Repeated measures

Philip Dixon

11/20/2022

Analyzing repeated measures data in R

This information was completely rewritten in Nov 2022 to use the mmrm package instead of the nlme package. mmrm provides a much more convenient interface and a much faster model fitting algorithm. mmrm is listed as experimental, but the authors and contributors are a highly respected group of statisticians.

Computing summary statistics requires some of the R data manipulation functions. Fitting models with correlated observations requires new libraries and new functions. lmer() only allows random effects. It doesn't allow user-specified correlation matrices. The new library are mmrm and nlme. mmrm fits lots of different repeated measures models; nlme is the older library for correlated observations that we will use for a couple of models.

emmeans() can process both mmrm() and nlme() results. Remember that when given an lmer() fit, emmeans uses the KR algorithm to get df. The actual computations are done by functions in the pbkrtest library. These functions work for random effects models but not correlated data. Because mmrm() results could be from a correlated data model, both mmrm() and emmeans() play chicken and don't use KR. Instead, emmeans uses the Satterthwaite algorithm.

For some models, emmeans decides it can't calculate df and ignores them. I.e., it uses normal distributions instead of t or F distributions. You will see this a Z or Chi-square statistic.

I use the asparagus study as my example. The results shown in lecture were obtained with SAS. These R results mostly match. When there are differences, they are because of different df computations: KR (in SAS) or Satterthwaite (in R/mmrm).

```r
library(mmrm)
library(emmeans)

library(dplyr)  # for summary statistics

asp <- read.csv('../data/asparagus.csv')
asp$trt.f <- factor(asp$trt)
asp$year.f <- factor(asp$year)
asp$block.f <- factor(asp$block)
```

Computing summary statistics

These can be computed various ways. The simplest is to use data manipulation functions in the dplyr library. We will use three functions:

- %>% The "piping" operator. The required commands can be written without this, but using piping results in code that is much easier to read.
- group_by() which defines the groups in the data
- summarize() which computes summary statistics for each group. If the data aren't grouped, you get a summary for the entire data set

We want to compute the mean for each plot (i.e. averaging over years). We also want to retain the block and treatment of that plot. Since the combination of block and treatment uniquely identifies a plot, the simplest way to do this is to group by both block and treatment.

Here's the code to group the data, then compute means for each group of observations.

```
asp.means <- asp %>%
  group_by(block.f, trt.f) %>%
  summarize(meanYield = mean(yield))
```

```
## `summarise()` has grouped output by 'block.f'. You can override using the
## `.groups` argument.
```

```
nrow(asp.means)
```

```
## [1] 16
```

```
head(asp.means)
```

```
## # A tibble: 6 x 3
## # Groups:   block.f [2]
##   block.f trt.f meanYield
##   <fct>   <fct>     <dbl>
## 1 1       1          355.
## 2 1       2          404.
## 3 1       3          320.
## 4 1       4          233.
## 5 2       1          327
## 6 2       2          319
```

You see that asp.means has 16 observations, one for each treatment in each block.

You then do the desired analysis on the means.

```
asp.mean.lm <- lm(meanYield ~ block.f + trt.f, data=asp.means)
anova(asp.mean.lm)
```

```
## Analysis of Variance Table
##
## Response: meanYield
##           Df Sum Sq Mean Sq F value    Pr(>F)
## block.f    3   6086  2028.6   3.187 0.0772053 .
## trt.f      3  42593 14197.7  22.306 0.0001672 ***
## Residuals  9   5729   636.5
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
asp.mean.emm <- emmeans(asp.mean.lm, 'trt.f')
asp.mean.emm
```

```
##  trt.f emmean   SE df lower.CL upper.CL
##  1        352 12.6  9      323      380
##  2        331 12.6  9      302      359
##  3        311 12.6  9      282      339
##  4        217 12.6  9      188      245
##
## Results are averaged over the levels of: block.f
## Confidence level used: 0.95
```

Modeling correlated observations

mmrm() is the main function in the mmrm package. It fits a model where you specify the fixed effect structure (the treatment design + blocking variables) and the form of the correlation matrix for repeated observations on the same subject.

One detail is a limitation on how subjects are identified. In an lmer() model, you can write (1 | block.f:trt.f) to indicate a random effect for each combination of block and trt, i.e., for each plot. You can't do this with mmrm(). You need to create one variable with a unique value for each plot.

I find the easiest way to create that unique value for each subject is to use paste() to concatenate strings. I find this the easiest because it works with both numeric and character variables. sep= specifies the separator between strings.

```
asp$bt <- factor(paste(asp$block, asp$trt, sep='/'))
table(asp$bt)
```

```
##
## 1/1 1/2 1/3 1/4 2/1 2/2 2/3 2/4 3/1 3/2 3/3 3/4 4/1 4/2 4/3 4/4
##   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
```

```
# this is completely equivalent
asp$bt2 <- factor(paste(asp$block.f, asp$trt.f, sep='/'))
table(asp$bt2)
```

```
##
## 1/1 1/2 1/3 1/4 2/1 2/2 2/3 2/4 3/1 3/2 3/3 3/4 4/1 4/2 4/3 4/4
##   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
```

```
# result doesn't have to be coerced to a factor, but it doesn't hurt
```

split-plot in time using mmrm()

You specify the fixed effects in the usual way, using the same specifications used in lm() or lmer(). The correlation among repeated measurements on the same subject is specified using a function that gives the name of the repeated variable and the name of the subject variable. For example, in the asparagus data, year.f is the repeated variable. ** This must be a factor. ** The subject variable does not have to be a factor. Two examples of covariance structure functions:

- cs(year.f | bt) compound symmetric structure (split-plot-in-time). Observations on the same subject share the same value of bt.
- un(year.f | bt) Unstructured covariance matrix.

In both cases, the variable on the left of the | indicates which the time of each observation. It has to be a factor. This is required for all covariance models even though it isn't needed for cs. The variable on the right of the | indicates the subject. Observations on the same subject (i.e., same value of bt) are correlated. Observations with different values of bt are independent.

For comparison, here is the syntax for the lmer fit of this model:

```
library(lme4)
```

```
## Loading required package: Matrix
```

```
asp.lmer1 <- lmer(
  yield ~ block.f + trt.f + year.f + trt.f:year.f + (1 | bt),
  data=asp)
```

Here is how to fit the split-plot-in-time model in mmrm, extract marginal and cell means, and obtain the type III tests:

```
asp.fit1 <- mmrm(
  yield ~ block.f + trt.f + year.f + trt.f:year.f + cs(year.f | bt),
  data=asp)

# marginal means
asp.trt <- emmeans(asp.fit1, 'trt.f')
```

```
## NOTE: Results may be misleading due to involvement in interactions
```

```
asp.trt
```

```
##  trt.f emmean   SE df lower.CL upper.CL
##  1        352 12.6  9      323      380
##  2        331 12.6  9      302      359
##  3        311 12.6  9      282      339
##  4        217 12.6  9      188      245
##
## Results are averaged over the levels of: block.f, year.f
## Confidence level used: 0.95
```

```
asp.year <- emmeans(asp.fit1, 'year.f')
```

```
## NOTE: Results may be misleading due to involvement in interactions
```

```
asp.year
```

```
##  year.f emmean   SE   df lower.CL upper.CL
##  1         180 7.76 18.7      163      196
##  2         299 7.76 18.7      283      316
##  3         428 7.76 18.7      412      445
##
## Results are averaged over the levels of: block.f, trt.f
## Confidence level used: 0.95
```

```
# test trt 1 - trt 2 = 0, then year 1 - year 2 = 0
contrast(asp.trt, list('trt 1-2' = c(1, -1, 0, 0)))
```

```
##  contrast estimate   SE df t.ratio p.value
##  trt 1-2      21.2 17.8  9   1.187  0.2658
##
## Results are averaged over the levels of: block.f, year.f
```

```
contrast(asp.year, list('year 1-2' = c(1, -1, 0)))
```

```
##  contrast estimate   SE df t.ratio p.value
##  year 1-2     -120 7.82 24 -15.340  <.0001
##
## Results are averaged over the levels of: block.f, trt.f
```

```
# cell means
asp.emm1 <- emmeans(asp.fit1, c('trt.f','year.f'))

# test trt 1 - trt 2 in year 1
contrast(asp.emm1, list('trt 1 - 2 in year 1' =
    c(1, -1, 0,0, rep(0, 8))))
```

```
##  contrast            estimate   SE   df t.ratio p.value
##  trt 1 - 2 in year 1     40.5 21.9 18.7   1.846  0.0807
##
```

```
## Results are averaged over the levels of: block.f
```

```
# type III tests
joint_tests(asp.emm1)
```

```
##  model term   df1 df2 F.ratio p.value
##  trt.f          3   9  22.306  0.0002
##  year.f         2  24 506.598  <.0001
##  trt.f:year.f   6  24   8.948  <.0001
```

Since this dataset is balanced, the only non-magic df are for cell means and some contrasts among cell means.

Other useful functions

- summary() of an mmrm object includes the AIC, BIC and log Likelihood statistics as the first line of results. You can extract those individually using:
- logLik(): returns the log likelihood
- AIC(): returns the AIC statistic = -2 logLik + 2p
- BIC(): returns the BIC statistic = -2 logLik + p ln(N)

These can also be used on lmer() results.

- summary() of an mmrm object includes the variance-covariance matrix for each subject. You can obtain the correlation matrix by extracting that VC matrix then using cov2cor() to compute the correlation matrix.

```
asp.cov <- asp.fit1$cov
cov2cor(asp.cov)
```

```
##           1         2         3
## 1 1.0000000 0.4919337 0.4919337
## 2 0.4919337 1.0000000 0.4919337
## 3 0.4919337 0.4919337 1.0000000
```

emmeans() is aware of mmrm() results, so all the usual things can be done with those objects. The big difference between objects is the degrees of freedom computations.

Fitting other correlation models

My code for each is in each description. The only difference between the models is the name of correlation model function. (Note: This is much simpler and easier to understand than the older way of using functions in the nlme library).

ar(1): This is the ar1() function in mmrm. There is also an ar1h() that allows different variances for each time.

```
asp.ar1 <- mmrm(
  yield ~ block.f + trt.f + year.f + trt.f:year.f + ar1(year.f | bt),
  data=asp)
```

ar(1) with heterogeneous variances Most of the mmrm covariance models have heterogeneous variance extensions. All are named with an h at the end of the model function. ar1 is the ar(1) model; ar1h is the ar(1) with heterogeneous variances model. The only two models that don't (as of 2022) have heterogeneous variance extensions are us (doesn't need it, because automatically includes heterogeneous variances and sp_exp (not currently included).

```
asp.ar1h <- mmrm(
  yield ~ block.f + trt.f + year.f + trt.f:year.f + ar1h(year.f | bt),
  data=asp)
```

ar(1) + random effects: mmrm() doesn't include this model, but a very close equivalent is the Toeplitz model. This is toep() and toeph() in mmrm().

```
asp.toep <- mmrm(
  yield ~ block.f + trt.f + year.f + trt.f:year.f + toep(year.f | bt),
  data=asp)
```

You can fit the ar(1) + re model using the lme function in the nlme library. This allows you to specify both a random effect and a correlation structure. The default AIC computation includes the fixed effect parameters. If you want to compare AIC from lme() to that from mmrm(), you need to remove the number of fixed effect parameters.

Antedependence This is the ad() function in mmrm. ad() has a single variance for all time points (an often reasonable simplification of the usual antedependence model). If you want the usual antedependence, with different variances for each time point, use adh().

```
asp.ante <- mmrm(
  yield ~ block.f + trt.f + year.f + trt.f:year.f + ad(year.f | bt),
  data=asp)
```

unequally spaced time intervals: The spatial exponential model generalizes ar(1) to arbitrary time intervals. This is sp_exp() in mmrm(). The difference in time between two observations is treated as a distance. You tell sp_exp() which variable contains the observation time. It computes the time lags for you.

For this model, the variable indicating the time for each observation needs to be numeric.

```
asp.spExp <- mmrm(
  yield ~ block.f + trt.f + year.f + trt.f:year.f + sp_exp(year | bt),
  data=asp)
```

unstructured: This is us()

```
asp.us <- mmrm(
  yield ~ block.f + trt.f + year.f + trt.f:year.f + us(year.f | bt),
  data=asp)
```

independence I saved this for last because this is actually the hardest to get the correct logLikelihood and AIC/BIC values. You would think you want to use lm(), but you can't compare lm() values to those from lmer() or mmrm().

This is because of a "gotcha". The log likelihood (the starting point for AIC and BIC) includes some constants. Some functions ignore those constants; some include them in the log Likelihood. This doesn't affect comparisons between results produced by the same function but it really matters when you compare results from different functions. The documentation almost never tells you exactly what is being computed.

A second issue is how the function counts parameters. Specifically, are the fixed effect parameters included or not. mmrm() does not, lm() does.
The consequence of both issues is that you need to be very careful comparing AIC or BIC computed by different functions. You can't compare lm() and mmrm() or lmer().

To illustrate this, I use a made-up data set where the observations are independent.

Here are log-Likelihood values, AIC values, and effective number of parameters for an independence model fit four different ways.

Table 1: logLik, AIC, and # parameters

	logLik	AIC	Npar
mmrm	-2137.5	4276.9	1
lmer	-2137.5	4336.9	31
gls	-2137.5	4336.9	31
lm	-2106.1	4274.1	31

You see that lm() returns a very different log Likelihood, that's because of it ignores a constant that is included in the log Likelihoods by the other three functions.

You see that lmer() and gls() return different AIC values than does mmrm(). That's because mmrm() counts the variance-covariance parameters (only σ^2 here) and ignores the fixed effect parameters. The other two include the fixed effect parameters (30 for this test data set).

The most straightforward way to get the AIC for the independence model is to use the gls() function in the nlme library. This is the "older" way to fit models with correlated errors. It can also fit the independence model. mmrm() will not. mmrm() requires you to specify a non-independent model for the correlated observations. You then need to "fix up" the number of parameters to not count those for the fixed effects. The code below includes a function to do that for AIC.

```
library(nlme)

asp.gls <- gls(yield ~ block.f + trt.f + year.f + trt.f:year.f,
  data=asp)

AICindep <- function(m) {
  # compute AIC adjusting for fixed effect parameters
  AIC(m) - 2*m$dims$p
  }

c(logLik = logLik(asp.gls), AIC=AICindep(asp.gls) )
```

```
##    logLik       AIC
## -169.9785  341.9570
```

Collecting all the AIC statistics Let's look at all the AIC statistics: I use a named vector to associate the AIC value and its model. round() rounds values to the specified # decimal places.

```
round(
  c(indep=AICindep(asp.gls), ar1=AIC(asp.ar1), ar1h = AIC(asp.ar1h),
  toep=AIC(asp.toep), ante=AIC(asp.ante), spExp=AIC(asp.spExp),
  un=AIC(asp.us) ),
  1)
```

```
## indep   ar1  ar1h  toep  ante spExp    un
## 342.0 333.1 335.8 334.3 333.7 333.1 335.7
```

The ar(1) model is the best fitting of the models being considered. It has the smallest AIC.

These values match those in my notes. There is one additional reason why there might be differences among packages: how to count variance parameters that are estimated as 0. SAS doesn't count a parameter when its variance component is 0, so the AIC statistics for the ar(1) and ar(1)+re are the same. R would count that random effect. If you look beyond the AIC statistics, i.e. at the estimated variance components, the data suggest no evidence for an additional random effect.

Which model comparison statistic should you use?

R does not compute small-sample corrected AIC, the AICc statistic. In general, BIC tends to select a simpler model than does AIC. That's because the penalty for an additional parameter is larger for BIC than for AIC. General opinion is that AIC is more likely to select the most appropriate covariance model.

Important information about reporting results

You have probably gotten the sense that results from fitting repeated measures models can differ (at least in details) among software implementations. That's absolutely right - there are many approximations and ad hoc choices made by all the softwares. There is no single correct way for these analyses. So what should a user do? **Report what software you used**