slim: Singular Linear Models

Daniel Farewell

2017-05-15

Introduction

The `slim` package fits singular linear models to longitudinal data. In particular, for given sets $\{x\}$ ($m \times p$ design matrices, where m is the number of observations for a particular subject), $\{V_0\}$ ($m \times m$ possibly *singular* covariance matrices), $\{V_*\}$ ($m \times m$ positive-definite covariance matrices), and $\{y\}$ (length m vectors containing the longitudinal data for each subject), `slim` computes $\hat{\beta} = \lim_{\omega \to 0} \hat{\beta}(\omega)$ where $\hat{\beta}(\omega)$ satisfies

$$\sum x^\top V(\omega)^{-1}\{y - x\hat{\beta}(\omega)\} = 0$$

and $V(\omega) = V_*\omega + V_0(1 - \omega)$.

For a few more details you can consult Farewell (2010). But for now let's load up the package and see how it works.

```r
library(slim)
```

```
## Loading required package: data.table
```

You'll see that `data.table` is a required package. That's because the main `slim` modelling function expects to receive the data in the form of a keyed `data.table`, not just an ordinary `data.frame`. The dialysis data, included with the package, has two keys: `id` (the individual or subject identifier) and `month` (the time variable).

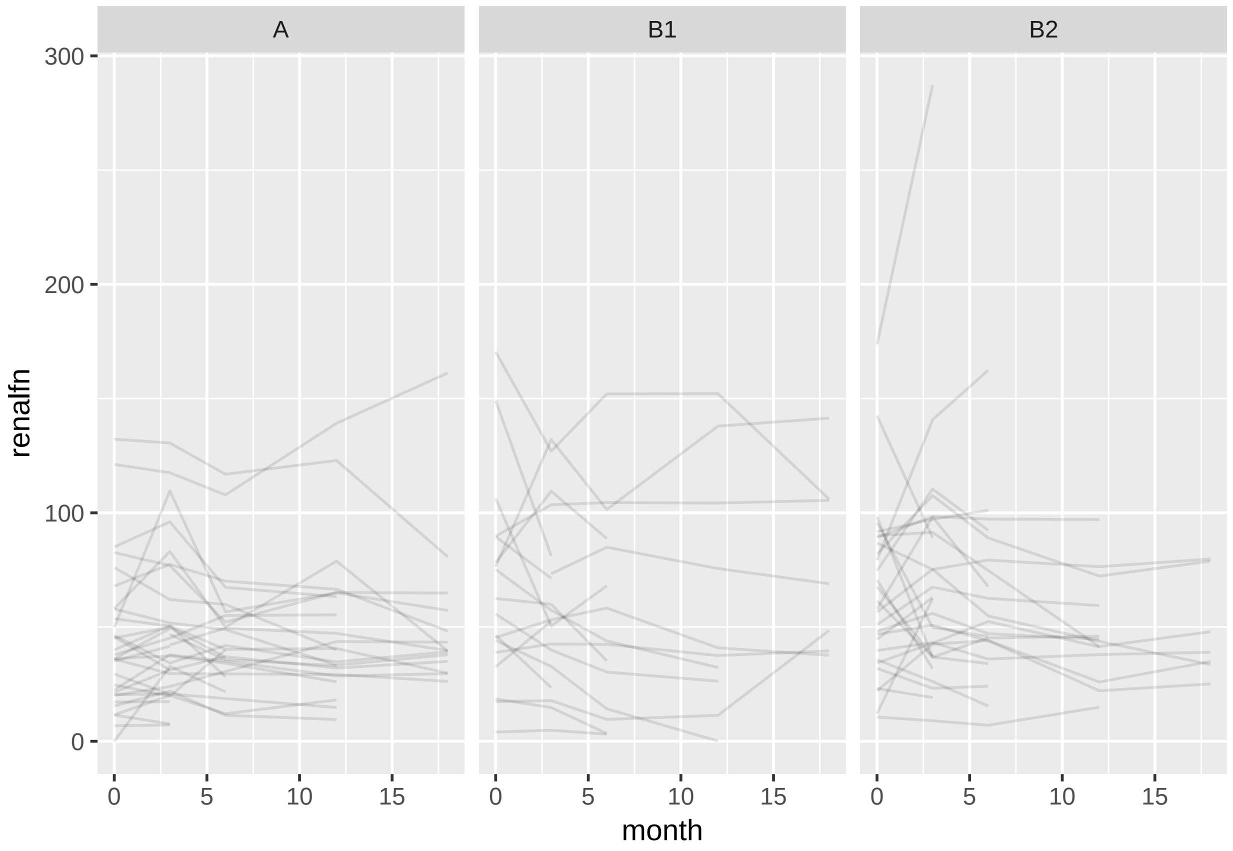
```r
data(dialysis)
str(dialysis)
```

```
## Classes 'data.table' and 'data.frame':   325 obs. of  5 variables:
##  $ id     : chr  "G-01-001" "G-01-001" "G-01-001" "G-01-001" ...
##  $ group  : chr  "A" "A" "A" "A" ...
##  $ vintage: int  596 596 596 596 1081 1081 1081 1081 655 655 ...
##  $ month  : int  3 6 12 18 0 3 6 12 0 3 ...
##  $ renalfn: num  42.2 49.5 47.2 39.7 36.1 ...
##  - attr(*, "sorted")= chr  "id" "month"
##  - attr(*, ".internal.selfref")=<externalptr>
```

The `slim` function makes use of these two keys in a couple different ways. First, by having these rather fundamental aspects of longitudinal data specified within the data, we can avoid having to ask for them each time we want to fit a model. Second, it ensures that the data are sorted in a consistent way (by individual, then time), so that associated quantities derived from calls to other modelling packages can be reliably re-aligned with the right subject and time.

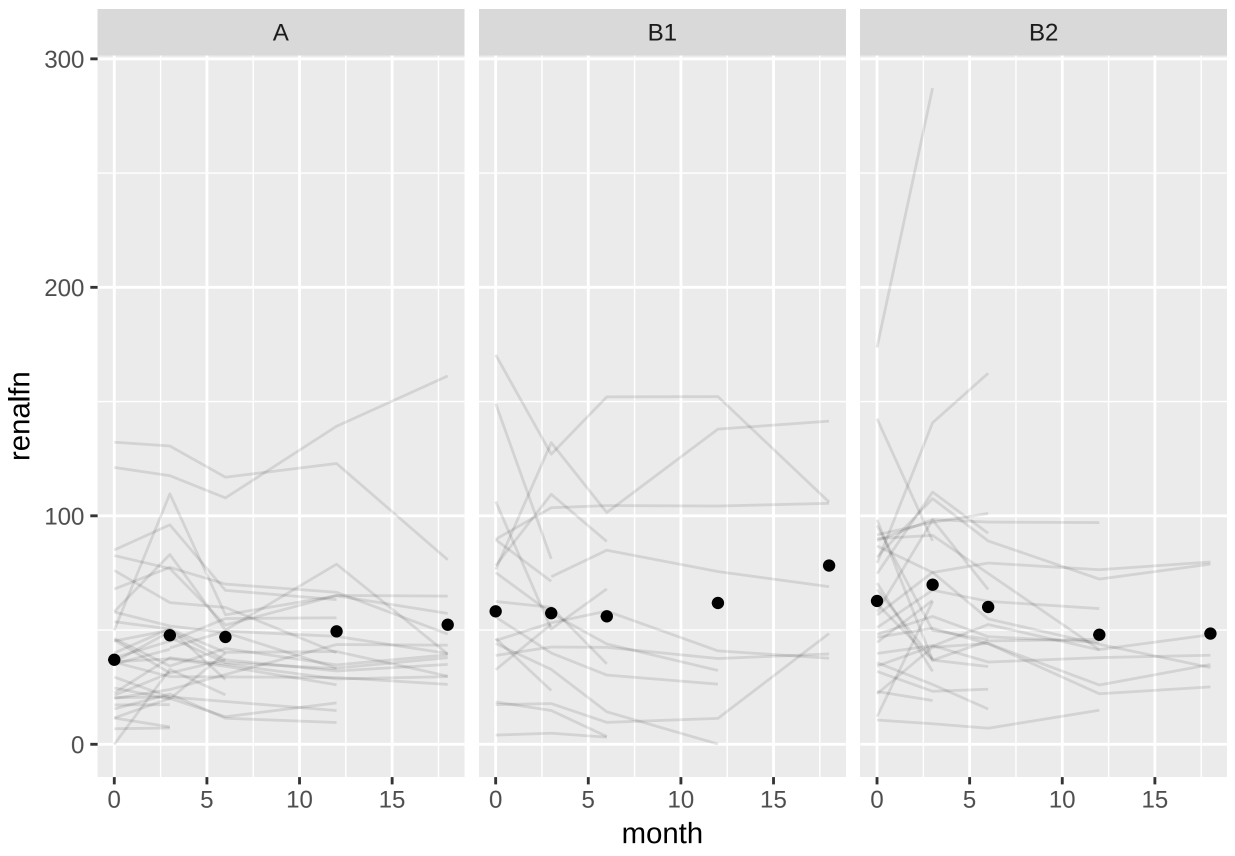
Let's look at the data.

```r
library(ggplot2)

print(p <- ggplot(dialysis, aes(x = month, y = renalfn, group = id)) +
        geom_line(alpha = 0.1) + facet_grid(~ group))
```



Most individuals have a generally downward trajectory, towards poorer renal function. However, if we overlay the average renal function at each measurement occasion, we see the opposite pattern:

```
naive_means <- dialysis[, list(id = group, renalfn = mean(renalfn)),
        by = list(group, month)]

print(p <- p + geom_point(data = naive_means))
```



A strong possibility, then, is that the individuals with poorest renal function are dropping out of the study earlier. This is the kind of effect `slim` models are designed to adjust for.

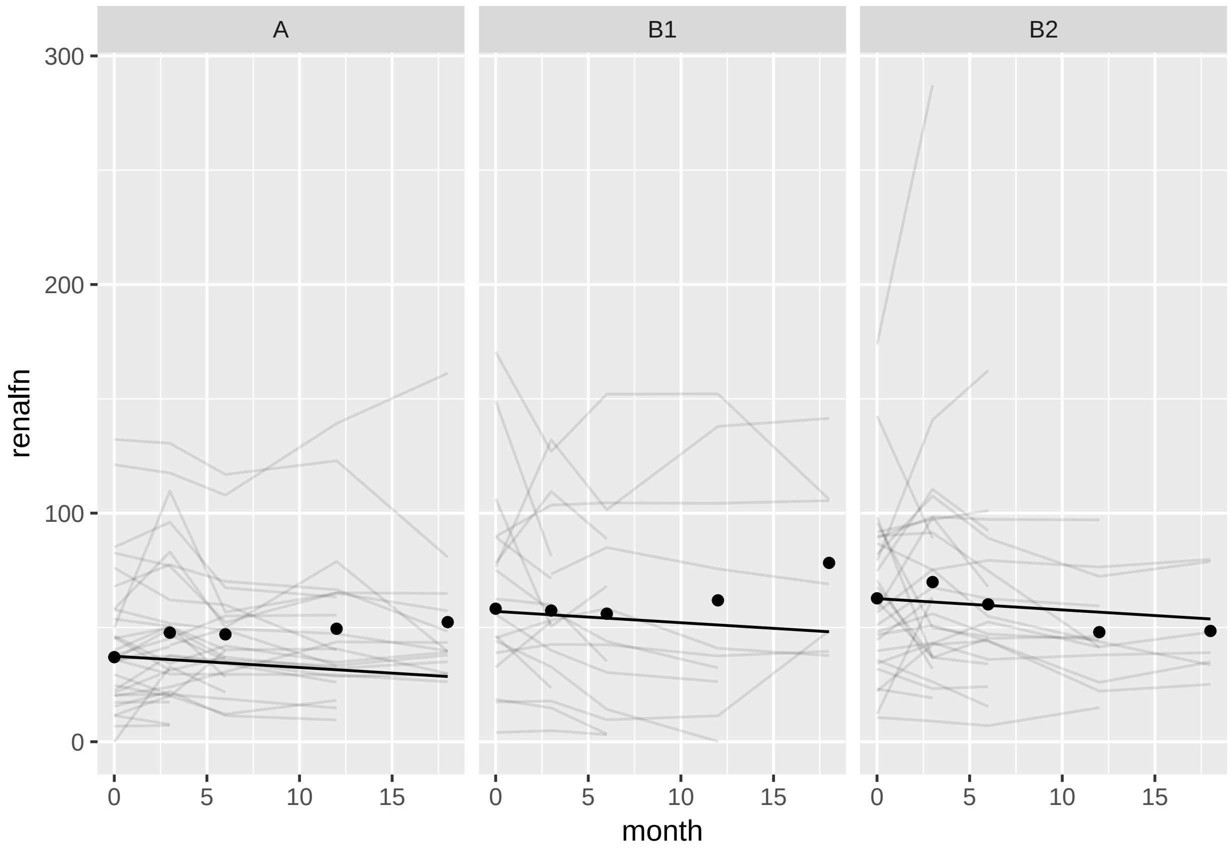
```
slim_basic_fit <- slim(renalfn ~ group + month, dialysis)

summary(slim_basic_fit)
```

```
## Singular Linear Model Fit
## Call:
## slim(formula = renalfn ~ group + month, data = dialysis)
##
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)    37.4212     4.5476   8.229  < 2e-16 ***
## groupB1        19.5866     8.4709   2.312 0.020766 *
## groupB2        25.1858     7.0129   3.591 0.000329 ***
## month          -0.4941     0.2694  -1.834 0.066664 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Notice that the sign of the estimated time trend (the coefficient associated with `month`) now matches the predominant individual pattern. Let's overlay this simple fit on the data:

```
# identify one fully observed individual in each group
dialysis[, fully_observed := (length(month) == 5), by = id]
group_reps <- dialysis[(fully_observed), list(id = min(id)), by = group]
setkey(group_reps, id)

dialysis[, slim_basic := fitted(slim_basic_fit)]
```

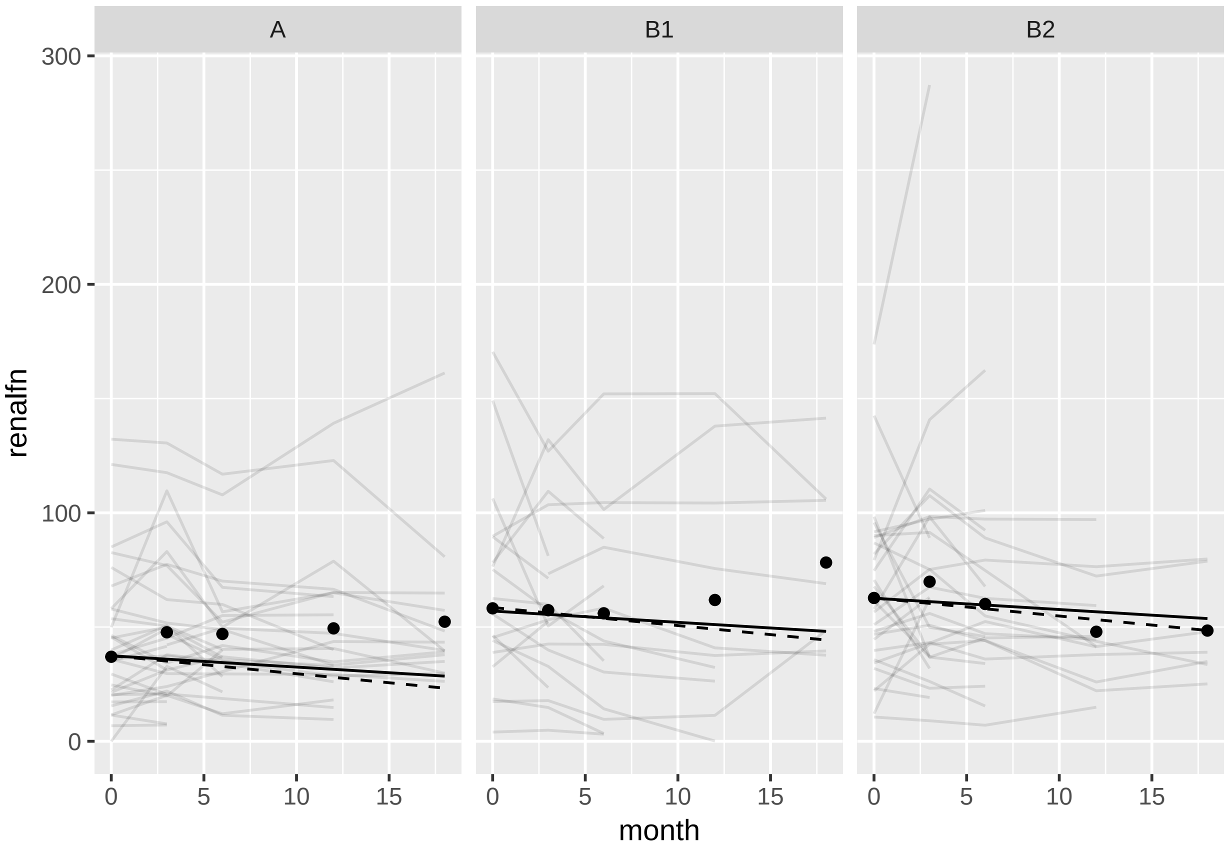
```
print(p <- p + geom_line(aes(y = slim_basic), data = dialysis[group_reps]))
```



The `slim` function uses two "working" covariance matrices for each individual: the initial, positive definite, matrix V_* and the limiting, possibly singular, matrix V_0. The default value of V_0 is the singular matrix of ones, but we can change this by passing a formula to the `limit` argument:

```
slim_intercept_slope_fit <- slim(renalfn ~ group + month, dialysis,
        limit = ~ 1 + month)

dialysis[, slim_intercept_slope := fitted(slim_intercept_slope_fit)]
```

```
print(p <- p + geom_line(aes(y = slim_intercept_slope),
                data = dialysis[group_reps], linetype = "dashed"))
```



For each individual, the matrix V_0 is constructed as the product of the columns of the model matrix generated by the `limit` formula with the transpose of the same model matrix. For example, for an individual with observations at months 0, 3, and 18,

```
small_example <- data.table(month = c(0, 3, 18))
z <- model.matrix(~ 1, small_example)
z %*% t(z)
```

```
##   1 2 3
## 1 1 1 1
## 2 1 1 1
## 3 1 1 1
```

```
z <- model.matrix(~ 1 + month, small_example)
z %*% t(z)
```

```
##   1  2   3
## 1 1  1   1
## 2 1 10  55
## 3 1 55 325
```

There are several ways to specify V_*. The default (`covariance = "randomwalk"`) is to use the covariance matrix of a random walk with unit-variance increments, that is $\text{cov}(y_j, y_k) = \min(j, k)$, but but other options include the identity matrix (`covariance = "identity"`), the covariance of Brownian motion ($\text{cov}(y_j, y_k) = \min(t_j, t_k)$, `covariance = "brownian"`) or the symmetric Pascal matrix given by

$$\text{cov}(y_j, y_k) = \binom{j+k-2}{j-1}$$

and specified by `covariance = "pascal"`. All these matrices are employed for their *structure* and make no reference to the actual variance of the data. When using such working covariance structures, therefore, it is important to use empirical standard errors to obtain reasonable estimates of parameter uncertainty:

```r
vcov(slim_basic_fit)
```

```
##             (Intercept)     groupB1     groupB2       month
## (Intercept)  20.6807381 -20.6001138 -20.6735421 -0.10931842
## groupB1     -20.6001138  71.7563078  20.6248393 -0.33281974
## groupB2     -20.6735421  20.6248393  49.1804719  0.11089517
## month        -0.1093184  -0.3328197   0.1108952  0.07258216
```

```r
vcov(slim_basic_fit, empirical = FALSE) # a bad idea for unmodelled covariances!
```

```
##               (Intercept)       groupB1       groupB2         month
## (Intercept)  2.381464e-02 -2.380832e-02 -0.0238119552 -3.583074e-05
## groupB1     -2.380832e-02  5.322157e-02  0.0238089517 -8.430761e-06
## groupB2     -2.381196e-02  2.380895e-02  0.0488106787  1.701960e-05
## month       -3.583074e-05 -8.430761e-06  0.0000170196  2.508151e-04
```

```r
summary(slim_basic_fit)
```

```
## Singular Linear Model Fit
## Call:
## slim(formula = renalfn ~ group + month, data = dialysis)
##
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  37.4212     4.5476   8.229  < 2e-16 ***
## groupB1      19.5866     8.4709   2.312 0.020766 *
## groupB2      25.1858     7.0129   3.591 0.000329 ***
## month        -0.4941     0.2694  -1.834 0.066664 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
summary(slim_basic_fit, empirical = FALSE) # still a bad idea
```

```
## Singular Linear Model Fit
## Call:
## slim(formula = renalfn ~ group + month, data = dialysis)
##
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 37.42124    0.15432   242.5   <2e-16 ***
## groupB1     19.58658    0.23070    84.9   <2e-16 ***
## groupB2     25.18579    0.22093   114.0   <2e-16 ***
## month       -0.49408    0.01584   -31.2   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Use of `empirical = FALSE` is intended for cases when the covariance V_* has itself been modelled from the same data. This can be accomplished by passing another model fit to the `covariance` argument. Methods exist to handle fits from `lmer` (package `lme4`, class `lmerMod`) and `jmcm` (package `jmcm`, class `jmcmMod`). For example, using a mixed effects model fit by `lmer` with a random intercept and slope, we can pass the estimated covariance matrices to `slim` via the `covariance` argument:

```r
library(lme4)
```

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

```r
lmer_fit <- lmer(renalfn ~ group + month + (1 + month | id), dialysis)
slim_lmer_fit <- slim(renalfn ~ group + month, dialysis, covariance = lmer_fit)

summary(slim_lmer_fit)
```

```
## Singular Linear Model Fit
## Call:
## slim(formula = renalfn ~ group + month, data = dialysis, covariance = lmer_fit)
##
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  39.2079     4.4639   8.783  < 2e-16 ***
## groupB1      16.0911     8.0633   1.996 0.045977 *
## groupB2      25.5848     7.4128   3.451 0.000558 ***
## month        -0.5703     0.2225  -2.563 0.010385 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
summary(slim_lmer_fit, empirical = FALSE) # this now makes sense
```

```
## Singular Linear Model Fit
## Call:
## slim(formula = renalfn ~ group + month, data = dialysis, covariance = lmer_fit)
##
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  39.2079     5.5127   7.112 1.14e-12 ***
## groupB1      16.0911     8.2660   1.947  0.05158 .
## groupB2      25.5848     7.8861   3.244  0.00118 **
## month        -0.5703     0.2242  -2.544  0.01096 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The `jmcm` function uses a more comprehensive formula object to specify the response, subject identifier, time variable and mean and covariance structures. These models provide even more flexible ways to estimate covariances:

```r
library(jmcm)

jmcm_fit <- jmcm(renalfn | id | month ~ group | 1, data = dialysis,
        triple = rep(2L, 3), cov.method = "mcd")
slim_jmcm_fit <- slim(renalfn ~ group + month, dialysis, covariance = jmcm_fit)

summary(slim_jmcm_fit)
```

```
## Singular Linear Model Fit
## Call:
## slim(formula = renalfn ~ group + month, data = dialysis, covariance = jmcm_fit)
##
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  41.8482     4.9235   8.500  < 2e-16 ***
## groupB1      15.5031     7.9189   1.958  0.05026 .
## groupB2      23.1299     7.9796   2.899  0.00375 **
## month        -0.6052     0.2024  -2.990  0.00279 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

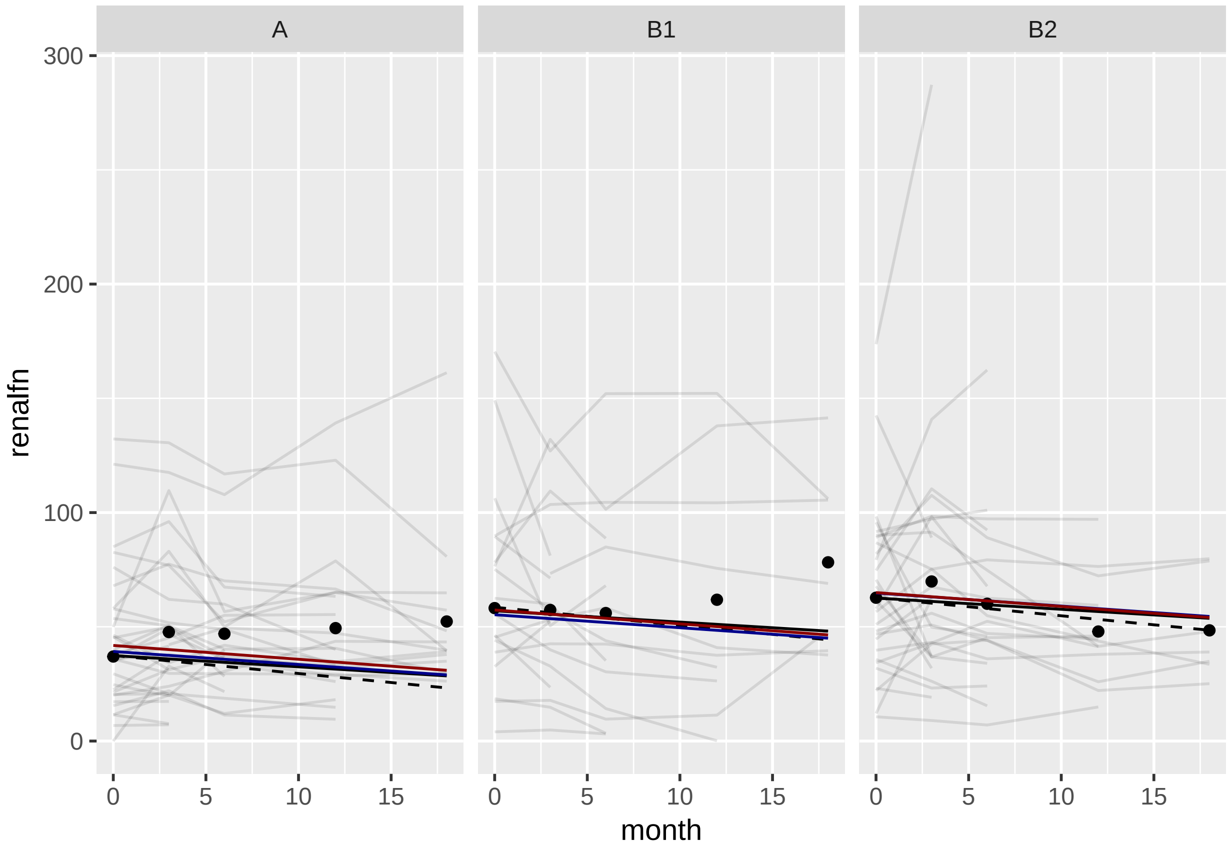
```r
summary(slim_jmcm_fit, empirical = FALSE)
```

```
## Singular Linear Model Fit
## Call:
## slim(formula = renalfn ~ group + month, data = dialysis, covariance = jmcm_fit)
##
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  41.8482     4.9861   8.393  < 2e-16 ***
## groupB1      15.5031     7.6808   2.018  0.04355 *
## groupB2      23.1299     7.2582   3.187  0.00144 **
## month        -0.6052     0.1770  -3.418  0.00063 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We can add these fits to our plots:

```r
dialysis[, slim_lmer := fitted(slim_lmer_fit)]
dialysis[, slim_jmcm := fitted(slim_jmcm_fit)]
```

```r
print(p <- p + geom_line(aes(y = slim_lmer), data = dialysis[group_reps],
                colour = "darkblue") +
        geom_line(aes(y = slim_jmcm), data = dialysis[group_reps],
                colour = "darkred"))
```

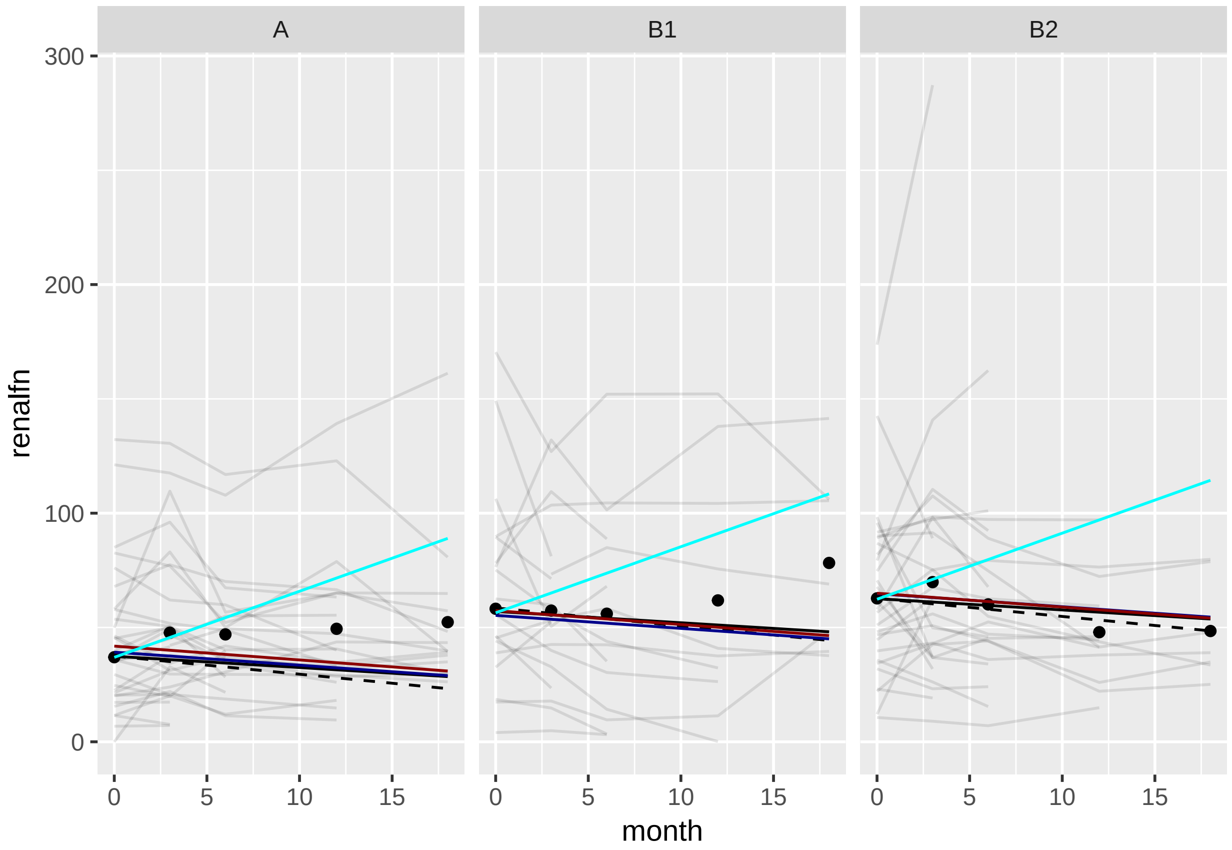


There is reasonable consistency across all these `slim` model fits. It is possible to demonstrate sensitivity to choice of covariance using more extreme structures (the Pascal matrix is very extreme):

```r
slim_pascal_fit <- slim(renalfn ~ group + month, dialysis,
        covariance = "pascal")
```

```
dialysis[, slim_pascal := fitted(slim_pascal_fit)]
```

```
print(p <- p + geom_line(aes(y = slim_pascal), data = dialysis[group_reps],
                 colour = "cyan"))
```



Notice the greatly increased standard error associated with the time trend in this fit:

```
extract_se <- function(fit) sqrt(diag(vcov(fit)))
sapply(list(se_basic = slim_basic_fit, se_pascal = slim_pascal_fit), extract_se)
```

```
##              se_basic se_pascal
## (Intercept) 4.5476080  4.482040
## groupB1     8.4709095  8.526003
## groupB2     7.0128790  6.993315
## month       0.2694108  1.892062
```

This sensitivity is indicative of the bias-variance trade-off associated with choice of covariance structure. Roughly, the Pascal structure allows the timings of observations to depend on a random intercept, random slope, random curvature and so on to higher derivatives as far as the number of observations on an individual allow. This consistency across a wider class of models comes at the price of greatly increased variance, although observe that the 'baseline' parameters of the three groups are basically unaffected, relating as they do to the time when all individuals remain in the study.

We end on this cautionary note: it does matter what covariance matrices you choose! Our (limited) experience suggests that random walk or well-modelled covariance structures lead to "sensible" answers, but we would greatly value hearing your insights or experience, too.

References

Farewell, D.M. 2010. “Marginal Analyses of Longitudinal Data with an Informative Pattern of Observations.” *Biometrika*.