Book Club: Statistical Rethinking

Chapter 9: Markov Chain Monte Carlo

Carles Milà

2022-06-23

Outline

Welcome to a **VERY PACKED** chapter!

Contents:

1. Intuition behind MCMC
2. MCMC Algorithms
3. Hamiltonian Monte Carlo
4. HMC in practice
5. HMC practical advice
6. Homework

Intuition: Computing the posterior and MCMC

How can we compute the posterior?

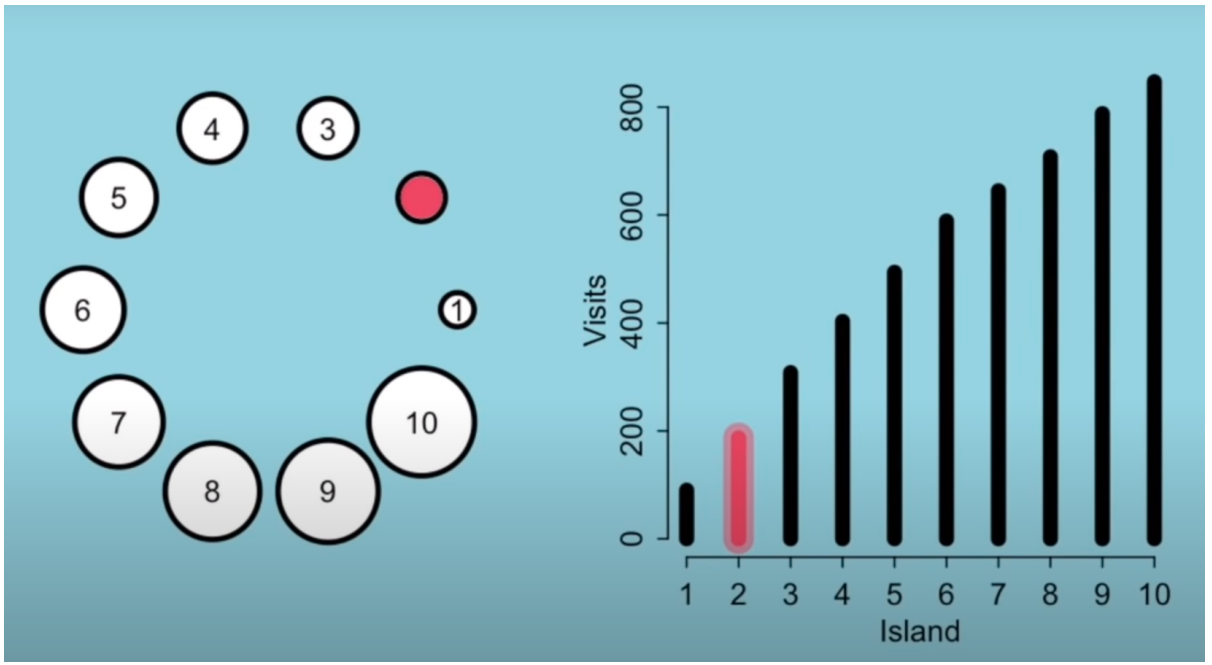
1. Analytical approach: Often impossible
2. Grid approximation: very intensive
3. Quadratic approximation: limited
4. Markov Chain Monte Carlo: intensive

Deciphering MCMC:

- Chain: Sequence of events (sort of a discrete time series)
- Markov Chain: sequence at *t* only depends on the value at the previous step *t-1*
- Monte Carlo: random simulation

Intuition: King Markov

- 10 islands, each 2 neighbours, population equal to their ID
- Each week, the King tosses a coin, if heads he considers moving clockwise, if tails counterclockwise.
- To decide if he moves or stays, he counts the population of the island where he is (stones) and that of the candidate (shells). If the the number of shells is higher than the number of stones, he'll always move. If not, he takes a random draw from a bag where there are all the shells and (#stones - #shells) stones.



Intuition: King Markov

In code format

```
num_weeks <- 1e5 # Number of time steps
positions <- rep(0,num_weeks) # vector to record the positions
current <- 10 # starting island
for ( i in 1:num_weeks ) {
  ## record current position
  positions[i] <- current
  ## flip coin to generate proposal
  proposal <- current + sample( c(-1,1) , size=1 )
  ## now make sure he loops around the archipelago
  if ( proposal < 1 ) proposal <- 10
  if ( proposal > 10 ) proposal <- 1
  ## move?
  prob_move <- proposal/current # Key step. We take a ratio, not difference!
  current <- ifelse( runif(1) < prob_move , proposal , current )
}
```

How does this translate to Bayes posterior estimation?

- Islands = parameter values
- population size = posterior probability

This code is an example of the metropolis algorithm for MCMC.

MCMC algorithms: Options

4 MCMC variants mentioned in the book:

1. **Metropolis algorithm**: see King Markov example, it requires symmetric proposal distributions (?).

2. **Metropolis-Hastings algorithm**: allows for asymmetric proposal distributions (?).

3. **Gibbs sampling**: the distribution of the proposed parameters is adjusted to the current values of the parameters, i.e. the simulation of a parameter at a given step is conditional on the current values of the rest of the parameters. Gibbs require conjugate pairs (i.e. likelihood and and prior combinations that can be solved analytically). It is the option used in BUGS and JAGS.

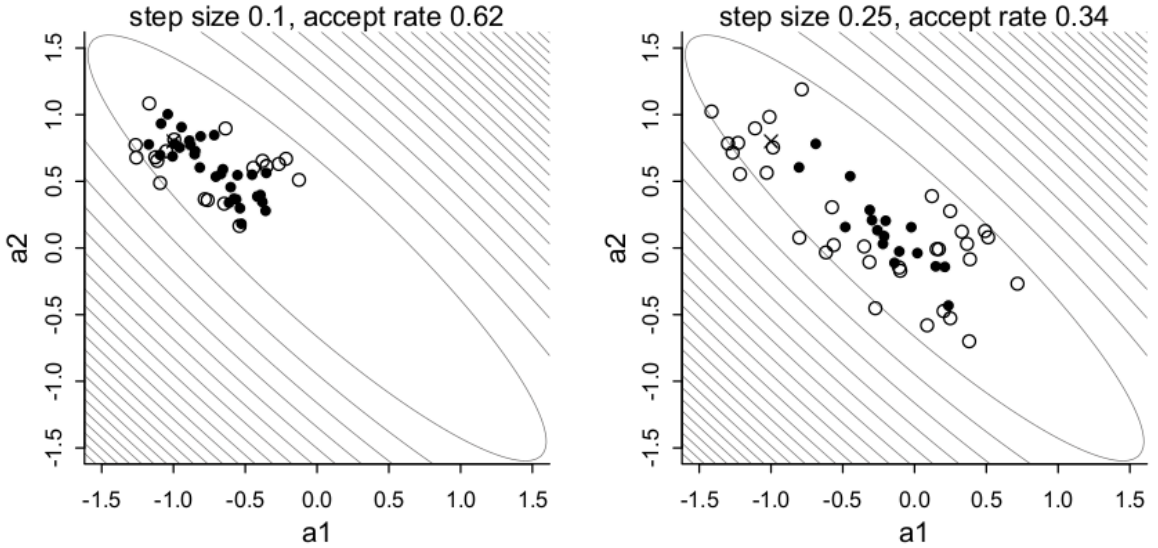
4. **Hamiltonian Monte Carlo**: The superior choice and the one used in STAN and in this book (see next section).

MCMC algorithms: Problems in high dimensionality

Complex models mean a lot of correlated parameters (i.e. narrow valleys), which make Metropolis, Metropolis-Hasting and Gibbs samplers get stuck in low probability areas.

This is known as **concentration of measure** and it is due to the fact that these algorithms do not know the global shape of the posterior and are unable to make sensible proposals.

Metropolis algorithm example where this is shown, also see the trade-off between step size and acceptance rate in this figure which leads to inefficiency.



Hamiltonian Monte Carlo

- More computationally expensive than Metropolis and Gibbs for each iteration
- However, much less shorter chains are needed and there's a net gain
- The key is that HMC takes into account the full posterior distribution to make sensible more sensible proposals than the other methods and will have a much higher acceptance rate, and therefore it is "less random".
- HMC mimics a physics system that considers the gradient of the log-posterior at a given position.
- Methaphor: Skate park where the surface is the log-posterior.
- This strategy results in a much lower autocorrelation of the simulated posteriors.
- Please, let's ignore the "King Monty’s Royal Drive" example, it's confusing -.- Let's focus on the Bayesian stats

Hamiltonian Monte Carlo

Let's consider a simple example with two parameters:

$$
\begin{aligned}
x_i &\sim \text{Normal}(\mu_x, 1) \\
y_i &\sim \text{Normal}(\mu_y, 1) \\
\mu_x &\sim \text{Normal}(0, 0.5) \\
\mu_y &\sim \text{Normal}(0, 0.5)
\end{aligned}
$$

Ingredients we need to run HMC:

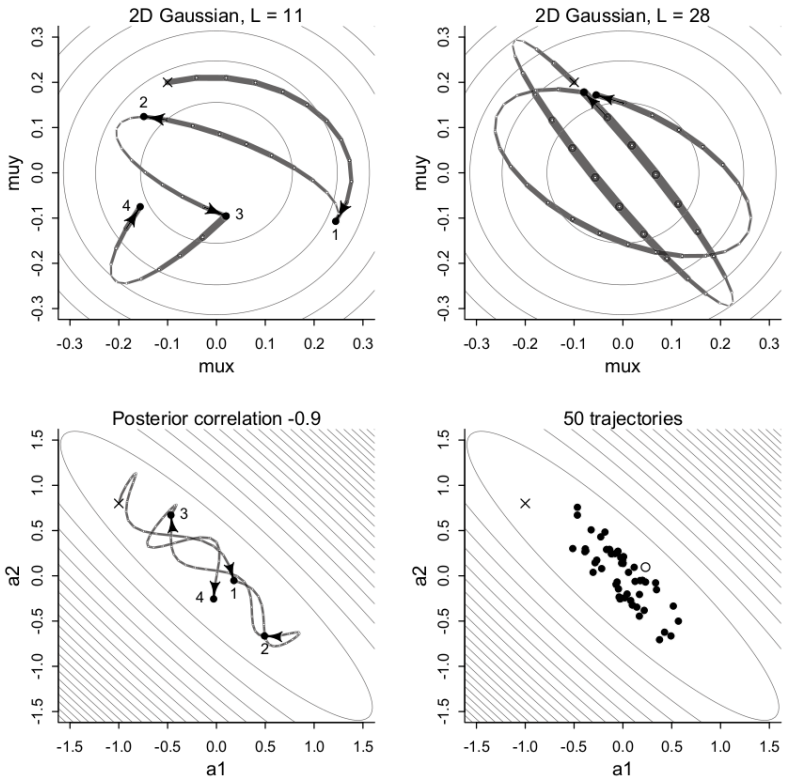
1. Log-probability of the data and parameters:
$$\sum_i \log p(y_i|\mu_y, 1) + \sum_i \log p(x_i|\mu_x, 1) + \log p(\mu_y|0, 0.5) + \log p(\mu_x, 0, 0.5)$$
2. Gradient, i.e. the derivative in all directions (in our case, 2) at the current position.
$$\frac{\partial U}{\partial \mu_x} = \frac{\partial \log \text{N}(x|\mu_x, 1)}{\partial \mu_x} + \frac{\partial \log \text{N}(\mu_x|0, 0.5)}{\partial \mu_x} = \sum_i \frac{x_i - \mu_x}{1^2} + \frac{0 - \mu_x}{0.5^2}$$
3. Leapfrog steps (self-tuned during warming phase): number of leaps in each step.
4. Step size (self-tuned during warming phase): length of those steps.

Limitations:

- Can't handle discrete parameters directly: We need to take derivatives!
- Difficult posterior distributions regardless of the MCMC sampler.

Hamiltonian Monte Carlo

- Fig 1: Leapfrog jumps and momentum
- Fig 2: U-turn danger and why we need to tune the number of jumps and step size
- Fig 3: HMC shines with autocorrelated data
- Fig 4: HMC has a really low rejection rate



Hamiltonian Monte Carlo

I think that HMC from both a theoretical and applied point of view is impossible to 100% understand unless we dig deeper into the lit, which I haven't had time to do :(

```
library(shape) # for fancy arrows
Q <- list()
Q$q <- c(-0.1,0.2)
pr <- 0.3
plot( NULL , ylab="muy" , xlab="mux" , xlim=c(-pr,pr) , ylim=c(-pr,pr) )
step <- 0.03
L <- 11 # 0.03/28 for U-turns --- 11 for working example
n_samples <- 4
path_col <- col.alpha("black",0.5)
points( Q$q[1] , Q$q[2] , pch=4 , col="black" )
for ( i in 1:n_samples ) {
  Q <- HMC2( U , U_gradient , step , L , Q$q )
  if ( n_samples < 10 ) {
    for ( j in 1:L ) {
      K0 <- sum(Q$ptraj[j,]^2)/2 # kinetic energy
      lines( Q$traj[j:(j+1),1] , Q$traj[j:(j+1),2] , col=path_col , lwd=1+2*K0 )
    }
    points( Q$traj[1:L+1,] , pch=16 , col="white" , cex=0.35 )
    Arrows( Q$traj[L,1] , Q$traj[L,2] , Q$traj[L+1,1] , Q$traj[L+1,2] ,
            arr.length=0.35 , arr.adj = 0.7 )
    text( Q$traj[L+1,1] , Q$traj[L+1,2] , i , cex=0.8 , pos=4 , offset=0.4 )
  }
  points( Q$traj[L+1,1] , Q$traj[L+1,2] , pch=ifelse( Q$accept==1 , 16 , 1 ) ,
          col=ifelse( abs(Q$dH)>0.1 , "red" , "black" ) )
}

HMC2 <- function (U, grad_U, epsilon, L, current_q) {
  q = current_q
  p = rnorm(length(q),0,1) # random flick - p is momentum.
  current_p = p
  # Make a half step for momentum at the beginning
  p = p - epsilon * grad_U(q) / 2
  # initialize bookkeeping - saves trajectory
  qtraj <- matrix(NA,nrow=L+1,ncol=length(q))
  ptraj <- qtraj
```

HMC implementation

Here things get simpler, we'll just use `rstan` (HMC MCMC) through the `ulam` function in `rethinking` using the county-wise GDP, ruggedness dataset.

Remember we hypothesised that the association of country-level GDP and ruggedness was different in Africa vs. the rest of the world.

First, we create a new list objects with the standardised vars and no NAs:

```
# Data prep
data(rugged)
d <- rugged
d$log_gdp <- log(d$rgdppc_2000)
dd <- d[ complete.cases(d$rgdppc_2000) , ]
dd$log_gdp_std <- dd$log_gdp / mean(dd$log_gdp)
dd$rugged_std <- dd$rugged / max(dd$rugged)
dd$cid <- ifelse( dd$cont_africa==1 , 1 , 2 )
dat_slim <- list(
  log_gdp_std = dd$log_gdp_std,
  rugged_std = dd$rugged_std,
  cid = as.integer( dd$cid )
)
str(dat_slim)
```

```
## List of 3
##  $ log_gdp_std: num [1:170] 0.88 0.965 1.166 1.104 0.915 ...
##  $ rugged_std : num [1:170] 0.138 0.553 0.124 0.125 0.433 ...
##  $ cid        : int [1:170] 1 2 2 2 2 2 2 2 1 ...
```

HMC implementation

The model is fit using a similar call than the one we had for `quap`, but now using `ulam`:

```
m9.1 <- ulam(
  alist(
    log_gdp_std ~ dnorm( mu , sigma ) ,
    mu <- a[cid] + b[cid]*( rugged_std - 0.215 ) ,
    a[cid] ~ dnorm( 1 , 0.1 ) ,
    b[cid] ~ dnorm( 0 , 0.3 ) ,
    sigma ~ dexp( 1 )
  ) , data=dat_slim , chains=4, cores=4) # We run 4 different chains in parallel
```

HMC run info:

```
show( m9.1 )
```

```
## Hamiltonian Monte Carlo approximation
## 2000 samples from 4 chains
##
## Sampling durations (seconds):
##         warmup sample total
## chain:1   0.04   0.03  0.06
## chain:2   0.04   0.03  0.07
## chain:3   0.04   0.03  0.07
## chain:4   0.04   0.03  0.07
##
## Formula:
## log_gdp_std ~ dnorm(mu, sigma)
## mu <- a[cid] + b[cid] * (rugged_std - 0.215)
## a[cid] ~ dnorm(1, 0.1)
## b[cid] ~ dnorm(0, 0.3)
## sigma ~ dexp(1)
```

HMC implementation

Parameter info:

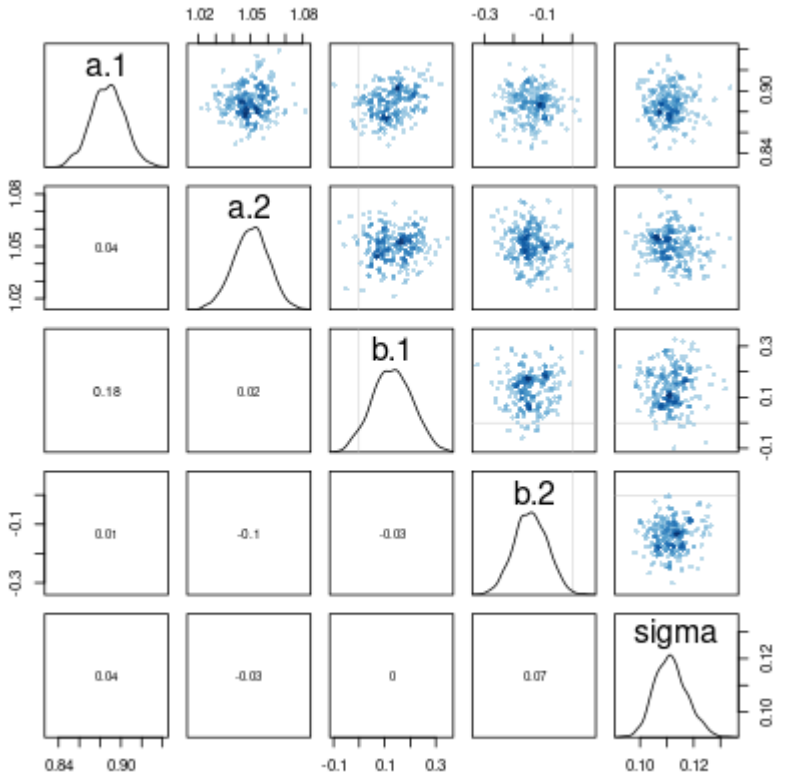
```
precis( m9.1 , 2)
```

```
##             mean          sd         5.5%       94.5%    n_eff     Rhat4
## a[1]   0.8872974 0.016131253  0.861654020  0.91263092 2552.426 0.9993882
## a[2]   1.0503005 0.010085920  1.033767800  1.06611165 2817.680 1.0002291
## b[1]   0.1302304 0.077430610  0.003933119  0.25543363 2552.208 0.9992614
## b[2]  -0.1411658 0.056897720 -0.233109325 -0.04992887 2475.159 0.9989559
## sigma  0.1115433 0.006113859  0.102582560  0.12221258 2421.178 0.9994340
```

HMC implementation

Posterior viz:

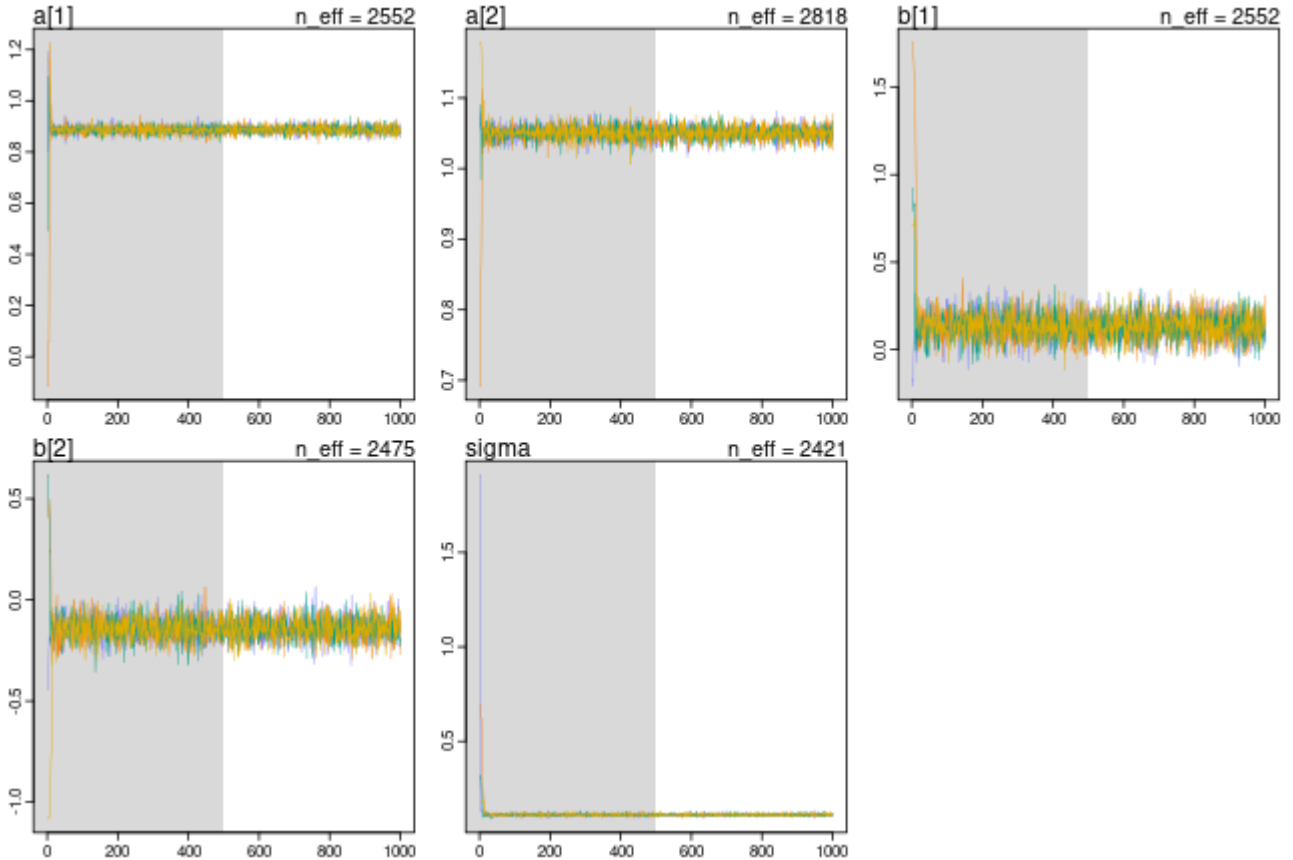
```
pairs( m9.1 )
```



HMC implementation

Chain check (healthy convergence):

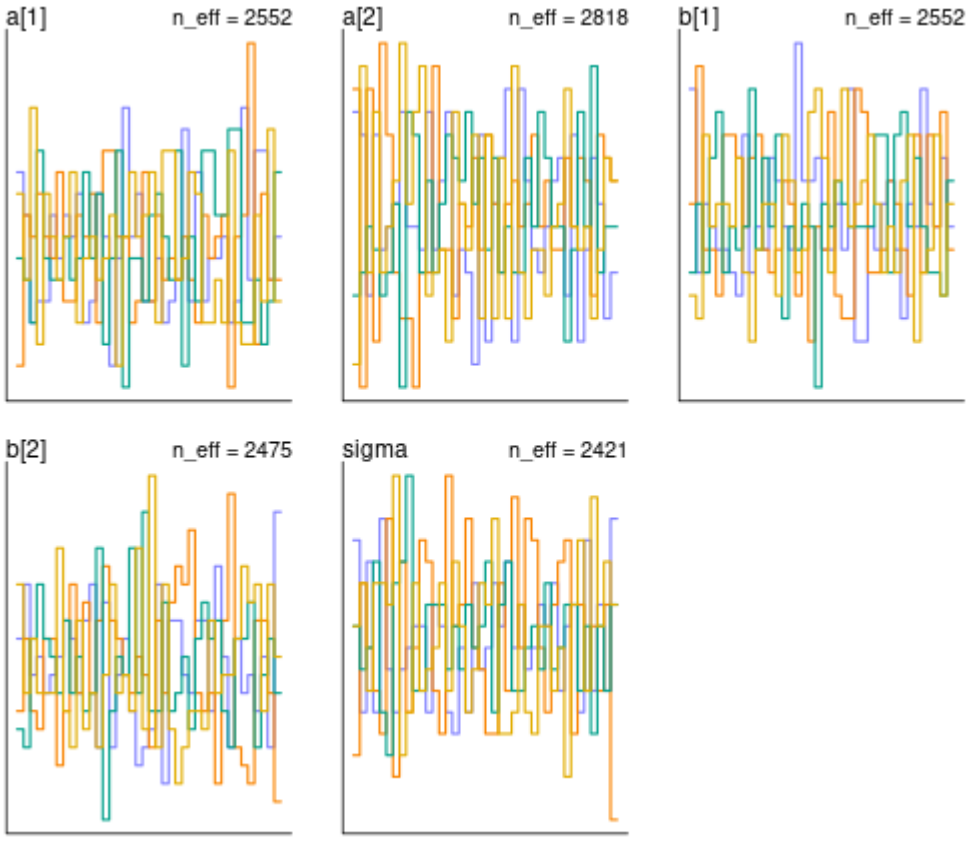
```
traceplot_ulam( m9.1 )
```



HMC implementation

Trace rank (Trank) plot (healthy convergence):

```
trankplot( m9.1 )
```



HMC implementation

Next step: Writing STAN code directly without `rethinking`:

```
data{
  vector[170] log_gdp_std;
  vector[170] rugged_std;
  int cid[170];
}
parameters{
  vector[2] a;
  vector[2] b;
  real<lower=0> sigma;
}
model{
  vector[170] mu;
  sigma ~ exponential( 1 );
  b ~ normal( 0 , 0.3 );
  a ~ normal( 1 , 0.1 );
  for ( i in 1:170 ) {
    mu[i] = a[cid[i]] + b[cid[i]] * (rugged_std[i] - 0.215);
  }
  log_gdp_std ~ normal( mu , sigma );
}
```

HMC practical advice

How many samples?

- Default: 500 warm up, 500 actual.
- What really matters are the **effective samples** (shown in `precis`)
- *n_eff* can be interpreted as the number of samples in a chain with no autocorrelation
- number of effective samples is relative to the objective: 200 is ok for the mean, but we need more to estimate the tails
- warm up period, where the number of jumps and step size parameters are tuned:
    - Generally the shorter, the better.
    - However, for complex models, it may need to be longer.

HMC practical advice

How many chains?:

- Debugging: 1 chain.
- Chain validation: >1 chain, ideally ~4, as we need to compare them.
- Analysis:
    - 1 long chain (saves warm up computing time)
    - several chains in parallel (e.g. HPC environment)

HMC practical advice

Detecting problems in chains: extremely flat priors and low sample size

```
y <- c(-1,1) # 2 samples
set.seed(11)
m9.2 <- ulam(
  alist(
    y ~ dnorm( mu , sigma ) , # We want to estimate the posterior mean and SD
    mu <- alpha ,
    alpha ~ dnorm( 0 , 1000 ) ,
    sigma ~ dexp( 0.0001 )
  ) , data=list(y=y) , chains=3 )
```

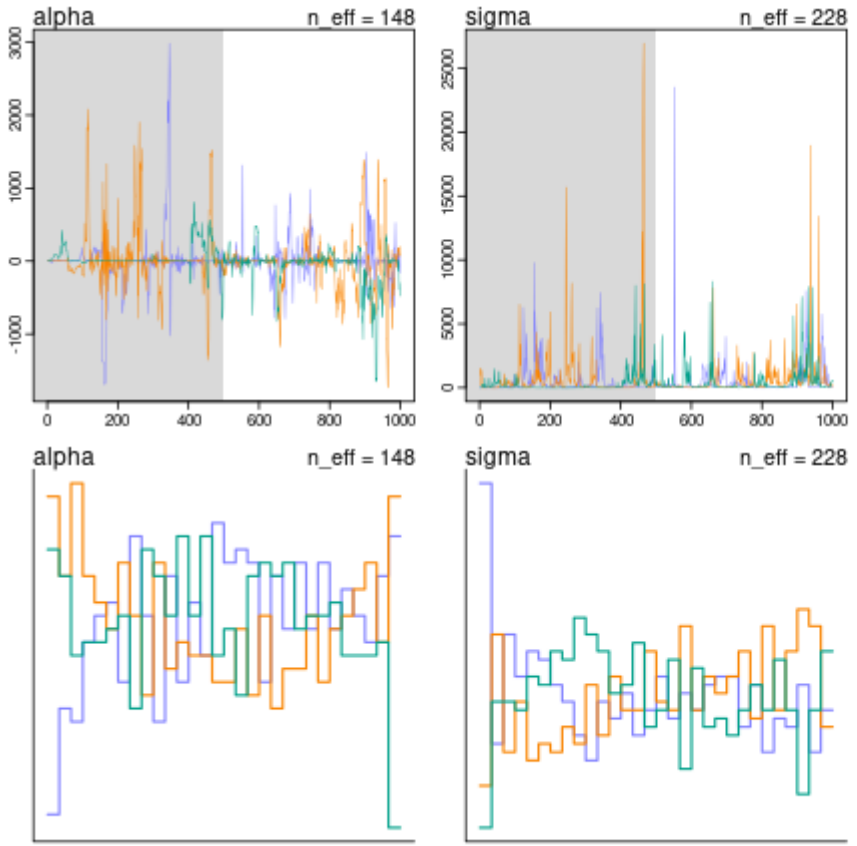
```
precis( m9.2 )
```

```
##            mean        sd      5.5%     94.5%    n_eff    Rhat4
## alpha -32.03234  303.6359 -562.9610  317.1984 147.8436 1.017110
## sigma 509.27442 1274.1540   11.7258 2053.1678 227.5873 1.012459
```

Something is clearly wrong, let's look at the traceplot and trankplot

Homework

```
traceplot_ulam(m9.2)
trankplot(m9.2)
```



The chains are not stationary and they are not mixing well. Don't use them!

HMC practical advice

Let's refit the model with more sensible weakly informative priors:

```
y <- c(-1,1) # 2 samples
set.seed(11)
m9.2 <- ulam(
  alist(
    y ~ dnorm( mu , sigma ) , # We want to estimate the posterior mean and SD
    mu <- alpha ,
    alpha ~ dnorm( 1 , 10 ) ,
    sigma ~ dexp( 1 )
  ) , data=list(y=y) , chains=3 )
```

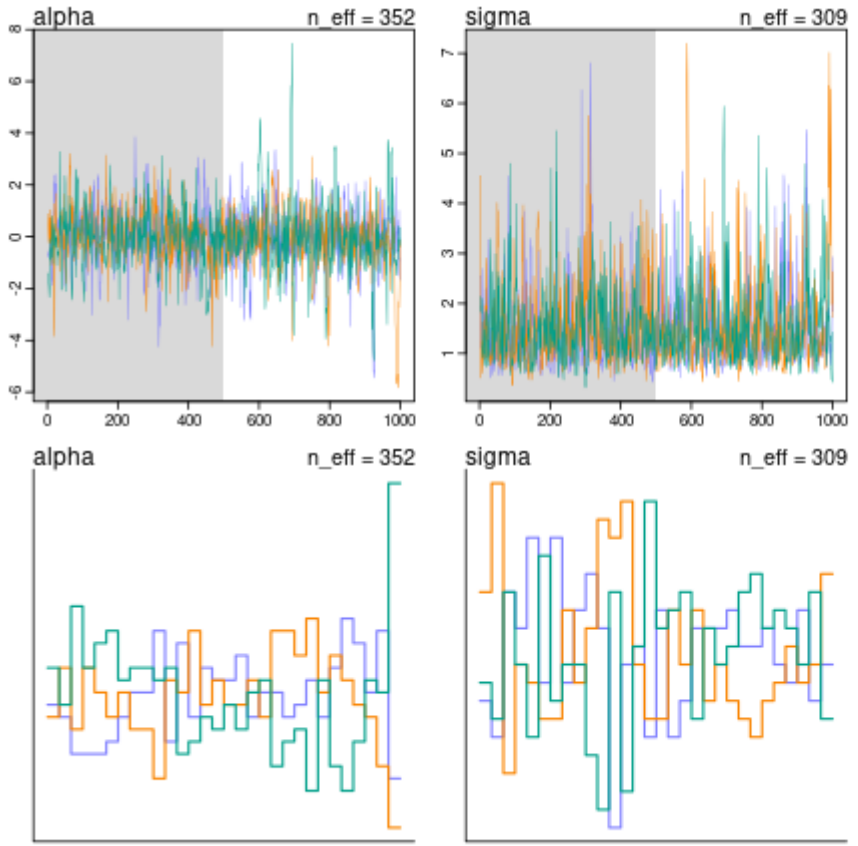
```
precis( m9.2 )
```

```
##              mean        sd      5.5%    94.5%    n_eff    Rhat4
## alpha -0.06754232 1.2989894 -2.019287 1.808546 352.0788 1.007502
## sigma  1.57302104 0.8826424  0.703107 3.182312 309.0911 1.001023
```

Now everything looks good!

Homework

```
traceplot_ulam(m9.2)
trankplot(m9.2)
```



HMC practical advice

Unidentifiable parameters: A model with two intercepts

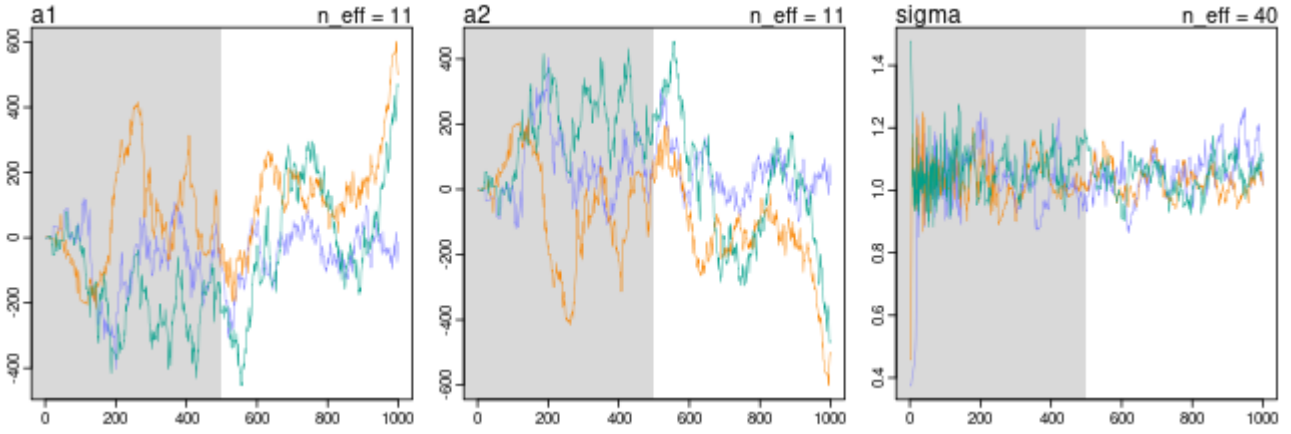
```
set.seed(41)
y <- rnorm( 100 , mean=0 , sd=1 )
set.seed(384)
m9.4 <- ulam(
  alist(
    y ~ dnorm( mu , sigma ) ,
    mu <- a1 + a2 ,
    a1 ~ dnorm( 0 , 1000 ),
    a2 ~ dnorm( 0 , 1000 ),
    sigma ~ dexp( 1 )
  ) , data=list(y=y) , chains=3 )
```

```
precis( m9.4 )
```

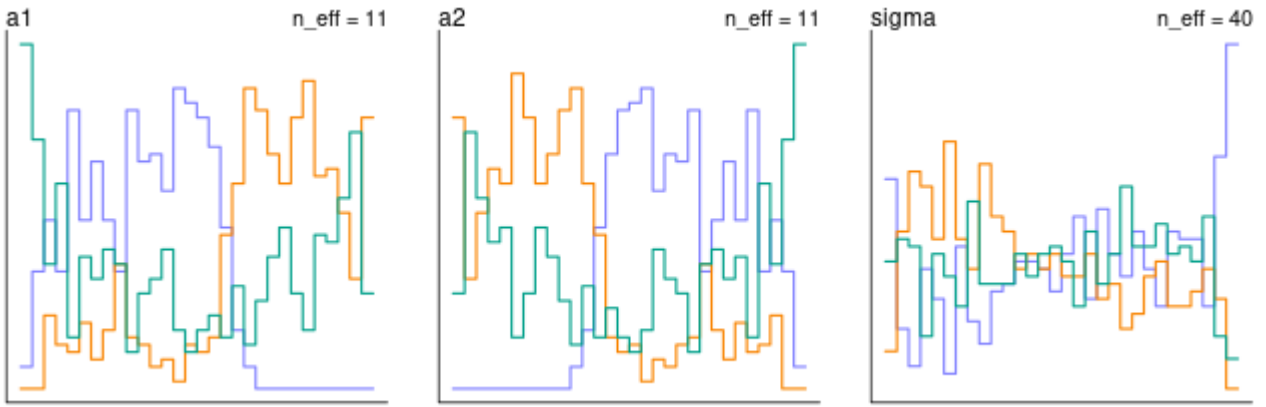
```
##             mean           sd         5.5%      94.5%    n_eff    Rhat4
## a1     32.404801 169.71687188 -230.8415950 282.878385 10.97090 1.249336
## a2    -32.216218 169.71871740 -282.6370100 230.987735 10.97245 1.249321
## sigma   1.043683   0.06013378    0.9560946   1.141507 39.96240 1.109095
```

Homework

```
traceplot_ulam(m9.4)
```




```
trankplot(m9.4)
```



HMC practical advice

Same model but with regularizing priors

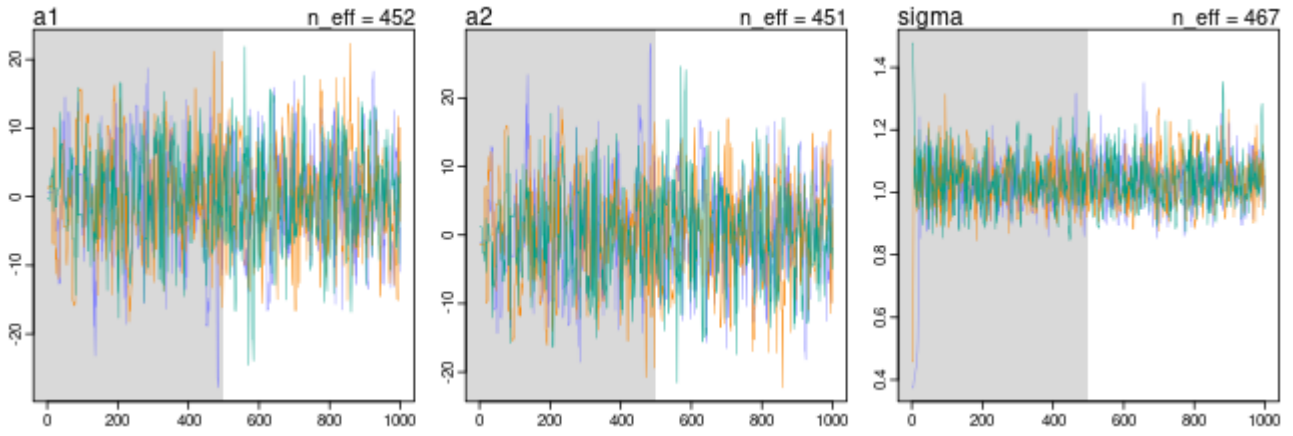
```
set.seed(41)
y <- rnorm( 100 , mean=0 , sd=1 )
set.seed(384)
m9.4 <- ulam(
  alist(
    y ~ dnorm( mu , sigma ) ,
    mu <- a1 + a2 ,
    a1 ~ dnorm( 0 , 10 ),
    a2 ~ dnorm( 0 , 10 ),
    sigma ~ dexp( 1 )
  ) , data=list(y=y) , chains=3 )
```

```
precis( m9.4 )
```

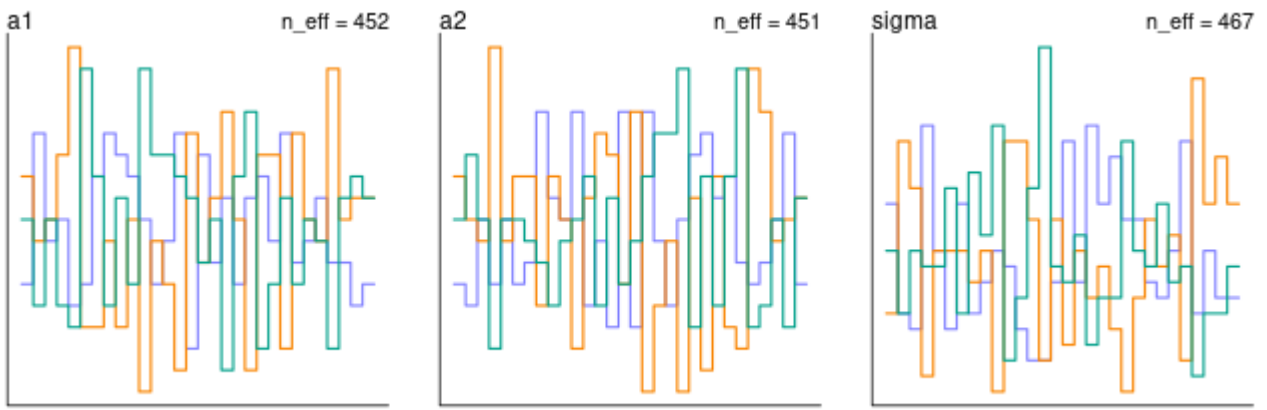
```
##             mean         sd        5.5%     94.5%    n_eff    Rhat4
## a1    -0.3529090 6.92199408 -11.4279485 10.782956 451.8703 1.000927
## a2     0.5451897 6.92796936 -10.5963640 11.648545 451.1268 1.000912
## sigma  1.0368838 0.07697601   0.9203446  1.164551 466.9935 1.002262
```

Homework

```
traceplot_ulam(m9.4)
```




```
trankplot(m9.4)
```



Homework

Revisit the marriage, age, and happiness collider bias example from Chapter 6 (Remember: Happiness -> Marriage <- Age). Run models m6.9 and m6.10 again (pages 178–179). Compare these two models using both PSIS and WAIC. Which model is expected to make better predictions, according to these criteria? On the basis of the causal model, how should you interpret the parameter estimates from the model preferred by PSIS and WAIC?

```
# Model 1: Happiness as a function of age and marriage
# We wrongly adjust for a collider and induce an association between age and happiness
m6.9 <- quap(
  alist(
    happiness ~ dnorm( mu , sigma ),
    mu <- a[mid] + bA*A,
    a[mid] ~ dnorm( 0 , 1 ),
    bA ~ dnorm( 0 , 2 ),
    sigma ~ dexp(1)
  ) , data=d2 )

# Model 2: Happiness as a function of age
# We don't adjust for the collider and consquentlythere's no association between age and happiness
m6.10 <- quap(
  alist(
    happiness ~ dnorm( mu , sigma ),
    mu <- a + bA*A,
    a ~ dnorm( 0 , 1 ),
    bA ~ dnorm( 0 , 2 ),
    sigma ~ dexp(1)
  ) , data=d2 )
```

Homework

```
# Compare them based on PSIS and WAIC
compare(m6.9, m6.10, func = PSIS)
```

```
##           PSIS       SE   dPSIS      dSE    pPSIS       weight
## m6.9  2713.996 37.56527   0.000       NA 3.751076 1.000000e+00
## m6.10 3101.925 27.75875 387.929 35.40121 2.350136 5.784794e-85
```

```
compare(m6.9, m6.10, func = WAIC)
```

```
##           WAIC       SE    dWAIC      dSE    pWAIC       weight
## m6.9  2714.256 37.51051   0.0000       NA 3.861508 1.000000e+00
## m6.10 3101.883 27.68399 387.6271 35.34372 2.328445 6.727359e-85
```

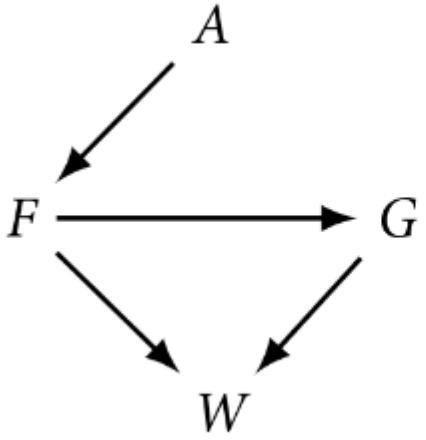
Both PSIS and WAIC indicate that model m6.9 is expected to make better predictions.

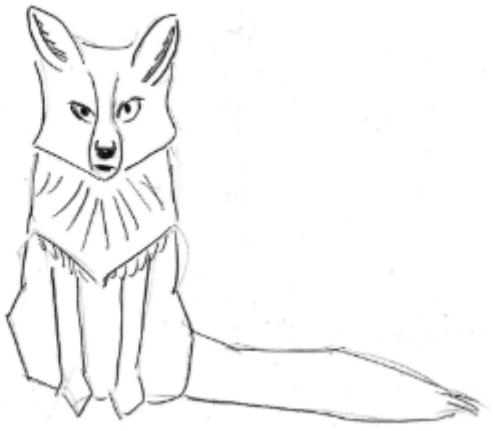
We cannot interpret the effect of age causally since we are adjusting for a collider.

We can't interpret the effect of marriage causally either since the arrow goes from happiness to marriage, and not the other way round.

Homework

Reconsider the urban fox analysis from last week's homework (W: weight, A: area, F: food, G: group size). On the basis of PSIS and WAIC scores, which combination of variables best predicts body weight? How would you interpret the estimates from the best scoring model?



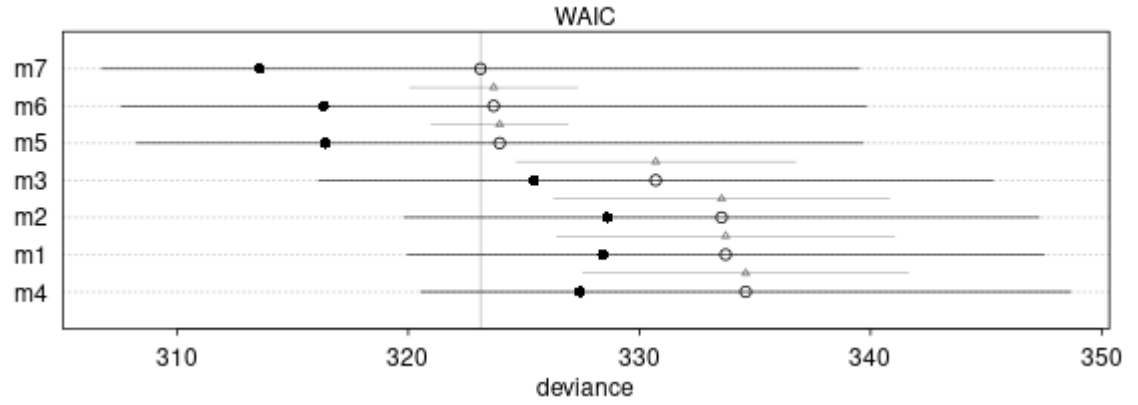


7 possible models:

1. W~A
2. W~F
3. W~G
4. W~A+F
5. W~A+G
6. W~G+F
7. W~A+F+G

Homework

```
plot(compare(m1, m2, m3, m4, m5, m6, m7, func=WAIC))
```




```
precis(m7)
```

```
##                mean         sd         5.5%      94.5%
## a     -3.781545e-06 0.07936018 -0.126836677  0.1268291
## bF     2.969014e-01 0.20959594 -0.038073424  0.6318762
## bG    -6.397409e-01 0.18160973 -0.929988318 -0.3494935
## bA     2.784050e-01 0.17010803  0.006539526  0.5502705
## sigma  9.311809e-01 0.06099587  0.833697688  1.0286641
```

It looks like model 7: W~A+F+G is the better choice (black: in sample, white: out-of-sample WAIC).

The coefficient of F is the direct effect of F on W. The coefficient of G is the total effect of G on W. Not sure about the coefficient of A to be honest, what do you think?

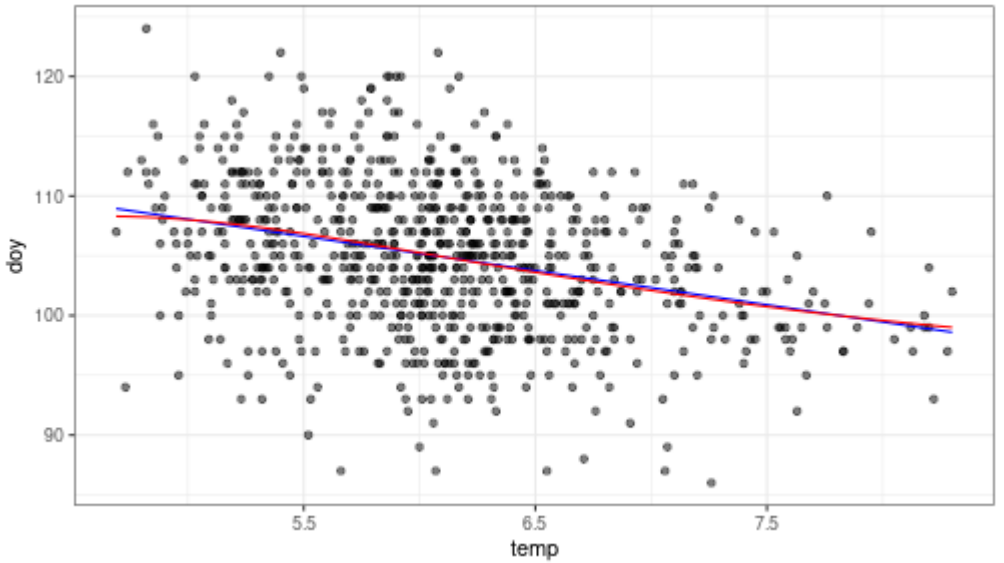
Homework

Build a predictive model of the relationship shown on the cover of the book, the relationship between the timing of cherry blossoms and March temperature in the same year. The data are found in data(cherry_blossoms) . Consider at least two functions to predict doy with temp. Compare them with PSIS or WAIC.

```
set.seed(1234)
data(cherry_blossoms)
d <- cherry_blossoms
d2 <- d[ complete.cases(d$doy) & complete.cases(d$temp) , ] # complete cases on doy
# Linear model
m1 <- quap(
  alist(
    doy ~ dnorm( mu , sigma ),
    mu <- a + b1*temp,
    a ~ dnorm(103,10), # Day 103: April 15th
    b1 ~ dnorm(0,5), # Not sure TBH
    sigma ~ dexp(1)
  ), data=d2)
# Smooth model
num_knots <- 3
knot_list <- quantile( d2$temp , probs=seq(0,1,length.out=num_knots) )
B <- bs(d2$temp,
        knots=knot_list[-c(1,num_knots)] ,
        degree=3 , intercept=TRUE )
m2 <- quap(
  alist(
    doy ~ dnorm( mu , sigma ),
    mu <- a + B %*% w ,
    a ~ dnorm(103,10), # Day 103: April 15th
    w ~ dnorm(0,10),
    sigma ~ dexp(1)
  ), data=list( doy=d2$doy, temp=d2$temp, B=B),
  start=list( w=rep( 0 , ncol(B) ) ) )
```

Homework

Let's see and compare the fitted models




```
compare(m1, m2)
```

```
##        WAIC       SE    dWAIC      dSE    pWAIC     weight
## m1 5035.798 41.06153 0.000000       NA 2.824132 0.94063324
## m2 5041.323 41.28119 5.525638 1.706777 5.690119 0.05936676
```

The linear model seems to be better.

Homework

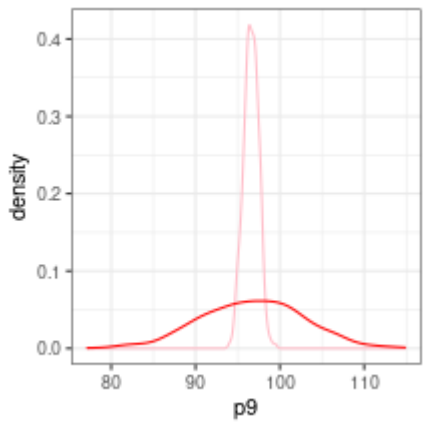
Suppose March temperatures reach 9 degrees by the year 2050. What does your best model predict for the predictive distribution of the day-in-year that the cherry trees will blossom?

```
musim <- data.frame(p9 = link(m1, data=data.frame(temp=9)))
ppsim <- data.frame(p9 = sim(m1, data=data.frame(temp=9)))
precis(musim); precis(ppsim)
```

```
##        mean        sd     5.5%    94.5%    histogram
## p9 96.58913 0.8855801 95.09484 97.89724 ▁▂▅▇▇▃▁▁▁▁
```

```
##        mean       sd     5.5%    94.5% histogram
## p9 96.73499 6.097821 87.15802 106.4441 ▁▂▅▇▃▁▁▁
```

```
ggplot() +
  geom_density(data = musim, aes(x = p9), col = "pink") +
  geom_density(data = ppsim, aes(x = p9), col = "red")+
  theme_bw()
```



Homework

The data in data(Dinosaurs) are body mass estimates at different estimated ages for six different dinosaur species. See ?Dinosaurs for more details. Choose one or more of these species (at least one, but as many as you like) and model its growth. To be precise: Make a predictive model of body mass using age as a predictor. Consider two or more model types for the function relating age to body mass and score each using PSIS and WAIC. Which model do you think is best, on predictive grounds? On scientific grounds? If your answers to these questions differ, why?