Supplementary Tutorial for "Power analysis for generalized linear mixed models in ecology and evolution" using R

PCD Johnson, SJE Barry, HM Ferguson & P Müller
Correspondence: paul.johnson@glasgow.ac.uk

Introduction

The aim of this document is to complement the article "Power analysis for generalized linear mixed models in ecology and evolution" with an R tutorial based on four examples:

1. Power analysis for a two-sample t-test: a simple example intended to bridge the gap between conventional power analysis and the more advanced examples presented in the article which use simulations to do power analysis for generalized linear mixed models (GLMMs). Power is estimated by three methods:
    1.1. Using simulations directly.
    1.2. Using simulations in the `sim.glmm` function available at `https://github.com/pcdjohnson/sim.glmm`.
    1.3. Using the `power.t.test` function.
2. The first power analysis example in the article: predicting precision in estimating tick burden using a Poisson GLMM.
    2.1. Using Wald Z confidence intervals.
    2.2. Using parametric bootstrap confidence intervals.
3. The second power analysis example from the article: predicting power for a binomial GLMM comparing mortality in malaria mosquitoes between different types of insecticidal nets.
4. Not a power analysis, but an example illustrating a capability of `sim.glmm` not covered in the article: simulating from a random intercepts-and-slopes GLMM.

This is a "how to" document, and to avoid repetition it does not cover the the background to simulation-based power analysis, which is briefly covered in the introduction to the main article. A more comprehensive guide to using simulations for power analysis and more can be found in Chapter 5 of *Ecological Models and Data in R* by Ben Bolker.

Examples

1 Power analysis for a two-sample t-test

1.1 Power analysis for a two-sample t-test using simulations

In this example we are planning an experiment to compare mean swimming speed, y, between young and old larval snails. We will use a two-sample t-test to choose between two hypotheses, the null hypothesis (H_0) that there is no difference between the two means, and the alternative hypothesis (H_1), that there is a difference. In order to estimate power for a two-sample t-test, we need to consider five quantities:

- n: the sample size *in each group*. Setting the sample size to be the same in each group is an efficient design choice because equal group sizes maximise power for a given total sample size.
- δ: the smallest difference between the group means that we would like to be able to detect under H_1.
- σ: the within-group standard deviation, which is assumed to be the same in each group.
- α: the significance level or type I error probability, which is the probability of a significant test result where H_0 is true (the two groups have the same mean). Throughout this tutorial we set $\alpha = 5\%$.
- Power: the probability of a significant test result where H_0 is false and the two groups differ by δ. Power can be defined in terms of β, the type II error probability: β is the probability of a *non*-significant test result where the two groups differ by δ, hence power $= 1 - \beta$. In this tutorial we will aim for power $\geq 80\%$.

We make the following assumptions:

- Swimming speed is normally and independently distributed with a standard deviation of $\sigma = 0.5$ mm/s within each group, and means of 2 mm/s in young larvae and $2 + \delta$ mm/s in old larvae.
- The experiment should be sufficiently sensitive to detect a difference of at least $\delta = 0.2$ mm/s between young and old larvae.
- We plan to study $n = 30$ larvae in each group.
- The significance level $\alpha = 5\%$.

In a real power analysis these assumptions would be based on existing data (for the assumptions of σ and normality), judgement (for δ, α and target power) and an assessment of available resources (for n). We will use simulations to ask if this design will give us adequate ($\geq 80\%$) power.

Simulation-based power analysis follows three basic steps:

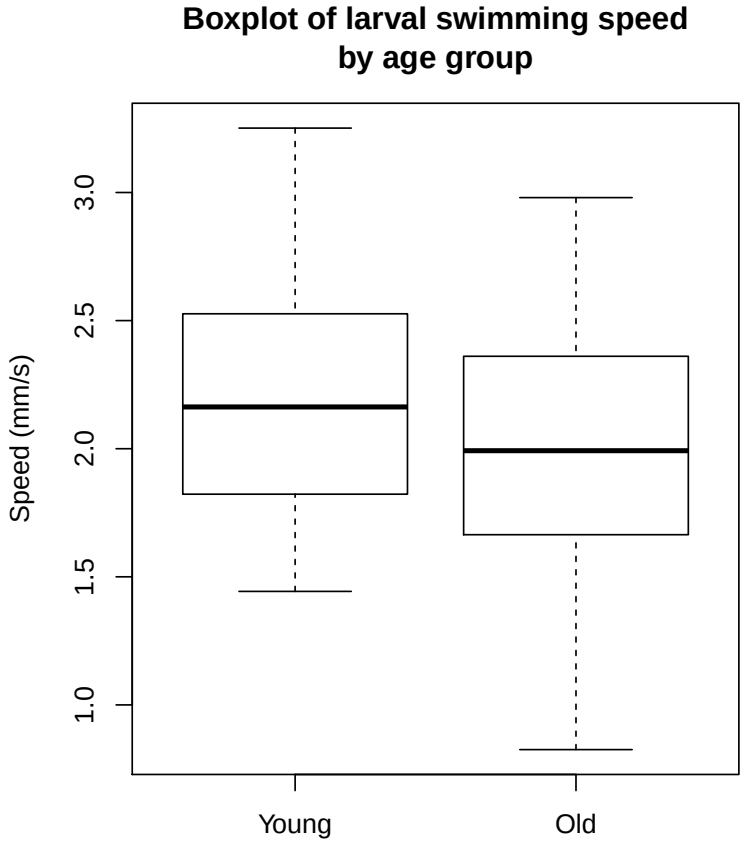
- Simulate data under the alternative hypothesis
- Test the null hypothesis
- Estimate power by averaging over multiple repeats of the first two steps

The assumptions above provide all the information necesary to simulate the data. After setting a seed for the random number generator,[1] simulate the responses, y, by using the function `rnorm` to simulate $n = 30$ normally distributed observations in each group, with means $\mu_{young} = 2$ in the young group and $\mu_{old} = \mu_{young} + \delta = 2.2$ in the treatment group, and $\sigma = 0.5$.

```
> set.seed(123456) # set seed for random number generator to give repeatable results
> y.young <- rnorm(n = 30, mean = 2, sd = 0.5)
> y.old <- rnorm(n = 30, mean = 2 + 0.2, sd = 0.5)
```

Plot the simulated data:

```
> boxplot(list(Young = y.young, Old = y.old), ylab = "Speed (mm/s)",
+         main = "Boxplot of larval swimming speed\nby age group")
```



Plotting the simulated data shows no evidence that old larvae swim faster than young ones; in fact, in this sample the young larvae are faster on average. We test the null hypothesis using a two-sample t-test, appending `$p.value` to return the P-value (see `?t.test` for details):

```
> t.test(x = y.young, y = y.old, alternative = "two.sided", paired=FALSE,
+        var.equal = TRUE)$p.value
```

```
[1] 0.2500745
```

[1] This will be done periodically throughout this tutorial to ensure that readers who run the code exactly as it is presented can reproduce the same results if desired. Unless you require exactly reproducible results, there is no need to set a seed in your own power analysis.

$P > 0.05$, so we do not reject the null hypothesis in this instance. However, we cannot draw reliable conclusions about power frow a single simulated data set. We must test many simulated data sets and estimate power as the proportion where the null hypothesis is rejected.

First, to make it easier to repeat the simulation process, we put it inside a function:

```
> sim.t.test <-
+   function(n, delta, sigma) {
+     y.young <- rnorm(n = n, mean = 0, sd = sigma)
+     y.old <- rnorm(n = n, mean = delta, sd = sigma)
+     t.test(x = y.young, y = y.old, alternative = "two.sided", paired = FALSE,
+            var.equal = TRUE)$p.value
+     }
```

Note that the mean in the young group has been set to zero rather than 2. We could have set it to any value without affecting the results, because the power of a t-test depends on the *difference* between the group means, not the group means themselves.

```
> sim.t.test(n = 30, delta = 0.2, sigma = 0.5)
```

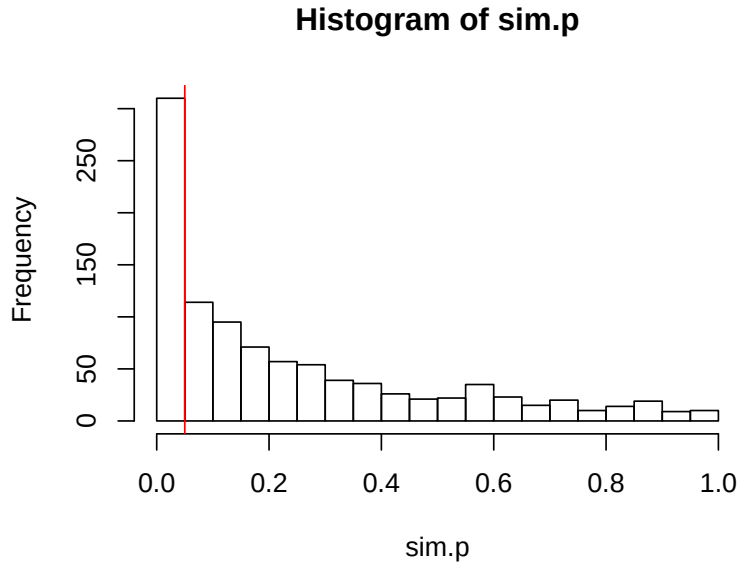
```
[1] 0.002214583
```

This time $P < 0.05$, so the null hypothesis is rejected. Use `sapply` to run the simulation function `nsim` $= 1000$ times:

```
> nsim <- 1000
> sim.p <-
+   sapply(1:nsim, function(...) sim.t.test(n = 30, delta = 0.2, sigma = 0.5))
```

`sim.p` is a sample of size `nsim` $= 1000$ from the distribution of P-values when $\delta = 0.2$. We can show with a histogram that although the P-values show a tendency towards being small, the power estimate, which is the proportion below 0.05 (to the left of the red line), is much less than the target value of 80%:

```
> hist(sim.p, nclass = 20)
> abline(v = 0.05, col = "red")
```



Power is estimated as the number of significant test results divided by the total number of tests, `nsim`. Entering the code `sim.p < 0.05` returns a logical vector that is `TRUE` when P < 0.05 and `FALSE` otherwise. Conveniently, R allows us to perform arithmetic operations on logical vectors, treating `TRUE` as 1 and `FALSE` as 0, so the power estimate can be calculated as:

```
> sum(sim.p < 0.05) / nsim
```

```
[1] 0.31
```

or equivalently

```
> mean(sim.p < 0.05)
```

```
[1] 0.31
```

The power estimate is 31%, which is too low; there only a 1 in 3 chance of detecting the specified treatment effect. There is a little sampling error in this estimate (95% confidence interval: 28-34%), which we could reduce by increasing `nsim`, but the conclusion would be the same: we need a larger sample. How large? We can find out by repeating the simulations with higher n until we achieve 80% power.

Create a vector of increasing n from 30 to 120 in steps of 5:

```
> n.vec <- seq(30, 120, by = 5)
> n.vec
```

```
 [1]  30  35  40  45  50  55  60  65  70  75  80  85  90  95 100 105 110 115 120
```

Use `sapply` to loop the power simulation code over each n:

```
> sim.power.over.n <-
+   sapply(n.vec, function(n.new) {
+     sim.p <-
+       sapply(1:nsim,
+         function(...) sim.t.test(n = n.new, delta = 0.2, sigma = 0.5))
+     mean(sim.p < 0.05)
+     })
```

Show the power estimates for each n:

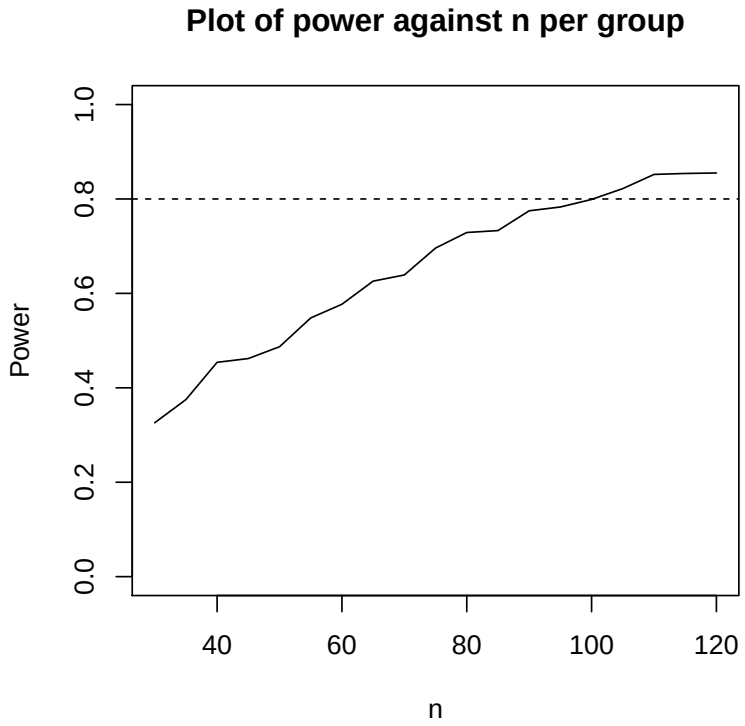
```
> names(sim.power.over.n) <- paste("n =", n.vec)
> sim.power.over.n
```

```
 n = 30  n = 35  n = 40  n = 45  n = 50  n = 55  n = 60  n = 65  n = 70  n = 75
  0.326   0.375   0.454   0.462   0.487   0.548   0.577   0.626   0.639   0.696
 n = 80  n = 85  n = 90  n = 95 n = 100 n = 105 n = 110 n = 115 n = 120
  0.729   0.733   0.775   0.783   0.799   0.822   0.852   0.854   0.855
```

Power is closest to 80% where $n = 100$, i.e. 100 larvae will be required in each group to give 80% power. The power estimate of 79.9% at $n = 100$ is very close to the target value of 80%. We can gauge the precision of the power estimate by calculating a 95% confidence interval (CI). For 799 significant results out of 1000 simulated data sets, the 95% CI is 77-82% (you can check this using the `binom.test` function). This is adeqate precision, particularly considering that almost all power analyses ignore more important sources of uncertainty in the values assumed for influential parameters such as σ. However, more precise estimates of power could easily achieved, and used to refine the sample size estimate, by increasing `nsim`.

Plot a power curve illustrating the relationship between power and n:

```
> plot(n.vec, sim.power.over.n, ylab = "Power", xlab = "n", ylim = 0:1,
+         main = "Plot of power against n per group", type="l")
> abline(h = 0.8, lty = 2)
```



The power curve is a little noisy and could be smoothed by increasing `nsim`. However even with only 1000 simulations we can see that the power curve crosses the dashed line indicating power = 80% at $n \approx 100$

1.2 Power analysis for a two-sample t-test using simulations with `sim.glmm`

The model underlying a t-test is a very simple example of a GLMM, with a single binary explanatory variable, Gaussian (normally distributed) responses and no random effects. The snail larvae t-test example can therefore provide a simple introduction to simulating from a GLMM using the `sim.glmm` function, which can be downloaded from `https://github.com/pcdjohnson/sim.glmm`.[2]

To simulate data for a t-test, the `sim.glmm` function requires four inputs:

- `design.data`: the template data set, which is a data frame with the predictor variables present but the responses missing. This data frame defines the design of the study. For the snail larvae example it will have $2 \times n$ rows, i.e. n rows per group, and one column categorizing larvae as young or old.
- `fixed.eff`: the fixed effects, which for the snail example are the intercept, which is the mean speed of young larvae (2 mm/s), and the differences between the intercept and the two group means (0 and 0.2 mm/s,respectively).

[2]Since `lme4` version 1.0, the `simulate` function, which has a method for `merMod` model fits, has had the capacity to simulate flexibly from GLMMs via the standard R formula syntax. See `?simulate.merMod` and `http://rpubs.com/bbolker/11703` for details.

- `distribution`: the distribution of the responses, which for a t-test is Gaussian.
- `SD`: the residual standard deviation, which must be supplied if the distribution is Gaussian.

For detailed instructions on how to use `sim.glmm`, see the header of the sim.glmm.R script file at `https://github.com/pcdjohnson/sim.glmm`.

Create the template data frame:

```
> snaildata <-
+   data.frame(group = factor(rep(c("Young", "Old"), each=100),
+     levels= c("Young", "Old")))
> snaildata
```

```
    group
1   Young
2   Young
3   Young
.       .
.       .
200   Old
```

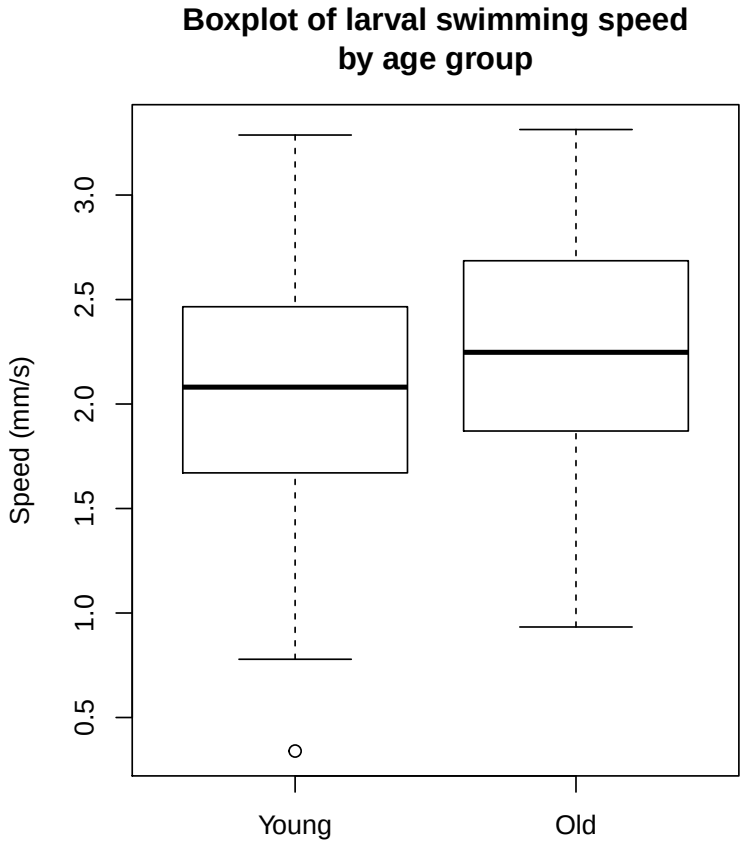
Simulate the data:

```
> snaildata <-
+   sim.glmm(design.data = snaildata,
+     fixed.eff = list(intercept = 2, group = c(Young = 0, Old = 0.2)),
+     distribution = "gaussian", SD = 0.5)
> snaildata
```

```
    group response
1   Young 3.247005
2   Young 2.524338
3   Young 1.824097
.       .        .
.       .        .
200   Old 2.850735
```

As above, inspect the simulated data on a boxplot...

```
> boxplot(response ~ group, data = snaildata, ylab = "Speed (mm/s)",
+   main = "Boxplot of larval swimming speed\nby age group")
```



...and do the t-test:

```
> t.test(response ~ group, data = snaildata, var.equal = TRUE)$p.value
```

```
[1] 0.00350535
```

This time P < 0.05 so we reject the null hypothesis. As before, one instance is not enough so we run the simulation `nsim` $= 1000$ times:

```
> sim.p <-
+   sapply(1:nsim, function(...) {
+     snaildata <-
+       sim.glmm(design.data = snaildata,
+         fixed.eff = list(intercept = 2, group = c(Young = 0, Old = 0.2)),
+         distribution = "gaussian", SD = 0.5)
+     t.test(response ~ group, data = snaildata, var.equal = TRUE)$p.value
+     })
> mean(sim.p < 0.05)
```

```
[1] 0.811
```

The power estimate of 81.1% is in close agreement with the estimate of 79.9% calculated in the previous section.

1.3 Analytical power analysis for a two-sample t-test using `power.t.test`

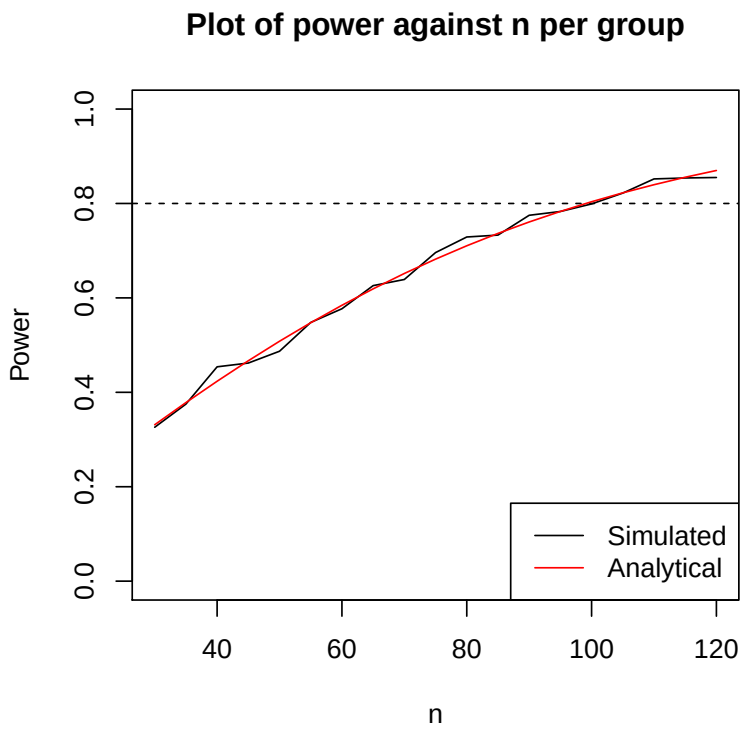
We could also have found the required sample size per group using the standard formula implemented in `power.t.test`:

```
> power.t.test(delta = 0.2, sd = 0.5, power = 0.8, sig.level = 0.05)

     Two-sample t test power calculation

              n = 99.08057
          delta = 0.2
             sd = 0.5
      sig.level = 0.05
          power = 0.8
    alternative = two.sided

NOTE: n is number in *each* group
```

`power.t.test` agrees with the simulation results that we need $n = 100$ larvae per group (sample size is always rounded up to ensure that power is always at least as high as the target value).

Illustrate the agreement between the analytical and simulation-based methods by adding the analytical power estimates to the power curve:

```
> Power <- power.t.test(n = n.vec, delta = 0.2, sd = 0.5, sig.level = 0.05)$power
> plot(n.vec, sim.power.over.n, ylab = "Power", xlab = "n", ylim = 0:1,
+         main = "Plot of power against n per group", type="l")
> abline(h = 0.8, lty = 2)
> lines(n.vec, Power, col = "red")
> legend("bottomright", legend = c("Simulated", "Analytical"), lty = 1, col = 1:2)
```



Clearly, there is no advantage to be gained from using simulations for scenarios such as this one for which simple and fast power analysis methods are available and implemented in most statistical software. The advantage of the simulation approach is that it can be easily extended to more complex scenarios where formulae and software are not available, as illustrated by the next two examples.

2 Predicting precision in estimating tick burden using a Poisson GLMM

2.1 Using Wald Z confidence intervals

The main article gives the background to this example, which illustrates the steps required to predict the precision of an estimate from a Poisson GLMM. Briefly, the aim of the power analysis is to determine the amount of sampling effort required (i.e. the number of locations sampled) to estimate the mean tick burden on grouse chicks on a grouse moor to a specified level of precision, while accounting for correlations between chicks within broods and between broods within locations. The level of precision required is that the expected confidence limits should be ±25% of the estimated mean tick burden, which we refer to as a margin of error of 25%.

The same basic steps are used as in the t-test example, adapted to the aim of estimating the mean margin of error (which we define as half the width of the 95% CI[3]):

- Simulate tick burden data
- Calculate the margin of error
- Average results across multiple repeats of the first two steps

Simulate a template data set that defines sampling of chicks within broods within locations, assuming that there are 3 chicks per brood, 2 broods per location, and that we will sample 10 locations.

```
> tickdata <- expand.grid(chick = 1:3, brood = 1:2, location = 1:10)
> tickdata
```

```
    chick brood location
1       1     1        1
2       2     1        1
3       3     1        1
.       .     .        .
.       .     .        .
60      3     2       10
```

Make brood and chick labels unique, otherwise random effects will be simulated as crossed, not nested.

```
> tickdata$location <- factor(paste("loc", tickdata$location, sep = ""))
> tickdata$brood <- factor(paste(tickdata$location, "brd", tickdata$brood, sep = ""))
> tickdata$chick <- factor(paste(tickdata$brood, "chk", tickdata$chick, sep = ""))
> tickdata
```

```
             chick     brood location
1    loc1brd1chk1  loc1brd1     loc1
2    loc1brd1chk2  loc1brd1     loc1
3    loc1brd1chk3  loc1brd1     loc1
.               .         .        .
```

[3] A 95% CI for the mean of a Poisson distribution is not symmetrical, so the confidence limits cannot be ±25% of the estimate. This definition gives limits which are ±25% on average.

```
.                .          .         .
60  loc10brd2chk3 loc10brd2    loc10
```

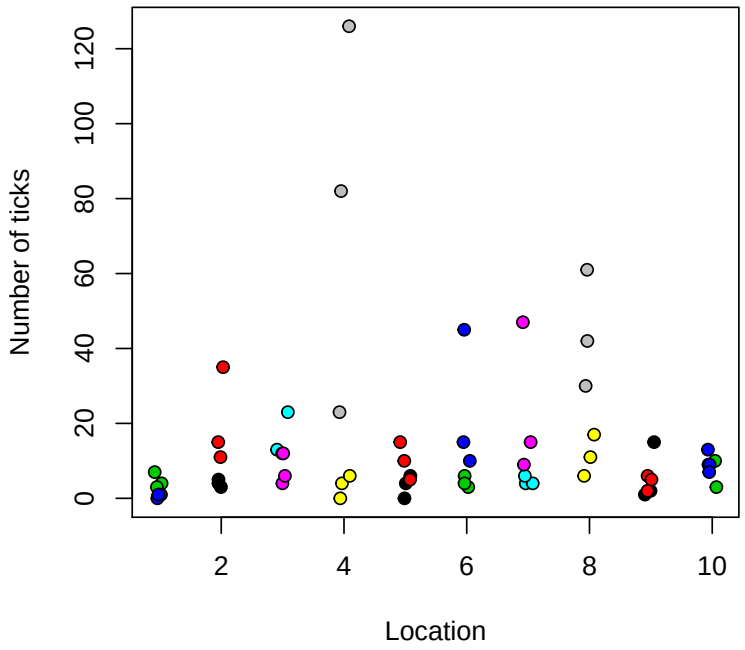
Simulate tick counts as Poisson-distributed with a mean burden of 10 ticks per chick. Random effect variances of 1, 0.7 and 0.3 for location, brood and chick, respectively, are supplied to `sim.glmm` as a vector with names corresponding to the columns of `design.data` that contain the relevant IDs.

```
> set.seed(121121) # set seed for random number generator to give repeatable results
> tickdata <-
+   sim.glmm(design.data = tickdata, fixed.eff = list(intercept = log(10)),
+     rand.V = c(location = 1, brood = 0.7, chick = 0.3), distribution = "poisson")
> tickdata
```

```
             chick     brood location response
1    loc1brd1chk1  loc1brd1     loc1        4
2    loc1brd1chk2  loc1brd1     loc1        7
3    loc1brd1chk3  loc1brd1     loc1        3
.              .         .        .        .
.              .         .        .        .
60  loc10brd2chk3 loc10brd2    loc10       35
```

Plot counts of ticks on individual chicks against locations. Each point represents a simulated tick count from a single chick. Different colours within locations distinguish broods, and the points are "jittered" horizontally to prevent points with the same coordinates being obscured.

```
> plot(response ~ jitter(as.numeric(location), factor = 0.5), pch = 21,
+   bg = as.numeric(brood), xlab = "Location", ylab = "Number of ticks",
+   data = tickdata)
```



Put the simulation code inside a function:

```
> sim.tickdata <- function(...){
+   sim.glmm(design.data = tickdata,
+     fixed.eff = list(intercept = log(10)),
+     rand.V = c(location = 1, brood = 0.7, chick = 0.3),
+     distribution = "poisson")
+ }
> sim.tickdata()
```

```
            chick     brood location response
1    loc1brd1chk1  loc1brd1     loc1       44
2    loc1brd1chk2  loc1brd1     loc1       14
3    loc1brd1chk3  loc1brd1     loc1       16
.               .         .        .        .
.               .         .        .        .
60  loc10brd2chk3 loc10brd2    loc10        3
```

Running the function a few times will show that the `response` column, which stores the number of ticks collected from each chick, changes randomly.

The next step is to calculate the margin of error by fitting a Poisson GLMM. The data simulation function can be supplied to the model-fitting function as a data set.

```
> set.seed(135791) # set seed for random number generator to give repeatable results
> fit <-
+   glmer(response ~ (1 | location) + (1 | brood) + (1 | chick),
+     family = "poisson", data = sim.tickdata())
```

The estimates of mean tick count and its standard error are used to calculate a 95% CI as exp(estimate $\pm$ 1.96 $\times$ standard error). The resulting interval is called a Wald Z interval and relies on the same assumptions as a Wald Z-test. Wald intervals and tests can be unacceptably inaccurate, and Monte Carlo Markov Chain (MCMC) or parametric bootstrapping should be used instead. However, these methods are slow, making them inconvenient for power analysis when using simulations. We therefore recommend the approach of monitoring the coverage of the Wald 95% CIs (that is, the proportion of simulations where the parameter value is within the 95% CI), and using slower, more accurate methods only where coverage deviates markedly from 95%, as illustrated in Section 2.2.

```
> intercept <- fixef(fit)
> estimate <- exp(intercept)
> se <- coef(summary(fit))[,"Std. Error"]
> ci.lo <- exp(intercept - qnorm(0.975) * se)
> ci.hi <- exp(intercept + qnorm(0.975) * se)
> c(estimate = estimate, ci.lo = ci.lo,  ci.hi = ci.hi)
```

```
estimate.(Intercept)    ci.lo.(Intercept)    ci.hi.(Intercept)
            9.301022             3.986976            21.697902
```

Calculate the margin of error as defined in the article: half the 95% CI width as a percentage of the estimated mean tick count.

```
> 100 * 0.5 * (ci.hi - ci.lo) / estimate
```

```
(Intercept)
   95.20956
```

We can also wrap these lines up in a function.

```
> sim.tick.err <- function(...){
+   fit <-
+     glmer(response ~ (1 | location) + (1 | brood) + (1 | chick),
+       family = "poisson", data = sim.tickdata())
+   intercept <- fixef(fit)
+   estimate <- exp(intercept)
+   se <- coef(summary(fit))[,"Std. Error"]
+   ci.lo <- exp(intercept - qnorm(0.975) * se)
+   ci.hi <- exp(intercept + qnorm(0.975) * se)
+   as.vector(100 * 0.5 * (ci.hi - ci.lo) / estimate)
+ }
> sim.tick.err()

[1] 67.41549
```

Again, you could run the function a few times to get a rough impression of the distribution of the margin of error values. In order to predict the margin of error we will average over 50 simulated data sets.

```
> sim.err <- sapply(1:50, sim.tick.err)
> mean(sim.err)

[1] 77.56467
```

The margin of error averaged over the 50 data sets is 78%, that is, an average 95% CI extends to ±78% of the estimate. This is (approximately) the expected margin of error. The margin of error has a wide distribution...

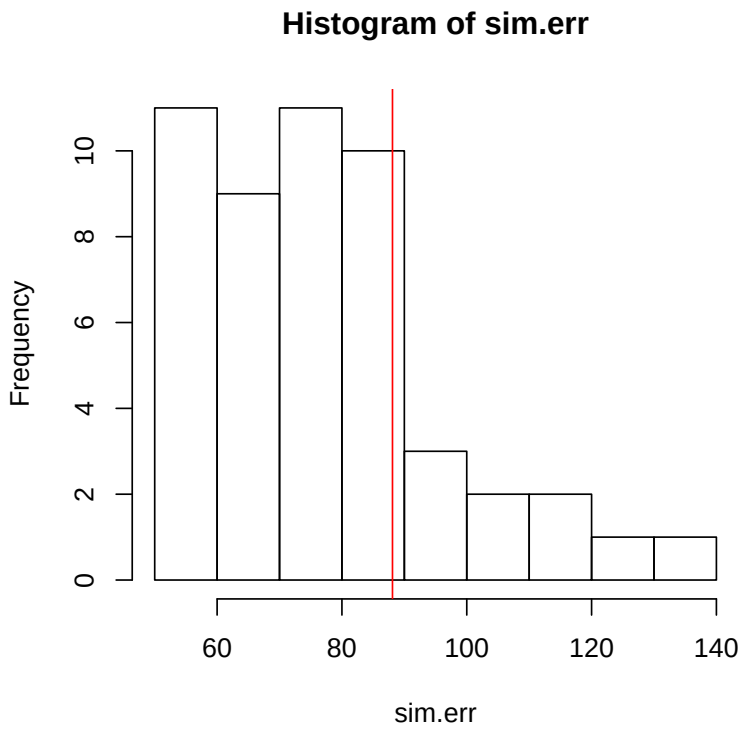
```
> hist(sim.err)
```

...so any individual simulation could yield a margin of error quite far from the expected value of 78%. In fact, because in this instance the mean and the median margin of error are both 75% (as can be checked using `median(sim.err)`), there is a 50% chance of the estimate of mean tick burden being less precise than predicted. If we wanted to be more cautious, we could ask instead: what margin of error are we 80% confident of doing better than?

```
> quantile(sim.err, 0.8)

 80%
88.1
```

The answer is 88%. Plot it on the histogram:

```
> hist(sim.err)
> abline(v = quantile(sim.err,0.8), col = "red")
```



In conclusion, if we performed the tick survey with the assumptions specified above, we would expect the 95% CI limits to be ±78% of the mean tick burden, and we would be 80% confident that they would be narrower than ±88% of the mean tick burden (these are potentially quite imprecise estimates of precision, due to only having used 50 data sets; re-running the analysis using 1000 simulations gave more trustworthy estimates of 75% and 92%). The target margin of error was 25% so we need to sample more locations. Figure 1 in the article shows that to achieve our target precision we need to sample 100 locations, regardless of whether the tick burden is assumed to be 1, 5 or 10 ticks per chick.

2.2 Using parametric bootstrap confidence intervals

As noted above, the Wald method of calculating confidence intervals used in Section 2.1 is potentially inaccurate. More accurate (but much slower) methods are available, including MCMC and

parametric bootstrapping.[4] In this section we illustrate how to use parametric bootstrapping to calculate 95% CIs in a scenario where poor coverage indicates that the Wald Z CIs are performing badly. MCMC CIs are not currently available for models fitted with `glmer`.

Supplementary Figures 3 and 4 show that the performance of the Wald method in terms of 95% CI coverage is worse for the tick abundance example (range 88% to 96%) than for the binomial (mosquito net trial) example (range 92% to 96%). We will focus on the tick abundance survey scenario where the mean tick burden is 1, 50 locations are surveyed, and location, brood and chick variances are 0, 0 and 0.3, respectively (Supplementary Figure 3, top right panel, red circle, solid line). Coverage in this scenario was 90.9%, (95% CI: 88.9% to 92.6%), significantly below the nominal rate of 95%. We therefore suspect that the estimated margin of error of 13% (main article Figure 1) is an underestimate. We will investigate this concern by repeating the precision analysis of Section 2.1, changing the CI code to use parametric bootstrapping rather than the Wald Z method.

First, we re-define the `sim.tickdata` function to simulate from the new scenario. This code is not shown below because all that is required is to re-run the code in Section 2.1 that generates the `tickdata` data and the `sim.tickdata` function, but with changes to the lines setting the number of locations (`expand.grid(chick = 1:3, brood = 1:2, location = 1:50)`), fixed effects (`fixed.eff = list(intercept = log(1))`) and the variances (`rand.V = c(location = 0, brood = 0, chick = 0.3)`).

```
> sim.tickdata()

             chick     brood location response
1    loc1brd1chk1  loc1brd1     loc1        0
2    loc1brd1chk2  loc1brd1     loc1        1
3    loc1brd1chk3  loc1brd1     loc1        1
.               .         .        .        .
.               .         .        .        .
300 loc50brd2chk3 loc50brd2    loc50        0
```

Fit the GLMM to a single simulated data set:

```
> set.seed(654321) # set seed for random number generator to give repeatable results
> fit <-
+   glmer(response ~ (1 | location) + (1 | brood) + (1 | chick),
+     family = "poisson", data = sim.tickdata())
```

We can calculate a Wald Z CI for the tick burden estimate and margin of error as before:

```
> intercept <- fixef(fit)
> estimate <- exp(intercept)
> se <- coef(summary(fit))[,"Std. Error"]
> ci.lo <- exp(intercept - qnorm(0.975) * se)
> ci.hi <- exp(intercept + qnorm(0.975) * se)
> c(estimate = estimate, ci.lo = ci.lo,  ci.hi = ci.hi)

estimate.(Intercept)    ci.lo.(Intercept)    ci.hi.(Intercept)
           1.0062055            0.8668192            1.1680055

> 100 * 0.5 * (ci.hi - ci.lo) / estimate
```

[4] See `http://glmm.wikidot.com/faq` and Bolker et al. (2009) Generalized linear mixed models: a practical guide for ecology and evolution *Trends in Ecology & Evolution*, 24, 127-135.

```
(Intercept)
   14.96644
```

For this single simulated data set, the 95% CI is 0.87 to 1.17 and the margin of error is 15%. Because of the evidence of low (91%) coverage in this scenario, we suspect that this CI might be too narrow, so we will re-estimate it using parametric bootstrapping. Parametric bootstrapping works by first simulating the sampling distribution of a parameter estimate from a fitted model and then calculating the confidence limits directly from this distribution. The simplest method for estimating a 95% CI from the sample of simulated estimates is to take the 2.5% and 97.5% quantiles. The following loop, which should take about a minute to run, performs the simulation and re-fitting steps 100 times. Responses are simulated from the fitted model using `sim.glmm` and the model is re-fitted to the simulated responses using `update`:[5]

```
> boot.est <-
+   sapply(1:100, function(...) {
+     sim.fit <- update(fit, data = sim.glmm(fit))
+     exp(fixef(sim.fit))
+   })
```

The distribution of these estimates represents the uncertainty around the estimate of 1.01 ticks per chick, and the 2.5th and 97.5th centiles give a 95% CI of 0.88 to 1.14 and a margin of error of 13%:

```
> ci95 <- quantile(boot.est, c(0.025, 0.975))
> ci95

     2.5%     97.5%
0.8849597 1.1441082

> 100 * 0.5 * diff(ci95) / estimate

   97.5%
12.87752

> hist(boot.est)
> abline(v = ci95, col="red")
```

[5]Rather than following the DIY approach to parametric bootstrapping presented here, we recommend using the `confint` function, which works with model fits of the class `merMod`. This function is not only more convenient, but also offers easy access to alternative bootstrap methods and parallel processing for greater speed. See `?confint.merMod` for details. However, at the time of writing `confint` is not working for this example. The `lme4` developers are working on this problem; for details and progress see `https://github.com/lme4/lme4/issues/231`.

The histogram shows the distribution of bootstrapped estimates with the 95% CI indicated by red vertical lines. One hundred bootstrap samples are too few to give stable bootstrapped confidence limits, so we cannot trust this CI; at least 1,000 and preferably closer to 10,000 samples are required. We used 20,000 samples to show that the stable 95% CI is 0.85 to 1.15, giving a margin of error of 15%. Thus, for this data set, the Wald and bootstrap CIs are similar and the margins of error identical. It might therefore appear that our efforts have been wasted and we should have stuck with the Wald CI, because the Wald method performs as well bootstrapping for this particular data set. However, we know from the the low coverage of 91% that data sets simulated in the same scenario yield Wald CIs that are, on average, too narrow, and without estimating the bootstrap CIs we could not have known which Wald CIs to trust.

Before concluding that the bootstrap CI is more trustworthy than the Wald CI, we should check that the bootstrap 95% CI does give coverage close to 95%. If it does, we can proceed to calculate the expected margin of error. We will calculate both coverage and expected margin of error by calculating bootstrap CIs from 50 simulated data sets.

For convenience, put the bootstrap CI code inside a function:

```
> paraboot.ticks.ci <-
+   function(mer.fit, nsim) {
+     boot.est <-
+       sapply(1:nsim, function(...) {
+         sim.fit <- update(mer.fit, data = sim.glmm(mer.fit))
+         exp(fixef(sim.fit))
+       })
+     c(estimate = exp(fixef(mer.fit)),
+       ci = quantile(boot.est, c(0.025, 0.975)))
+   }
> paraboot.ticks.ci(mer.fit = fit, nsim = 100)
```

```
estimate.(Intercept)              ci.2.5%             ci.97.5%
            1.006206             0.861024             1.157219
```

The next step is to apply this function to multiple simulated data sets. For this example we use only five simulated data sets and only 100 bootstrap samples per data set. These numbers are low

enough to allow the example to complete within a few minutes, and consequently not nearly high enough to give reliable results.

```
> sim.err.boot <-
+   sapply(1:5, function(...) {
+     fit <-
+       glmer(response ~ (1 | location) + (1 | brood) + (1 | chick),
+         family = "poisson", data = sim.tickdata())
+     paraboot.ticks.ci(mer.fit = fit, nsim = 100)
+   })
> sim.err.boot
```

```
                          [,1]      [,2]      [,3]      [,4]      [,5]
estimate.(Intercept) 0.8810299 1.0492180 0.9060356 0.9283756 0.9849593
ci.2.5%              0.7399866 0.8668291 0.7590703 0.7927512 0.8185644
ci.97.5%             0.9978969 1.1720755 1.0316971 1.0881383 1.1208047
```

This code outputs a table where each column stores the tick abundance estimate and bootstrap 95% CI for each simulated data set. Coverage is estimated as the percentage of CIs that contain the true tick abundance value of 1:

```
> 100 * mean(1 > sim.err.boot[2,] & 1 < sim.err.boot[3,])
```

```
[1] 80
```

The mean margin of error is:

```
> 100 * 0.5 * mean((sim.err.boot[3,] - sim.err.boot[2,]) / sim.err.boot[1,])
```

```
[1] 15.09598
```

These results are of course not reliable. We need to use much higher numbers of simulated data sets and bootstrap samples to obtain reliable estimate of coverage and expected margin of error. We repeated this analysis with 1000 bootstrap samples and 1000 simulated data sets. Coverage was 93.9%, significantly higher than the 90.9% coverage using Wald CIs, and not significantly different from 95%, showing that the bootstrap method outperforms the Wald method for this scenario. In line with this conclusion, the mean margin of error was 15%, higher than the 13% estimated using the Wald method. The computational cost of greater accuracy is high, however. This analysis took 32 hours, despite being run in parallel across 8 cores of a 2.7 GHz processor, although the number of simulated data sets could probably be reduced to 100 without substantial loss of accuracy.

Finally, it should be remembered that bootstrap CIs are not a panacea for poor (usually low) coverage. Bootstrapping is expected to correct the width of CIs that are too narrow, but poor coverage can also be caused by bias in the location of the CI. If the estimates from which the bootstrap CI is derived are biased, then the location of the CI will also be biased, causing low coverage even when CI width is accurate. For example, coverage in the scenario used in Section 2.1 (mean tick burden = 10; 10 locations surveyed; location, brood and chick variances of 1, 0.7 and 0.3, respectively) was 89% (Supplementary Figure 3), which most researchers would consider unacceptably low (in terms of P-values, 89% coverage for a 95% CI is equivalent to a type I error rate of 0.11 being presented as 0.05). We re-estimated coverage using bootstrap CIs (using 1000 bootstrap simulations and 1000 simulated data sets) as 88%, so no improvement on the Wald CIs. Supplementary Figure 1 shows that bias in this scenario is relatively high at 6%, and we suspect that this is the source of the poor coverage.

3 Predicting the power of a trial of insecticidal mosquito nets using a binomial GLMM

Like the t-test example, this example illustrates a "traditional" power analysis where the aim is to estimate the power of a hypothesis test. Six types of insecticidal mosquito net (one control net, C, and five experimental nets, E1-E5) are to be trialed against each other in six experimental huts using a standard experimental design as specified in the World Health Organization *Guidelines for laboratory and field testing of long-lasting insecticidal nets* (`http://apps.who.int/iris/bitstream/10665/80270/1/9789241505277_eng.pdf`). The outcome of interest is mosquito mortality, and the aim is to test the null hypothesis that there is no difference in mortality between the control net type and experimental net E1. The other four net types, E2-E5, are of secondary interest and are not tested here. To prevent confounding between the effects of huts and net types, net types are rotated through the huts weekly so that after six weeks a 6-week × 6-hut Latin square design has been performed (see Table 1 in main article).

As with the t-test example, the three steps to estimating power using simulations are:

- Simulate data under the alternative hythothesis
- Test the null hypothesis
- Estimate power by averaging over multiple repeats of the first two steps

First, simulate a single data set. Define the 6-week × 6-hut Latin square determining the rotation[6] of the six net types treatments through the six huts:

```
> latsq <-
+   rbind(
+     c("C",  "E1", "E2", "E3", "E4", "E5"),
+     c("E5", "C",  "E1", "E2", "E3", "E4"),
+     c("E4", "E5", "C",  "E1", "E2", "E3"),
+     c("E3", "E4", "E5", "C",  "E1", "E2"),
+     c("E2", "E3", "E4", "E5", "C",  "E1"),
+     c("E1", "E2", "E3", "E4", "E5", "C"))
> colnames(latsq) <- paste("hut", 1:nrow(latsq), sep = "")
> rownames(latsq) <- paste("week", 1:ncol(latsq), sep = "")
> latsq
```

```
      hut1 hut2 hut3 hut4 hut5 hut6
week1 "C"  "E1" "E2" "E3" "E4" "E5"
week2 "E5" "C"  "E1" "E2" "E3" "E4"
week3 "E4" "E5" "C"  "E1" "E2" "E3"
week4 "E3" "E4" "E5" "C"  "E1" "E2"
week5 "E2" "E3" "E4" "E5" "C"  "E1"
week6 "E1" "E2" "E3" "E4" "E5" "C"
```

Make a template data set with 6 huts × 6 weeks × 6 nights per week = 216 rows. Each row will record the mosquito mortality data for one hut on one night.

[6] In a real hut trial it would be advisable to deviate from simple rotation in order to balance against carry-over effects, e.g. where an insecticide-treated net leaves a residue in the hut that biases the response to the net that follows it (see Table 2 of the main article). The can be done usng the `get.plan` function in the `crossdes` package.

```
> mosdata <-
+   expand.grid(
+     hut = factor(1:ncol(latsq)),
+     week = factor(1:nrow(latsq)),
+     night = factor(1:6))
> mosdata <- mosdata[order(mosdata$hut, mosdata$week, mosdata$night),]
> mosdata
```

```
    hut week night
1     1    1     1
37    1    1     2
73    1    1     3
.     .    .     .
.     .    .     .
216   6    6     6
```

Add an observation (row ID) number, which will be used to simulate overdispersion:

```
> mosdata$observation <- factor(formatC(1:nrow(mosdata), flag="0", width=3))
> mosdata
```

```
    hut week night observation
1     1    1     1         001
37    1    1     2         002
73    1    1     3         003
.     .    .     .           .
.     .    .     .           .
216   6    6     6         216
```

Assign treatments to each row:

```
> mosdata$net <- factor(diag(latsq[mosdata$week, mosdata$hut]))
> mosdata
```

```
    hut week night observation net
1     1    1     1         001   C
37    1    1     2         002   C
73    1    1     3         003   C
.     .    .     .           .   .
.     .    .     .           .   .
216   6    6     6         216   C
```

We also need to specify the number of mosquitoes entering each hut each night, which `sim.glmm` expects to be stored as a column of the design data set labelled "n":

```
> mosdata$n <- 25
> mosdata
```

```
    hut week night observation net  n
1     1    1     1         001   C 25
37    1    1     2         002   C 25
73    1    1     3         003   C 25
.     .    .     .           .   .  .
.     .    .     .           .   .  .
216   6    6     6         216   C 25
```

The design data set is now ready to be input to `sim.glmm`, which will add a single column of responses (numbers of dead mosquitoes in each hut on each night) to `mosdata`. These responses will be simulated from a binomial GLMM specified by fixed and random effect parameter values. The fixed effects specify the mortality due to each net type. Here, as in the article, we assume that mortality is 70% with the control net and 80% with the five experimental nets. The difference in mortality between the control net and the experimental nets can be expressed as an odds ratio of approximately 1.7:

```
> (80 / 20) / (70 / 30)

[1] 1.714286
```

The fixed effects must be supplied on the logit scale, which is the scale on which they are estimated and output with standard GLM- and GLMM-fitting functions such as `glm` and `glmer`. The intercept is the logit of the mortality in the reference category, or `qlogis(0.7)`. The regression coefficients are log odds ratios, calculated as `log(1.7)`, for all the experimental nets. The odds ratio for the reference net type, C, is 1 by definition.

For the random effects, we assume variances of 0.5, 0.5 and 1 between huts, weeks and observations, respectively.[7] The inter-observation variance models overdispersion.

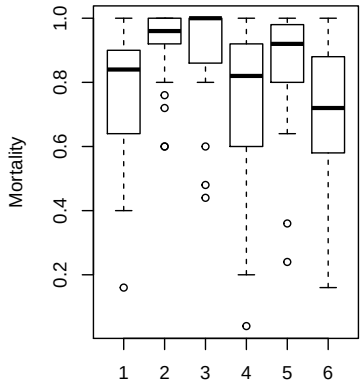
Input the design data set, the fixed and random effect assumptions, and the assumption that mortality is binomially distributed to `sim.glmm`:

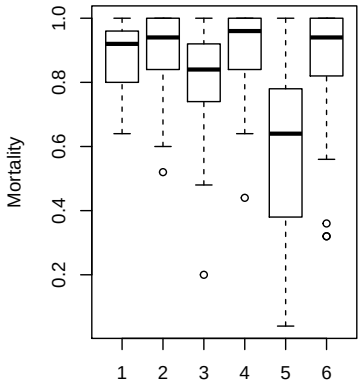
```
> set.seed(321123) # set seed for random number generator to give repeatable results
> mosdata <-
+   sim.glmm(design.data = mosdata,
+     fixed.eff = list(
+       intercept = qlogis(0.7),
+       net = log(c(C = 1, E1 = 1.7, E2 = 1.7, E3 = 1.7, E4 = 1.7, E5 = 1.7))),
+     rand.V = c(hut = 0.5, week = 0.5, observation = 1),
+     distribution = "binomial")
> mosdata

    hut week night observation net  n response
1     1    1     1         001   C 25       20
37    1    1     2         002   C 25       24
73    1    1     3         003   C 25       16
.     .    .     .           .   .  .        .
.     .    .     .           .   .  .        .
216   6    6     6         216   C 25       21
```

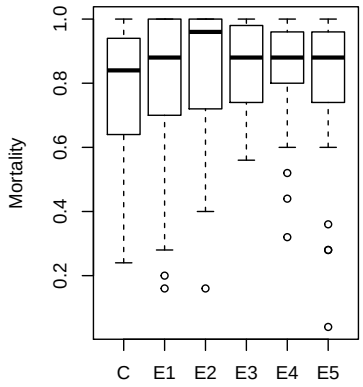
Plot mortality against the three factors expected to cause variation in mortality:

```
> par(mfrow = c(2, 2))
> plot(response/n ~ hut, ylab = "Mortality", data = mosdata)
> plot(response/n ~ week, ylab = "Mortality", data = mosdata)
> plot(response/n ~ net, ylab = "Mortality", data = mosdata)
```

[7] A variance of 0.5 between huts translates to a MOR of 2 (see footnote on p12). Interpretation: mortality differs between a typical pair of huts by a factor of 2.







The first two plots show evidence of variation in mortality among huts and over the six weeks. Based on the third plot, the difference in mortality between the C and E1 net types, which is the effect that we are interested in, is small in this simulated data set. We can test for this difference by fitting the GLMM from which the data were simulated:

```
> fit <-
+   glmer(
+     cbind(response, n - response) ~
+       net + (1 | hut) + (1 | week) + (1 | observation),
+     family = binomial, data = mosdata)
> summary(fit)

Generalized linear mixed model fit by maximum likelihood (Laplace
  Approximation) [glmerMod]
 Family: binomial  ( logit )
Formula: cbind(response, n - response) ~ net + (1 | hut) + (1 | week) +
    (1 | observation)
   Data: mosdata

     AIC      BIC   logLik deviance df.resid
  1047.9   1078.3   -514.9   1029.9      207
```

```
Scaled residuals:
    Min      1Q  Median      3Q     Max
-1.0660 -0.2643  0.0678  0.6205  1.2549

Random effects:
 Groups      Name        Variance Std.Dev.
 observation (Intercept) 1.1911   1.0914
 week        (Intercept) 0.6440   0.8025
 hut         (Intercept) 0.7643   0.8742
Number of obs: 216, groups:  observation, 216; week, 6; hut, 6

Fixed effects:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   1.6489     0.5283   3.121  0.00180 **
netE1         0.5025     0.3050   1.648  0.09942 .
netE2         0.9367     0.3123   2.999  0.00271 **
netE3         0.6311     0.3036   2.079  0.03764 *
netE4         0.5691     0.3024   1.882  0.05982 .
netE5         0.4472     0.3042   1.470  0.14145
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Correlation of Fixed Effects:
      (Intr) netE1  netE2  netE3  netE4
netE1 -0.270
netE2 -0.262  0.463
netE3 -0.273  0.475  0.468
netE4 -0.273  0.476  0.465  0.476
netE5 -0.273  0.475  0.464  0.481  0.480
```

The null hypothesis is that mortality is the same between net C, the control, and experimental net E1. This is the same as hypothesising that the mortality odds ratio for E1 relative to C is 1 (we know that the true odds ratio is 1.7 in the sampled population because we set this in the simulation). The P-value for a Wald Z-test of this null hypothesis is...

```
> coef(summary(fit))["netE1", "Pr(>|z|)"]

[1] 0.09942021
```

...providing no evidence of a difference in mortality. For convenience we put the simulation code inside a function:

```
> sim.mosdata <- function(...){
+   sim.glmm(design.data = mosdata,
+     fixed.eff = list(
+       intercept = qlogis(0.7),
+       net = log(c(C = 1, E1 = 1.7, E2 = 1.7, E3 = 1.7, E4 = 1.7, E5 = 1.7))),
+     rand.V = c(hut = 0.5, week = 0.5, observation = 1),
+     distribution = "binomial")
+   }
```

For greater flexibility, the "`...`" in `function(...)` could be replaced with arguments allowing the parameter values to be changed. Also, the generation of the template data set, `mosdata`, could be

brought inside the function to allow "design" aspects (e.g. the number of mosquitoes per night) to be altered easily.

Run the function a few times to show that the response column, which stores the number of dead mosquitoes, changes randomly.

```
> sim.mosdata()
```

```
    hut week night observation net  n response
1     1    1     1         001   C 25       25
37    1    1     2         002   C 25        5
73    1    1     3         003   C 25       19
.     .    .     .           .   .  .        .
.     .    .     .           .   .  .        .
216   6    6     6         216   C 25        8
```

`sim.mosdata()` can be supplied to the GLMM fitting function as a data set, just as `mosdata` was above (output not shown):

```
> glmer(
+   cbind(response, n - response) ~
+     net + (1 | hut) + (1 | week) + (1 | observation),
+   family = binomial, data = sim.mosdata())
```

Put these lines inside a function, adding the code used above to extract the P-value:

```
> sim.mos.pval <- function(...){
+   fit <-
+     glmer(
+       cbind(response, n - response) ~
+         net + (1 | hut) + (1 | week) + (1 | observation),
+       family = binomial, data = sim.mosdata())
+     coef(summary(fit))["netE1", "Pr(>|z|)"]
+   }
```

Running `sim.mos.pval()` once simulates a data set, fits the GLMM and returns the P-value for the null hypothesis that mortality is the same between net types C and E1:

```
> set.seed(978675) # set seed for random number generator to give repeatable results
> sim.mos.pval()
```

```
[1] 0.0009851087
```

Try running the function repeatedly to get an impression of the distribution of P-values.

The third step is to estimate power by averaging over multiple simulations. Simulate and test 20 data sets, then estimate power as the proportion of data sets giving $P < 0.05$:

```
> sim.pvals <- sapply(1:20, sim.mos.pval)
> mean(sim.pvals < 0.05)
```

```
[1] 0.4
```

The power estimate is 40%. However, 20 simulations are of course not sufficient to give an accurate power estimate, as can be seen from the width of the CI around the power estimate:

```
> binom.test(table(factor(sim.pvals < 0.05, c(T, F))))$conf.int
```

```
[1] 0.1911901 0.6394574
attr(,"conf.level")
[1] 0.95
```

Ideally you should run 1000 simulations, which gives a 95% CI of about ±2.5% at 80% power. For the scenario specified above, 1000 simulations should give a reasonably accurate power estimate, which should be close to 60% (this is the lowest power scenario in the lower panel of Figure 1 in the article). This should take between 5 and 60 minutes, depending on the computer used.[8]

[8]Due to the time taken to fit the models, the GLMM examples in this document are quite slow, the net trial example in particular. 1000 of the mosquito net simulations took less than 1 hour on a single core of an old Windows laptop (2.27 GHz Intel Core i3 CPU). On some computers, particularly those running Windows, this will probably be the best that can be done. However, given the value of power analysis to improving the design of an experiment, this will generally be time well spent. On some computers and operating systems, the simulations can easily be made much faster by parallelizing across multiple cores, if available. This is straightforward on multi-core Unix-alike machines, including Macs, by loading the `parallel` package and replacing `sapply(...)` in the code above with `unlist(mclapply(..., mc.cores = detectCores()))`.

4 Simulating from a random intercepts-and-slopes GLMM

All of the GLMMs presented in the article and, so far, in this tutorial have been random intercepts models; that is, the role of the random effects is to model variation in the intercept. Random intercepts-and-slopes models are a widely used class of GLMM in which not only the intercept but also the slope of the regression line vary between clusters. The aim of this section is to illustrate how to simulate from a random intercepts-and-slopes GLMM, where both the intercept and the covariate effects vary randomly between groups. We will use the `sleepstudy` data set from the `lme4` package, because it allows simpler and clearer illustration of random intercepts-and-slopes than the examples above (see `?sleepstudy` and the examples at `?lmer`). Briefly, as part of a study of the effects of sleep deprivation on performance reaction times (`Reaction`, measured in ms) of eighteen subjects (`Subject`) were recorded on ten consecutive days (`Days`), numbered zero to nine to indicate the number of consecutive sleep-deprived nights the subject has undergone at the time of testing.

```
> sleepstudy
```

```
    Reaction Days Subject
1   249.5600    0     308
2   258.7047    1     308
3   250.8006    2     308
.          .    .       .
.          .    .       .
180 364.1236    9     372
```

We will initially simulate reaction times from a random intercepts Gaussian GLMM without random slopes. The fixed effects assumptions are as follows: the intercept, which is the mean reaction time on day zero, is 250 ms, and the slope, which represents the linear increase in reaction time per day, is 10 ms/day. Subjects are expected to differ in mean reaction time; we assume normally distributed random variation among subjects with a variance of 1400 ms^2. The residual standard deviation is assumed to be 30 ms. As above, we supply these assumptions to `sim.glmm`...

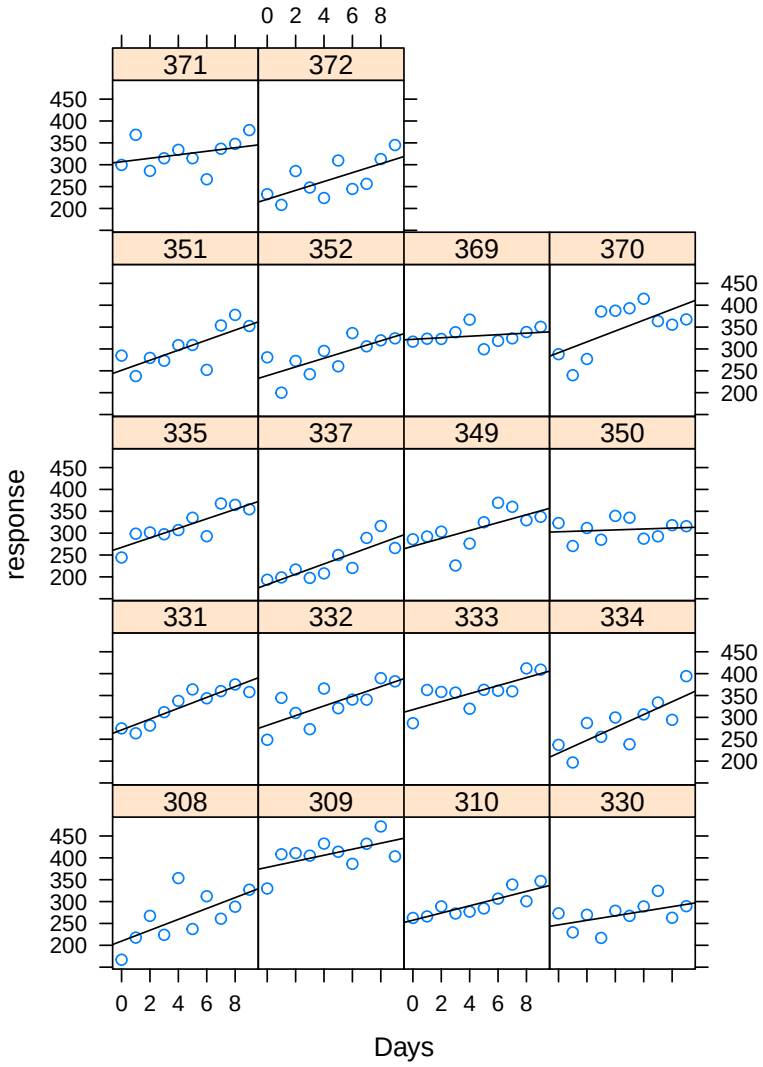
```
> set.seed(121212) # set seed for random number generator to give repeatable results
> sim.sleepstudy <-
+   sim.glmm(design.data=sleepstudy,
+     fixed.eff=c(intercept = 250, Days = 10), rand.V = c(Subject = 1400),
+     distribution = "gaussian", SD = 30)
> sim.sleepstudy
```

```
    Reaction Days Subject response
1   249.5600    0     308 166.9963
2   258.7047    1     308 217.5891
3   250.8006    2     308 267.3376
.          .    .       .        .
.          .    .       .        .
180 364.1236    9     372 345.0251
```

...creating a new data frame with simulated responses stored in `sim.sleepstudy$response`. Plot `response` against `Days` for each subject, adding subject-specific least-squares regression lines:

```
> library(lattice)
> xyplot(response ~ Days | Subject, sim.sleepstudy,
+   panel=
```

```
+     function(x, y){
+       panel.xyplot(x, y)
+       panel.abline(lm(y ~ x))
+     })
```



Despite the apparent variation in the slopes of the regression lines, we know that each subject has the same underlying slope of 10 ms/day because variation in slopes was not specified in the simulation code. To simulate random variation in slopes—that is, to simulate from a random intercepts-and-slopes model—we need to specify three variance parameters via the `rand.V` argument: the variance among random intercepts, the variance among random slopes, and the covariance between intercepts and slopes. These are supplied to `rand.V` as a covariance matrix with the variances on the diagonal and the covariance off the diagonal, with appropriate row and column names:

```
> covmat <- matrix(c(600, 10, 10, 35), nrow = 2)
> rownames(covmat) <- colnames(covmat) <- c("intercept", "Days")
> covmat
```

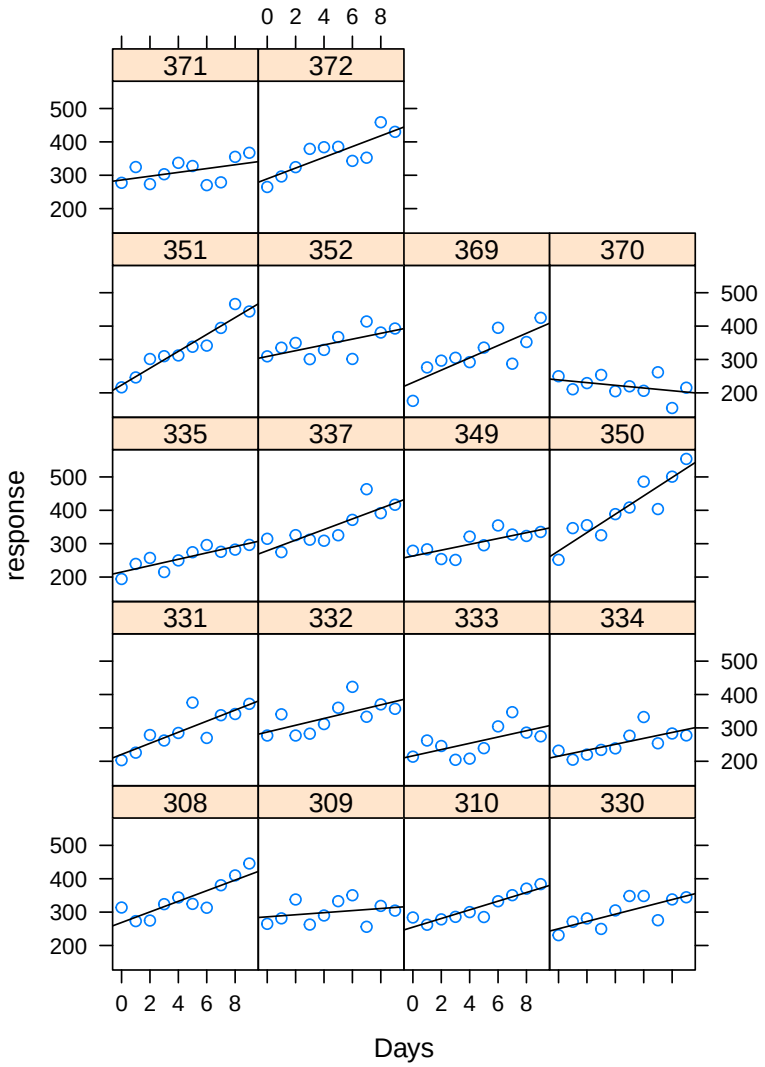
```
          intercept Days
intercept       600   10
Days             10   35
```

The assumptions for the intercept variance, the slope variance and the covariance of 600, 35 and 10, respectively, are based on fitting a random intercepts-and-slopes model (`Reaction ~ Days + (Days | Subject)`) to the `sleepstudy` data set. We can now update the code above to simulate from a random slopes model by replacing `rand.V = c(Subject = 1400)` with `rand.V = list(Subject = covmat)`:

```
> sim.sleepstudy <-
+   sim.glmm(design.data=sleepstudy,
+     fixed.eff=c(intercept = 250, Days = 10), rand.V = list(Subject = covmat),
+     distribution = "gaussian", SD = 30)
> sim.sleepstudy

    Reaction Days Subject response
1   249.5600    0     308 313.7525
2   258.7047    1     308 272.9681
3   250.8006    2     308 274.7453
.          .    .       .        .
.          .    .       .        .
180 364.1236    9     372 430.0944
```

Plot `response` against `Days` for each subject with subject-specific regression lines:

```
> library(lattice)
> xyplot(response ~ Days | Subject, sim.sleepstudy,
+   panel=
+     function(x, y){
+       panel.xyplot(x, y)
+       panel.abline(lm(y ~ x))
+     })
```



There is clearly much greater variation among subjects in the speed of decline in performance compared with the random intercepts simulation. The reaction times of some subjects slowed very rapidly (e.g. subjects 350 and 351), while others were unchanged or even improved over the ten days (e.g. subjects 309 and 370).

In this example, the parameter values assumed in the simulation were estimates obtained by fitting a GLMM to real data. `sim.glmm` allows a model fitted with `lmer` or `glmer` (i.e. an object of class `merMod`) to be supplied directly, as a single argument, rather than as separate estimates:

```
> fm1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
> sim.glmm(fm1)

    Reaction Days Subject response
1   249.5600    0     308 242.8795
2   258.7047    1     308 239.0938
3   250.8006    2     308 244.7493
.          .    .       .        .
.          .    .       .        .
180 364.1236    9     372 327.3402
```

Note that `simulate(fm1)` will do the same, with the minor difference that only the responses are returned. This function also allows multiple simulations to be run without the need for additional

lines of code. For example, a data frame with 5 columns of simulated responses is generated as follows:

```
> simulate(fm1, nsim = 5)
```

```
       sim_1    sim_2    sim_3    sim_4    sim_5
1   276.0321 221.7083 258.0632 261.3962 263.4468
2   298.9607 271.2039 243.4024 222.3215 316.6757
3   275.9521 255.2033 305.4879 245.0681 273.1689
.          .        .        .        .        .
.          .        .        .        .        .
180 312.2866 343.4309 324.6310 303.3323 560.1905
```