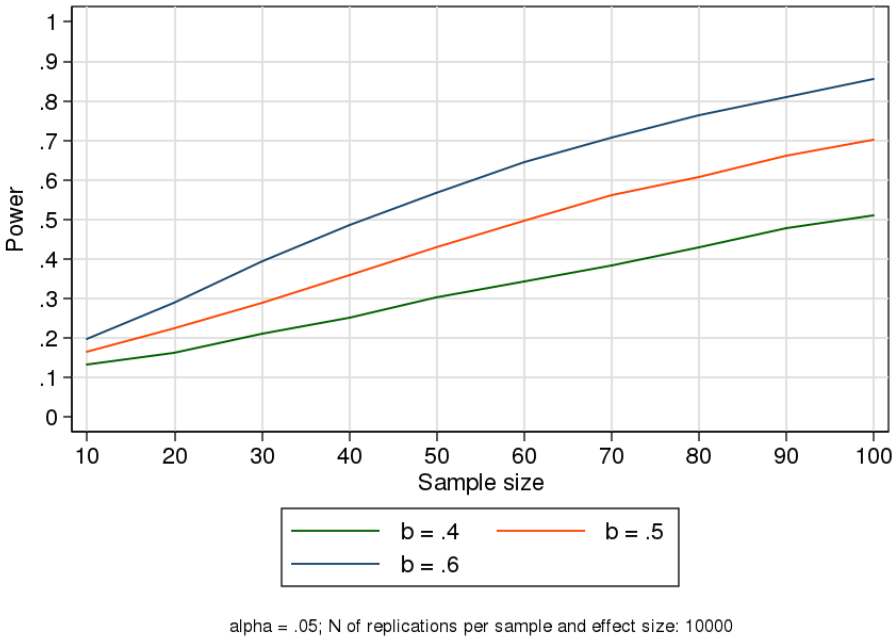

Figure 2: Simulated power curves for a mean comparison test

We could now use `powersimplot` as a post-estimation command in order to plot the power curves:

```
. powersimplot
```

The resulting graph is shown in Figure 2. We can see that we essentially get the same results as we did with the analytical approach.

Results from each Monte Carlo replication are available in a dataset and can be further analyzed. We will get back to this in one of the other examples below. The dataset contains the following variables:

```
. describe

Contains data from ex1b_results.dta
  obs:       300,000
 vars:             9                          24 Jul 2013 17:44
 size:    15,300,000
```

variable name	storage type	display format	value label	variable label
nd	double	%10.0g		Iteration ID
esize	double	%10.0g		Specified effect size
esize_id	byte	%8.0g	eid	Specified effect size (as factor variable)
n	double	%10.0g		Sample size
b	double	%10.0g		Estimated coefficient b
se	double	%10.0g		Standard error of b
p	double	%10.0g		p-value
power	byte	%8.0g		1 = p < .05
c95	byte	%8.0g		95% CI coverage (1=covered)

```
Sorted by:
```

Example 2: Interaction effects among correlated predictors in a linear model

Suppose now that we would be interested in the power of a test of an interaction effect between two normally distributed variables that are correlated with $\rho = 0.5$. We define our predictor variables using the `cov1()` and `cov2()` options, requesting two normally distributed variables. We use the `inter1()` option to specify the interaction effect and since this is the effect of interest we use the placeholder `_bp`. This model can be expressed as:

$$y = 5 - 0.5x_1 + 0.4x_2 + 0.1x_1x_2 + \epsilon \tag{2}$$

with $\epsilon \sim \mathcal{N}(0, 0.84^2)$ and $(x_1, x_2) \sim \mathcal{N}(\mu, \Sigma)$ with $\mu = \begin{pmatrix} 0 \\ 2 \end{pmatrix}, \Sigma = \begin{bmatrix} 1 & \\ 0.5 & 1.5^2 \end{bmatrix}$.

This time we will generate the data `inside` the simulation loop since, unlike in the previous example where we had a fixed design variable, we regard both covariates as random variables, as a part of the stochastic process.

Now, before we run this we could first check a few more things. With the `dryrun` option we request an overview of our data generating and analysis models, while with the `detail` option we also request a view on the do-file content which is used for generating the predictor data. Very important to check is the specified column position of the effect of interest in the analysis model. Since the variables in the analysis model command could be specified in arbitrary order and with different syntax, we have to tell `powersim` where to find the effect of interest. In this case we can see that we correctly specified 3 as the column position in the displayed `e(b)` matrix.

```
. powersim , ///
> b(0.1) ///
> pos(3) ///
> alpha(0.05) ///
> sample(200) ///
> nreps(1000) ///
> family(gaussian .84) ///
> link(identity) ///
> cons(5) ///
> cov1(x1 -0.5 normal 0 1) ///
> cov2(x2 0.4 normal 2 1.5) ///
> inter1(_bp x1*x2) ///
> corr12(0.5) ///
> inside ///
> seed(1234) ///
> addscalar(r2) ///
> dofile(example2_dofile, replace) ///
> dryrun detail : reg y c.x1##c.x2
```

```
powersim dry run:
do-file used for data generation:

. *--------------------------
. // Generating predictor data
.
```

```
. matrix __M=(0, 2)
. matrix __SD=(1, 1.5)
. matrix __C = (1, 0.5, 1)
.
. drawnorm x1 x2 , ///
> means(__M) sds(__SD) corr(__C) ///
> cstorage(upper) forcepsd double
.
.
. *---------------------------
.
. *----------------------------------------
. // Link function with specified parameters
. *  link = identity
.
. generate double xb = 5 + -0.5*x1 + 0.4*x2 + _bp*x1*x2
.
. *----------------------------------------
.
end of do-file
Data generating model: family(gaussian .84) link(identity)
Analysis model command: reg y c.x1##c.x2
```

y	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
x1	-.5	.	.	.	.	.
x2	.4	.	.	.	.	.
c.x1#c.x2	.1	.	.	.	.	.
_cons	5	.	.	.	.	.

```
Matrix e(b) from analysis model (check column position of effect of interest):
e(b)[1,4]
                    c.x1#
       x1     x2   c.x2  _cons
y1    -.5     .4     .1      5
Position specified in option position(#): 3
SD of Gaussian error: .84
```

Now we could still do some more checking by generating a single realization of a dataset which is created under the specified data generating model (using the `gendata` option). If more than one effect size is specified in `b()`, `powersim` will use the first from the provided list to generate the data. By default, a dataset with $N = 10,000$ observations is created which can be changed with the `nobs()` option. The following example also demonstrates that no analysis model need to be specified when one is only generating a single dataset. It would also be legal syntax to include the analysis model command (after a colon), although it will not do anything. After we have created the dataset we could, for example, fit the analysis model to the data just to check whether everything works out as expected:

```
. powersim , ///
> b(0.1) ///
> pos(3) ///
> alpha(0.05) ///
> sample(200) ///
> nreps(1000) ///
> family(gaussian .84) ///
> link(identity) ///
> cons(5) ///
> cov1(x1 -0.5 normal 0 1) ///
> cov2(x2 0.4 normal 2 1.5) ///
> inter1(_bp x1*x2) ///
> corr12(0.5) ///
> inside ///
> seed(1234) ///
> addscalar(r2) ///
> dofile(example2_dofile, replace) ///
> gendata
. reg y c.x1##c.x2
```

Source	SS	df	MS		Number of obs = 10000
					F(3, 9996) = 1320.69
Model	2872.10059	3	957.366865		Prob > F = 0.0000
Residual	7246.09372	9996	.724899332		R-squared = 0.2839
					Adj R-squared = 0.2836
Total	10118.1943	9999	1.01192062		Root MSE = .85141

y	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
x1	-.5000145	.0143256	-34.90	0.000	-.5280955	-.4719334
x2	.3941009	.0066056	59.66	0.000	.3811526	.4070492
c.x1#c.x2	.1061116	.005203	20.39	0.000	.0959126	.1163105
_cons	5.005941	.0161153	310.63	0.000	4.974352	5.037531

We could also inspect the data more closely (e.g., by plotting certain aspects of the data) in order to check whether the data are consistent with our ideas about the data generating process and our hypotheses. But now we finally run our simulations:

```
. powersim , ///
> b(0.1) ///
> pos(3) ///
> alpha(0.05) ///
> sample(200) ///
> nreps(1000) ///
> family(gaussian .84) ///
> link(identity) ///
> cons(5) ///
> cov1(x1 -0.5 normal 0 1) ///
> cov2(x2 0.4 normal 2 1.5) ///
> inter1(_bp x1*x2) ///
> corr12(0.5) ///
> inside ///
> seed(123) ///
> addscalar(r2) ///
> dofile(example2_dofile, replace) : reg y c.x1##c.x2
```

```
Power simulations:

Model:              reg y c.x1##c.x2
Effect:             b = 0.100
alpha:              .05

n of replications:  1000
sample size:        200
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
..................................................    50
(output omitted)
.................................................. 1000

Power analysis simulations

Effect sizes b:       .1
H0:                   b = 0
Sample sizes:         200
alpha:                .05
N of replications*:   1000
do-file used for data generation: example2_dofile
Model command:                    reg y c.x1##c.x2

Power by sample and effect sizes:

------------------------
           | Specified
           |  effect
           |    t
Sample     |  size
size       |    .1
-----------+------------
       200 |  0.778
------------------------

Total N of requested  MC replications: 1000
Total N of successful MC replications: 1000
* per sample and effect size combination
```

Note that we also added the `addscalar()` option which can be used to grab a certain scalar from what is stored in `e()` in the wake of a fitted analysis model. For example, in this case we grabbed the R^2 from each replication which is added to the results dataset. We could be interested in how the R^2 can be expected to be distributed under the defined model:

```
. scdensity r2
```

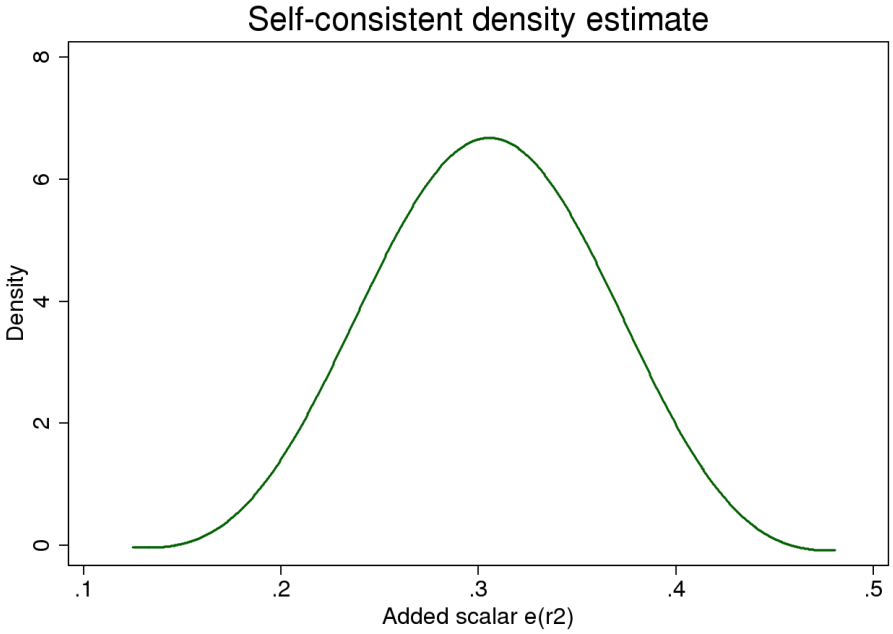

Figure 3: Non-parametrically estimated probability density of R^2 from $1,000$ MC replications (estimated with the `scdensity` command, available from SSC [Luedicke (2012)])

Example 3: Logistic regression model with an interaction effect in a 2x2 balanced block design

What follows is an example where we generate and analyze the data using a binomial model with a logit link function. Covariates are assumed to be fixed, arising from a 2x2 balanced block experimental design. We specify this design by using the `block22()` option. With the fifth argument of that option we specify that the interaction effect will be generated (by leaving out the fifth argument we would omit the interaction effect). Since this is our effect of interest, we call it `_bp`. We could have included a number after the family name with the `family()` option which would indicate the number of trials. Here we use the default of 1, i.e., we specify the Bernoulli model:

$$y \sim Binomial(1, p) \tag{3}$$

where p is

$$Pr(y = 1) = logit^{-1}(0.3x_1 + 0.5x_2 + \_bpx_1x_2) \tag{4}$$

```
. powersim , ///
> b(0(0.4)1.6) ///
> alpha(0.05) ///
> pos(8) ///
> sample(400 800) ///
> nreps(500) ///
> family(binomial) ///
> link(logit) ///
> block22(0.3 x1 0.5 x2 _bp) ///
> silent ///
> dofile(example3_dofile, replace) ///
> : glm y i.x1##i.x2, family(binomial) link(logit)
```

```
Power analysis simulations

Effect sizes b:       0 .4 .8 1.2 1.6
H0:                   b = 0
Sample sizes:         400 800
alpha:                .05
N of replications:    500

do-file used for data generation: example3_dofile
Model command:                    glm y i.x1##i.x2, family(binomial) link(logit
> ) iterate(200)

Power by sample and effect sizes:
```

Sample size	Specified effect size 0	.4	.8	1.2	1.6
400	0.050	0.142	0.440	0.802	0.918
800	0.064	0.302	0.758	0.962	0.994

.

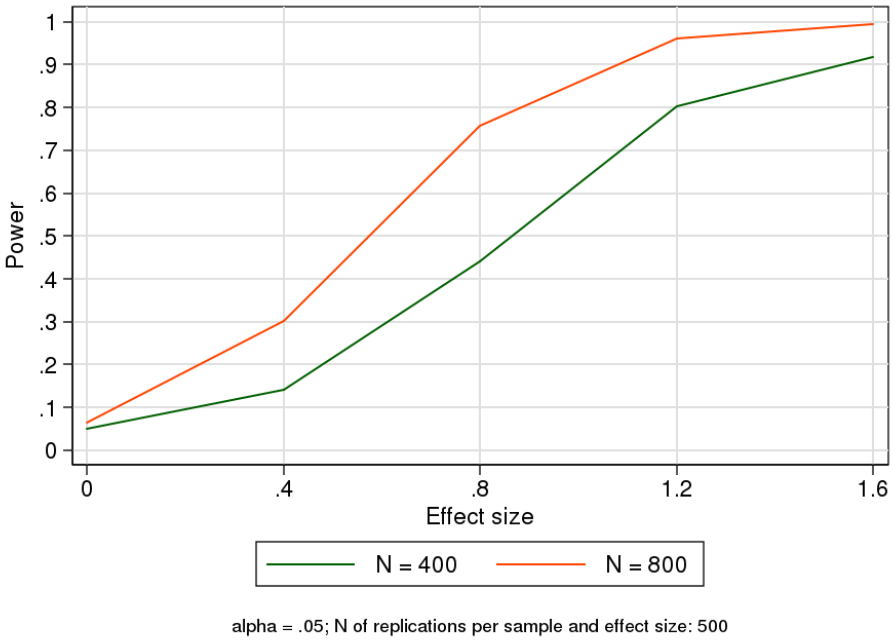

Figure 4: Simulated power as a function of effect size for two sample sizes

```
. powersimplot, esize
```

Now we used `powersimplot` with the `esize` option in order to plot the simulated power as a function of effect size, rather than as a function of sample size. The plot is shown in Figure 4.

Example 4: Extended use – using an existing do-file

Feiveson (2002) uses an example of an experiment with rats in which power is computed for the effect of a drug dose on a Poisson distributed outcome y. The assumed model is:

$$y \sim Poisson(exp(\_bp * x)) \tag{5}$$

where x is a covariate with three levels for dosages 0.2, 0.5, and 1.0 mg. The number of rats per dose level is supposed to be fixed and the hypothesized effect size is $\_bp = 0.64$, to be tested against zero. We first set up `powersim` to generate data with three equally sized groups using the `cov1(... block 3)` specification and request a detailed dry run:

```
. powersim , ///
> b(0.64) ///
> alpha(0.05) ///
> pos(1) ///
> sample(30(30)210) ///
> nreps(500) ///
> family(poisson) ///
> link(log) ///
> cov1(x1 _bp block 3) ///
> seed(1234) ///
> dry det ///
> dofile(example4_dofile, replace) ///
> : glm y x1, family(poisson) link(log)
```

```
powersim dry run:
do-file used for data generation:

. *---------------------------
. // Generating predictor data
.
.
. generate int x1 = cond(mod(_n-1, 3) == 1, 1, cond(mod(_n-1, 3) == 0, 2, 3))
.
. *---------------------------
.
. *----------------------------------------
. // Link function with specified parameters
. *  link = log
.
. generate double xb = exp( _bp*x1 )
.
. *----------------------------------------
.
end of do-file
Data generating model: family(poisson ) link(log)
Analysis model command: glm y x1, family(poisson) link(log) iterate(200)
```

---

y	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
y						

```
          x1 |         .64          .        .       .            .           .
       _cons |    1.60e-12          .        .       .            .           .
------------------------------------------------------------------------------

Matrix e(b) from analysis model (check column position of effect of interest):
e(b)[1,2]
            y:         y:
           x1      _cons
y1        .64  1.599e-12
Position specified in pos() option: 1
(Note: Displayed coefficients are approximate, use only for checking input.)
------------------------------------------------------------------------------
```

We can then open the do-file that we just generated by typing:

```
. doedit example4_dofile.do
```

and make the necessary changes. After saving it we have:

```
. clear
. noisily do example4_dofile
. *----------------------------
. // Generating predictor data
.
.
. generate int x1 = cond(mod(_n-1, 3) == 1, 1, cond(mod(_n-1, 3) == 0, 2, 3))
.
. qui recode x1 1=0.2 2=0.5 3=1
.
. *----------------------------
.
. *----------------------------------------
. // Link function with specified parameters
. *  link = log
.
. generate double xb = exp( _bp*x1 )
.
. *----------------------------------------
.
end of do-file
```

Now we can run the simulations by specifying `using` instead of the `dofile` option:

```
. powersim using example4_dofile , ///
> b(0.64) ///
> alpha(0.05) ///
> pos(1) ///
> sample(30(30)210) ///
> nreps(500) ///
> family(poisson) ///
> seed(1234) ///
> silent ///
> : glm y x1, family(poisson) link(log)
```

```
Power analysis simulations

Effect sizes b:       .64
H0:                   b = 0
Sample sizes:         30 60 90 120 150 180 210
alpha:                .05
N of replications:    500

do-file used for data generation: example4_dofile
Model command:                    glm y x1, family(poisson) link(log) iterate(2
> 00)

Power by sample and effect sizes:
```

Sample size	Specified effect size .64
30	0.284
60	0.532
90	0.670
120	0.842
150	0.880
180	0.936
210	0.968

Example 5: Advanced use – simulating power of a cross-level interaction effect in a linear mixed effects model

By default, only Stata's `regress` and `glm` commands can be used for the specification of an analysis model, but other estimation commands can be used by specifying the `force` option. However, users need to make sure that the estimation command is suitable and that `powersim` properly picks up the results that are stored by the analysis command. For an estimation command to work with `powersim` the following three conditions have to be met:

- The estimation command must store point estimates in `e(b)`.
- It must store the variance estimates in `e(V)` such that when, say, a coefficient of interest is stored in `e(b)[1,3]`, the corresponding variance estimate is stored in `e(V)[3,3]`.
- The variance estimate has to be on the variance scale such that its square root reflects the standard error of the corresponding point estimate.

Also note that, by default, significance tests (and 95% CI coverages) are based on the assumption of a normally distributed test statistic for *any* analysis model other than `regress` (for which Student's t distribution is used). This default behavior can be changed with the `df()` option.

The following example shows how to use `powersim` in combination with Stata's `mixed` command for simulating power of a cross-level interaction effect in a linear multilevel/mixed effects model. Suppose that we assume the following data generating model:

$$y_{ij} = 1 + 0.3x_{1ij} - 0.5x_{2ij} + 0.4x_{3i} + 0.05x_{3i}x_{2ij} + \zeta_{1i} + \zeta_{2i}x_{2ij} + \epsilon_{ij} \qquad (6)$$

where x_{1ij} and x_{2ij} are level-1 predictors with $(x_{1ij}, x_{2ij}) \sim \mathcal{N}(\mu, \Sigma)$,

$$\mu = \begin{pmatrix} 5 \\ 5 \end{pmatrix}, \Sigma = \begin{bmatrix} 1 & \\ 0.5 & 1 \end{bmatrix}$$

and x_{3i} is a level-2 predictor with $x_{3i} \sim Gamma(\alpha = 5, \beta = 2)$. The three error terms correspond to the varying intercept variation $\zeta_{1i} \sim \mathcal{N}(0, 0.5^2)$, the variation of the varying coefficient $\zeta_{2i} \sim \mathcal{N}(0, 0.3^2)$, and the level-1 residual errors $\epsilon_{ij} \sim \mathcal{N}(0, 0.8^2)$. Thus, we have a 2-level model with two correlated level-1 predictor variables and one level-2 predictor variable and are interested in computing power for the cross-level interaction effect between x_{3i} and x_{2ij} while the effect of x_{2ij} on y_{ij} is assumed to vary across level-2 units. Based on this assumed data generating process we can set up a do file with which we define our predictor data as well as our data generating model (except for the level-1 errors which we will specify via `powersim`'s `family()` option):

```
. noisily do example5_dofile
. *---------------------------
. // Generating predictor data
.
. * Level 2 ID
. gen ID2 = _n
.
. * Level 2 errors (uncorrelated)
. gen u = rnormal(0,0.5)
. gen rc = rnormal(0,0.3)
.
. * Level 2 covariate
. gen x3 = rgamma(5, 2)
.
. * N observations per cluster
. expand 10
(270 observations created)
.
. * Level 1 covariates
. matrix __M=(5, 5)
. matrix __SD=(1, 1)
. matrix __C = (1, 0.5, 1)
. drawnorm x1 x2 , ///
> means(__M) sds(__SD) corr(__C) ///
> cstorage(upper) forcepsd double
.
. *---------------------------
.
. *----------------------------------------
. // Link function with specified parameters
. *  link = identity
.
. generate double xb = 1 + 0.3*x1 + -0.5*x2 + 0.4*x3 + _bp*x3*x2 + x2*rc + u
.
. *----------------------------------------
.
end of do-file
```

We can now use this do-file to generate a single dataset and, for instance, fit our analysis model to the fabricated data:

```
. powersim using example5_dofile, ///
> pos(4) ///
> b(0.05) ///
> sample(25 40) ///
> family(gaussian 0.8) ///
> gen nobs(40) seed(123)

.
. mixed y x1 c.x2##c.x3 || ID2 : x2 , nolog
Mixed-effects ML regression                     Number of obs      =       400
Group variable: ID2                             Number of groups   =        40
                                                Obs per group: min =        10
                                                               avg =      10.0
                                                               max =        10
                                                Wald chi2(4)       =    215.50
Log likelihood = -559.75811                     Prob > chi2        =    0.0000
------------------------------------------------------------------------------
           y |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          x1 |   .3445344   .0511637     6.73   0.000     .2442554    .4448134
          x2 |  -.3400779   .1588752    -2.14   0.032    -.6514676   -.0286881
          x3 |   .4239126   .0603022     7.03   0.000     .3057224    .5421028
             |
   c.x2#c.x3 |   .0369216    .014156     2.61   0.009     .0091763    .0646668
             |
       _cons |   .8227497    .669868     1.23   0.219    -.4901674    2.135667
------------------------------------------------------------------------------

------------------------------------------------------------------------------
  Random-effects Parameters |   Estimate   Std. Err.     [95% Conf. Interval]
----------------------------+-------------------------------------------------
ID2: Independent            |
                  var(x2) |   .0584067    .017283      .0327029    .1043133
                var(_cons) |   .4074977   .2997653      .0963741    1.723018
----------------------------+-------------------------------------------------
             var(Residual) |   .6740335   .0516584      .5800225     .783282
------------------------------------------------------------------------------
LR test vs. linear regression:       chi2(2) =   386.04   Prob > chi2 = 0.0000
Note: LR test is conservative and provided only for reference.

.
. * SD of level 1 residuals:
. di exp([lnsig_e]_b[_cons])
.82099544
. * SD of varying intercepts:
. di exp([lns1_1_2]_b[_cons])
.63835543
. * SD of varying coefficients:
. di exp([lns1_1_1]_b[_cons])
.24167486
```

Now we could further inspect the data in order to check whether they are consistent with our hypothesis. For example, we could visualize our cross-level interaction using Stata's `margins` and `marginsplot` commands:

```
. foreach v of varlist x1 x2 x3 {
  2.         sum `v´, meanonly
  3.         gen c_`v´ = `v´ - r(mean)
  4. }
. mixed y c_x1 c.c_x2##c.c_x3 || ID2 : x2 , nolog
Mixed-effects ML regression                     Number of obs      =       400
Group variable: ID2                             Number of groups   =        40
                                                Obs per group: min =        10
                                                               avg =      10.0
                                                               max =        10
                                                Wald chi2(4)       =    215.50
Log likelihood = -559.75811                     Prob > chi2        =    0.0000
```

y	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
c_x1	.3445344	.0511637	6.73	0.000	.2442554	.4448134
c_x2	.0486189	.0641461	0.76	0.448	-.0771051	.174343
c_x3	.6102728	.0542813	11.24	0.000	.5038834	.7166623
c.c_x2#c.c_x3	.0369216	.014156	2.61	0.009	.0091763	.0646668
_cons	7.265125	.2215881	32.79	0.000	6.830821	7.69943

Random-effects Parameters	Estimate	Std. Err.	[95% Conf.	Interval]
ID2: Independent				
var(x2)	.0584067	.017283	.0327029	.1043133
var(_cons)	.4074977	.2997653	.0963741	1.723018
var(Residual)	.6740335	.0516584	.5800225	.783282

```
LR test vs. linear regression:       chi2(2) =   386.04   Prob > chi2 = 0.0000
Note: LR test is conservative and provided only for reference.
.
. margins , dydx(c_x2) at(c_x3=(-8(2)14))
(output omitted)
.
. marginsplot, ylabel(-1(0.25)1, angle(0) grid)
  Variables that uniquely identify margins: c_x3
```

The resulting plot is shown in Figure 5. We can see how the effect of x_{2ij} on y_{ij} is expected to vary as a function of x_{3i}.

Finally, we run our simulations for 25 and 40 level-2 units with 10 level-1 units in each cluster:

```
. powersim using example5_dofile, ///
> b(0.05) ///
> pos(4) ///
> sample(25 40) ///
> nreps(1000) ///
> family(gaussian 0.8) ///
> inside ///
> seed(123) ///
> force : mixed y x1 c.x2##c.x3 || ID2 : x2 , iterate(20)
```

(output omitted)

```
Power analysis simulations

Effect sizes b:       .05
H0:                   b = 0
Sample sizes:         25 40
alpha:                .05
N of replications*:   1000
do-file used for data generation: example5_dofile
Model command:                    mixed y x1 c.x2##c.x3 || ID2 : x2 , iterate(2
> 0)

Power by sample and effect sizes:
```

Sample size	Specified effect size .05
250	0.715
400	0.911

```
Total N of requested  MC replications: 2000
Total N of successful MC replications: 1994
* per sample and effect size combination
```

We can see that we reach a statistical power of around 0.7 ($N = 250$) and 0.9 ($N = 400$) for the test of our cross-level interaction effect. The output also shows that only $1,994$ of the requested $2,000$ Monte Carlo replications were successful. In this case, this is due to six models not converging within the specified limit of 20 iterations.

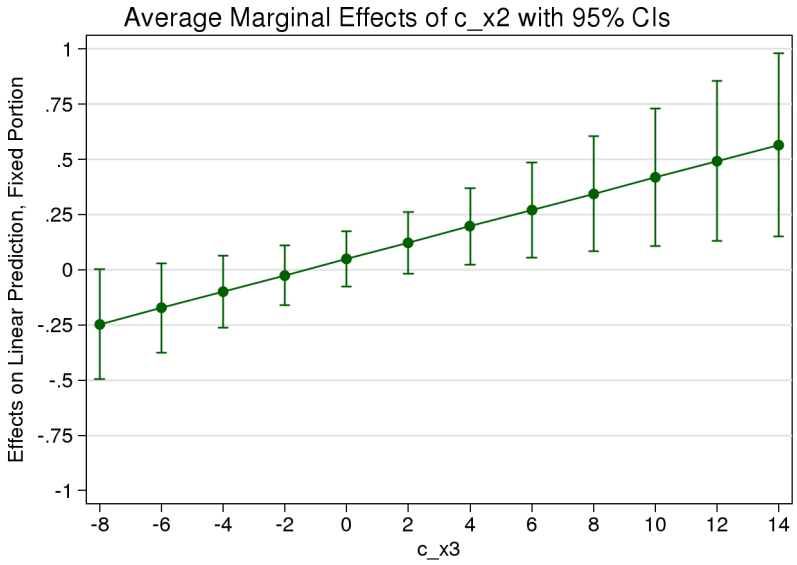

Figure 5: Visualization of the cross-level interaction effect from the linear two-level mixed-effects model

Example 6: Beyond power – using `powersim` for simple Monte Carlo experiments

We can also use `powersim` to perform simple Monte Carlo experiments in a regression modeling context. Suppose that we would like to learn about the performance of conventional standard errors in Poisson regressions as compared to robust standard errors when the data are slightly overdispersed. With the following powersim code we specify a data generating model from the negative binomial family with overdispersion $\alpha = 0.2$ in order to generate overdispersed count data. We then loop over `powersim` twice to fit Poisson models with conventional standard errors the first time, and models with robust standard errors the second time around. We use the `saving()` option to save the results from each loop:

```
. forvalues i = 0/1 {
  2.
.         if `i´ == 1 local robust robust
  3.
.         powersim , ///
>         b(0) ///
>         pos(1) ///
>         sample(500) ///
>         nreps(4000) ///
>         family(nbinomial 0.2) ///
>         link(log) ///
>         cov1(x1 _bp beta 4 2) ///
>         cov2(x2 -0.5 gamma 2 1) ///
>         cons(1) ///
>         inside ///
>         seed(123) ///
>         saving(psim_data`i´, replace) ///
>         dofile(example5_dofile, replace) ///
>         : glm y x1 x2, fam(poisson) link(log) `robust´
  4. }
```

```
(output omitted)
```

```
. gen method = "ROBUST"
. append using psim_data0.dta, nolabel
. quietly replace method = "OIM" if method == ""
```

After appending the data we can now analyze the results for which we could use the user-written Stata modules `simpplot` (Buis (2012)) and `simsum` (White (2010)), for example. Since we generated our data under the true null-hypothesis, we know that the p-values should follow a uniform distribution on $(0, 1)$ if point and variance estimates are unbiased. Since we have the same point estimates in this example, differences between the estimators with respect to their discrepancies from the uniform distribution will be due to the estimated standard errors. With `simpplot` we can nicely plot such discrepencies and Figure 6 shows that our significance test indeed does not perform very well with conventional variance estimation. Note that we would not have to reshape the

Figure 6: Distribution of p-values (shown as deviations from their nominal level) from Poisson models fit to overdispersed count data (`simpplot` example)

data but that way we can overlay the two lines within one graph. Without reshaping we could use `simpplot`'s `by()` option.

```
. keep nd method b se p
. reshape wide b se p, i(nd) j(method) string
(note: j = OIM ROBUST)
(output omitted)
. label variable pOIM "OIM SE"
. label variable pROBUST "robust SE"
. simpplot p*, overall
not enough replications to compute overall bounds; the returned bounds have an
approximate overall error rate of 0.125
```

Finally, we can use `simsum` in order to summarize our simulation results more formally:

```
. simsum b*, se(se*) true(0) format(%6.0g)
Starting to process results ...
```

Statistic	OIM b	ROBUST b
Non-missing point estimates	4000	4000
Non-missing standard errors	4000	4000
Bias in point estimate	.0054	.0054
Empirical standard error	.2616	.2616
% gain in precision relative to method OIM b	.	0
RMS model-based standard error	.2296	.2606
Relative % error in standard error	-12.25	-.3852
Coverage of nominal 95% confidence interval	91.15	94.43
Power of 5% level test	8.85	5.575

References

Buis, M. L. 2012. SIMPPLOT: Stata module creating a plot describing p-values from a simulation by comparing nominal significance levels with the coverages. Statistical Software Components, Boston College Department of Economics. URL `http://ideas.repec.org/c/boc/bocode/s457467.html`.

Feiveson, A. H. 2002. Power by simulation. *Stata Journal* 2(2): 107–124(18). URL `http://www.stata-journal.com/article.html?article=st0010`.

Gelman, A., and J. Hill. 2007. *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge et al.: Cambridge University Press.

Hardin, J., and J. Hilbe. 2012. *Generalized Linear Models and Extensions - Third Edition*. College Station, TX: Stata Press.

Hilbe, J., and W. Linde-Zwirble. 1995. Random number generators. *Stata Technical Bulletin* 28: 20–21. URL `http://www.stata.com/products/stb/journals/stb28.pdf`.

Luedicke, J. 2012. SCDENSITY: Stata module to perform univariate self-consistent density estimation. Statistical Software Components, Boston College Department of Economics. URL `http://ideas.repec.org/c/boc/bocode/s457486.html`.

White, I. 2010. SIMSUM: Stata module to perform analyses of simulation studies including Monte Carlo error. Statistical Software Components, Boston College Department of Economics. URL `http://ideas.repec.org/c/boc/bocode/s457170.html`.