

Research topic:

Sample size calculations in complex random effects models

(“Using simulation in parallel computing for faster sample size calculations in complex random effects models”)

Toni Price, University of Bristol

MLPowSim – recap

- Developed in a separate ESRC-funded project
- Generates both MLwiN macro code and R language code for performing sample size calculations on multilevel models
 - Text-based interface
 - Uses C code to gather user input and generate output
- Works for a selection of multilevel nested and crossed designs (balanced and unbalanced)
- Slow to run but very flexible

C:\ powerupdate.exe

MLPowSim version 1.0 Beta 1

coded by William J. Browne and Mousa Golalizadeh (c) March 2009

MLPowSim is free software and comes with absolutely NO WARRANTY
MLPowSim produces output files that can be used by the R or MLwiN
packages.

We make no guarantees that the files produced are in any sense correct
or will run in these packages.

The further use of any files generated by MLPowSim is the responsibility
of the user for whatever purposes they may be used.

To continue using this program having read and understood this
disclaimer please input 1 : 1

Welcome to MLPowSim

Please input 0 to generate R code and 1 to generate MLwiN macros: 1

Please choose model type

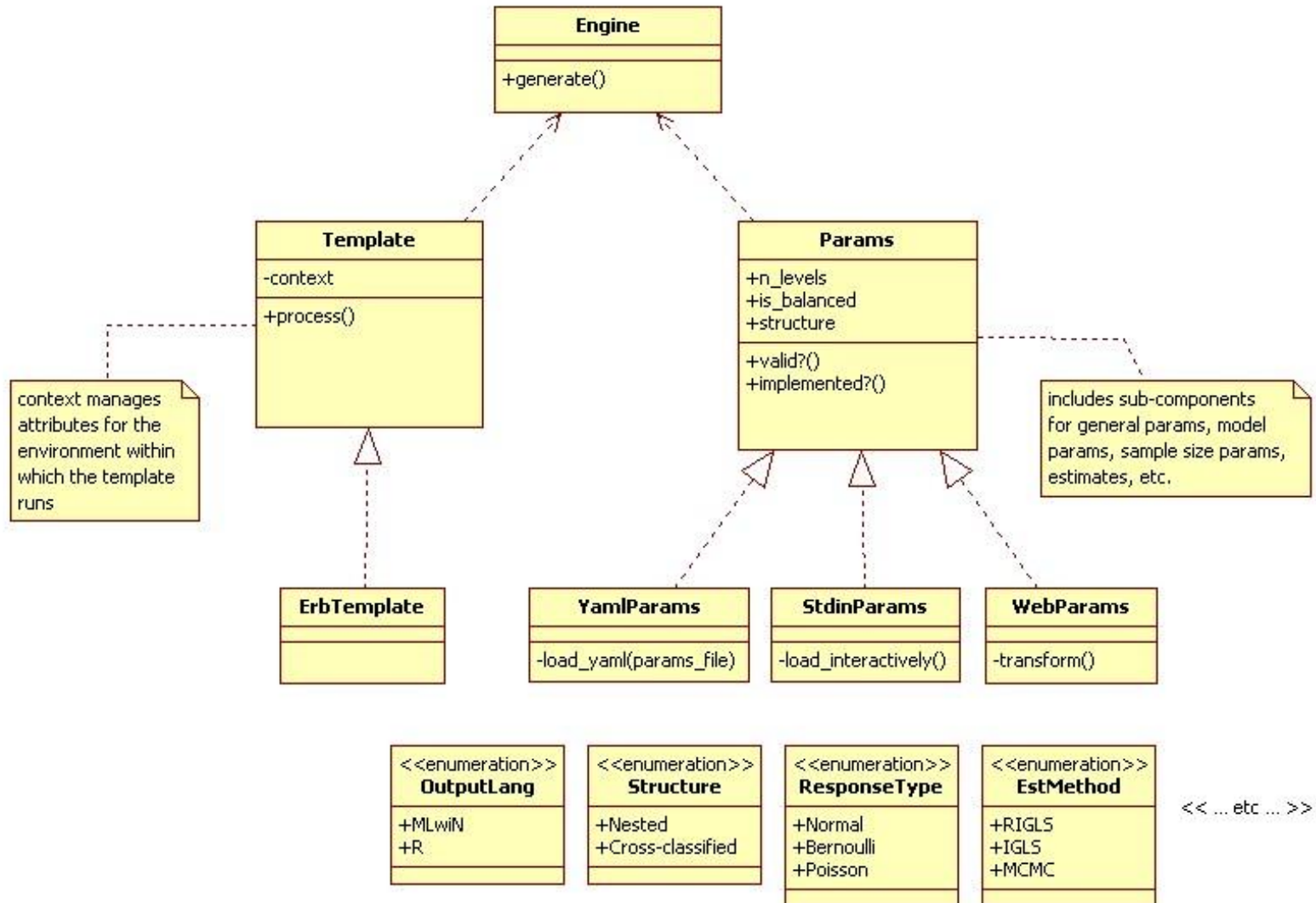
1. 1-level model
2. 2-level balanced data nested model
3. 2-level unbalanced data nested model
4. 3-level balanced data nested model
5. 3-level unbalanced data nested model
6. 3-classification balanced cross-classified model
7. 3-classification unbalanced cross-classified model

Model type : _

First step: started putting into a cohesive framework to:

- Streamline duplicated code (e.g. for user input which is similar across different models)
 - Improves code maintenance (e.g. bug fixes impacting fewer lines of code)
- Improve input validation
 - Makes for a better user experience and reduces crashes
- Automate testing of generated code and results
- Add multiple user interfaces, e.g. command line/file input/web-based

MLPowSim – UML (partial) Class Diagram



Running MLPowSim on the command line

```
toni@goji:~/workspace/svn/dev/phd/mlpowsim/codegen$ ./src/mlpowsim.rb -h
[INFO] -> Running mlpowsim.rb
Run 'mlpowsim.rb --help' for help with command line options.
[INFO]
Usage: mlpowsim.rb [options]

Generates code for calculating sample sizes in MLwiN or the R language.

-l, --log-level STRING      Logging level
                             (default: info)
-f STRING                  Path to parameter input file
                             (if not supplied, parameters will be input from the command line)
-y, --yaml-only            Load yaml file and exit
                             (requires option -f as well; irrelevant options will be ignored)
-p, --file STRING          Path to default parameter file *** Note: only used for command line input
                             (if not supplied, will be set to /home/toni/workspace/
                             svn/dev/phd/mlpowsim/codegen/test/fixtures/eg008/mlwin/params.yaml)
-o, --output STRING        Output location
                             (default: /home/toni/workspace/svn/dev/phd/mlpowsim/codegen/output)
-c, --confirm-disclaimer  Suppress interactive disclaimer confirmation
-v, --version              Display version info
-h, --help                 Display this help message
toni@goji:~/workspace/svn/dev/phd/mlpowsim/codegen$
```

File input – Example for a 2-level model

```
# Input params
#
# Example p. 39 in MLPowSim user manual
# MLwiN code output

# See http://www.yaml.org/YAML\_for\_ruby.html

general:
  output_lang:          mlwin

  rnd_num_seed:         1
  sig_level:            0.025
  n_sims:               1000

model:
  n_levels:             2
  is_balanced:          yes
  structure:            nested #=> nested | cross-classified
  response_type:       normal
  est_method:          igls

  include_fixed_intercept: yes

  include_random_intercept: yes

  n_explanatory_vars:   0

  estimates:
    beta_0:             -0.177
    sigma_sq_u:         0.151
    sigma_sq_e:         0.916

sample_size:
  level_2:
    low:                10
    hi:                 50
    step:               10
  level_1:
    low:                10
    hi:                 60
    step:               10
```

```

toni@goji:~/workspace/svn/dev/phd/mlpowsim/codegen$ ./src/mlpowsim.rb -c -f test/fixtures/eg008/mlwin/params.yaml
[INFO] -> Running mlpowsim.rb
Run 'mlpowsim.rb --help' for help with command line options.
[INFO]
[INFO] User-supplied params file: 'test/fixtures/eg008/mlwin/params.yaml'
.
<snip>
.
[INFO] =====
[INFO] Params:
[INFO] -----
[INFO] General:
[INFO]   Output language:          MLwiN
[INFO]   Random no. seed:          1
[INFO]   Significance level:        0.025000
[INFO]   No. of simulations per setting: 1000
[INFO] Model:
[INFO]   No. of levels:            1
[INFO]   Balanced?                  Yes
[INFO]   Structure:                  Nested
[INFO]   Response type:              Normal
[INFO]   Estimation method:          IGLS
[INFO]   Include fixed intercept?    Yes
[INFO]   No. of explanatory vars:    0
[INFO] Estimates:
[INFO]   beta_0:                     -0.140000
[INFO]   sigma_sq_e:                 1.051000
[INFO] Sample size:
[INFO]   Level_1:
[INFO]     Lower value:              20
[INFO]     Upper value:              600
[INFO]     Step:                      20
[INFO] =====
toni@goji:~/workspace/svn/dev/phd/mlpowsim/codegen$

```


Running MLPowSim via Web interface

Sample size calculations for Multilevel Models - Mozilla Firefox

http://goji:3000/inputs/new

Sample size calculations for Multilev...

MLPowSim

a code generator for multilevel model
sample size calculations
using simulation

Centre for Multilevel Modelling
University of BRISTOL

Input parameters

Output language: MLwIN

Number of levels: 2

Balanced Unbalanced

Random number seed: 1

Significance level: 0.025

Number of simulations per setting: 1000

Sample Size

Level 2

Lower value: 10

Upper value: 50

Step size: 10

Level 1 (per Level 2 setting)

Lower value: 10

Upper value: 60

Step size: 10

Sample size calculations for Multilevel Models - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://goji:3000/inputs/new

Most Visited Getting Started Latest Headlines Google

Sample size calculations for Multilevel...

Sample Size

Level 2

Lower value:

Upper value:

Step size:

Level 1 (per Level 2 setting)

Lower value:

Upper value:

Step size:

Model setup

Response type:

Estimation method:

Include fixed intercept?

Include random intercept?


Number of explanatory vars:

Estimates

Beta 0:

Sigma squared u:

Sigma squared e:

Funded by 

[Centre for Multilevel Modelling \(CMM\)](#) | [MLPowSim & MCMC methods project](#)

Done

Sample size calculations for Multilevel Models - Mozilla Firefox

File Edit View History Bookmarks Tools Help



http://goji:3000/inputs

Most Visited Getting Started Latest Headlines Google

Sample size calculations for Multilev...

MLPowSim

a code generator for multilevel model
sample size calculations
using simulation

Invalid input

Input parameters

Output language:

Number of levels:

Balanced Unbalanced

Random number seed:

Significance level:

- Significance level must be between 0 and 1

Number of simulations per setting:

- No of simulations must be greater than or equal to 1

Sample Size

Level 2

Lower value:

Upper value:

Step size:

Level 1 (per Level 2 setting)

Lower value:

Upper value:

Step size:

Done

Sample size calculations for Multilevel Models - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://goji:3000/inputs/7feecead32b0/generated_code

Most Visited Getting Started Latest Headlines Google

MLPowSim

a code generator for multilevel model
sample size calculations
using simulation

Centre for Multilevel Modelling University of BRISTOL

Start again >

Generated Code

Download

File name	Generated code
simu.txt	<p>NOTE MLwiN macro code generated by MLPowSim on Mon 04 Apr 2011 16:07:35 BST NOTE This is outer code to be run directly in MLwiN NOTE You will also need simu2.txt, setup.txt and analyse.txt</p> <pre>SEED 1 ERASE C594-C598 NOTE set up the values of beta, sigma2u, sigma2e etc. JOIN C598 -0.177000 C598 JOIN C596 0.151000 C596 JOIN C596 0.916000 C596 NAME c209 'N-level 1' NAME c210 'N-level 2' NAME c211 'zpow0' c231 'spow0' NAME c251 'zlow0' c291 'slow0' NAME c271 'zupp0' c311 'supp0' CALC b41 = 1000 LOOP b22 10 50 10 OBEY simu2.txt ENDL</pre>
simu2.txt	<p>NOTE MLwiN macro code generated by MLPowSim on Mon 04 Apr 2011 16:07:35 BST NOTE This code simply covers second level of looping!</p> <pre>LOOP b21 10 60 10 OBEY setup.txt ENDL</pre>

Done

- This gives an indication of how MLPowsim generates code, but ...

What does MLPowSim do?

- Takes a model specification
- Steps through a number of different sample size settings
- For each sample size setting:
 - Simulates a (user-specified) number of datasets according to model specification
 - For each simulated dataset, estimates model parameters and their standard errors
 - Computes the estimated power according to two different methods:
 - “Zero/One” method
 - “Standard error” method (suggested by Joop Hox, 2007)

Zero/One method

For a parameter θ of interest:

- Compute (Gaussian) confidence interval for parameter:

$$CI(\hat{\theta}_i) = \hat{\theta}_i \pm z_{1-\alpha/2} \cdot se(\hat{\theta}_i), i=1 \dots N$$

where N = no. of simulations

- Compute 'zero/one' value:

$$zero_one_i = \begin{cases} 0 & CI(\hat{\theta}_i) \text{ contains } 0 \\ 1 & \text{otherwise} \end{cases}$$

- Take average of zero/one values over simulations for sample size setting:

$$\widehat{power}(z) = \frac{1}{N} \sum_{i=1}^N zero_one_i$$

Zero/One method – contd.

Then compute confidence interval for power:

$$\begin{aligned} CI(\widehat{power}(z)) \\ = \widehat{power}(z) \pm z_{1-\alpha/2} \sqrt{\frac{\widehat{power}(z)(1-\widehat{power}(z))}{N}} \end{aligned}$$

Standard error method

For each parameter of interest:

- Take average of estimated standard errors over simulations:

$$\widehat{se}(\theta) = \frac{1}{N} \sum_{i=1}^N se(\widehat{\theta}_i) \text{ where } \theta = \text{'true' effect size}$$

- Use approximation:

$$\frac{\gamma}{se(\gamma)} \approx z_{1-\alpha} + z_{1-\beta}$$

where γ is the effect size

- this formula is for a one-sided t-test for γ with reasonably large d.f. (say, $d.f. \geq 10$) [Snijders and Bosker (1999) p. 142]

Standard error method – contd.

- Solve for power $(1 - \beta)$:

$$\widehat{power}_{(s)} = \Phi \left(\frac{\theta}{\widehat{se}(\theta)} - z_{1-\alpha/2} \right)$$

where $\Phi(\cdot)$ is the Standard Normal distribution function

Standard error method – contd.

Then compute confidence interval for power:

$$\begin{aligned} & \text{LowerCI}(\widehat{\text{power}}(s)) \\ &= \Phi \left(\frac{\theta}{\widehat{\text{se}}(\theta) + z_{1-\alpha/2} \cdot \sqrt{\frac{\text{var}(\widehat{\text{se}}(\theta_i))}{N}}} - z_{1-\alpha/2} \right) \end{aligned}$$

$$\begin{aligned} & \text{UpperCI}(\widehat{\text{power}}(s)) \\ &= \Phi \left(\frac{\theta}{\widehat{\text{se}}(\theta) - z_{1-\alpha/2} \cdot \sqrt{\frac{\text{var}(\widehat{\text{se}}(\theta_i))}{N}}} - z_{1-\alpha/2} \right) \end{aligned}$$

Finally:

- For each sample size setting, tabulate computed power values
- Read off required sample sizes for desired level of power!

Improving speed

- Objective: to speed up run-time for generated power calculation code
- Previously started taking a look at using capabilities of multi-core processors and parallelization to execute more than one run simultaneously
- Exploratory code made use of Unix (Linux) ‘forking’ to create sub-processes
- No more done on this yet, but would like to try using snow (“Simple Network of Workstations”) which provides support for parallel computing in R
<http://cran.r-project.org/web/packages/snow/>
 - Would have the advantage of doing everything in R (in cases where power calculations are being run in R that is)

Cross-Classified Model – Example

- Unbalanced design, sampling from a pupil lookup table
- Data from Fife in Scotland
- Response: Exam results at age 16 of 3435 school children (cross-classification of primary/secondary schools)
- 19 secondary schools & 148 primary schools

Cross-Classified Model – Example contd.

- Model (cross-classified variance components):

$$y_i = \beta_0 + u_{sec_school(i)}^{(2)} + u_{pri_school(i)}^{(3)} + e_i$$

$$u_{sec_school(i)}^{(2)} \sim N(0, \sigma_{u(2)}^2)$$

$$u_{pri_school(i)}^{(3)} \sim N(0, \sigma_{u(3)}^2)$$

$$e_i \sim N(0, \sigma_e^2)$$

- Implemented in MLPowSim in R (since MCMC option in MLwiN was slow)

```
y ~ 1 + (1 | level_1id) + (1 | level_2id)
```

Cross-Classified Model – contd.

- (From MLPowSim manual) Estimated power values increase very quickly for small sample sizes but tend to plateau at roughly 0.8 after around 3,000 pupils
- Potential enhancement:
 - Investigate further alternatives to drawing individuals from a 2 way table – e.g. when generating the data, set a probability of a family moving into an area (in which case their child's primary school may be different from most of the other children's primary school for the relevant secondary school)

Cross-Classified Model – contd.

- After implementing unit tests to check expected power calculation output, ran this example on both Linux and Windows
- Got different results so have been looking at possible causes for the differences
- After quite a lot of digging into detail of the example, it seems this might be due to computational differences in R on 64-bit and 32-bit platforms
- From r-devel Mailing List:
(<https://stat.ethz.ch/mailman/listinfo/r-devel>)
[Amongst other platform-specific differences, an important difference is]
“... the number of registers available on the CPU, which differs between i386 and x86_64. Hence computations get done in different orders by optimizing compilers.”
– Prof Brian Ripley (Feb 10, 2011 03:12pm)

[Demo of current MLPowSim
incl. automated tests for verifying actual
numerical output]

What next?

- Extend cross-classified unbalanced data model (and look more closely at convergence)
- Add support for more models
- Continue investigating speed improvements through parallelization
- Look at gaining speed improvements through use of an algorithm to 'focus in' on required power values rather than running a complete series of grid-like settings