March 8, Multilevel II

Stephen R. Proulx

3/1/2021

Today's objectives:

- Simulate data to observe shrinkage (aka partial pooling)
- Use the posterior distribution from a multilevel model in posetior predictive plots

Simulating to show an advantage of partial pooling

Define the parameters to use in this simulation

```
## R code 13.7
a_bar <- 1.5
sigma <- 1.5
nponds <- 60
```

Simulate the data

```
set.seed(5005)
sim_data <- tibble(Ni = rep(c(5,10,25,35),each=15),
                   a_pond = rnorm( nponds , mean=a_bar , sd=sigma )) %>%
  mutate(
    p_true = inv_logit(a_pond),
    Si = rbinom(n(),prob=inv_logit(a_pond),size=Ni)) %>%
  rowid_to_column("pond")
```

Calulate the full-pooling and no pooling maximum likelihood estimates (i.e. maximum posterior values with a flat prior). These are just the global average survival (full-pooling) and the survival within each pond (no-pooling).

```
sim_data <- mutate(sim_data,
                   p_nopool=Si/Ni,
                   p_fullpool=sum(Si)/sum(Ni) )
```

Run the mutli-level model.

```
## R code 13.13
dat <- select( sim_data, pond, Ni, Si )
m13.3 <- ulam(
    alist(
```

```
        Si ~ dbinom( Ni , p ),
        logit(p) <- a_pond[pond],
        a_pond[pond] ~ dnorm( a_bar , sigma ),
        a_bar ~ dnorm( 0 , 1.5 ),
        sigma ~ dexp( 1 )
    ), data=dat , chains=4 )
```

```
## Trying to compile a simple C file

## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Library/Frameworks/R.frame
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'd8ab7f3d807d0eb7d841894a7fbd4a24' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 4.6e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.46 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.151587 seconds (Warm-up)
## Chain 1:                0.087886 seconds (Sampling)
```

```
## Chain 1:                0.239473 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'd8ab7f3d807d0eb7d841894a7fbd4a24' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.3e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.13 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.141168 seconds (Warm-up)
## Chain 2:                0.088951 seconds (Sampling)
## Chain 2:                0.230119 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'd8ab7f3d807d0eb7d841894a7fbd4a24' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1.9e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.19 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 3: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 3: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 3: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 3: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 3: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 3: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 3: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 3: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 3: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 3: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 3: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.164915 seconds (Warm-up)
## Chain 3:                0.094503 seconds (Sampling)
## Chain 3:                0.259418 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'd8ab7f3d807d0eb7d841894a7fbd4a24' NOW (CHAIN 4).
```

```
## Chain 4:
## Chain 4: Gradient evaluation took 1.2e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.12 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 4: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 4: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 4: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 4: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 4: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 4: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 4: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 4: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 4: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 4: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.153232 seconds (Warm-up)
## Chain 4:                0.0993 seconds (Sampling)
## Chain 4:                0.252532 seconds (Total)
## Chain 4:
```

[prob cut this]

```
## R code 13.14


precis( m13.3 , depth=2 )
```

```
##                   mean        sd         5.5%       94.5%     n_eff
## a_pond[1]   1.64265068 1.0356282  0.081009165  3.45941681 2968.7949
## a_pond[2]   2.92923854 1.2810424  1.078457397  5.02118928 1970.7189
## a_pond[3]  -0.67314140 0.9175905 -2.168981898  0.68788537 3883.2224
## a_pond[4]   2.90611469 1.2217809  1.100054296  4.97610950 1913.0250
## a_pond[5]   2.94660821 1.3002287  1.073481334  5.09430149 1282.2404
## a_pond[6]   2.88533459 1.2391868  1.095669691  5.01505793 2221.3819
## a_pond[7]   0.08815240 0.8499278 -1.281120804  1.44415190 3760.3445
## a_pond[8]   2.94225104 1.3304832  1.029972480  5.25481551 2486.6333
## a_pond[9]   1.66675404 1.0297987  0.103698741  3.35040384 2393.5623
## a_pond[10]  1.66657621 1.0167732  0.099647008  3.35903828 3236.4530
## a_pond[11]  2.93375967 1.2612183  1.106617376  5.03928316 1511.4873
## a_pond[12]  0.07154634 0.8438531 -1.295394320  1.39366734 3151.3940
## a_pond[13]  2.89672819 1.3263634  1.031297272  5.23815431 2177.2377
## a_pond[14]  2.92735776 1.2868050  1.026738291  5.05124054 1848.7839
## a_pond[15]  2.91414498 1.2854902  1.030379754  5.20993468 2371.0590
## a_pond[16]  1.57458977 0.7594127  0.422242965  2.84026205 3542.3464
## a_pond[17] -1.45535187 0.7468167 -2.660023521 -0.36435201 2573.2288
## a_pond[18]  1.04576351 0.7012123 -0.025788359  2.22962519 3239.5407
## a_pond[19] -0.98463315 0.6708178 -2.078920497  0.07333676 1893.8861
## a_pond[20]  1.58047760 0.7662972  0.441243084  2.88239716 2407.0742
## a_pond[21] -0.15946304 0.6006262 -1.099304163  0.78135323 4059.1401
```

```
## a_pond[22]  2.27584280 0.9373636  0.962214978  3.88119304 2255.7529
## a_pond[23]  3.29323957 1.1751620  1.650509373  5.38749975 1983.0947
## a_pond[24]  0.62768741 0.6355917 -0.359263456  1.69422303 2985.3247
## a_pond[25]  3.27928574 1.1509724  1.629850879  5.27253272 3231.3100
## a_pond[26]  2.28151719 0.8970625  0.960348236  3.79911603 2600.9672
## a_pond[27]  1.03251011 0.6630596 -0.019558499  2.12769158 2854.9291
## a_pond[28]  2.27555463 0.9210970  0.952820004  3.82490599 1836.1869
## a_pond[29]  1.58879427 0.7669109  0.449064497  2.85936175 2353.2648
## a_pond[30]  1.05890954 0.6820824  0.009327097  2.18032137 3475.2811
## a_pond[31]  2.47126846 0.6973111  1.482781354  3.62178748 2629.0752
## a_pond[32]  2.08355993 0.6193480  1.175420205  3.13595798 2553.0027
## a_pond[33]  1.74081822 0.5411478  0.923032524  2.66006478 2870.7444
## a_pond[34]  1.24772760 0.4491990  0.574077183  1.98877699 2294.1726
## a_pond[35]  0.67266127 0.4047366  0.043577699  1.32024915 2863.3369
## a_pond[36]  3.83899953 0.9953745  2.475218907  5.55731550 2317.7462
## a_pond[37] -0.99392043 0.4494302 -1.742064585 -0.27727603 3222.5668
## a_pond[38] -1.20764835 0.4706775 -1.988100721 -0.48895235 3193.8050
## a_pond[39]  0.66330278 0.4126044  0.031622145  1.34363570 3253.6605
## a_pond[40]  3.92049521 1.1154674  2.331747091  5.87707193 1649.1317
## a_pond[41]  3.87910717 1.0835129  2.329397640  5.77576198 1947.6991
## a_pond[42]  2.46630823 0.7012689  1.446427629  3.63101934 2445.6094
## a_pond[43] -0.14798128 0.4041842 -0.816453307  0.49851192 2964.1228
## a_pond[44]  0.66100088 0.4170658  0.007787988  1.31228215 3110.6138
## a_pond[45] -1.20077530 0.4642994 -1.956615322 -0.49211594 2713.2468
## a_pond[46]  0.01135802 0.3382143 -0.516326521  0.56117726 3509.4048
## a_pond[47]  4.14241516 1.0996839  2.557080111  6.05868298 1630.0494
## a_pond[48]  2.10133834 0.5078335  1.328583627  2.94853799 2806.5197
## a_pond[49]  1.85865730 0.4967727  1.120313604  2.71541920 3206.3120
## a_pond[50]  2.81624935 0.6556448  1.845453462  3.91352562 2537.6779
## a_pond[51]  2.38836908 0.5613644  1.552991486  3.32235938 2740.9475
## a_pond[52]  0.35077055 0.3236406 -0.159768980  0.88186789 3170.4936
## a_pond[53]  2.09566308 0.5149501  1.371624000  2.96255280 2272.8982
## a_pond[54]  4.08142517 1.0491323  2.594523722  5.92930868 2334.2530
## a_pond[55]  1.14979745 0.3958041  0.550284646  1.78678105 2383.6951
## a_pond[56]  2.76910806 0.6557371  1.815790367  3.85236396 3405.9783
## a_pond[57]  0.70756945 0.3516705  0.154908607  1.28811398 3502.3832
## a_pond[58]  4.12633333 1.0893250  2.602171088  6.02679997 1821.8023
## a_pond[59]  1.63938938 0.4422554  0.971010139  2.38780176 4110.3815
## a_pond[60]  2.39627910 0.5829820  1.506716194  3.35878029 3269.9330
## a_bar       1.68058106 0.2586859  1.272612758  2.10698657 1451.2674
## sigma       1.69623322 0.2418526  1.350834362  2.10770210  709.5072
##               Rhat4
## a_pond[1]  0.9986268
## a_pond[2]  0.9988555
## a_pond[3]  1.0001283
## a_pond[4]  0.9992844
## a_pond[5]  1.0020999
## a_pond[6]  1.0010340
## a_pond[7]  0.9983187
## a_pond[8]  0.9988459
## a_pond[9]  0.9989553
## a_pond[10] 0.9988492
## a_pond[11] 1.0010892
## a_pond[12] 0.9989917
```

```
## a_pond[13] 0.9989776
## a_pond[14] 1.0003997
## a_pond[15] 1.0023858
## a_pond[16] 0.9988362
## a_pond[17] 0.9993887
## a_pond[18] 0.9983209
## a_pond[19] 1.0001558
## a_pond[20] 0.9991569
## a_pond[21] 0.9989129
## a_pond[22] 1.0013343
## a_pond[23] 1.0001028
## a_pond[24] 0.9998116
## a_pond[25] 0.9991724
## a_pond[26] 1.0016241
## a_pond[27] 0.9998248
## a_pond[28] 0.9995966
## a_pond[29] 0.9995523
## a_pond[30] 0.9992587
## a_pond[31] 0.9997590
## a_pond[32] 1.0000365
## a_pond[33] 1.0015746
## a_pond[34] 1.0003845
## a_pond[35] 0.9992302
## a_pond[36] 1.0003812
## a_pond[37] 1.0002971
## a_pond[38] 0.9984768
## a_pond[39] 0.9985526
## a_pond[40] 1.0020636
## a_pond[41] 0.9994754
## a_pond[42] 1.0030461
## a_pond[43] 0.9987306
## a_pond[44] 1.0002467
## a_pond[45] 0.9990094
## a_pond[46] 0.9996593
## a_pond[47] 0.9995521
## a_pond[48] 0.9983311
## a_pond[49] 0.9990433
## a_pond[50] 0.9995757
## a_pond[51] 0.9993249
## a_pond[52] 0.9988033
## a_pond[53] 0.9987853
## a_pond[54] 1.0010179
## a_pond[55] 0.9999798
## a_pond[56] 0.9985748
## a_pond[57] 0.9983128
## a_pond[58] 1.0004902
## a_pond[59] 0.9989856
## a_pond[60] 0.9988992
## a_bar      1.0019235
## sigma      1.0056629
```

Extract the samples from the model, convert them to the probability scale, and caclulate the error from true

```
## R code 13.15
post <- extract.samples( m13.3 ) %>% as_tibble()


# get the a_pond portion of the posterior and summarize
post_a_pond <- select(post,a_pond)%>%
  as.matrix()%>%  as_tibble() %>%
  gather( "ponds","a_pond",1:nponds)%>%
  separate(ponds,c("name","pond"),sep=7) %>%
  mutate(pond=as.numeric(pond),
         p_pond = inv_logit(a_pond)) %>%
  select(-name)
```

```
## Note: Using an external vector in selections is ambiguous.
## i Use `all_of(nponds)` instead of `nponds` to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

```
post_a_pond_summarized <- group_by(post_a_pond,pond) %>%
  summarise(
    p_partpool = mean(p_pond),
  )%>%
  ungroup()



# get the a_bar portion of the posterior and summarize
post_a_bar <- select(post,a_bar)

post_a_bar_summarized <- summarise(post_a_bar,
    a_bar = mean(a_bar))


# put them all in one dataframe
sim_data <- left_join(sim_data,post_a_pond_summarized) %>%
  mutate(partpool_a_bar = post_a_bar_summarized$a_bar[1],
         nopool_p_bar = sum(p_nopool)/nponds,
         nopool_a_bar = logit(nopool_p_bar),
         true_a_bar = sum(a_pond)/nponds
         ) %>%
  select(-nopool_p_bar)
```

```
## Joining, by = "pond"
```

Take a minute to look at the error.

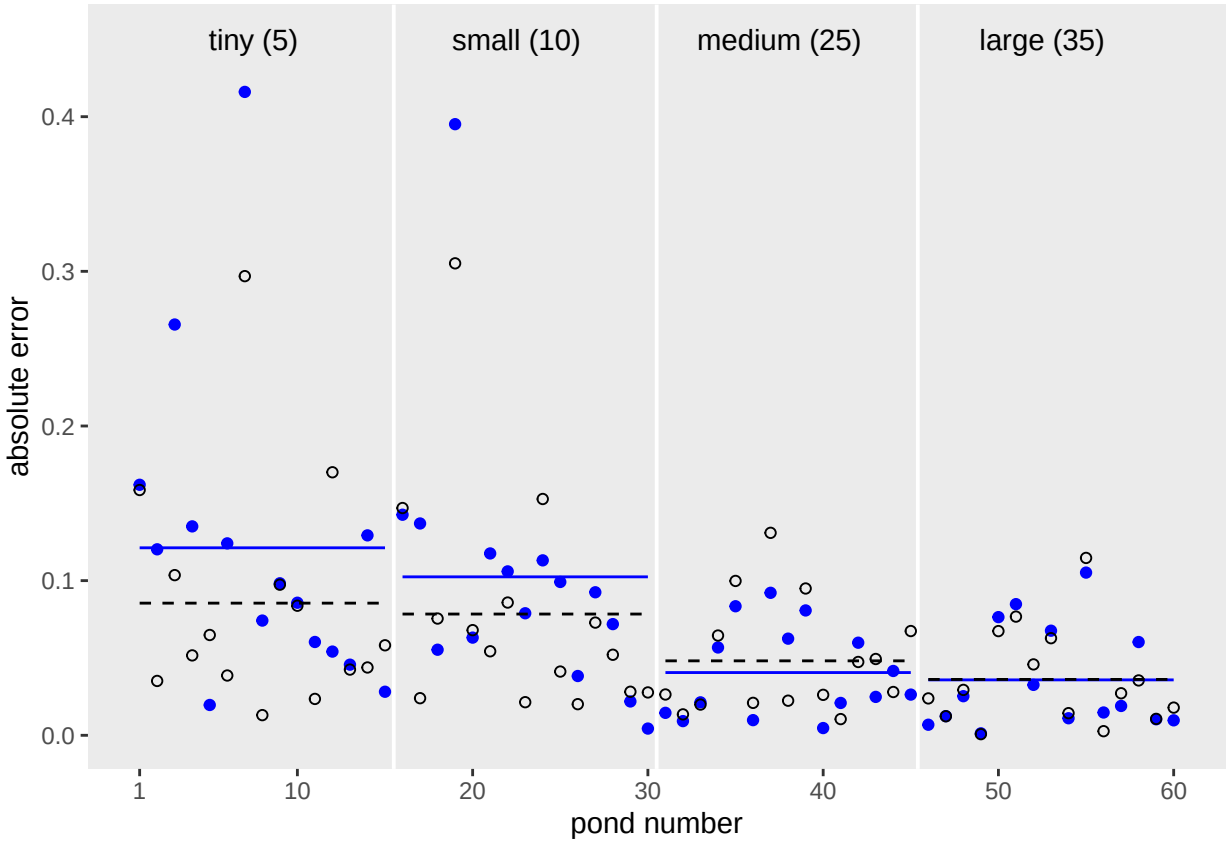
```
view(sim_data)
```

add the error measurement and plot

```
## R code 13.17
sim_data <- sim_data %>%
  mutate(nopool_error = abs(p_nopool-p_true), partpool_error = abs(p_partpool-p_true))


dfline <-
  sim_data %>%
  select(Ni, nopool_error:partpool_error) %>%
  gather(key, value, -Ni) %>%
  group_by(key, Ni) %>%
  summarise(mean_error = mean(value)) %>%
  mutate(x    = c( 1, 16, 31, 46),
         xend = c(15, 30, 45, 60))
```

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

```
 ggplot(sim_data,aes(x = pond)) +
  geom_vline(xintercept = c(15.5, 30.5, 45.4),
             color = "white", size = 2/3) +
  geom_point(aes(y = nopool_error), color = "blue") +
  geom_point(aes(y = partpool_error), shape = 1) +
  geom_segment(data = dfline,
               aes(x = x, xend = xend,
                   y = mean_error, yend = mean_error),
               color = rep(c("blue", "black"), each = 4),
               linetype = rep(1:2, each = 4)) +
  scale_x_continuous(breaks = c(1, 10, 20, 30, 40, 50, 60)) +
  annotate("text", x = c(15 - 7.5, 30 - 7.5, 45 - 7.5, 60 - 7.5), y = .45,
           label = c("tiny (5)", "small (10)", "medium (25)", "large (35)")) +
  labs(y        = "absolute error",
       x = "pond number")+
  theme(panel.grid    = element_blank(),
        plot.subtitle = element_text(size = 10))
```



re-simulate and look again

```
sim_data <- tibble(Ni = rep(c(5,10,25,35),each=15),
                   a_pond = rnorm( nponds , mean=a_bar , sd=sigma )) %>%
  mutate(
    p_true = inv_logit(a_pond),
    Si = rbinom(n(),prob=inv_logit(a_pond),size=Ni)) %>%
  rowid_to_column("pond")


sim_data <- mutate(sim_data,
                   p_nopool=Si/Ni,
                   p_fullpool=sum(Si)/sum(Ni) )



m13.3new <- stan( fit=m13.3@stanfit , data=sim_data , chains=4 )
```

```
##
## SAMPLING FOR MODEL 'd8ab7f3d807d0eb7d841894a7fbd4a24' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 2.9e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.29 seconds.
## Chain 1: Adjust your expectations accordingly!
```

```
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.188385 seconds (Warm-up)
## Chain 1:                0.230777 seconds (Sampling)
## Chain 1:                0.419162 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'd8ab7f3d807d0eb7d841894a7fbd4a24' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 2.9e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.29 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.241123 seconds (Warm-up)
## Chain 2:                0.194085 seconds (Sampling)
## Chain 2:                0.435208 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'd8ab7f3d807d0eb7d841894a7fbd4a24' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1.7e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.17 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
```

```
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.182031 seconds (Warm-up)
## Chain 3:                0.176057 seconds (Sampling)
## Chain 3:                0.358088 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'd8ab7f3d807d0eb7d841894a7fbd4a24' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.7e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.17 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.196651 seconds (Warm-up)
## Chain 4:                0.112902 seconds (Sampling)
## Chain 4:                0.309553 seconds (Total)
## Chain 4:
```

```
## R code 13.15
post <- extract.samples( m13.3new ) %>% as_tibble()


# get the a_pond portion of the posterior and summarize
post_a_pond <- select(post,a_pond)%>%
  as.matrix()%>%  as_tibble() %>%
  gather( "ponds","a_pond",1:nponds)%>%
  separate(ponds,c("name","pond"),sep=7) %>%
  mutate(pond=as.numeric(pond),
         p_pond = inv_logit(a_pond)) %>%
  select(-name)
```

```
post_a_pond_summarized <- group_by(post_a_pond,pond) %>%
  summarise(
    p_partpool = mean(p_pond),
  )%>%
  ungroup()



# get the a_bar portion of the posterior and summarize
post_a_bar <- select(post,a_bar)

post_a_bar_summarized <- summarise(post_a_bar,
    a_bar = mean(a_bar))


# put them all in one dataframe
sim_data <- left_join(sim_data,post_a_pond_summarized) %>%
  mutate(partpool_a_bar = post_a_bar_summarized$a_bar[1],
         nopool_p_bar = sum(p_nopool)/nponds,
         nopool_a_bar = logit(nopool_p_bar),
         true_a_bar = sum(a_pond)/nponds
         ) %>%
  select(-nopool_p_bar)
```

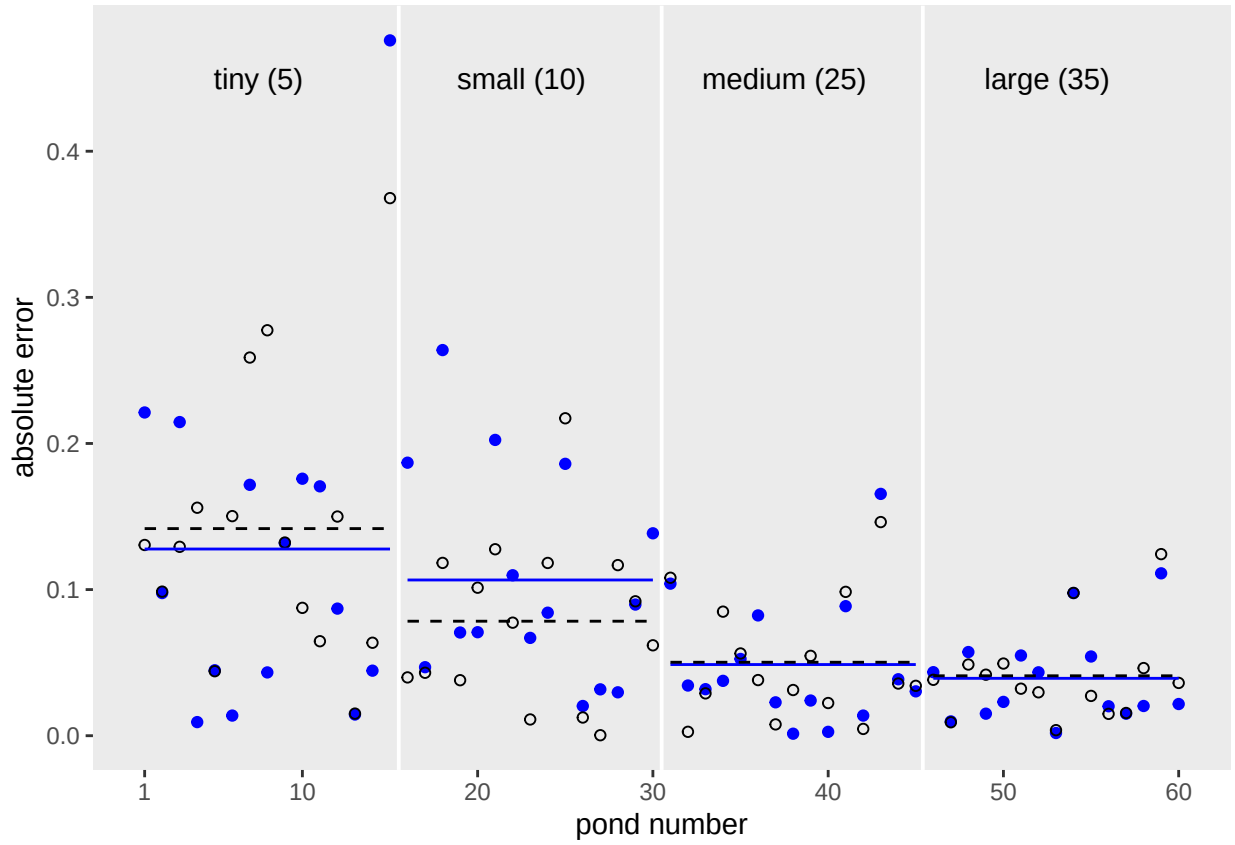
```
## Joining, by = "pond"
```

```
## R code 13.17
sim_data <- sim_data %>%
  mutate(nopool_error = abs(p_nopool-p_true), partpool_error = abs(p_partpool-p_true))


dfline <-
  sim_data %>%
  select(Ni, nopool_error:partpool_error) %>%
  gather(key, value, -Ni) %>%
  group_by(key, Ni) %>%
  summarise(mean_error = mean(value)) %>%
  mutate(x    = c( 1, 16, 31, 46),
         xend = c(15, 30, 45, 60))
```

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

```
 ggplot(sim_data,aes(x = pond)) +
  geom_vline(xintercept = c(15.5, 30.5, 45.4),
             color = "white", size = 2/3) +
  geom_point(aes(y = nopool_error), color = "blue") +
  geom_point(aes(y = partpool_error), shape = 1) +
  geom_segment(data = dfline,
               aes(x = x, xend = xend,
                   y = mean_error, yend = mean_error),
               color = rep(c("blue", "black"), each = 4),
               linetype = rep(1:2, each = 4)) +
```

```
  scale_x_continuous(breaks = c(1, 10, 20, 30, 40, 50, 60)) +
  annotate("text", x = c(15 - 7.5, 30 - 7.5, 45 - 7.5, 60 - 7.5), y = .45,
           label = c("tiny (5)", "small (10)", "medium (25)", "large (35)")) +
  labs(y        = "absolute error",
       x = "pond number")+
  theme(panel.grid    = element_blank(),
        plot.subtitle = element_text(size = 10))
```



‘

```
view(sim_data)
```

error in the full estimate of a?

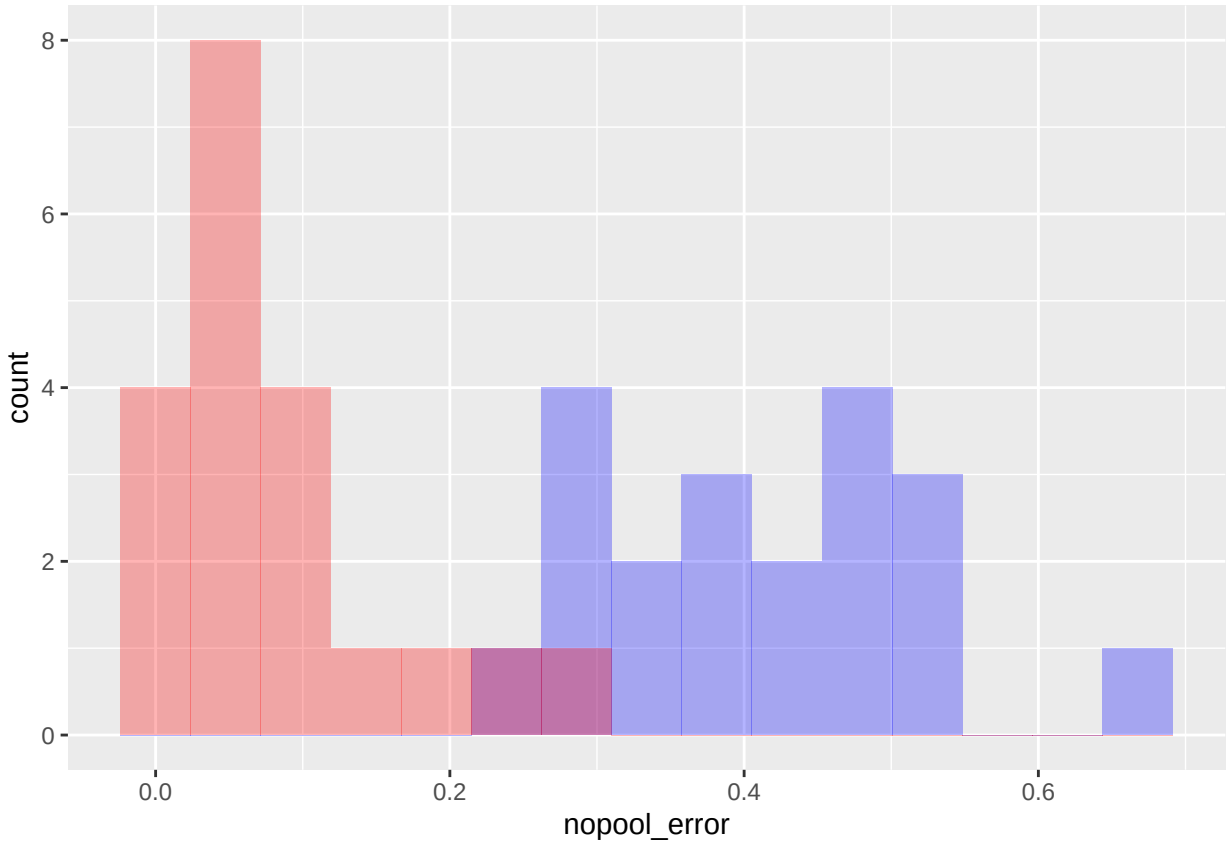
```
abar_error<-tibble(nopool_error=rep(0,20),partpool_error=rep(0,20))


for(i in 1:20){
  sim_data <- tibble(Ni = rep(c(5,10,25,35),each=15),
                     a_pond = rnorm( nponds , mean=a_bar , sd=sigma )) %>%
  mutate(
    p_true = inv_logit(a_pond),
    Si = rbinom(n(),prob=inv_logit(a_pond),size=Ni)) %>%
```

```
  rowid_to_column("pond")


sim_data <- mutate(sim_data,
                   p_nopool=Si/Ni,
                   p_fullpool=sum(Si)/sum(Ni) )



m13.3new <- stan( fit=m13.3@stanfit , data=sim_data , chains=4 )



## R code 13.15
post <- extract.samples( m13.3new ) %>% as_tibble()


# get the a_pond portion of the posterior and summarize
post_a_pond <- select(post,a_pond)%>%
  as.matrix()%>%  as_tibble() %>%
  gather( "ponds","a_pond",1:nponds)%>%
  separate(ponds,c("name","pond"),sep=7) %>%
  mutate(pond=as.numeric(pond),
         p_pond = inv_logit(a_pond)) %>%
  select(-name)

post_a_pond_summarized <- group_by(post_a_pond,pond) %>%
  summarise(
    p_partpool = mean(p_pond),
  )%>%
  ungroup()



# get the a_bar portion of the posterior and summarize
post_a_bar <- select(post,a_bar)

post_a_bar_summarized <- summarise(post_a_bar,
    a_bar = mean(a_bar))


# put them all in one dataframe
sim_data <- left_join(sim_data,post_a_pond_summarized) %>%
  mutate(partpool_a_bar = post_a_bar_summarized$a_bar[1],
         nopool_p_bar = sum(p_nopool)/nponds,
         nopool_a_bar = logit(nopool_p_bar),
         true_a_bar = sum(a_pond)/nponds
         ) %>%
  select(-nopool_p_bar)



abar_error$partpool_error[i]<-abs(sim_data$partpool_a_bar[1]-sim_data$true_a_bar[1])
```

```
abar_error$nopool_error[i]<-abs(sim_data$nopool_a_bar[1]-sim_data$true_a_bar[1])

}
```

```
abar_error <- tibble(nopool_error=c( 0.4182332, 0.5081091, 0.3939369, 0.3559052, 0.5133194, 0.2753575, 
                     partpool_error=c(0.0683,   0.091,  0.0524, 0.0545, 0.221,  0.0431, 0.125,  0.0289,
```

```
ggplot(abar_error,aes(x=nopool_error)) +
         geom_histogram(bins=15,fill="blue",alpha=0.3) +
         geom_histogram(aes(x=partpool_error), bins=15,fill="red",alpha=0.3)
```



multilevel chimps and posterior sampling

We'll revisit the lever-pulling chimps to see how to work with posetrior samples from a multilevel model. Here's the data, ready for handing to `ulam`

```
library(rethinking)
data(chimpanzees)
d <- chimpanzees %>%
  mutate(treatment = 1+ prosoc_left + 2*condition)

dat_list <- list(
```

```
    pulled_left = d$pulled_left,
    actor = d$actor,
    block_id = d$block,
    treatment = as.integer(d$treatment) )
```

Here is the reparmeterized model where actor, block_id, and treatment each contribute additively. Remember though, treatment is already coded to measure the interaction between where the pro-social food is and the presence of another chimp.

```
set.seed(13)
m13.4nc <- ulam(
    alist(
        pulled_left ~ dbinom( 1 , p ) ,
        logit(p) <- a_bar + z[actor]*sigma_a + # actor intercepts
                    x[block_id]*sigma_g +      # block intercepts
                    b[treatment] ,
        b[treatment] ~ dnorm( 0 , 0.5 ),
        z[actor] ~ dnorm( 0 , 1 ),
        x[block_id] ~ dnorm( 0 , 1 ),
        a_bar ~ dnorm( 0 , 1.5 ),
        sigma_a ~ dexp(1),
        sigma_g ~ dexp(1),
        gq> vector[actor]:a <<- a_bar + z*sigma_a,
        gq> vector[block_id]:g <<- x*sigma_g
    ) , data=dat_list , chains=4 , cores=4 )
```

```
## Trying to compile a simple C file
```

```
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Library/Frameworks/R.frame
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                 ^
##                 ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
```

Now we would like to depict two types of information from this posterior, the average treatment effect and the treatment outcome including individual variability.

The first, the average treatment effect, aims to show how the treatment fundamentally affects the outcome on the probability scale. We already have the informaiton to say how treatment affects the oucome on the logit scale, because it is an additive model and we have the distribution of treatment effects. We just want to convert back to the probabilty scale, but do this for the "average" chimp/block. This means that we need to set the individual effect and block effect to zero, and then compute the range of values for the probability of pulling left.

The second, called the "marginal of actors" means that we have to simulate new actors, assuming that our small-world model is correct, and that new individuals have normally distributed actor effects with the posterior of the standard deviation coming from our posterior samples.

Construct these two graphs, which will look like the left and center panel of figure 13.7.