STATISTICAL RETHINKING WINTER 2019
HOMEWORK, WEEK 10 SOLUTIONS

1. Running the model without measurement error (the code was given in the assignment), we get:

```
precis( m1.1 )
```

```
      mean   sd 5.5% 94.5% n_eff Rhat
a     0.43 0.06 0.33  0.53   745 1.01
b     0.78 0.01 0.76  0.81   802 1.01
sigma 0.29 0.02 0.27  0.32   990 1.00
```

We'll plot this implied relationship in a moment, after fitting the measurement error model too. To build the measurement error model, all we really need to do is add the observation process. This means that the observed values arise from their own distribution, each having a true value as the mean. We use these unknown true values in the regression. It looks like this:

```
m1.2 <- ulam(
    alist(
    # B model
        B ~ normal( B_true , Bse ),
        vector[N_spp]:B_true ~ dlnorm( mu , sigma ),
        mu <- a + b*log( M_true[i] ),

    # M model
        M ~ normal( M_true , Mse ),
        vector[N_spp]:M_true ~ normal( 0.5 , 1 ),

    # priors
        a ~ normal(0,1),
        b ~ normal(0,1),
        sigma ~ exponential(1)
    ) ,
    data=dat_list ,
    start=list( M_true=dat_list$M , B_true=dat_list$B ) ,
    chains=4 , cores=4 , control=list(max_treedepth=15) )
```

The top chunk is the model for the B values. The first line is the measurement process. Then the next two lines are the same regression as before, but with `B_true` replacing the observed `B` values. Likewise `M_true` replaces the observed `M` in the linear model.

The second chunk is the measurement model for `M`. The prior for `M_true` covers the entire range of the normalize variable—it ranges from 0 to 1 now, recall, because we scaled it that way to start by dividing by the maximum observed value.

The last chunk holds the same priors as before.

Note the `control` list at the bottom. If you run this model without that, it will work, but be inefficient and warn about exceeding maximum "treedepth." This is not a concern for the validity of the chains, just how well they run. Treedepth is a control parameter for NUTS algorithm. The Stan manual contains more detail, if you want it.

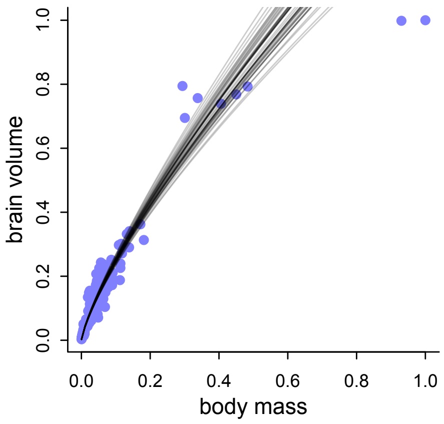
Once it finishes:

```
precis( m1.2 )
```

```
364 vector or matrix parameters hidden. Use depth=2 to show them.
      mean   sd 5.5% 94.5% n_eff Rhat
a     0.42 0.06 0.32  0.51  1133    1
b     0.78 0.01 0.76  0.81  1126    1
sigma 0.26 0.02 0.24  0.29  1483    1
```

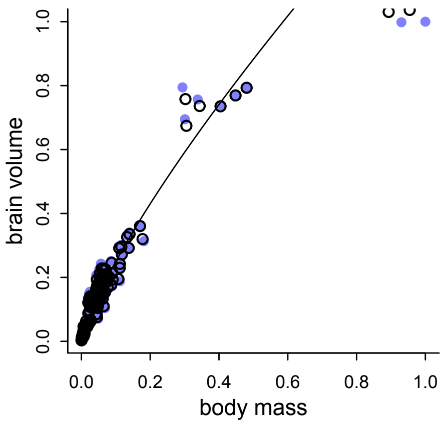
Those 364 hidden parameters are the estimated true values. We can look at those later on. For now, notice that the posterior distributions of `a` and `b` are nearly identical to `m1.1`. Adding measurement error hasn't changed a thing! Plotting the regression against the observed values:

```
plot( B ~ M , xlab="body mass" , ylab="brain volume" , col=rangi2 , pch=16 )
post <- extract.samples(m1.2)
for ( i in 1:50 ) curve( exp(post$a[i])*x^(post$b[i]) , add=TRUE , col=grau(0.2) )
```



The two points in the upper right are gorillas. Most primates are small, and obviously gorillas have something special going on. Now let's plot the estimated values on this:

```
B_est <- apply( post$B_true , 2 , mean )
M_est <- apply( post$M_true , 2 , mean )

plot( B ~ M , xlab="body mass" , ylab="brain volume" , col=rangi2 , pch=16 )
points( M_est , B_est , pch=1 , lwd=1.5 )

x_seq <- seq( from=0 , to=1 , length.out=100 )
EB <- sapply( x_seq , function(x) mean( exp(post$a)*x^(post$b) ) )
lines( x_seq , EB )
```



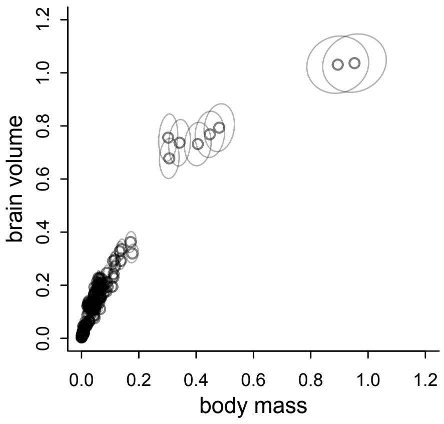
The open points are the posterior mean estimates. Notice that they have moved towards the regression line, as you'd expect. But even the outlier gorillas haven't moved much. The assumed error just isn't big enough to get them any closer.

If you increase the amount of error, you can get all of the species to fall right on the regression line. Try for example 30% error. The model will mix poorly, but take a look at the inferred true values.

The truth of this example is that there are just so many small primates that they dominate the relationship. And their measurement errors are also smaller (in absolute terms). So adding plausible amounts of measurement error here doesn't make a big difference. We still don't have a good explanation for gorillas.

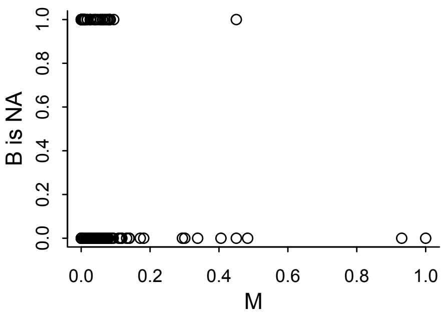
Before moving on, I'll also plot the estimated species values with 50% compatibility ellipses.

```
library(ellipse)
plot( B_est ~ M_est , xlab="body mass" , ylab="brain volume" , lwd=1.5 ,
    col=grau() , xlim=c(0,1.2) , ylim=c(0,1.2) )
for ( i in 1:length(B_est) ) {
    SIGMA <- cov( cbind( post$M_true[,i] , post$B_true[,i] ) )
    el <- ellipse( SIGMA , centre=c(M_est[i],B_est[i]) , level=0.5 )
    lines( el , col=grau(0.3) )
}
```



2. First, let's see where the missing values are, to get some idea about the missingness mechanism. If missing brain sizes are associated with certain ranges of body sizes, then it isn't plausibly MCAR (dog eats any homework). Let's plot body size against missingness:

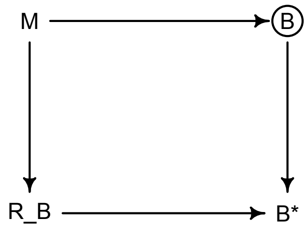
```
Bna <- is.na(d$brain[cc])
plot( Bna ~ M , ylab="B is NA" )
```



Looks like the missing brain values are almost all for small bodied species. This implies at least a MAR (dog eats students' homework) mechanism. Let's try a DAG to express it:

```
library(dagitty)
dag2 <- dagitty('dag{
    M -> R_B -> "B*" <- B
    M -> B
}')
```

```
coordinates(dag2) <- list( x=c(M=0,B=1,R_B=0,"B*"=1),
                           y=c(M=0,B=0,R_B=1,"B*"=1) )
drawdag( dag2 , shapes=list(B="c") )
```



M here is body mass, B (unobserved, suggested by the circle) is brain size, R_B is the missingness mechanism, and B* is the observed brain sizes (with missing values). The arrow from M to R_B indicates that body size influences missingness. In this case, it would imply that small body size makes a missing brain value more likely.

Now let's do some imputation. Remember that the model for imputation is really no different than an ordinary model. It just needs a prior for any variable with missing values. In this case, the missing values are in the outcome, so the likelihood is the prior we need. So the model doesn't change at all. In `ulam`:

```
dat_list <- list(
    B = B,
    M = M )

m2.1 <- ulam(
    alist(
        B ~ dlnorm( mu , sigma ),
        mu <- a + b*log(M),
        a ~ normal(0,1),
        b ~ normal(0,1),
        sigma ~ exponential(1)
    ) ,
    data=dat_list , chains=4 , cores=4 ,
    start=list( B_impute = rep(0.5,56) ) )
```

`ulam` figures out how to do the imputation. But an equivalent model that is more explicit would be:

```
m2.1b <- ulam(
    alist(
        B_merge ~ dlnorm( mu , sigma ),
        mu <- a + b*log(M),
        B_merge <- merge_missing( B , B_impute ),
        a ~ normal(0,1),
        b ~ normal(0,1),
        sigma ~ exponential(1)
    ) ,
    data=dat_list ,  chains=4 , cores=4  ,
    start=list( B_impute = rep(0.5,56) ) )
```

It's a little more obvious now what `ulam` is doing. It constructs the merged vector of observed and imputed values, `B_merge`, and then uses that merged vector as the outcome. The outcome distribution at the top of the model is the prior for each `B_impute` parameter. That prior is adaptive—it has parameters inside it. Hence, shrinkage happens.

Here is the posterior summary for the imputation model:

```
precis( m2.1 )
```

```
56 vector or matrix parameters hidden. Use depth=2 to show them.
      mean   sd 5.5% 94.5% n_eff Rhat
a     0.43 0.06 0.34  0.52  1187    1
b     0.78 0.01 0.76  0.81  1217    1
sigma 0.29 0.02 0.27  0.32  1338    1
```

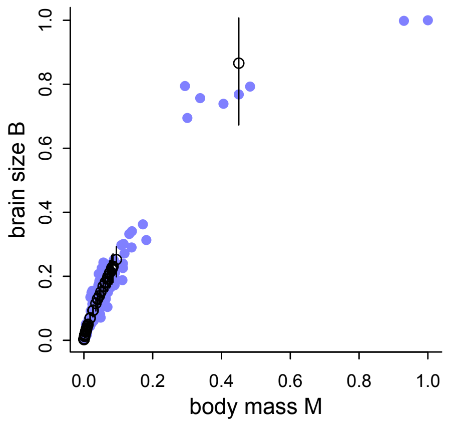
The same analysis on only complete cases:

```
cc2 <- complete.cases( B )

dat_list2 <- list(
    B = B[cc2],
    M = M[cc2] )

m2.2 <- ulam(
    alist(
        B ~ dlnorm( mu , sigma ),
        mu <- a + b*log(M),
        a ~ normal(0,1),
        b ~ normal(0,1),
        sigma ~ exponential(1)
    ) ,
    data=dat_list2 ,  chains=4 , cores=4 )

precis( m2.2 )
```

```
      mean   sd 5.5% 94.5% n_eff Rhat
a     0.43 0.06 0.33  0.52   801    1
b     0.78 0.01 0.76  0.81   813    1
sigma 0.29 0.02 0.27  0.32   960    1
```

Really no difference from before. Let's plot the imputed values:

```
post <- extract.samples(m2.1)
Bi <- apply( post$B_impute , 2 , mean )
miss_idx <- which( is.na(B) )

plot( M[-miss_idx] , B[-miss_idx] , col=rangi2 , pch=16 ,
    xlab="body mass M" , ylab="brain size B" )
points( M[miss_idx] , Bi )
Bi_ci <- apply( post$B_impute , 2 , PI , 0.5 )
for ( i in 1:length(Bi) ) lines( rep(M[miss_idx][i],2) , Bi_ci[,i] )
```



Black open points are the imputed values, with 50% compatibility intervals. Imputation hasn't done much, apparently because all but one of the missing values are in a very dense region of the body size range. So almost no information was lost—the missing info is redundant.