# March 3, Multilevel I

Stephen R. Proulx

3/1/2021

## Today's objectives:

- See how to write models with multiple layers of parameter distributions.
- Observe the way that multilevel models can improve model fit without creating over-fitting
- Include a multi-level component in a model that tests for a treatment effect

## Reedfrog datasetr

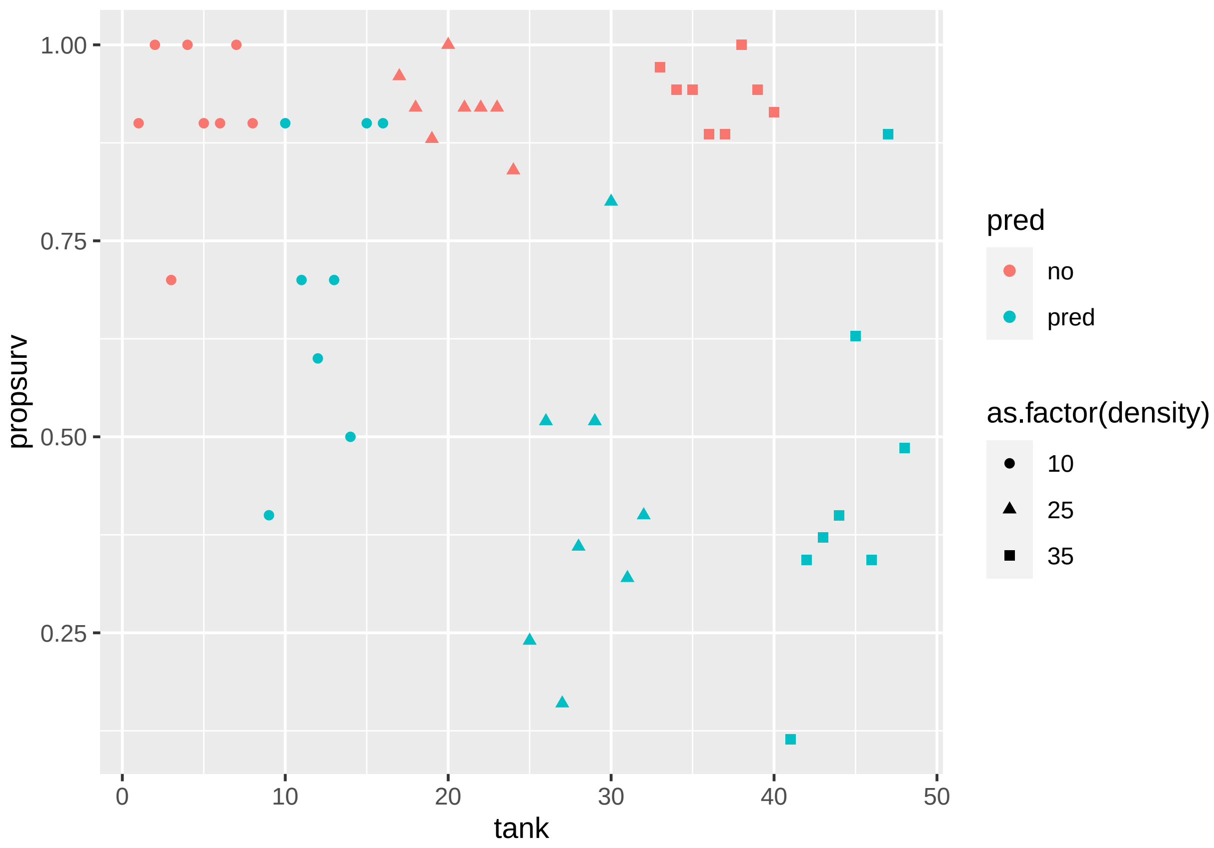
This data is from a study that looked at how tadpole density and size affected their predation rate. Those that survived did so because they didn't die naturally and also did not get eaten.

Load the data and have a general look.

```
data(reedfrogs)
d <- as_tibble(reedfrogs)%>%
  rowid_to_column("tank") %>%
  view()
```

Let's see how it looks, we'll add some features to the plot so we can visualize the effects.

```
ggplot(d, aes(x=tank, y= propsurv)) +geom_point( aes(color=pred,shape=as.factor(density)))
```




### Tank effects model with no predictors

Here we have a model you should be fairly used to, individual effects for each tank with a common prior for all tanks.

```
m13.1 <- ulam(
  alist(
    surv ~ dbinom( density, p),
    logit(p) <- a[tank],
    a[tank] ~ dnorm(0,1.5)
  ),
  data=select(d,surv,density,tank), chains=4, log_lik = TRUE
  )
```

```
## 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 'cbc8c5321cddc84992ceba311b8e48a7' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 2.6e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.26 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: 0.096974 seconds (Warm-up)
## Chain 1:                0.084693 seconds (Sampling)
## Chain 1:                0.181667 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'cbc8c5321cddc84992ceba311b8e48a7' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.6e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.16 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: 0.099047 seconds (Warm-up)
## Chain 2:                0.085112 seconds (Sampling)
## Chain 2:                0.184159 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'cbc8c5321cddc84992ceba311b8e48a7' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1.5e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.15 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: 0.089911 seconds (Warm-up)
## Chain 3:                0.087081 seconds (Sampling)
## Chain 3:                0.176992 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'cbc8c5321cddc84992ceba311b8e48a7' 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: 0.107653 seconds (Warm-up)
## Chain 4:                0.089339 seconds (Sampling)
## Chain 4:                0.196992 seconds (Total)
## Chain 4:
```

Inspect the summary, there are 48 parameters, and they all have good convergence stats.

```r
precis(m13.1 , depth =2)
```

```
##               mean        sd        5.5%       94.5%    n_eff     Rhat4
## a[1]   1.713849576 0.7483075  0.56108507  2.96037107 3904.966 0.9991906
## a[2]   2.421621685 0.9014988  1.10477875  4.01764124 3354.981 0.9997741
## a[3]   0.747838602 0.5983202 -0.15658551  1.72416853 4091.712 0.9984290
## a[4]   2.405719080 0.8859261  1.03296487  3.91121818 3593.318 0.9984299
## a[5]   1.741122604 0.7685478  0.57943346  2.96093267 4325.341 0.9986539
## a[6]   1.706239791 0.7710096  0.57180259  2.98903303 4376.920 0.9984254
## a[7]   2.407977155 0.8685397  1.11273933  3.90357125 3256.898 0.9987543
## a[8]   1.716143305 0.8139947  0.51844138  3.11806004 3669.414 0.9988127
## a[9]  -0.368792754 0.6244182 -1.38678409  0.63244942 4693.381 0.9985408
## a[10]  1.711741229 0.7437916  0.57604259  2.93657400 3787.103 0.9994363
## a[11]  0.754509798 0.6489284 -0.21533829  1.79846933 4637.979 0.9990083
## a[12]  0.375025269 0.6282610 -0.58242400  1.40774875 4751.028 0.9987624
## a[13]  0.771825601 0.6384459 -0.21691688  1.80913658 4292.612 0.9991580
## a[14]  0.008363493 0.6145732 -0.96126329  1.00395064 4825.748 0.9990827
## a[15]  1.712036542 0.7824670  0.52791166  3.02978616 4348.381 0.9987600
## a[16]  1.714906355 0.7478862  0.59248510  2.92672037 3899.192 1.0004312
## a[17]  2.539550406 0.6871309  1.53777561  3.70334299 3510.237 0.9990868
## a[18]  2.131337898 0.5994906  1.26310387  3.13599752 4634.643 0.9997532
## a[19]  1.814603458 0.5480127  0.99555160  2.72336892 4040.394 0.9987018
## a[20]  3.109524568 0.8531002  1.87176439  4.53640601 3948.494 0.9986629
## a[21]  2.133630839 0.5960822  1.23601182  3.13307901 5137.912 0.9989405
## a[22]  2.131905056 0.5995743  1.21056803  3.16773710 4165.123 0.9992935
## a[23]  2.128358008 0.5868930  1.28236676  3.14748860 4260.544 0.9991228
## a[24]  1.529885706 0.4910900  0.78865359  2.32404606 5000.313 0.9987636
## a[25] -1.087265429 0.4500444 -1.79826270 -0.39229998 4753.666 0.9993853
## a[26]  0.074179846 0.3783801 -0.51641739  0.67366247 4583.832 0.9991096
## a[27] -1.541883268 0.4892830 -2.36222204 -0.79674593 4114.352 0.9984471
## a[28] -0.559141238 0.4316515 -1.25346143  0.13009661 3865.076 0.9996989
## a[29]  0.063611834 0.4034127 -0.57834511  0.70427158 4566.309 0.9992405
## a[30]  1.303717098 0.5006323  0.52342515  2.11245937 4771.901 0.9986636
## a[31] -0.721056100 0.4207118 -1.39344775 -0.05081795 4485.228 0.9986233
## a[32] -0.393539358 0.3877345 -1.02735649  0.21116780 4279.858 1.0002212
## a[33]  2.853464546 0.6595559  1.88106675  3.97769735 2798.749 1.0007168
## a[34]  2.454356642 0.5635339  1.58271113  3.39991197 4002.756 0.9984490
## a[35]  2.455371414 0.5863697  1.58074964  3.43560507 3518.491 0.9992166
## a[36]  1.901977240 0.4750815  1.19297445  2.71425113 3804.192 0.9986176
## a[37]  1.902175508 0.4733054  1.18419547  2.67200100 4690.650 0.9983154
## a[38]  3.341703779 0.7687299  2.17687159  4.64344694 4084.841 0.9989463
## a[39]  2.458705051 0.5750981  1.62516072  3.42808059 5409.109 0.9990759
## a[40]  2.147373876 0.5474665  1.32692541  3.04616027 4723.666 0.9987695
## a[41] -1.908138159 0.4820916 -2.72755900 -1.18596910 3827.372 0.9983746
## a[42] -0.623911927 0.3417715 -1.17491811 -0.08775752 5327.168 0.9989769
```

```
## a[43] -0.513255982 0.3439469 -1.09019067  0.04171812 5229.552 0.9990075
## a[44] -0.391478155 0.3382076 -0.95567395  0.14237637 3916.795 0.9988265
## a[45]  0.505740828 0.3465947 -0.05388373  1.07726852 4721.060 0.9985697
## a[46] -0.625652885 0.3473636 -1.19134219 -0.07989764 4194.740 0.9988068
## a[47]  1.905064756 0.4776254  1.17226032  2.72349532 4132.852 0.9989743
## a[48] -0.057036057 0.3128564 -0.54294491  0.45343950 3292.555 0.9983899
```

Compute the WAIC. The effective number of parameters is lower than the true number, but it is more important to know how the number of parameters compares between models.

```r
WAIC(m13.1)
```

```
##       WAIC      lppd  penalty  std_err
## 1 214.9365 -81.71758 25.75065 4.480643
```

**Same system, but multi-level tank effects**

Here we model the tank-specific means as coming from a distribution themselves. We will end up with a parameter for each tank, and this parameter will have a mean and a distribution. But we also will have the more general parameters which describe where tank parameters themselves come from. This is great, we can now make predictions about tanks we have not yet seen without resorting to over-fitting.

```r
m13.2 <- ulam(
  alist(
    surv ~ dbinom( density, p),
    logit(p) <- a[tank],
    a[tank] ~ dnorm(abar,sigma),
    abar ~ dnorm(0,1.5),
    sigma ~ dexp(1)
  ),
  data=select(d,surv,density,tank), chains=4, log_lik = TRUE
  )
```

```
## 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.fram
## 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 'ea3715cecb3ee81391fb2ed20edc3c1d' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 4.9e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.49 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: 0.149768 seconds (Warm-up)
## Chain 1:                0.110299 seconds (Sampling)
## Chain 1:                0.260067 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'ea3715cecb3ee81391fb2ed20edc3c1d' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.8e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.18 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: 0.160568 seconds (Warm-up)
## Chain 2:                0.102061 seconds (Sampling)
## Chain 2:                0.262629 seconds (Total)
## Chain 2:
```

```
##
## SAMPLING FOR MODEL 'ea3715cecb3ee81391fb2ed20edc3c1d' 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: 0.138036 seconds (Warm-up)
## Chain 3:                0.109351 seconds (Sampling)
## Chain 3:                0.247387 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'ea3715cecb3ee81391fb2ed20edc3c1d' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.5e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.15 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: 0.137752 seconds (Warm-up)
## Chain 4:                0.100177 seconds (Sampling)
## Chain 4:                0.237929 seconds (Total)
## Chain 4:
```

This model now has 50 parameters, a few more than the last.

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

```
##              mean        sd        5.5%        94.5%    n_eff     Rhat4
## a[1]    2.13401327 0.8912617  0.81297510  3.656988072 3125.517 0.9987600
## a[2]    3.06195766 1.0887989  1.51868171  4.958429714 2396.970 0.9993097
## a[3]    1.02148102 0.7090577 -0.06078966  2.143325728 4627.202 0.9983255
## a[4]    3.04028282 1.0981918  1.41935494  4.922898245 2564.109 0.9988110
## a[5]    2.13095078 0.8496222  0.87431099  3.549277704 3900.292 0.9985321
## a[6]    2.13302403 0.8660815  0.83985056  3.631432403 3374.118 0.9990516
## a[7]    3.07419997 1.0740595  1.54005347  4.901866629 2732.741 0.9998173
## a[8]    2.13119818 0.8554838  0.89433254  3.533384157 4415.013 0.9985058
## a[9]   -0.18378914 0.6480440 -1.21888298  0.859413953 5410.329 0.9987143
## a[10]   2.12909605 0.8926149  0.78928694  3.671924056 3578.394 0.9989093
## a[11]   0.99995412 0.6670357 -0.01075502  2.096576465 4240.653 0.9993242
## a[12]   0.57862679 0.6591246 -0.41704173  1.630430400 4408.061 0.9989333
## a[13]   1.00914809 0.7055192 -0.04842157  2.172742375 3916.957 0.9987306
## a[14]   0.20392673 0.6178936 -0.76865378  1.188596065 4754.595 0.9989135
## a[15]   2.13339140 0.8583665  0.87683330  3.508991841 3336.395 0.9986607
## a[16]   2.13336894 0.8519803  0.84195880  3.636978083 3488.085 0.9987380
## a[17]   2.91733650 0.8114632  1.74781398  4.321872380 2639.902 0.9995272
## a[18]   2.39866800 0.6690148  1.44214344  3.565649422 3002.332 0.9990385
## a[19]   2.01652203 0.5897447  1.12607536  2.983464427 3689.696 0.9982147
## a[20]   3.65094260 0.9928058  2.22435255  5.306170426 3676.632 0.9986206
## a[21]   2.39888796 0.6745088  1.40380598  3.522222693 3128.675 0.9993641
## a[22]   2.40262779 0.6593089  1.43340494  3.575603722 3207.728 0.9986856
## a[23]   2.41299836 0.6668088  1.46399233  3.527401370 2797.143 1.0009457
## a[24]   1.71520006 0.5669028  0.86132105  2.630699298 2955.124 0.9990888
## a[25]  -1.00728107 0.4304005 -1.71950379 -0.351027062 3443.886 0.9983181
## a[26]   0.16681645 0.3996833 -0.45722793  0.802465347 5365.149 0.9990804
## a[27]  -1.42588877 0.4725105 -2.16706707 -0.703121771 4036.548 0.9991824
## a[28]  -0.45914152 0.4206020 -1.16822275  0.206224000 3668.086 0.9991141
## a[29]   0.16084111 0.3893403 -0.44994600  0.775928380 5005.591 0.9983668
## a[30]   1.46257770 0.4919737  0.70981234  2.289040539 3926.451 0.9991743
## a[31]  -0.64034995 0.4091516 -1.32098677 -0.007703717 3843.713 0.9987162
## a[32]  -0.30613184 0.3873125 -0.92587611  0.312988904 5046.839 0.9988796
## a[33]   3.19520063 0.7805167  2.07608524  4.541146919 3376.725 0.9990481
## a[34]   2.70832665 0.6247607  1.77732502  3.753621784 3463.561 1.0000111
## a[35]   2.69135972 0.6298230  1.75498362  3.738583314 4673.173 0.9985963
## a[36]   2.06547506 0.5001847  1.30839668  2.915582570 3789.622 0.9997253
## a[37]   2.05787411 0.4898476  1.33364469  2.907718059 3732.151 0.9991964
## a[38]   3.91083304 0.9928979  2.50362187  5.613188773 2309.589 0.9999422
## a[39]   2.72238678 0.6590058  1.77259031  3.876561879 2536.195 0.9988680
## a[40]   2.37172924 0.5706489  1.51858176  3.314946653 2779.930 1.0002585
## a[41]  -1.80203919 0.4681027 -2.58549472 -1.100250356 4165.848 0.9987265
## a[42]  -0.56968442 0.3484213 -1.13741180 -0.028393701 3130.309 0.9995459
## a[43]  -0.46541406 0.3511367 -1.02879813  0.089716992 5482.911 0.9988577
## a[44]  -0.34039186 0.3468811 -0.90082924  0.195139994 4957.006 0.9984385
## a[45]   0.58523697 0.3321176  0.07015204  1.116129510 4179.910 0.9983141
## a[46]  -0.56622118 0.3501143 -1.13369579 -0.022093095 4333.806 0.9990299
## a[47]   2.05920000 0.5209400  1.29111410  2.905300936 3772.071 0.9991675
## a[48]   0.01315129 0.3151749 -0.47734085  0.512463610 3752.465 0.9990974
## abar    1.34277223 0.2628107  0.94000691  1.774864361 3540.512 0.9989133
## sigma   1.61369231 0.2065927  1.30963799  1.960881405 1800.485 0.9995699
```

We compute WAIC and see that the effective number of parameters has actually gone down!

```r
WAIC(m13.2)
```

```
##      WAIC      lppd  penalty  std_err
## 1 200.697 -79.16727 21.18123 7.233985
```

And we can compare them. The multilevel model here does better, and does more-better than the SE of the WAIC scores, so we can be confident that it improves fit to the data without overfitting. And as we've already noted it does this by having fewer effective parameters.

```r
compare(m13.1,m13.2)
```

```
##           WAIC       SE    dWAIC      dSE    pWAIC       weight
## m13.2 200.6970 7.233985  0.00000       NA 21.18123 0.9991916784
## m13.1 214.9365 4.480643 14.23948 4.128826 25.75065 0.0008083216
```

### A multi-level model with contrasts

In the reedfrog example we may actually want to know some things about how tadpole survival depends on the imposed experimental conditions. While there are many aspects of the data that are relevant, we will focus on one, the influence of tank size on predation rate.

Setup the data: We need to add an index variable for density (1,2,3) and an indicator variable for predation (0,1).

```r
d<- mutate(d, D=as.numeric( as.factor(density)),
           P=(pred=="no")*0 + (pred=="pred")*1) %>% view()
```

Here's a reasonable model. In the absence of predators, there is a baseline mortality rate that is tank density dependent. The presence of predators has an additive effect, on the logit scale, but this slope depends on tank density. And each tank also has "noise", variance in the binomial parameter itself that is not due to a measured predictor, but is common to the tadpoles within each tank (i.e. tank temperature, predator vigor, etc.)

The "noise" is centered at 0, because it is noise. How much noise is there between tanks? We don't know this ahead of time, so we make a multi-level model. The noise parameter itself is normally distributed, but we use the data to fit the sigma associated with this.

We do, however, have to have a prior for sigma itself, we can't kick the can down the road forever.

```r
m.tank.size.prior <- ulam(alist(
  surv ~ dbinom(density , p),
  logit(p) <- a[D] + b[D]*P + noise[tank],
  a[D] ~ dnorm(1,2),
  b[D] ~ dnorm(0,1),
  noise[tank] ~ dnorm(0,sigma), #this is what makes it multilevel!
  sigma ~ dexp(1)
), data=select(d,surv,D,P,tank) , cores=4 , chains=4 , iter=3000, sample_prior = TRUE)
```

```
## 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
```

```
## Warning: There were 4 chains where the estimated Bayesian Fraction of Missing Information was low. Se
## http://mc-stan.org/misc/warnings.html#bfmi-low
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
## Warning: The largest R-hat is 1.15, indicating chains have not mixed.
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#r-hat
```

```
## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#bulk-ess
```

```
## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quant
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#tail-ess
```

Simulate from the prior: Some work is required to re-associate the output with the tank number and calculate the propotion that survived

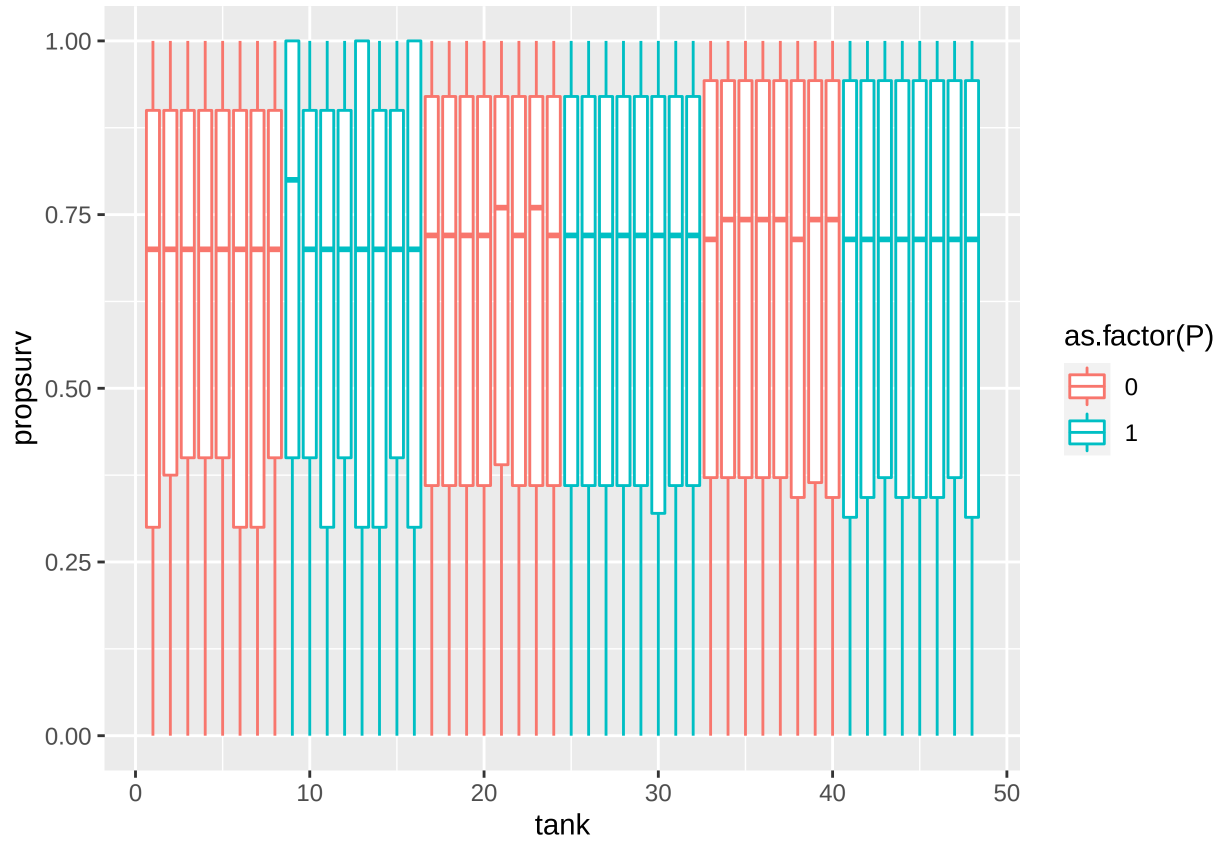
```
prior.sim.tank.size <- sim(m.tank.size.prior, data=d) %>%
  as_tibble() %>%
  gather("tanks","surviving",1:48)%>%
  separate(tanks,c("V","tank"), sep=1) %>%
  mutate(tank=as.numeric(tank)) %>%
  left_join(select(d,tank,density,P))%>%
  mutate(propsurv = surviving/density)
```

```
## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if `.name_repair` is
## Using compatibility `.name_repair`.
```

```
## Joining, by = "tank"
```

Our prior seems broad enough to capture the data, without getting pegged at 0 or 1.

```r
ggplot(prior.sim.tank.size , aes(x=tank, y=propsurv, group=as.factor(tank), color=as.factor(P)))+
  geom_boxplot(outlier.alpha = 0)
```




Now we are ready to try our full model!

```r
m.tank.size <-  ulam(alist(
  surv ~ dbinom(density , p),
  logit(p) <- a[D] + b[D]*P + noise[tank],
  a[D] ~ dnorm(1,2),
  b[D] ~ dnorm(0,1),
  noise[tank] ~ dnorm(0,sigma), #this is what makes it multilevel!
  sigma ~ dexp(1)
), data=select(d,surv,density,D,P,tank) , cores=4 , chains=4 , iter=3000, sample_prior = FALSE,
sample = TRUE)
```

```
## 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/incluc
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/incluc
## /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/incluc
## /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
```

Check the chain stats, everything seems fine.

```
precis(m.tank.size,depth=2)
```

```
##                  mean        sd        5.5%       94.5%     n_eff
## a[1]       2.26500688 0.4295914  1.58559207  2.95209743  5698.120
## a[2]       2.22145166 0.3597987  1.65024126  2.79224600  3953.707
## a[3]       2.45365631 0.3603172  1.88569233  3.03204530  3898.942
## b[1]      -1.13980521 0.5186630 -1.97136183 -0.31210743  5556.735
## b[2]      -2.35496917 0.4531619 -3.05685087 -1.62630320  3055.523
## b[3]      -2.43719576 0.4565907 -3.14664525 -1.68986989  2890.508
## noise[1]   0.04355333 0.6581509 -0.98720570  1.11001048 10199.570
## noise[2]   0.48038275 0.7118745 -0.58866971  1.63009729  7885.311
## noise[3]  -0.69482099 0.6378630 -1.71753726  0.32801237  8668.283
## noise[4]   0.48478720 0.7175608 -0.58693263  1.69715485  7688.583
## noise[5]   0.03674986 0.6628282 -0.98949651  1.11958926  9665.280
## noise[6]   0.03362759 0.6597896 -1.01587120  1.09600121  9845.176
## noise[7]   0.46590843 0.7014273 -0.57204735  1.62669502  8403.534
## noise[8]   0.04456407 0.6569401 -0.99686753  1.12818336 10469.401
## noise[9]  -0.91962933 0.5759457 -1.85924675 -0.02796906  6573.523
## noise[10]  0.50218424 0.6210620 -0.45174042  1.50829757  9397.758
## noise[11] -0.12797284 0.5682212 -1.02334748  0.78486221  9848.370
## noise[12] -0.40138062 0.5697088 -1.31682934  0.48253322  9765.515
## noise[13] -0.12758673 0.5762110 -1.04098979  0.78460137  8800.966
## noise[14] -0.66385351 0.5709701 -1.59166334  0.22319174  6662.546
## noise[15]  0.50556146 0.6362049 -0.46520001  1.55924888 10197.575
## noise[16]  0.49625738 0.6175076 -0.45737687  1.48576113  9186.470
## noise[17]  0.49509729 0.6151085 -0.44092985  1.51394976  8034.585
## noise[18]  0.18624732 0.5767786 -0.69848173  1.12775698  7820.852
## noise[19] -0.07661383 0.5573044 -0.95263002  0.80785264  7717.989
## noise[20]  0.83167703 0.6671931 -0.16144486  1.95671620  5689.095
## noise[21]  0.19464505 0.5819445 -0.71097648  1.14702389  7451.126
## noise[22]  0.18309547 0.5776669 -0.70614102  1.12648779  7865.969
## noise[23]  0.18328049 0.5708504 -0.67334710  1.12125198  7925.271
## noise[24] -0.30576742 0.5300316 -1.16137045  0.54597808  7484.644
```

```
## noise[25] -0.78910914 0.4814627 -1.57882933 -0.04030788  4288.111
## noise[26]  0.16795164 0.4435268 -0.54818844  0.87090726  5016.983
## noise[27] -1.10624756 0.5041216 -1.95140146 -0.33984508  3915.032
## noise[28] -0.35914933 0.4513169 -1.10440956  0.34445638  4890.552
## noise[29]  0.16549236 0.4455727 -0.54287686  0.86477336  5013.810
## noise[30]  1.13439909 0.4694427  0.41592503  1.91319955  6404.464
## noise[31] -0.49688604 0.4541524 -1.23180374  0.22111570  4738.045
## noise[32] -0.22637201 0.4537001 -0.97319979  0.47856710  5035.167
## noise[33]  0.56137194 0.5948228 -0.35229658  1.54117439  7151.031
## noise[34]  0.26647243 0.5502076 -0.59439602  1.17575369  6457.501
## noise[35]  0.27072959 0.5831444 -0.61024306  1.22712621  8510.281
## noise[36] -0.21642608 0.5213621 -1.04133312  0.61468076  6964.013
## noise[37] -0.21353503 0.5227161 -1.03341887  0.63641993  6521.956
## noise[38]  0.90061775 0.6614213 -0.07121882  2.03620916  5285.137
## noise[39]  0.26820708 0.5687330 -0.60676928  1.21795751  7089.135
## noise[40]  0.01874529 0.5364286 -0.81404423  0.88423796  7401.518
## noise[41] -1.53745982 0.4995284 -2.37637983 -0.78284005  3678.493
## noise[42] -0.56200203 0.4199512 -1.24491101  0.08931944  4059.454
## noise[43] -0.46192616 0.4189954 -1.14161797  0.19516803  4077.639
## noise[44] -0.35998216 0.4100220 -1.02748820  0.26438490  4210.680
## noise[45]  0.43155906 0.4142893 -0.22437522  1.09250422  5215.604
## noise[46] -0.56222328 0.4274168 -1.25901226  0.09724261  4053.037
## noise[47]  1.50714143 0.4740836  0.76367579  2.29621729  6377.049
## noise[48] -0.06672878 0.4130152 -0.72975649  0.57827309  4043.676
## sigma      0.80031523 0.1562846  0.57529381  1.06361032  1767.559
##               Rhat4
## a[1]      1.0005043
## a[2]      1.0016412
## a[3]      1.0002620
## b[1]      1.0000724
## b[2]      1.0016533
## b[3]      1.0000765
## noise[1]  0.9998043
## noise[2]  0.9998036
## noise[3]  1.0001098
## noise[4]  0.9999792
## noise[5]  1.0000355
## noise[6]  0.9997418
## noise[7]  0.9999447
## noise[8]  0.9997996
## noise[9]  0.9994650
## noise[10] 0.9999560
## noise[11] 0.9994635
## noise[12] 0.9997690
## noise[13] 0.9996807
## noise[14] 0.9996384
## noise[15] 0.9995101
## noise[16] 1.0000416
## noise[17] 0.9999032
## noise[18] 0.9999267
## noise[19] 1.0000973
## noise[20] 1.0001326
## noise[21] 1.0004283
## noise[22] 1.0006159
```

```
## noise[23] 1.0001834
## noise[24] 1.0002333
## noise[25] 0.9998385
## noise[26] 0.9998594
## noise[27] 0.9999627
## noise[28] 0.9998330
## noise[29] 0.9996787
## noise[30] 0.9994904
## noise[31] 1.0000774
## noise[32] 0.9999280
## noise[33] 0.9996695
## noise[34] 0.9999254
## noise[35] 0.9999020
## noise[36] 0.9998738
## noise[37] 0.9998968
## noise[38] 0.9998696
## noise[39] 0.9996639
## noise[40] 0.9998192
## noise[41] 1.0004032
## noise[42] 1.0002042
## noise[43] 1.0005036
## noise[44] 0.9995727
## noise[45] 0.9998763
## noise[46] 0.9999076
## noise[47] 1.0001944
## noise[48] 1.0001631
## sigma     1.0008910
```

Simulate from the posterior: Some work is required to re-associate the output with the tank number and calculate the propotion that survived

```
post.sim.tank.size <- sim(m.tank.size, data=d) %>%
  as_tibble() %>%
  gather("tanks","surviving",1:48)%>%
  separate(tanks,c("V","tank"), sep=1) %>%
  mutate(tank=as.numeric(tank)) %>%
  left_join(select(d,tank,density,P))%>%
  mutate(propsurv = surviving/density)
```

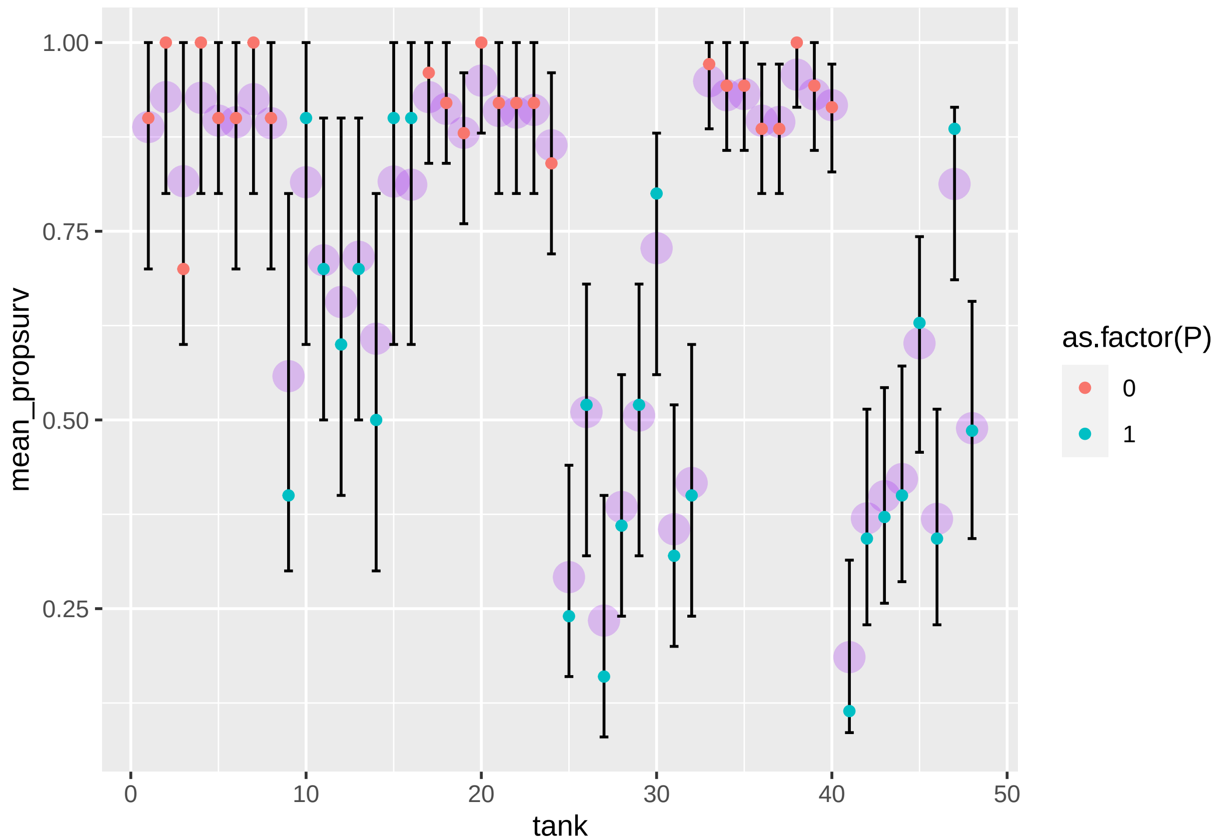
```
## Joining, by = "tank"
```

Now we summarize and plot alongside of the original data

```
summary.post.sim.tank.size <-group_by( post.sim.tank.size ,tank) %>%
  summarise(
    mean_propsurv = mean(propsurv),
    lower_propsurv = quantile(propsurv,0.1),
    upper_propsurv = quantile(propsurv,0.9),
  ) %>%
  ungroup()
```

Here the big purple dots are the mean of the posterior simulations. They have a specific pattern relative to the small red and blue dots that are the ovserved data. For tanks that all have the same preditors (density of tadpoles and presence of predators), the purple dots are always closer to the mean of that group than the actual data. THIS is shrinkage. The model is skeptical of extreme datapoints, but at least acknowledges that each tank is skewed away from the mean.

```
ggplot(summary.post.sim.tank.size, aes(x=tank,y=mean_propsurv))+
  geom_point(color="purple",alpha=0.25,size=5)+
  geom_errorbar(aes(ymin=lower_propsurv,ymax=upper_propsurv),width=0.5) +
  geom_point(data=d,aes(x=tank,y=propsurv,group=as.factor(tank), color=as.factor(P)))
```




**What happens when we remove the added variance**

Re-run this analysis, but artificially remove the tank-specific noise in the model. You can do this simply by setting the prior on `sigma` to be tightly focused on 0. Compare the posterior simulation, what has changed? Do you think this creates over-fitting?

**Does density effect predation dependent survival**

What is the effect of density on predation? i.e. what are the contrasts in the `b` values?

**Compare to a model where tank density is not included as a predictor of the effect of predation on survival**

Write a model where tank density is not a predictor of survival and compare the WAIC score to our full model. How does it relate to our result in terms of the contrasts above?