The great random effects debate rises again

Function for simulating observations along a line from multiple gruops

```
### Function for generating (simulating) sample observations across multiple groups
make_points = function(x_sd = 0, y_sd = 0, npoints, ngroups, x_width, group_spacing) {

  # x_width: how wide a range of x values should be spanned by all the points?
  # group_spacing: what percentage of the x_width goes toward spacing between groups? Negative for over

  # Assume a 1:1 straight line
  intercept = 0
  slope = 1

  ## Divide the requested n points among all groups
  n_pts_per_grp = npoints / ngroups
  remainder = n_pts_per_grp - floor(n_pts_per_grp) # used to randomly assign an integer number of point

  x_width_per_group = x_width/ngroups
  spacing_per_group = x_width_per_group * group_spacing

  ## Simulate points for each group

  group_start = 0 # to keep track of x range from one group to the next
  points = list() # to hold the points from all groups

  for(i in 1:ngroups) {

      group_name = as.character(i)

      # get x range
      xmin = group_start
      xmax = xmin + x_width_per_group

      # prep for next group
      group_start = xmax + spacing_per_group

      # how many points in this group?
      npoints_group = n_pts_per_grp + rbinom(1,1,remainder)

      x_vals_noerr = seq(from = xmin, to = xmax, length.out = npoints_group)

      points_group = data.frame(x_vals_noerr, group = group_name)
      points[[i]] = points_group
  }

  points = bind_rows(points)
```

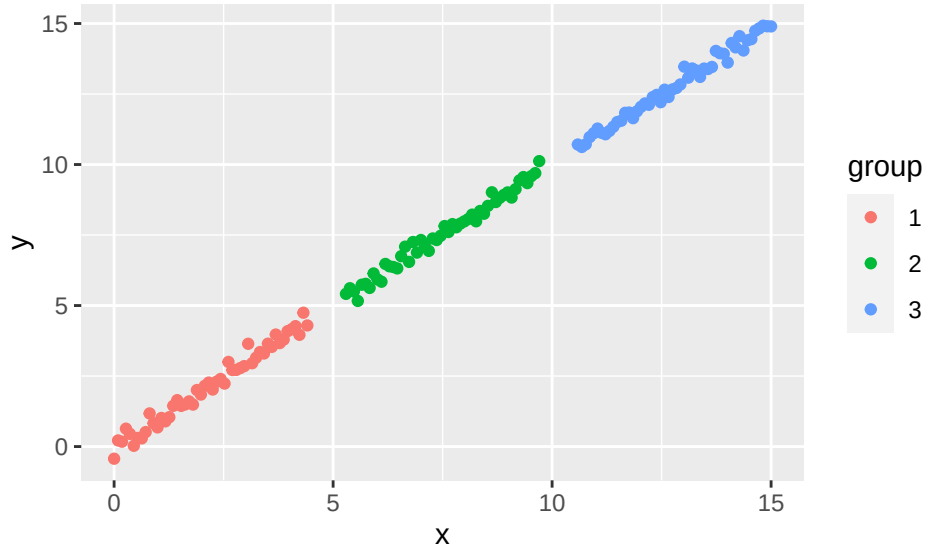
```
  # scale the x axis range to the range specified, and add error and calc y values and add err
  max_x = max(points$x_vals_noerr)
  points = points %>%
    mutate( x_vals_noerr = x_vals_noerr / max_x * x_width,
            x_vals_err = x_vals_noerr + rnorm(n=length(x_vals_noerr), sd = x_sd),
            y_vals_noerr = x_vals_noerr * slope + intercept,
            y_vals_err = y_vals_noerr + rnorm(n=length(y_vals_noerr), sd = y_sd)
    ) %>%
    select(x = x_vals_err, y = y_vals_err, group)


  return(points)
}
```

Simulating three groups of points which fall on the same line

First, only minor error (SD = 0.2) in the response variable (residual error)

```
points = make_points(x_sd = 0, y_sd = 0.2, npoints = 150, ngroups = 3, x_width = 15, group_spacing = 0.
```

```
ggplot(points,aes(x=x,y=y,color=group)) +
  geom_point()
```




```
m = lmer(y ~ x + (1| group), points)
fixef(m)
```

Fit the model and get the overall slope and intercept and the group-level intercepts

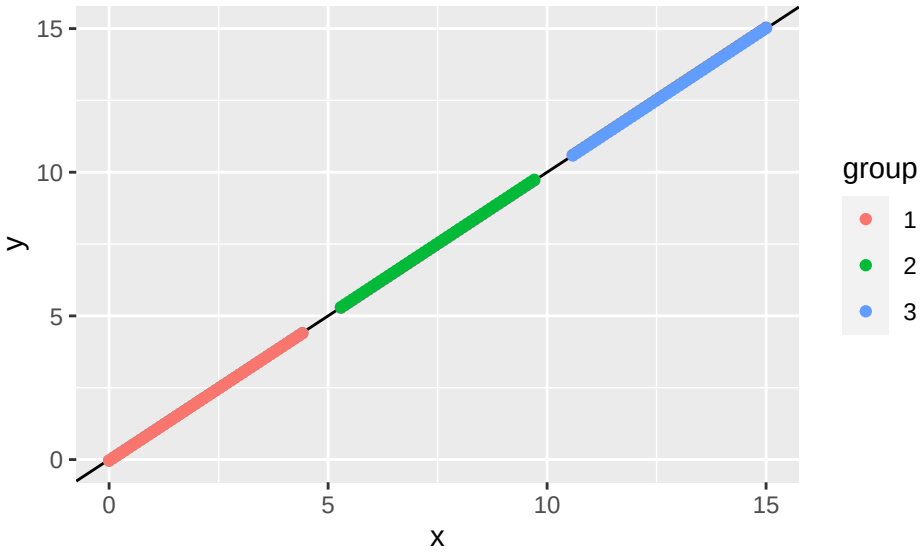
```
## (Intercept)           x
## -0.03196358  1.00429347
```

```
ranef(m)$group
```

```
##    (Intercept)
## 1 -0.005199037
## 2  0.022327531
## 3 -0.017128493
```

Here, the random intercepts are not absorbing the group-level intercept; it is being explained entirely by the fixed effect slope.

```
points$pred_conditional = predict(m) # predictions made for the actual groups
points$pred_fixed = predict(m, re.form = NA) # predictions made for the overall (fixed-effect) pattern
ggplot(points,aes(x=x,y=pred_conditional,color=group)) +
    geom_abline(intercept=0,slope=1)  +
    geom_point(aes(y = pred_fixed), color="grey50") +
    geom_point() +
    labs(y = "y")
```



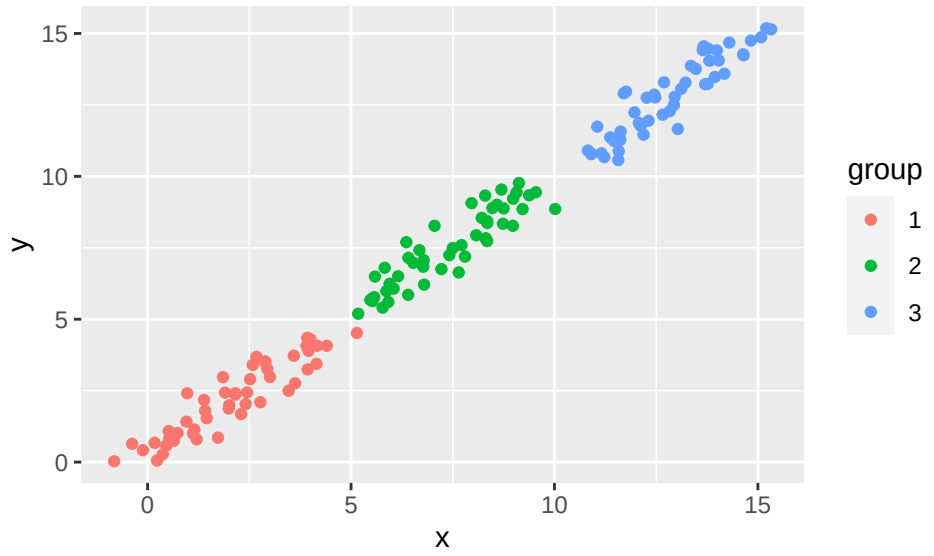
Plot model predictions

The black line is the underlying pattern that was simulated. Predictions look good for both conditional and fixed-effects only (they are on top of each other)

Adding (measurement) error (SD = 0.5) in the predictor variable.

Again, only minor error (SD = 0.2) in the response variable

```
points = make_points(x_sd = 0.5, y_sd = 0.2, npoints = 150, ngroups = 3, x_width = 15, group_spacing =

ggplot(points,aes(x=x,y=y,color=group)) +
  geom_point()
```




```
m = lmer(y ~ x + (1| group), points)
```

Fit the model and get the overall slope and intercept and the group-level intercepts

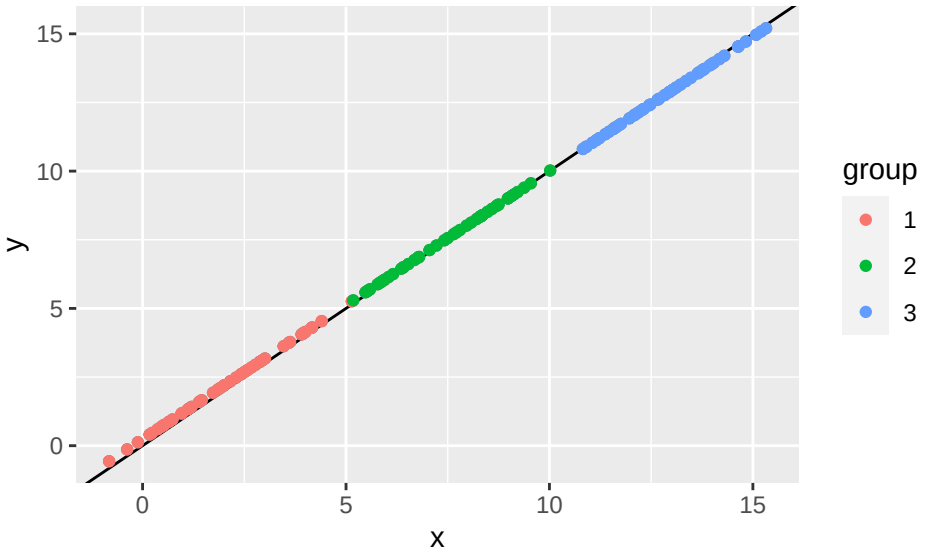
```
## boundary (singular) fit: see ?isSingular
```

```
fixef(m)
ranef(m)$group
```

```
## (Intercept)           x
##   0.2372217   0.9766137
##   (Intercept)
## 1           0
## 2           0
## 3           0
```

Now the overall slope is less and the random intercepts are absorbing some of the group-level variation.

```
points$pred_conditional = predict(m) # predictions made for the actual groups
points$pred_fixed = predict(m, re.form = NA) # predictions made for the overall (fixed-effect) pattern
ggplot(points,aes(x=x,y=pred_conditional,color=group)) +
    geom_abline(intercept=0,slope=1)  +
    geom_point(aes(y = pred_fixed), color="grey50") +
    geom_point() +
    labs(y = "y")
```



Plot model predictions

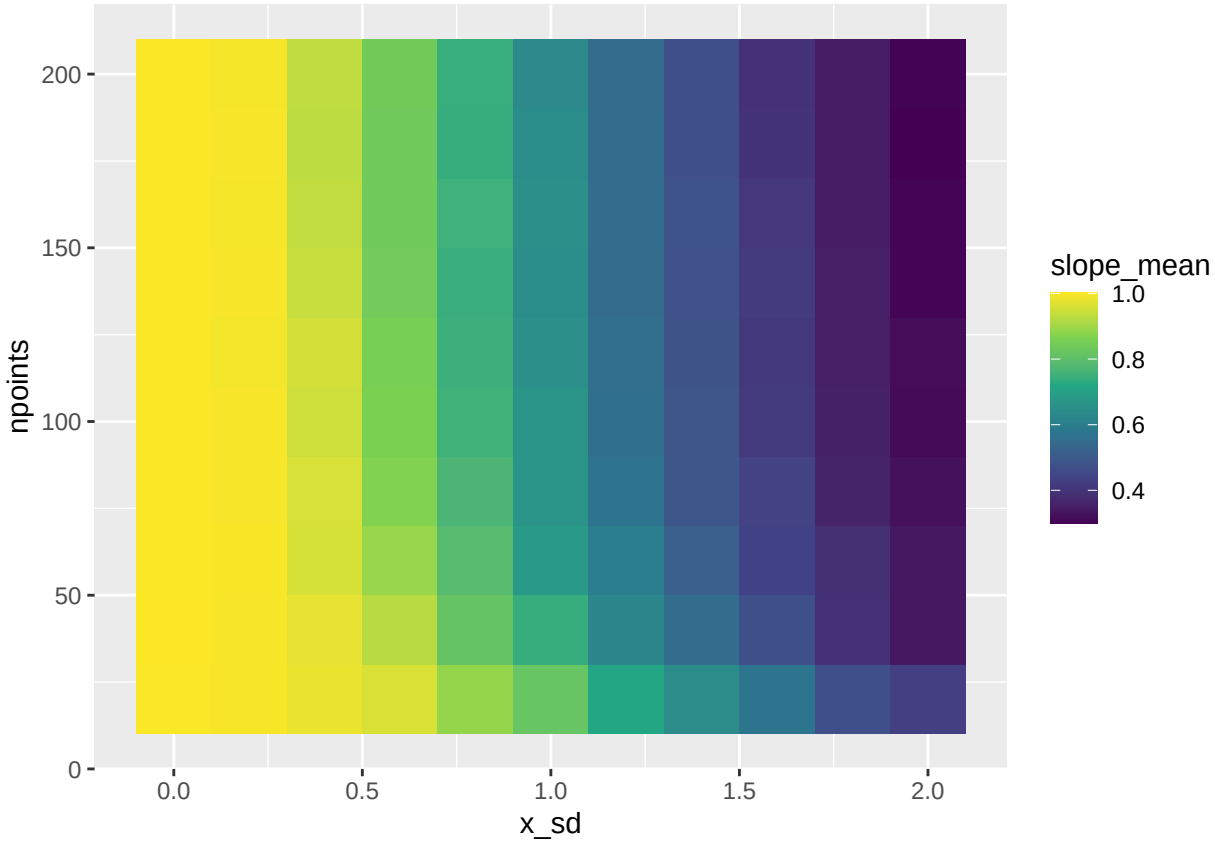
The conditional (including random intercept) predictions coarsely track the true pattern across groups, but the slope is off. The fixed-effects-only prediction has a shallower slope than the true pattern. This seems to be more problematic the more x variance there is, but it doesn't seem to depend on y (residual) variance.

How does the underestimation of the overall slope depend on x (predictor) measurement error and sample size?

Still assuming a 1:1 straight line is the true pattern and y error is 0.2 SD

```
### Function for getting the overall model slope for a given sample size and x standard deviation

slope_fun = function(npoints, x_sd) {

  points = make_points(x_sd = x_sd, y_sd = 0.2, npoints = npoints, ngroups = 3, x_width = 15, group_spa

  # Fit model and get fixed-effect slope
  m = lmer(y ~ x + (1| group), points)
  slope = fixef(m)[["x"]]

}

### Get this slope for pairwise combinations of sample size and y_sd
npoints = seq(20,by=20, length.out = 10)
x_sd = seq(0,by=0.2, length.out = 11)

params = expand_grid(npoints,x_sd)

# repeat it 100 times to get mean and variance of the slope-reduction effect
params = map_dfr(seq_len(50), function(x) params)

params_to_pass = list(npoints = as.list(params$npoints), x_sd = as.list(params$x_sd))
slope_pred = future_pmap_dbl(params_to_pass, slope_fun)

params$slope = slope_pred
```

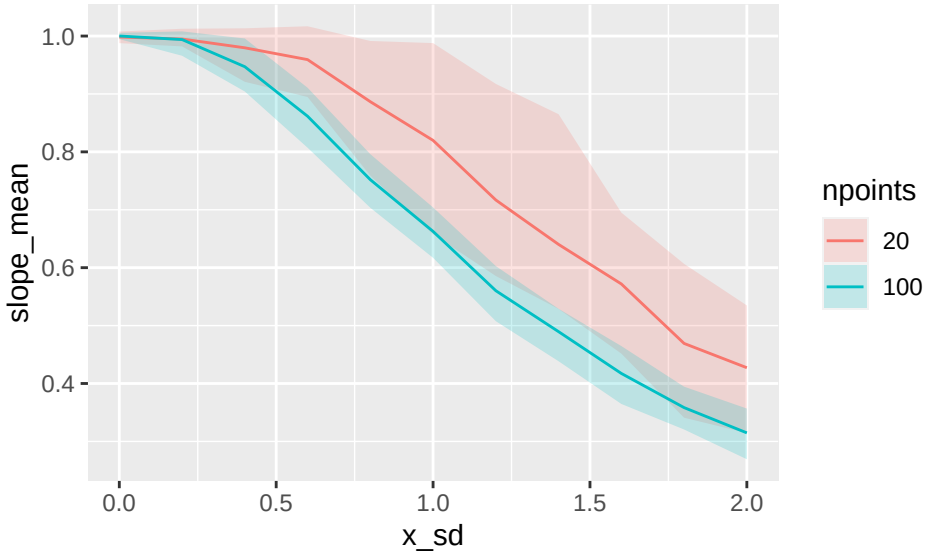
```
# summarize (get mean and 10th-90th percentiles of predicted slope)
params_summ = params %>%
  group_by(x_sd, npoints) %>%
  summarize(slope_mean = mean(slope),
            slope_lwr = quantile(slope,0.10),
            slope_upr = quantile(slope,0.90))

ggplot(params_summ,aes(x=x_sd, y = npoints, fill = slope_mean)) +
  geom_tile() +
  scale_fill_viridis_c()
```



So when the measurement error on the predictor variable is 1 SD, the slope is (on average) reduced from 1.0 to about 0.6. This appears largely independent of smaple size, except at very smallest sample sizes (n = 5 samples per group).

How variable is this slope reduction effect as a consequence of random variation in the sample?

```
params_summ_plot = params_summ %>%
  filter(npoints %in% c(20, 100)) %>%
  mutate(npoints = as.factor(npoints))


ggplot(params_summ_plot, aes(x = x_sd, y = slope_mean, color=npoints, fill = npoints)) +
    geom_ribbon(aes(ymin = slope_lwr, ymax = slope_upr), alpha = 0.2, color=NA) +
    geom_line()
```



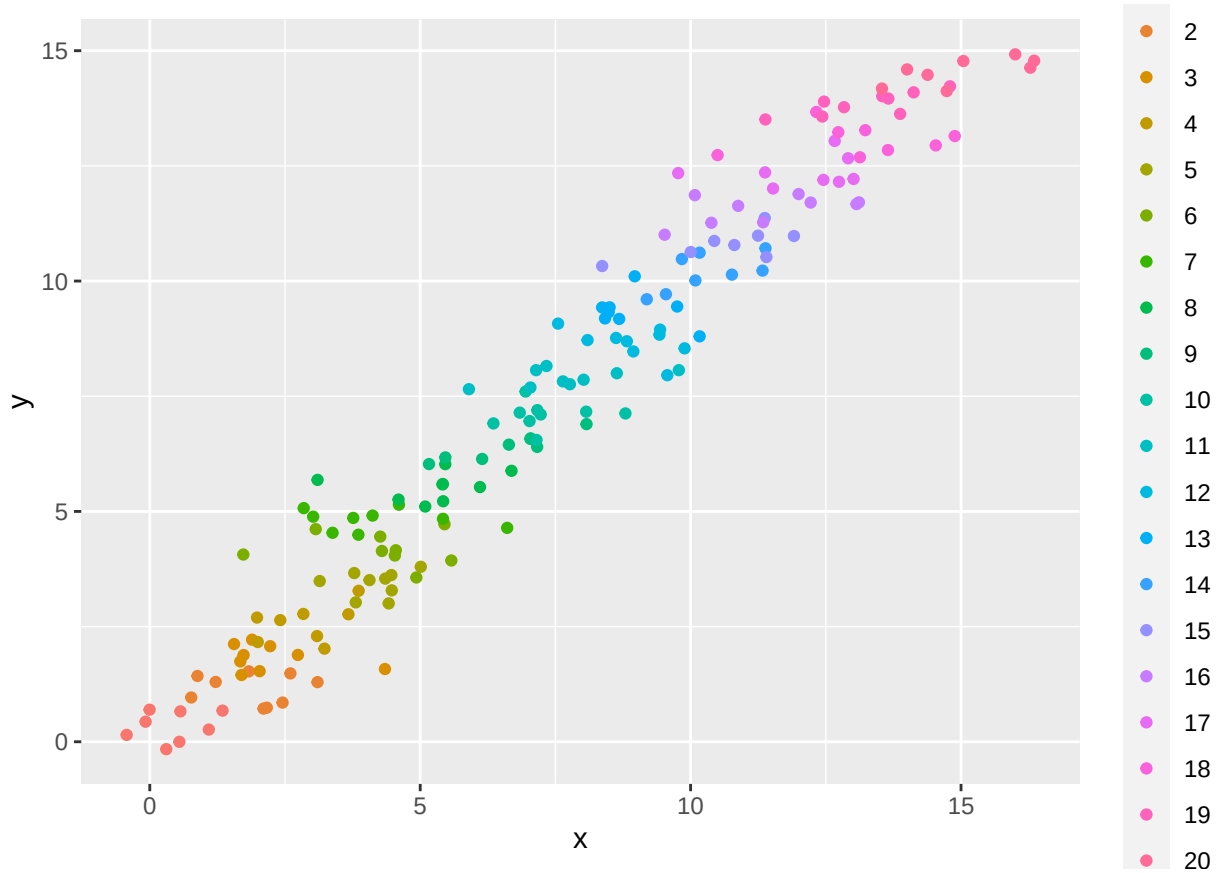
Not a huge amount of variation from one random simulation of points to the next. The variation is greater when the sample size is smaller.

How much does the slope reduction depend on the number of groups?

Assuming x_sd = 1.0

Here's an example of 20 groups with 150 points (same as the earlier datasets shown)

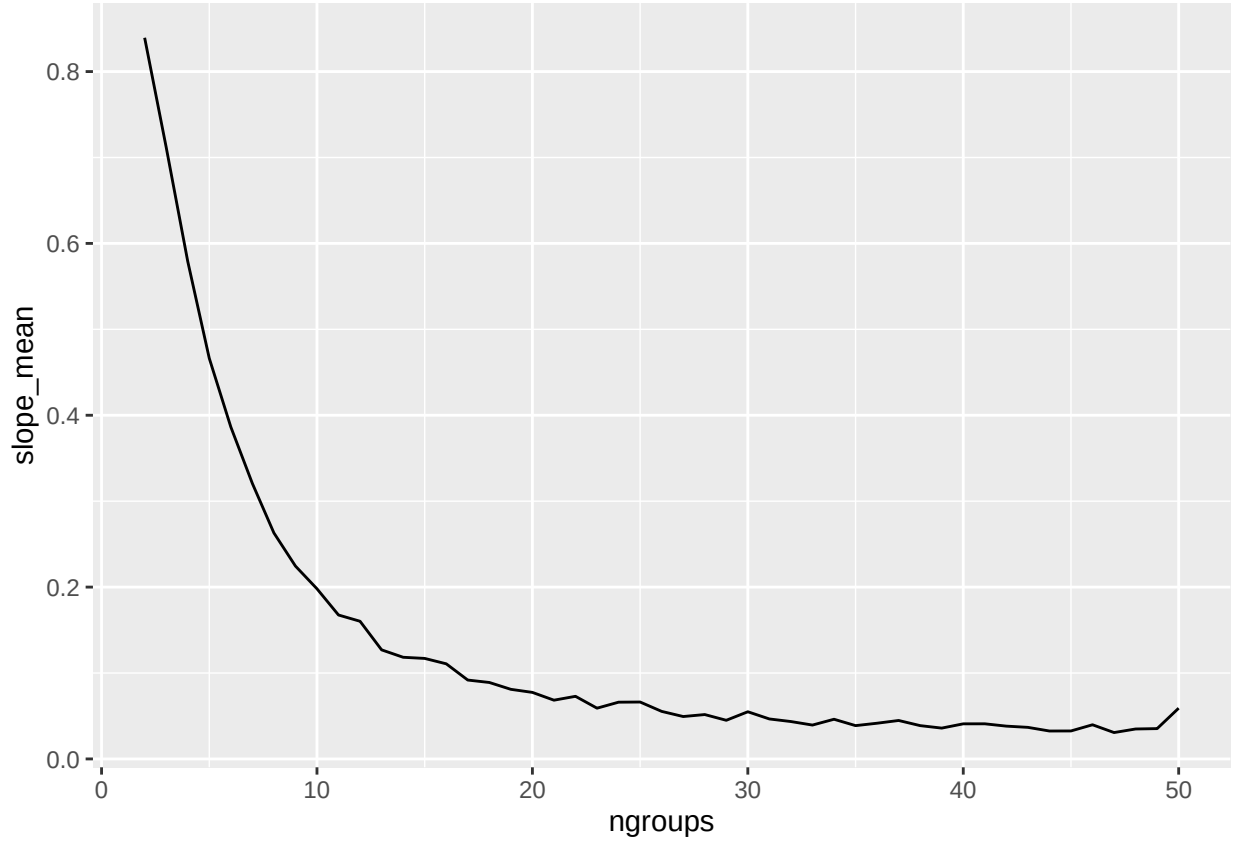
```
  points = make_points(x_sd = 1.0, y_sd = 0.2, npoints = 150, ngroups = 20, x_width = 15, group_spacing

ggplot(points,aes(x=x,y=y,color=group)) +
  geom_point()
```



Test over a range of n groups. Same total number of points each time (just divide the points evenly among the groups – as in figure above where the same number of 150 points was divided among 20 groups)

```
### Function for getting the overall model slope for a given sample size and x standard deviation

slope_fun = function(ngroups) {

  points = make_points(x_sd = 1.0, y_sd = 0.2, npoints = 150, ngroups = ngroups, x_width = 15, group_spa

  # Fit model and get fixed-effect slope
  m = lmer(y ~ x + (1| group), points)
  slope = fixef(m)[["x"]]

}

### Get this slope for a range of ngroups
ngroups = 2:50

# repeat it 100 times to get mean and variance of the slope-reduction effect
ngroups = rep(ngroups,50)

slope_pred = future_map_dbl(ngroups, slope_fun)

params = data.frame(ngroups,slope = slope_pred)

# summarize (get mean and 10th-90th percentiles of predicted slope)
```

```r
params_summ = params %>%
  group_by(ngroups) %>%
  summarize(slope_mean = mean(slope),
            slope_lwr = quantile(slope,0.10),
            slope_upr = quantile(slope,0.90))

ggplot(params_summ,aes(x=ngroups, y = slope_mean)) +
  #geom_ribbon(aes(xmin = slope_lwr, xmax = slope_upr),alpha = 0.5) +
  geom_line()
```

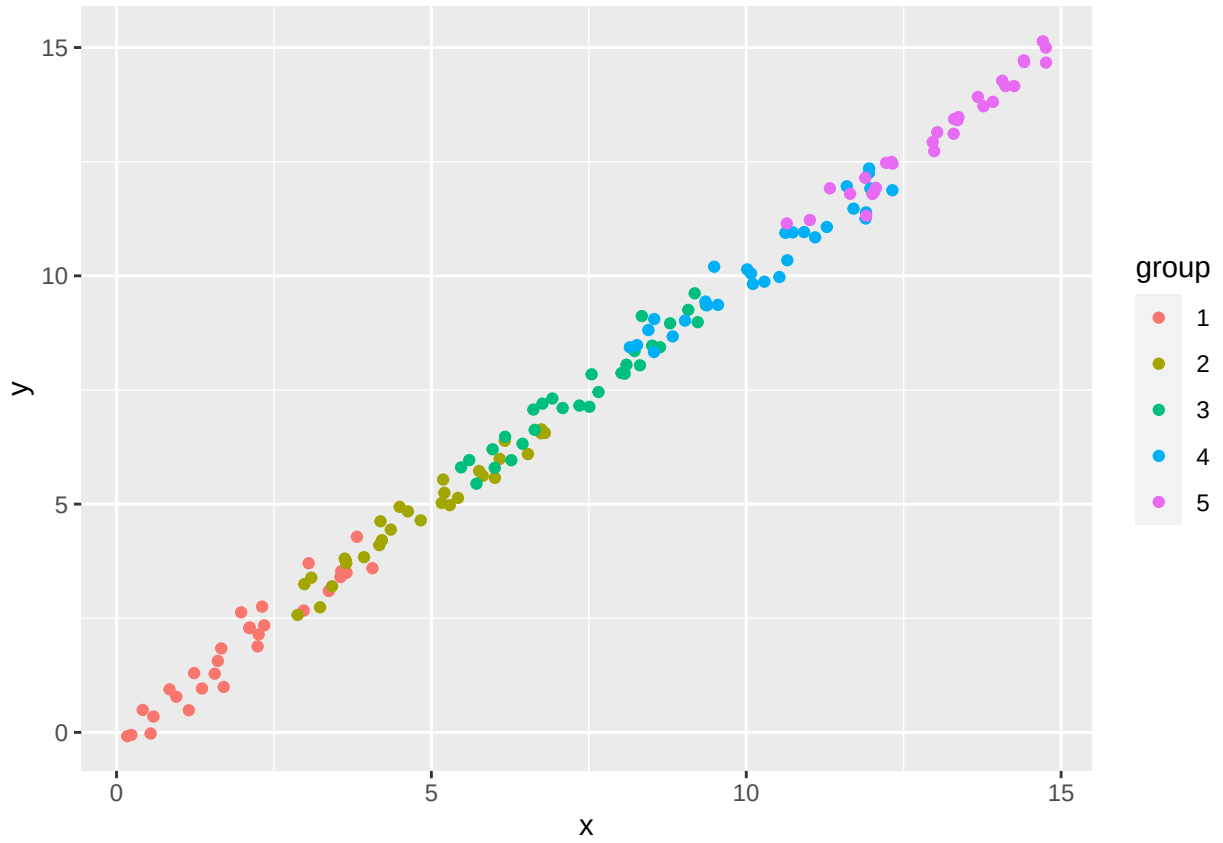


The more groups, the more the slope is reduced – with 20 groups, over 90% of the overall variation gets absorbed by the group intercepts

What if there is overlap between the groups?

Here's an example of overlap

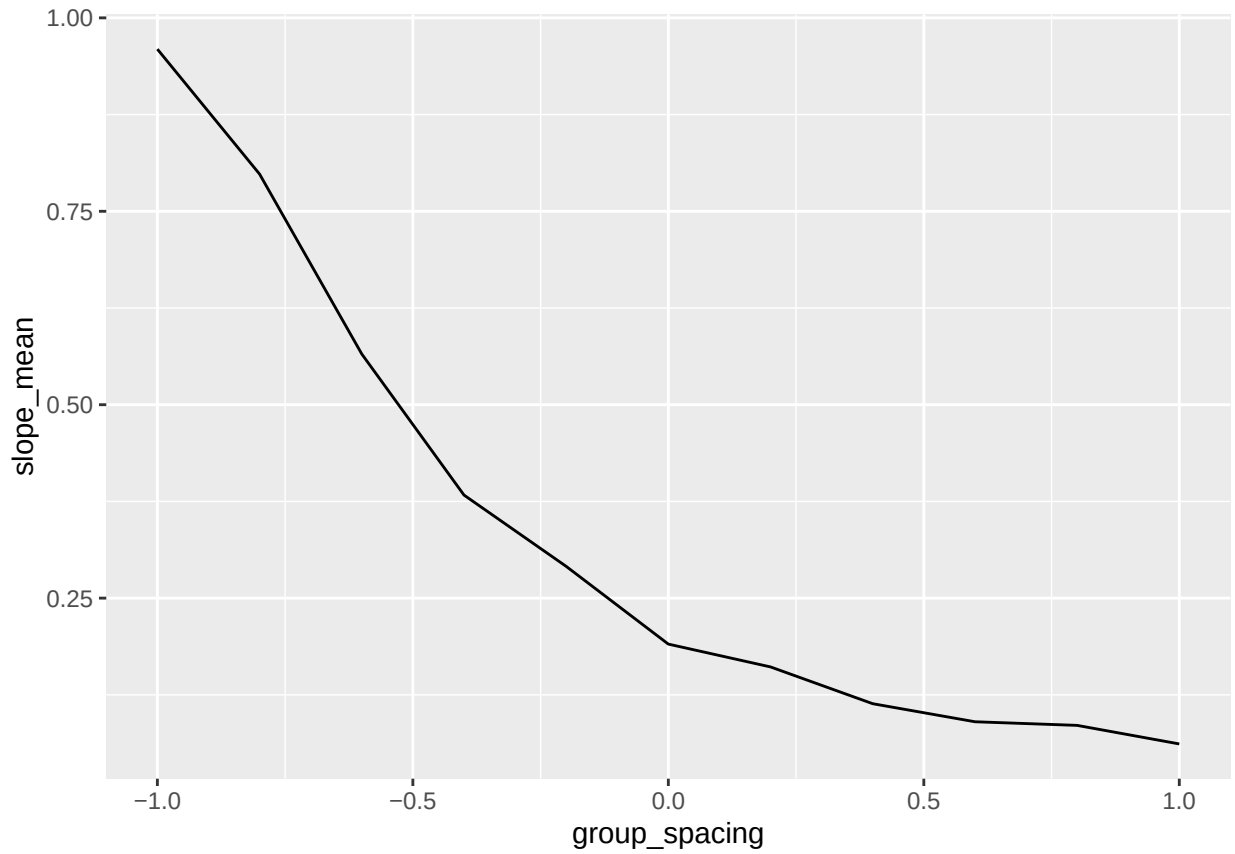
```r
  points = make_points(x_sd = 0.2, y_sd = 0.2, npoints = 150, ngroups = 5, x_width = 15, group_spacing =

ggplot(points,aes(x=x,y=y,color=group)) +
  geom_point()
```



Test along a range in spacing. Assuming 10 groups.

```
### Function for getting the overall model slope for a given sample size and x standard deviation

slope_fun = function(group_spacing) {

  points = make_points(x_sd = 1.0, y_sd = 0.2, npoints = 150, ngroups = 10, x_width = 15, group_spacing

  # Fit model and get fixed-effect slope
  m = lmer(y ~ x + (1| group), points)
  slope = fixef(m)[["x"]]

}

### Get this slope for a range of ngroups
group_spacing = seq(from=-1, to = 1, by = 0.2)

# repeat it 100 times to get mean and variance of the slope-reduction effect
group_spacing = rep(group_spacing,20)

slope_pred = future_map_dbl(group_spacing, slope_fun)

params = data.frame(group_spacing,slope = slope_pred)

# summarize (get mean and 10th-90th percentiles of predicted slope)
params_summ = params %>%
```

```
  group_by(group_spacing) %>%
  summarize(slope_mean = mean(slope),
            slope_lwr = quantile(slope,0.10),
            slope_upr = quantile(slope,0.90))

ggplot(params_summ,aes(x=group_spacing, y = slope_mean)) +
  #geom_ribbon(aes(xmin = slope_lwr, xmax = slope_upr), alpha = 0.5) +
  geom_line()
```



With complete overlap (spacing = -1), there is very little absorption by the random intercepts. With 50% overlap between adjacent groups, the slope is still reduced by > 50% due to absorption by random effects.

Does the same thing happen using BRMS?

Assuming x_sd = 1, group spacing = 0, 10 groups.

```
  points = make_points(x_sd = 1, y_sd = 0.2, npoints = 150, ngroups = 10, x_width = 15, group_spacing =

ggplot(points,aes(x=x,y=y,color=group)) +
  geom_point()
```

Starting with lmer.

```
  m = lmer(y ~ x + (1| group), points)
  slope = fixef(m)[["x"]]
  slope
```

```
## [1] 0.1842517
```

Now with BRMS

```
m = brm(y ~ x + (1| group), points, cores=4)
```

```
fixef(m)
```

```
##             Estimate  Est.Error      Q2.5     Q97.5
## Intercept 6.0788321 1.38470857 3.2113708 8.6107897
## x         0.1831661 0.03560311 0.1149787 0.2563907
```

```
ranef(m)
```

```
## $group
## , , Intercept
##
##      Estimate Est.Error       Q2.5      Q97.5
```

```
## 1  -5.3847748  1.384546 -7.9339207 -2.5176031
## 2  -4.2153704  1.371347 -6.6949265 -1.4378876
## 3  -3.0255090  1.370331 -5.5006022 -0.2004186
## 4  -1.6234687  1.367662 -4.1393889  1.1688023
## 5  -0.5713347  1.366682 -3.0668213  2.1248994
## 6   0.6177461  1.367264 -1.8719831  3.3390684
## 7   1.8975989  1.367173 -0.5528503  4.5941040
## 8   3.0476413  1.375259  0.5647353  5.7867370
## 9   4.2038896  1.383090  1.7240780  6.9424032
## 10  5.5794807  1.390848  3.0952501  8.3118250
```

Yes, same thing hapens. Very similar fit. Now what happens if we specify a known x error?

```
m = brm(y ~ me(x,1) + (1| group), points, cores=4)
```

```
fixef(m)
```

```
##            Estimate  Est.Error      Q2.5     Q97.5
## Intercept 6.1121762 1.43066397 3.3875618 9.0235131
## mex1      0.1926087 0.03876238 0.1202721 0.2701714
```

```
ranef(m)
```

```
## $group
## , , Intercept
##
##      Estimate Est.Error       Q2.5      Q97.5
## 1  -5.4894484  1.428231 -8.3974371 -2.7605879
## 2  -4.3159677  1.413491 -7.1888036 -1.6098157
## 3  -3.1313839  1.405199 -5.9631218 -0.4257581
## 4  -1.7305454  1.403800 -4.5236470  1.0189895
## 5  -0.6769871  1.401737 -3.4010714  2.0761125
## 6   0.5092708  1.396615 -2.2569263  3.2438566
## 7   1.7870880  1.399838 -0.9632252  4.5503097
## 8   2.9455833  1.402194  0.1694178  5.7005903
## 9   4.1004469  1.408973  1.3332524  6.8625846
## 10  5.4735045  1.411808  2.6559021  8.2160339
```

```
summary(m)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: y ~ me(x, 1) + (1 | group)
##    Data: points (Number of observations: 150)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Group-Level Effects:
## ~group (Number of levels: 10)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     4.29      1.12     2.66     6.96 1.01      949     1900
```

```
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     6.11      1.43     3.39     9.02 1.00      543     1091
## mex1          0.19      0.04     0.12     0.27 1.00     1311     2400
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     0.44      0.03     0.37     0.51 1.00     1066     1629
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

This didn't seem to fix it.