STATISTICAL RETHINKING WINTER 2020/2021
HOMEWORK, WEEK 10 SOLUTIONS

1. To simulate from the DAG, we can just assume everything is Gaussian and get to work. I'll simulate 1000 cases, so that sampling variance isn't an issue.

```
N <- 1000
X <- rnorm(N)   # X has no parents, so just normal(0,1)
Y <- rnorm(N,X) # X -> Y
Z <- rnorm(N,Y) # Y -> Z
dat <- list(Y=Y,X=X,Z=Z)
```

Now we have 1000 triplets of X,Y,Z. First let's run a model that ignores Z. This will give us the causal effect of X on Y:

```
library(rethinking)
m1.1 <- quap(
    alist(
        Y ~ dnorm(mu,sigma),
        mu <- a + bX*X,
        c(a,bX) ~ dnorm(0,1),
        sigma ~ dexp(1)
    ), data=dat )
precis(m1.1)
```

```
      mean   sd  5.5% 94.5%
a     0.00 0.03 -0.05  0.05
bX    0.94 0.03  0.89  0.99
sigma 0.99 0.02  0.95  1.02
```

No trouble there. bX is near 1, which is the correct effect from the simulation. Now let's add Z and see what happens:

```
m1.2 <- quap(
    alist(
        Y ~ dnorm(mu,sigma),
        mu <- a + bX*X + bZ*Z,
        c(a,bX,bZ) ~ dnorm(0,1),
        sigma ~ dexp(1)
    ), data=dat )
precis(m1.2)
```

```
      mean   sd 5.5% 94.5%
a     0.04 0.02 0.00  0.07
```

```
bX    0.47 0.03 0.42  0.51
bZ    0.50 0.02 0.48  0.53
sigma 0.70 0.02 0.67  0.72
```

The coefficient bX has halved in size. The coefficient bZ is the same size. This is clearly misleading, since bX is too small and bZ isn't even a causal path. So what has happened here? Let's look at the DAG again:

$$X \rightarrow Y \rightarrow Z$$

There is no collider here. And there are no back-doors.

This scenario is sometimes called "case-control bias" or just "selection bias". Those names may not help though. What is happening is just that since Z contains information about Y, once you know Z, learning X doesn't provide as much new information about Y. In the extreme case that Z is just a copy of Y, the coefficient of X goes to zero.

Try it! You can't solve the model, if you make Z exactly equal to Y, but you can if you just restrict the error on Z like: Z <- rnorm(N,Y,0.1). Re-run the models and you'll see.

The reason this phenomenon is related to selection bias (or case-control bias) is that when we condition on Z, we stratify the population by values of Z. Then within each stratum, we consider the association between X and Y. This is like a badly run experiment in which we first select cases based on their outcome (Y) values, creating a too-homogenous population. They are selected on their outcome values, because Z has a lot of information about outcome. It is caused by the outcome.

As a simple logical example, suppose the DAG is:

$$\text{cloudy} \rightarrow \text{raining} \rightarrow \text{ground wet}$$

Now it doesn't always rain, when it is cloudy. But the ground always gets wet, when it rains. So if we condition on wet ground before we examine the association between clouds and rain, we have removed the variation in rain. So among those days when the ground was not wet, it did not rain, and all variation in cloudiness was unrelated to rain! No association. But in the full sample of days, ignoring wet ground, the is an association.

This can happen in more realistic examples. Suppose we want to know the relationship between education (X) and income (Y). We also observe health (Z), which is influenced by income. Now suppose we divide the population in half by health status (Z). Among the healthy individuals, the average income is higher and the range of variation is smaller, than in the total population. So any relationship between education and income is also smaller, because there is less variation in income for education to explain.

Note that this is similar to, but mechanistically different from, selection bias that results from conditioning on a collider.

2. To start, here is the simulation code from the chapter, just to produce the data:

```
library(rethinking)
set.seed(73)
N <- 500
U_sim <- rnorm( N )
Q_sim <- sample( 1:4 , size=N , replace=TRUE )
E_sim <- rnorm( N , U_sim + Q_sim )
W_sim <- rnorm( N , U_sim + 0*E_sim )
dat_sim <- list(
    W=standardize(W_sim) ,
    E=standardize(E_sim) ,
    Q=standardize(Q_sim) )
```

We want a model that expresses both the relationships E ~ Q + U and W ~ E + U. So that's two simultaneous regressions. The U values are not observed, but we'll just make parameters for them, since that is all a parameter is, an unobserved variable. Here's my code:

```
m14.6b <- ulam(
    alist(
        W ~ normal( muW , sigmaW ),
        E ~ normal( muE , sigmaE ),
        muW <- aW + bEW*E + bUW*U[i],
        muE <- aE + bQE*Q + bUE*U[i],
        vector[500]:U ~ normal(0,1),
        c(aW,aE) ~ normal( 0 , 0.2 ),
        c(bEW,bQE) ~ normal( 0 , 0.5 ),
        c(bUE,bUW) ~ normal( 0 , 0.5 ),
        c(sigmaW,sigmaE) ~ exponential(1)
    ), data=dat_sim , chains=4 , cores=4 , cmdstan=TRUE )
```

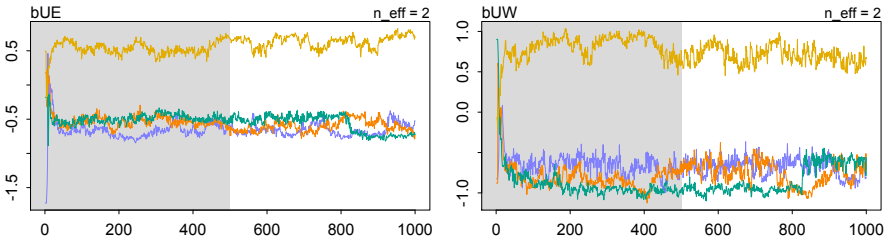
The two regressions appear at top, which U appearing as U[i] in each. The [i] is needed in ulam, just because otherwise it doesn't know whether you want the entire vector, in some matrix algebra fiesta, or just one element of it. Then the vector[500]:U line declares the 500 parameters for the 500 missing values of U.

Okay, run it and get:

```
        mean   sd  5.5% 94.5% n_eff Rhat4
aE      0.00 0.04 -0.06  0.06   585  1.00
aW      0.00 0.05 -0.07  0.07   571  1.00
bQE     0.59 0.03  0.53  0.65   560  1.00
bEW    -0.05 0.07 -0.17  0.07   316  1.01
bUW    -0.39 0.65 -0.99  0.79     2  6.02
bUE    -0.30 0.55 -0.75  0.71     2  6.74
sigmaE  0.50 0.13  0.29  0.68    20  1.24
sigmaW  0.66 0.17  0.30  0.85    12  1.43
```

The coefficient bEW is the same as in model m14.6 in the text. That is, it should be zero, and it has the same standard deviation as by the other method. The n_eff and Rhat values on some of the parameters are terrifying though. What is going on here?

What is going on is called "label switching". Let's look at some trace plots:

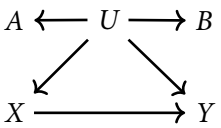


The gold chain has the opposite sign as the other chains, but otherwise explores the same range of absolute values. This is a result of the fact that the sign (plus or negative) on each U value can combine with the sign on each coefficient bE or bW to produce the same predictions. So we have two different posterior distributions, in essence, and they are equivalent in terms of inference. Some chains explore one. Some chains explore another, depending upon the starting values.

This is not ruining inference in our case, but it isn't producing ideal samples and it is disturbing. Unfortunately, there are many popular classes of models that exhibit this type of label switching. Factor analysis, for example, but also network models, finite state space models, stochastic block models, etc.

To address label switching, sometimes you can fix a couple of cases or fix the sign of a parameter. Or you can post-process the chains to flip the signs and collapse them to the same solution. If you are confident they are exploring the same solution, you can just run one long chain for inference, assuming the solution doesn't switch within a single chain (sometimes it does!).

This approach of using a big vector of parameters, instead of a covariance structure, doesn't always exhibit label switching. The details matter. Sometimes it works better than using a covariance matrix, or it would be difficult to construct the covariance structure, because it is rather constrained. Here's a weird example to consider. Take this DAG:



We want to estimate $X \rightarrow Y$, but U is unobserved and is a confound. However we have observed A and B, which are descendants of U. How can we use A and B to get a better estimate of $X \rightarrow Y$?

The answer is not to put them into a multiple regression with X. Let's simulate, assuming X has no influence on Y in reality:

```
N <- 200
U <- rnorm(N)
A <- rnorm(N,U)
B <- rnorm(N,U)
X <- rnorm(N,U)
Y <- rnorm(N,0*X+U) # no effect of X
dat2 <- list(Y=Y,X=X,A=A,B=B)
```

Now if we just do ordinary regression, we get a strong bias:

```
precis(lm(Y~X))
precis(lm(Y~X+A+B))
```

```
             mean   sd  5.5% 94.5%
(Intercept) -0.06 0.09 -0.20  0.08
X            0.53 0.06  0.42  0.63

             mean   sd  5.5% 94.5%
(Intercept) -0.07 0.08 -0.20  0.06
X            0.27 0.07  0.15  0.38
A            0.24 0.07  0.13  0.35
B            0.26 0.07  0.16  0.37
```

So that's no good. The way to go forward is to express the DAG as a model. Like this:

```
m2 <- ulam(
    alist(
        A ~ normal(U,1),         # U -> A
        B ~ normal(U,1),         # U -> B
        X ~ normal(muX,sigmaX),  # U -> X
        muX <- aX + bUX*U[i],
        Y ~ normal(muY,sigmaY),  # U -> Y, X -> Y
        muY <- aY + bUY*U[i] + bXY*X,
        vector[200]:U ~ normal(0,1),
        c(aX,aY,bXY) ~ normal(0,0.5),
        c(bUY,bUX) ~ normal(0,0.5),
        c(sigmaX,sigmaY) ~ exponential(1)
    ) , data=dat2 , chains=4 , cores=4 , cmdstan=TRUE )
precis(m2)
```

```
        mean   sd  5.5% 94.5% n_eff Rhat4
bXY     0.09 0.09 -0.07  0.24   754  1.00
aY     -0.08 0.08 -0.21  0.06  2673  1.00
aX     -0.03 0.08 -0.16  0.10  2050  1.00
bUX     0.87 0.09  0.72  1.01   988  1.00
bUY     0.88 0.14  0.65  1.10   594  1.01
```

```
sigmaY  1.05 0.07  0.94  1.17   850  1.00
sigmaX  1.04 0.07  0.94  1.15  1322  1.00
```

Smooth mixing and a much better inference for bXY. Now you can solve this problem using a covariance structure and a multi_normal. But it's hard, because you have to impose constraints on the covariance matrix. SEM software like blavaan constructs the proper constrained covariance matrix in cases like this one. But you can also build your own solution, just by writing the DAG down as a full structural causal model.

Okay, but what would the covariance matrix look like, if we did this problem the same way as the instrumental variable problem in the text? In this case, the covariance matrix is more complicated, because the entries in it are related to one another in special ways. First, consider that we want a 4-dimension multi-normal with [Y,X,A,B] in it. So we have a 4-by-4 covariance matrix. And it's entries are:

$$\begin{pmatrix} \sigma_Y^2 & rUY \times rUX + rXY & rUY \times rUA & rUY \times rUB \\ & \sigma_X^2 & rUX \times rUA & rUX \times rUB \\ & & \sigma_A^2 & rUX \times rUB \\ & & & \sigma_B^2 \end{pmatrix}$$

and the lower triangle, below the diagonal is symmetric with the upper. That is, [2,1] must be the same as [1,2]. Those r parameters in the matrix are path coefficients.

Okay, but where do these products come from? In a linear system like this one, there is a simple rule that gives the covariance between variables in a graph. The rule was figured out by Sewell Wright, the population geneticist, in the first part of the 20th century. The rule is to take each path that connects the two variables, and then multiply all the path coefficients along that path, then finally add together these products. So for example X and Y are connected by two paths: $X \rightarrow Y$ and $X \leftarrow U \rightarrow Y$. The first path yields rXY and the second $rUX \times rUY$. Adding these gives us [1,2] in the matrix above. The same procedure gives the rest of the entries.

You could code this strategy, using raw Stan, by building up the covariance matrix as a function of the other parameters. It's ugly code, but it does work. There is an example at this gist:

https://gist.github.com/rmcelreath/68fb40bf7a3e73bf2623e10a5973216d

3. This is a very hard problem. The point is the realize something of the structure of the a solution, not to fully code it.

Let's begin by keying in the data, the counts of the outcomes from 1 to 8:

```
y <- c( 18 , 19 , 22 , NA , NA , 19 , 20 , 22 )
```

You might just look at these numbers and intuit that 20 4s and 20 5s is the best guess. And you wouldn't be wrong. But we're going to justify that guess with probability theory now.

Let's begin by listing the unknown variables. Then we'll assign priors and update with the observations. The unobserved variables are (1) the number of spins, (2)

the vector of probabilities of each number 1-8, and (3) the counts of 4s and 5s. Let's consider each in turn.

Let S be the number of spins. We know that S is at least:

```
sum(y,na.rm=TRUE)
```

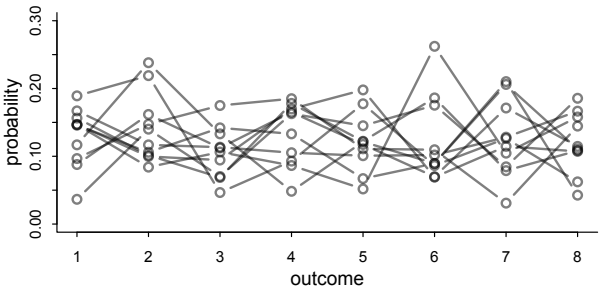
```
[1] 120
```

So that's the lower bound. There is no obvious upper bound. With only those facts to go on, one choice would be a Poisson prior for the number of spins. We also need an expected value for this Poisson prior. We could make that a parameter as well, with its own prior, or we could fix it. To keep things (relatively) simple, let's fix it at $8 \times 20 = 160$. So the prior is Poisson, but truncated below so that the minimum is 120. The easy way to approach this is to parameterize the number of spins above 120 and give that a Poisson distribution with a mean of 40. So the distribution of S is:

$$S \sim \text{Poisson}(40) + 120$$

Now we need a prior for the vector of probabilities. Let's call that vector p. As the problem suggested, we'll use a Dirichlet prior. Dirichlet was introduced in an earlier chapter, where we used it as a prior for categorical outcomes like this. The problem says we want the prior to encode the knowledge that none of the outcomes is twice as likely as the others. We can't express that directly in a Dirichlet, but we can find a parameterization that gets it qualitatively right. Let's try:

```
library(gtools)
p <- rdirichlet( 1e3 , alpha=rep(4,8) )
plot( NULL , xlim=c(1,8) , ylim=c(0,0.3) , xlab="outcome" , ylab="probability" )
for ( i in 1:10 ) lines( 1:8 , p[i,] , type="b" , col=grau() , lwd=2 )
```



Hard to tell exactly what is going on from that plot. But we can count the number of simulated draws in which any probability is more than twice as large as another. One way to do this is to sort each vector of probabilities and ask whether the largest is more than twice as big as the smallest.

```
twicer <- function( p ) {
    o <- order( p )
    if ( p[o][8]/p[o][1] > 2 ) return( TRUE ) else return( FALSE )
}
sum( apply( p , 1 , twicer ) )
```

```
[1] 978
```

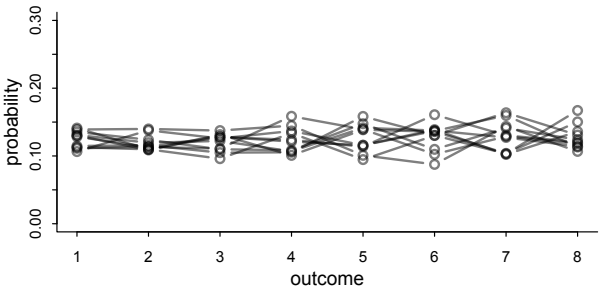
978 of 1000 simulations violate the criterion. So we need a tighter prior.

```
p <- rdirichlet( 1e3 , alpha=rep(50,8) )
sum( apply( p , 1 , twicer ) )
```

```
[1] 13
```

That's pretty good. It's also a very tight prior. Let's look at it:

```
plot( NULL , xlim=c(1,8) , ylim=c(0,0.3) , xlab="outcome" , ylab="probability" )
for ( i in 1:10 ) lines( 1:8 , p[i,] , type="b" , col=grau() , lwd=2 )
```



Now both the total number of spins S and the counts of 4s and 5s are both discrete variables. So we can't declare them in Stan as parameters. We'll instead have to marginalize over them in the model block and then use the generated quantities to invert the likelihoods to get posterior probabilities. Luckily, we can focus on the 4s and 5s, because that's all we were asked for.

I'll do this is raw Stan code, because we'll need some loops and fancier data structures. This book hasn't taught these coding techniques. But any attempt that gets the approach right—the notion that we need to marginalize over all combinations of 4s and 5s—is gold. Here's the code, and then I'll explain.

```
code15H7 <- '
data{
    int N;
    int y[N];
    int y_max; // consider at most this many spins for y4 and y5
    int S_mean;
```

```
}
parameters{
    simplex[N] p;   // probabilities of each outcome
}
model{
    vector[(1+y_max)*(1+y_max)] terms;
    int k = 1;

    p ~ dirichlet( rep_vector(50,N) );

    // loop over possible values for unknown cells 4 and 5
    // this code updates posterior of p
    for ( y4 in 0:y_max ) {
        for ( y5 in 0:y_max ) {
            int Y[N] = y;
            Y[4] = y4;
            Y[5] = y5;
            terms[k] = poisson_lpmf(y4+y5|S_mean-120)
                       + multinomial_lpmf(Y|p);
            k = k + 1;
        }//y5
    }//y4
    target += log_sum_exp(terms);
}
generated quantities{
    matrix[y_max+1,y_max+1] P45; // prob y4,y5 takes joint values
    // now compute Prob(y4,y5|p)
    {
        matrix[(1+y_max),(1+y_max)] terms;
        int k = 1;
        real Z;
        for ( y4 in 0:y_max ) {
            for ( y5 in 0:y_max ) {
              int Y[N] = y;
              Y[4] = y4;
              Y[5] = y5;
              terms[y4+1,y5+1] = poisson_lpmf(y4+y5|S_mean-120)
                                 + multinomial_lpmf(Y|p);
            }//y5
        }//y4
        Z = log_sum_exp( to_vector(terms) );
        for ( y4 in 0:y_max )
            for ( y5 in 0:y_max )
                P45[y4+1,y5+1] = exp( terms[y4+1,y5+1] - Z );
    }
}
```

```
'
```

At the top, the data block declares the observed variables and fixed parts of the priors. We feed these values in with this:

```
y <- c(18,19,22,-1,-1,19,20,22)
dat <- list(
    N = length(y),
    y = y,
    S_mean = 160,
    y_max = 40 )
```

Notice that I've replace the `NA` values in `y` with -1. This is because Stan can't accept `NA`.

The parameters block just declares the vector of outcome probabilities p. We'll assign the prior for this in the next block.

The model block is big. We'll take it one step at a time. The first line declares a vector to accumulate the marginalization terms for the probabilities of each count of 4s and 5s. We'll consider every possible combination of 4s and 5s from 0 to `y_max` (40 here) for each, for a total of `(y_max+1)*(y_max+1)` terms. When `y_max` is 40, this is 1681 terms. Stan can handle it. The counter `k` is there to help us manage this vector.

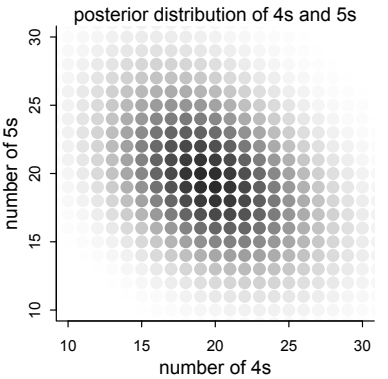
After assigning the Dirichlet prior to p, there is a big loop over all combinations of 4s and 5s. For each combination, we compute the log-probability of the entire vector `Y` (with the hypothetical 4s and 5s inserted) and that many total spins. The total spins have a Poisson probability and the vector of counts `Y` has a multinomial probability conditional on that number of spins and the vector `p`. This is the same strategy as at the end of the chapter where we marginalized over the unobserved cats. Now however there are many many (1681) terms to marginalize over. This code computes the likelihood of the observed counts, marginalizing over the unobserved 4s and 5s.

The generated quantities block repeats most of this code, in order to compute the posterior probability of each combination of 4s and 5s. It begins by declaring a matrix `P45` of these combinations. Then the loops come again. But now we store the individual log-probabilities in a big matrix of combinations of 4s and 5s. Each of these terms is $\log \text{P}(Y, S|p)$. At the bottom, we normalize these terms so that they all sum to 1 and we have a proper joint distribution over all combinations of 4s and 5s.

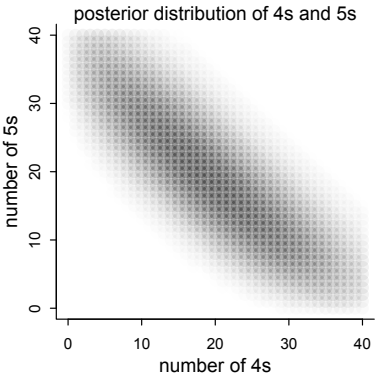
Now let's run the Stan model, extract the samples, and plot the joint distribution of 4s and 5s:

```
# use cstan() instead of stan() for cmdstan
m15H7 <- stan( model_code=code15H7 , data=dat , chains=4 , cores=4 )
post <- extract.samples(m15H7)
```

```
y_max <- dat$y_max
plot( NULL , xlim=c(10,y_max-10) , ylim=c(10,y_max-10) ,
    xlab="number of 4s" , ylab="number of 5s" )
mtext( "posterior distribution of 4s and 5s" )
for ( y4 in 0:y_max ) for ( y5 in 0:y_max ) {
    k <- grau( mean( post$P45[,y4+1,y5+1] )/0.01 )
    points( y4 , y5 , col=k , pch=16 , cex=1.5 )
}
```



So as you may have intuited, 20 4s and 20 5s is most plausible. But combinations of counts near there are almost equally plausible. Notice also the negative correlation in the counts of 4s and 5s. This is because more 4s makes more 5s less plausible, given that we have a prior on the total number of spins that keeps it from ballooning to infinity and that the prior on p vector keeps the counts of each outcome relatively close to one another. If you change the Dirichlet prior to a vector of 2s, for example, you'll get instead:



In the middle, 20 4s and 20 5s is still most plausible, but there is a lot of mass along the diagonal now.