Introduction to Bayesian Statistics in R

Kelly Heilman

5/12/2020

Overview

In this workshop, we will cover the very basics of Bayesian Statistics, and go over an example of how to specify your first bayesian linear regression in R using JAGS!

Prerequisites:

If you want to follow along with the code you need to:

- install JAGS itself http://mcmc-jags.sourceforge.net/
- have R/Rstudio installed
- have the "rjags", "coda", and "ggplot2" libraries installed
- clone or download this repository with this Rmd file here: https://github.com/Kah5/ResBaz_JAGS_tutorial.git

Objectives

We will cover the very basics of Bayesian Statistics, and go over an example of how to specify your first bayesian linear regression in R! At the end of this tutorial, learners should understand bayes theorum, and have some experience running a bayesian model in R.

I have also listed further resources and things to watch out for when working in the bayes frame of mind.

What are the advantages of Bayesian analysis?

- All parameters are/can be treated as as random variables, including our data!
  - rather than assuming our data and covariates are the single *true* estimate, we acknowlege that these values come from a distribution that represents the True (with a capital T) value of the process we are interested in.
- Allow for estimation of and parsing different sources of uncertainty
  - Bayesian predictive intervals can include different sources of uncertainty: unexplained process error, parameter uncertainty, data uncertainty, and driver uncertainty
- Bayesian methods can accomodate heiarchical and different process models
  - borrow strength across different groups or treatments
  - can specify appropriate distributions, and different process models, including state-space models, logistic functions, etc.

Bayes Theorem

Bayes theorem is the basis for Bayesian analyses, so we should introduce it. It states that the conditional probability of A, given B is given by the Probabiliity of B given A, multiplied by Probability of A, and normalized by the Probability of B

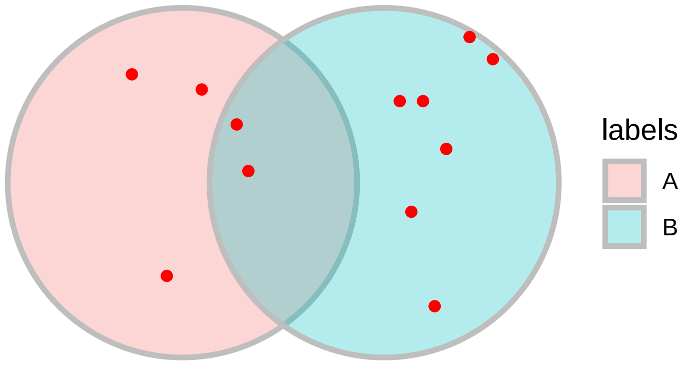
$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Where $P(B|A)$ is the likelihood function, $P(A)$ is the prior or marginal probabilty of A, $P(B)$ is the marginal probabiility of B

Now I am assuming you know a bit about probability, but lets review a minute. Lets reveiw using this venn diagram.

Based on the distribution of points in groups A and B:

1. P(A) Marginal probability of being in group A = (5/12)
2. P(B) Marginal probability of being in group B = (9/12)
3. P(A, B) Joint probability of being in A and B = (2/12)
4. P(A|B) Conditional probability that we are in A, given that we are in B? (2/9)
5. P(B|A) Conditional probaiblity that we are in B, given that we are in A? (2/5)



So what does Bayes theorum mean in the context of a model?

Lets say we have a regression model with parameters denoted by θ and data denoted by X we can use bayes theorem to determine the conditional probability of our model parameters, given the data $P(\theta|X)$, also called the *posterior*.

$$P(\theta|X) = \frac{P(X|\theta) * P(\theta)}{P(X)}$$

The posterior is equal to the the liklihood $P(X|\theta)$ (you might recognize this from frequentist/maximum liklihood methods) times the prior probability of the model parameters $P(\theta)$

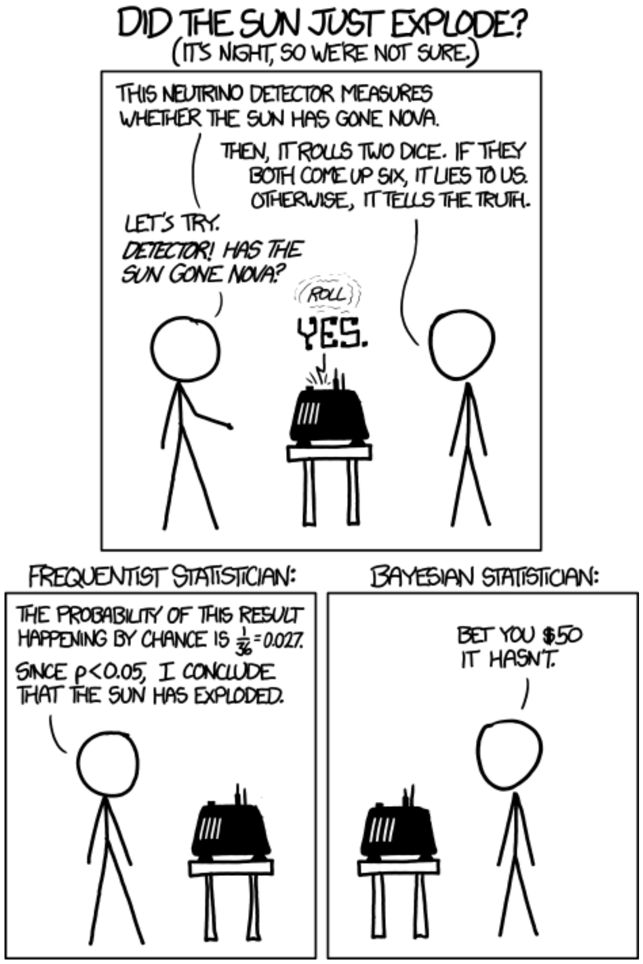
Because the marginal probability $P(X)$ is the probability of X, determined by the integral of the joint probability $P(X, \theta)$ over all possible states of θ, we can rewrite this as:

$$P(\theta|X) = \frac{P(X|\theta) * P(\theta)}{\int P(X|\theta) * P(\theta)}$$

Note that now the denominator serves to normalize the function.

Sun exploding example

Example From Ecological Forecasting, by Mike Dietze, inspired by comic below XKCDhttps://xkcd.com/1132/:


DID THE SUN JUST EXPLODE?
(IT'S NIGHT, SO WE'RE NOT SURE.)
THIS NEUTRINO DETECTOR MEASURES WHETHER THE SUN HAS GONE NOVA.
THEN, IT ROLLS TWO DICE. IF THEY BOTH COME UP SIX, IT LIES TO US. OTHERWISE, IT TELLS THE TRUTH.
LET'S TRY. DETECTOR! HAS THE SUN GONE NOVA?
(ROLL)
YES.
FREQUENTIST STATISTICIAN:
THE PROBABILITY OF THIS RESULT HAPPENING BY CHANCE IS 1/36 = 0.027. SINCE p<0.05, I CONCLUDE THAT THE SUN HAS EXPLODED.
BAYESIAN STATISTICIAN:
BET YOU $50 IT HASN'T.

A neutrino detector tells us whether the sun has exploded or not, but has a 1/36 chance of lying to us about the sun going nova. Our data (X) says Yes, the sun has gone nova. X = Yes, and we have two hypotheses (θ_1 and θ_2)

$$\theta_1 = novasun$$

$$\theta_2 = nonova$$

,

So lets derive 2 liklihoods: $P(X = yes|\theta_1 = nova) = 35/36$ # the probability that a yes from the detector is not a lie

$P(X = yes|\theta_1 = notnova) = 1/36$ # the probability that a yes from the detector is actually a lie

So to estimate the poseterior probabilities of the sun exploding θ_1 = nova, given our the neutrino detector telling us "yes" $X = Yes$:

$$P(\theta_1 = nova|X = YES) = \frac{P(X = YES|\theta_1 = nova) * P(\theta_1 = nova)}{P(X = YES|\theta_1 = nova) * P(\theta_1 = nova) + P(X = YES|\theta_2 = notnova) * P(\theta_2 = notnova)}$$

Lets assume that $P(\theta_1) = 1/10000$, or a low chance of the sun actually going nova, we can calculate the posterior probability of the sun going nova!

You can test out a few different priors (change prior_theta1 in th code below) to see how our prior belief that the sun has actullay exploded affects our posterior estimates.

```
likelihood.theta1 = 35/36 # probability of detector giving a yes, given the sun has actually exploded
likelihood.theta2 = 1/36 # probability of a detector giving a yes, when the sun has not exploded

prior_theta1 = 1/1000 # prior belief that the sun has gone nova
prior_theta2 = 1- prior_theta1

# plug in values for bayes theorem:
posterior = (likelihood.theta1*prior_theta1)/
  ((likelihood.theta1*prior_theta1)+(likelihood.theta2*prior_theta2))

# convert to a percentange & round
print (paste("Bayesian: There is a ", posterior*100, "% chance the sun just went nova!"))
```

```
## [1] "Bayesian: There is a  3.38491295938105 % chance the sun just went nova!"
```

Note that if we have a stronger prior belief that the sun exploded, then the prior has a stronger influence on the posterior.

In many (perhaps most) cases, practitioners will keep priors "non-informative," and rely mostly on the data. However, there might be some good reasons to use "informative" or strong priors in some cases. For example, if you have previous research or a posterior from a previous analsis that strongly suggests a high probability of the sun going nova. Or a meta analysis that gives some prior information about what the values of certain model parameters should be. The key here is to not double dip–you can't use your data from the analysis to inform the prior.

Using Bayes theorem to estimate model parameters:

Lets say we have a normal Gaussian liklihood with a mean = μ SD = σ, and that we observed a value of X = 15. We could use bayes theorem to determine the posetior probability of our parameter of interest μ, given our data X = 15:

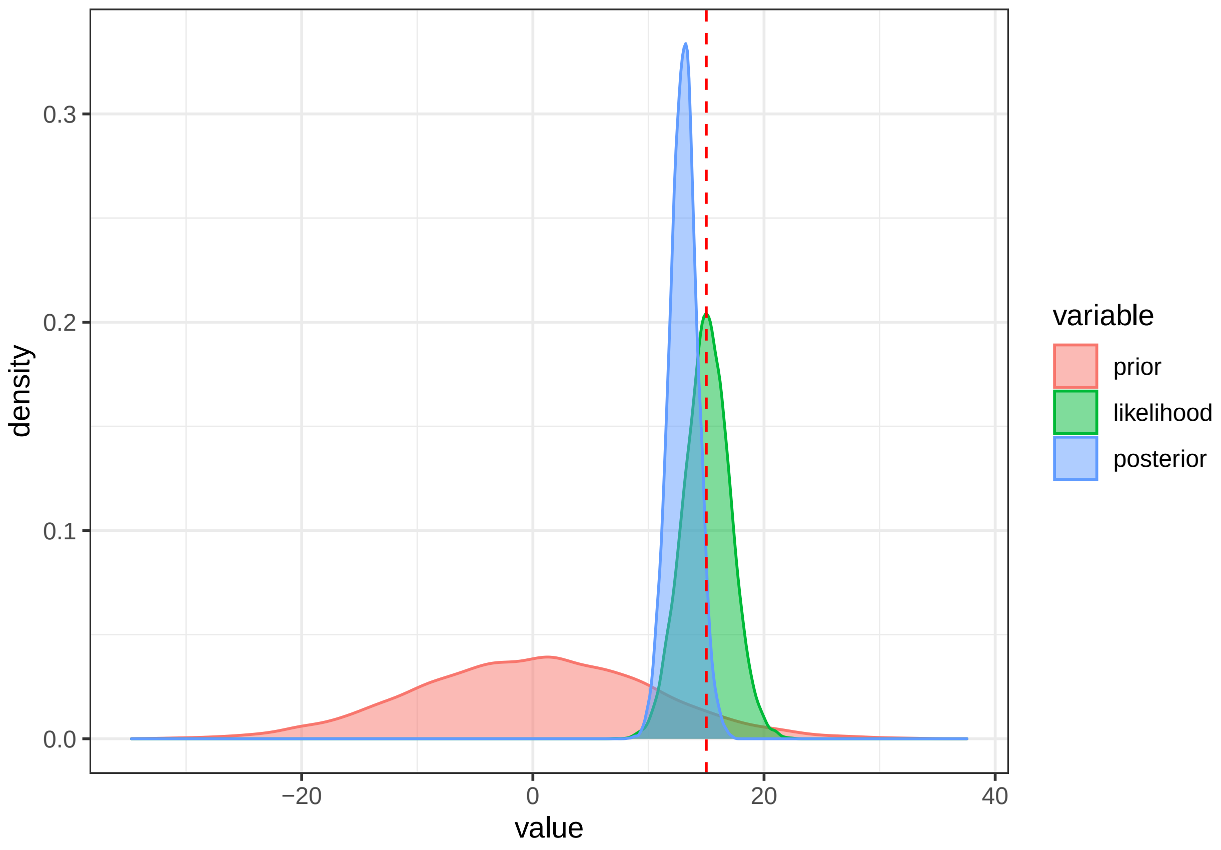
The posterior probability of μ given the data X = 15 is porportional to the likelihood (probability of the data, given the model) multipled by the prior probability of mu.We drop the normalizing constant, so the formula is not proporational to.

Assume that our prior probability of μ is represented by a normal distribution with a mean of μ_0 and variance τ. Subistituting these distributions gives us the posterior. If you were to analytically solve this (substituting the analytical form of the normal distribution), you would see that that this results in a normal distribution for our posterior, with a posterior mean = a weighted average of the data (X), and the prior mean (μ_0), and a posterior precision ($1/\sigma^2$) is the sum of the prior precision and the data precision.

$$P(\mu|X) \propto P(X|\mu) * P(\mu) = Normal(X|\mu, \sigma) * Normal(\mu|mu_0, \tau)$$

For those who are visual learners:

```
## No id variables; using all as measure variables
```



TLDR: The posterior mean is a weighted average of the prior and data, while posterior precision = sum of data precision and prior precision.

Question for thought: -What does this mean for the precision or uncertainty in the posterior?

Monte Carlo Methods:

In the two examples with bayes theorem above, we used examples where you could theoretically express the posterior using a commonly used distribution (normal). However, in more complex models, this is not always (not often) the case. However, we can rely on Monte Carlo Methods to approximate…

Numerical methods that allow for approximation of any probability distriubtion using random samples from that distribution, even if you can't write out the analytical form of that distribution. We won't get too much into the weeds about this here, as I want to focus on implemetation in R.

Just know that here we will use Markov Chain Monte Carlo methods to randomly sample our posterior parameter distributions. JAGS is one program that uses MCMC methods, but there are several.

Multiple programs to run Bayesian models in R

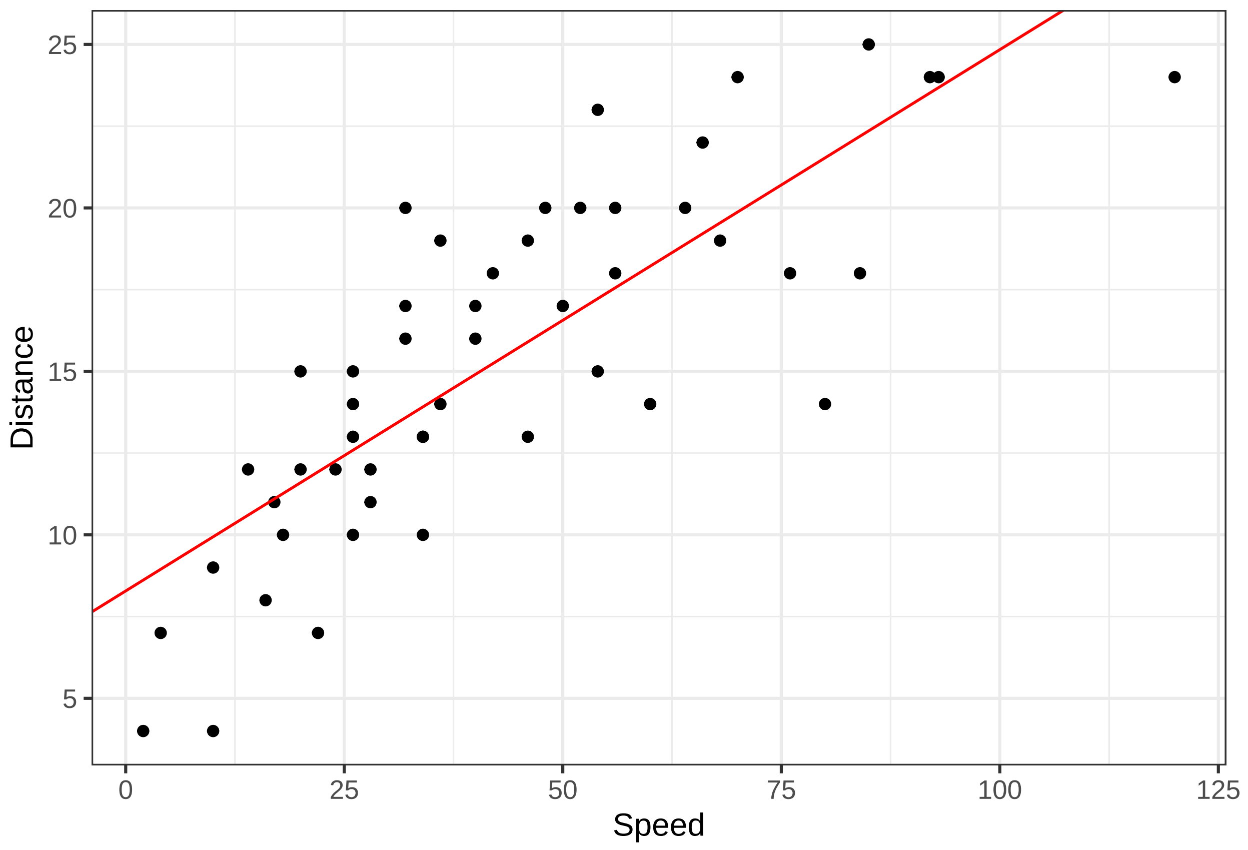
a. JAGS
b. BUGS/WINBUGS
c. STAN
d. NIMBLE

A simple Linear regression example

We will start off thinking about a simple linear regression approach, where we are estimating the slope and intercept of the relationship between two variables in a sample relationship. You might use `lm()` to fit this model using maximum likelihood, frequentist approach. We will use the cars dataset loaded into R.

Frequetist approaches often based on maximum likelhood methods, that is maximizing the probability of the data given the specified model parameters.

```
# do a basic linear regression using "lm"

speed.dist.lm <- lm(speed ~ dist, data = cars)
lm.results <- summary(speed.dist.lm)

summary(speed.dist.lm)
##
## Call:
## lm(formula = speed ~ dist, data = cars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.5293 -2.1550  0.3615  2.4377  6.4179
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  8.28391    0.87438   9.474 1.44e-12 ***
## dist         0.16557    0.01749   9.464 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.156 on 48 degrees of freedom
## Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12

# I am a big GGPLOT fan, so sorry base R plotters...
ggplot(cars, aes(dist, speed))+
  geom_point()+theme_bw(base_size = 12) + ylab("Distance")+xlab("Speed")+
  geom_abline(intercept = lm.results$coefficients[1,1],
              slope = lm.results$coefficients[2,1], color = "red")
```



A simple linear regression in JAGS

I mentioned that in Bayesian models, all variables are treated as random, that means that all variables are/can be estimated within the model. In our linear regression, this includes our dependant variable, y, and could include our independant variable x.

Why might this be useful? - Well y is a measurement taken by me, an imperfect human being. - We typically can't directly measure the "TRUE" process or value that we are interested in.

The Linear Regression Model

Data Model

$$y_i \ \sim normal(\mu, \sigma^2)$$

Process Model

We specify the linear regression with slope β, intercept α and our independent variable x

$$mu = \beta * x + \alpha$$

While we are specifying a very simple model, the process model could be any model that you can write out (logistic growth, muliple linear regression, state-space model etc)

Priors:

In Bayes models you specify distributions that you want to sample the parameters from. We wont go into detail on how to select priors in this short workshop, but keep in mind that prior selection is quite important. The benefit of Bayesian modelling is that you can choose "uninformative priors" or "informative priors". You typically want to specify 'conjugate priors,' which means that when the prior and the liklihood are combined, they generate a posterior distribution that has a named & defined distribution, such as a normal distribution, Poisson, etc. Typical examples of conjugacy include a normal distribution and either normal or inverse gamma priors. We won't be too concerned about this for this tutorial, but keep this in mind when you are building your own models.

Uninformative priors:

Specify a very broad distribution over which to sample for our parameter, and basically state that we have no prior knowledge of what value our parameter should be, besides the distribution shape.

Informative priors:

Specify that we have *some* prior knowledge of what value the parameter should be.

Why might this be useful?

Well, alot of science does not exist in a vacuum, and we likely already have some prior knowledge of what we expect. For example, this might be useful if we have already done a study where we quantified the relationship between y and x, and we know that the value of beta should fall within a given range. In the XKCD comic sun example we specified an informative prior based on the fact that the sun exploding is fairly unlikely.

This may not be appear to be super useful in a basic linear regression, it can also be useful in much more complex models, where multiple parameters could trade off.

In this model we will specify uninformative priors for our prarameters, which assume that the true value of beta is sampled from a normal distribution with mean of 0 and variance of 1000. We also specify a prior for σ^2

$$\beta \sim normal(0, 0.001)$$
$$\alpha \sim normal(0, 0.001)$$
$$\sigma^2 \sim invgamma(0.01, 0.01)$$

Note that variances must be non-negative, hence we cannot use a normal prior as a distribution

JAGS (Just Another Gibbs Sampler)

We will use JAGS ('Just Another Gibbs Sampler') to run this particular bayesian model. JAGS is designed to run Bayesian models using Markoc Chain Monte Carlo simulations (MCMC). JAGS is just one option among many for bayesian analysis. We can call and run jags from R utilizing the 'rjags' library. Note that there are other R libraries that can help with this including 'R2jags'.

Defining our JAGS model

To specify a JAGS model, you must define all the random distributions

```
asimple_linear_regression <- "model{

  # Likelihood
  for(i in 1:n){

    # data model
    y[i]   ~ dnorm(mu[i], tau)

    # process model for the linear regression
    mu[i] <- alpha + beta*xvals[i]
  }

 # Priors
  beta ~ dnorm(0,0.0001)
  alpha ~ dnorm(0,0.0001)


  tau   ~ dgamma(0.01, 0.01) # Prior for the inverse variance, the precision tau
  sigma     <- 1/sqrt(tau)

  # note that in JAGS the second argument of the normal
  # distribution is 1/sigma^2, or the precision Tau

}"
```

Formatting the data + inital values for each chain

Jags expects all data to in a list.

```
car.data <- list(xvals = cars$dist, y = cars$speed, n = length(cars$speed))
```

Initialize the model

Initializing the model using jags.model. Note that you can (and probably should generate intial values, but here we let jags do the default). This step sets up the JAGS model, loads the data, and specifies all paramets and priors.

```
jags.model   <- jags.model (file = textConnection(asimple_linear_regression),
                             data = car.data,
                             #inits = inits,
                             n.chains = 3)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 50
##    Unobserved stochastic nodes: 3
##    Total graph size: 180
##
## Initializing model
```

Running MCMC chains

JAGS can run mulitple MCMC chains, ideally with random starting values. This is a good idea beacuse it ensures that our model "converges" on the same parameter space consistently. Many times people run at least 3 chains.

The funciton "coda.samples" runs the MCMC chains of our model for a specified number of MCMC iteraations (specified by 'n.iter'), and is set up to 'trace' or output all parameters/variables of interest specified by variable.names. Often models are run for a high number of iterations, so 5000 MCMC samples is quite small, but we will use it for this tutorial.

```
jags.reg.out   <- coda.samples (model = jags.model,
                             variable.names = c("alpha","beta", "tau"),
                                 n.iter = 5000)

summary(jags.reg.out)
```

```
##
## Iterations = 1:5000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##         Mean      SD  Naive SE Time-series SE
## alpha 8.3106 0.89395 0.0072991      0.0189209
## beta  0.1650 0.01796 0.0001466      0.0003789
## tau   0.1006 0.02049 0.0001673      0.0001748
##
## 2. Quantiles for each variable:
##
##          2.5%     25%    50%    75%   97.5%
## alpha 6.57559 7.70950 8.3203 8.9104 10.0534
## beta  0.13006 0.15312 0.1648 0.1770  0.1995
## tau   0.06417 0.08602 0.0992 0.1131  0.1447
```

Assessing Model output

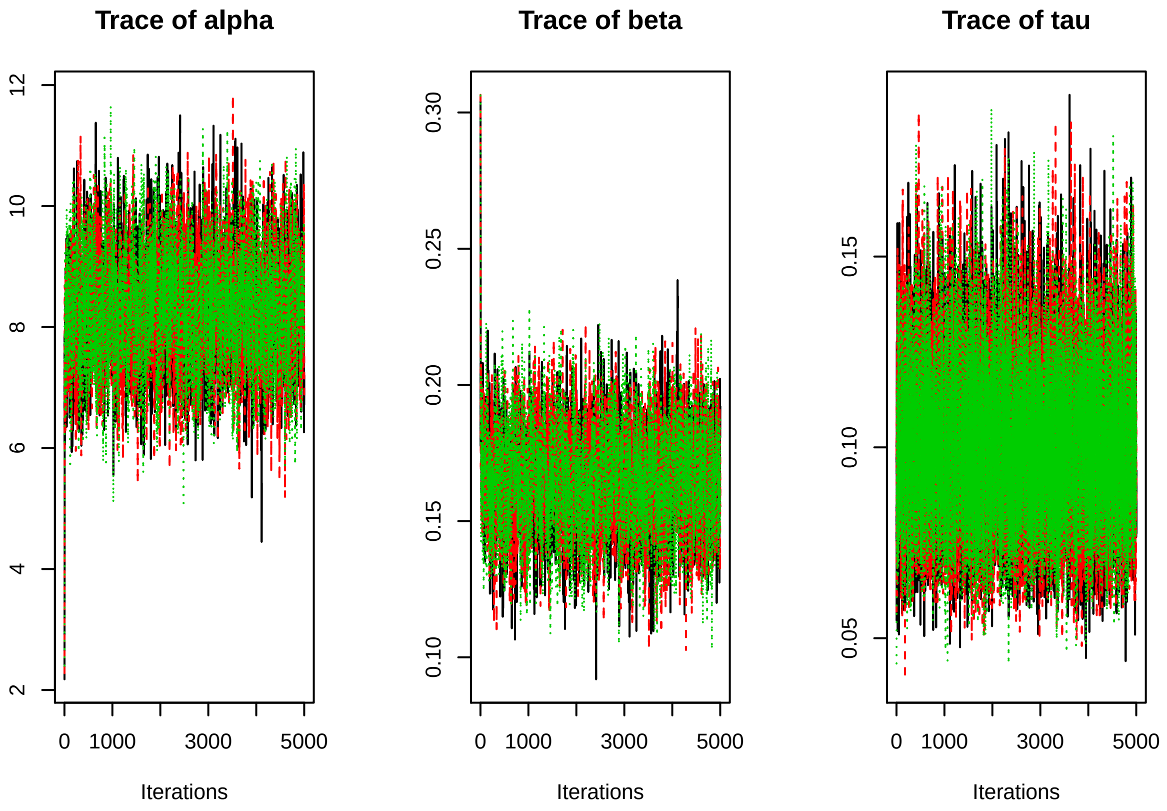
coda.samples gives us a `mcmc.list`

```
class(jags.reg.out)
## [1] "mcmc.list"
```

Have our chains converged?

We need to make sure that our random samples are converging to the same parameter space. Tools to do this include: -traceplots -GBR statistics (ideally < 1.01) -remove "burn-in"

```
par(mfrow = c(1,3))
traceplot(jags.reg.out)
```



```
gelman.diag(jags.reg.out)
```

```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## alpha          1       1.00
## beta           1       1.01
## tau            1       1.00
##
## Multivariate psrf
##
## 1
```

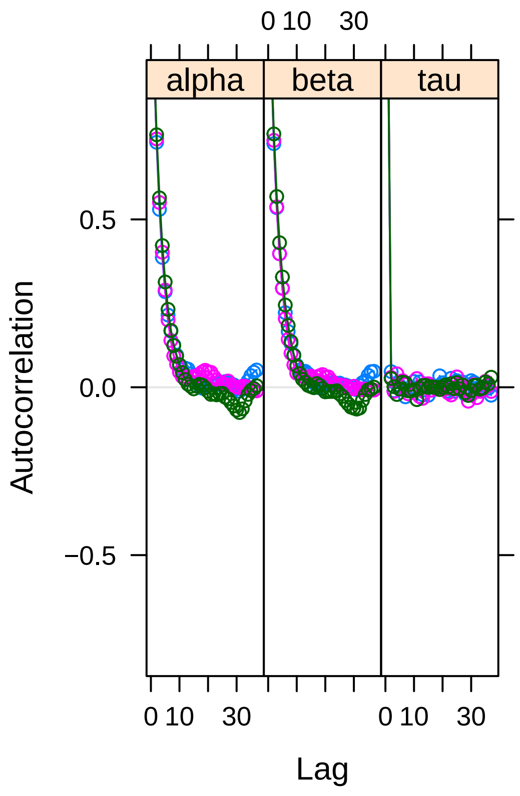
apply a burn in period:

Note that in the above traceplots, the first few samples have not quite converged, so we often run the model for a "burn in" period to remove these mcmc samples and only work with the samples that have converged. You can do it when we specify the model, or you can just remove these after the fact. We are removing them after the fact in the code below

Check for Autocorrelation

Many people thin their chains by taking only the "ith" MCMC iteration

```
acfplot(jags.reg.out)
```

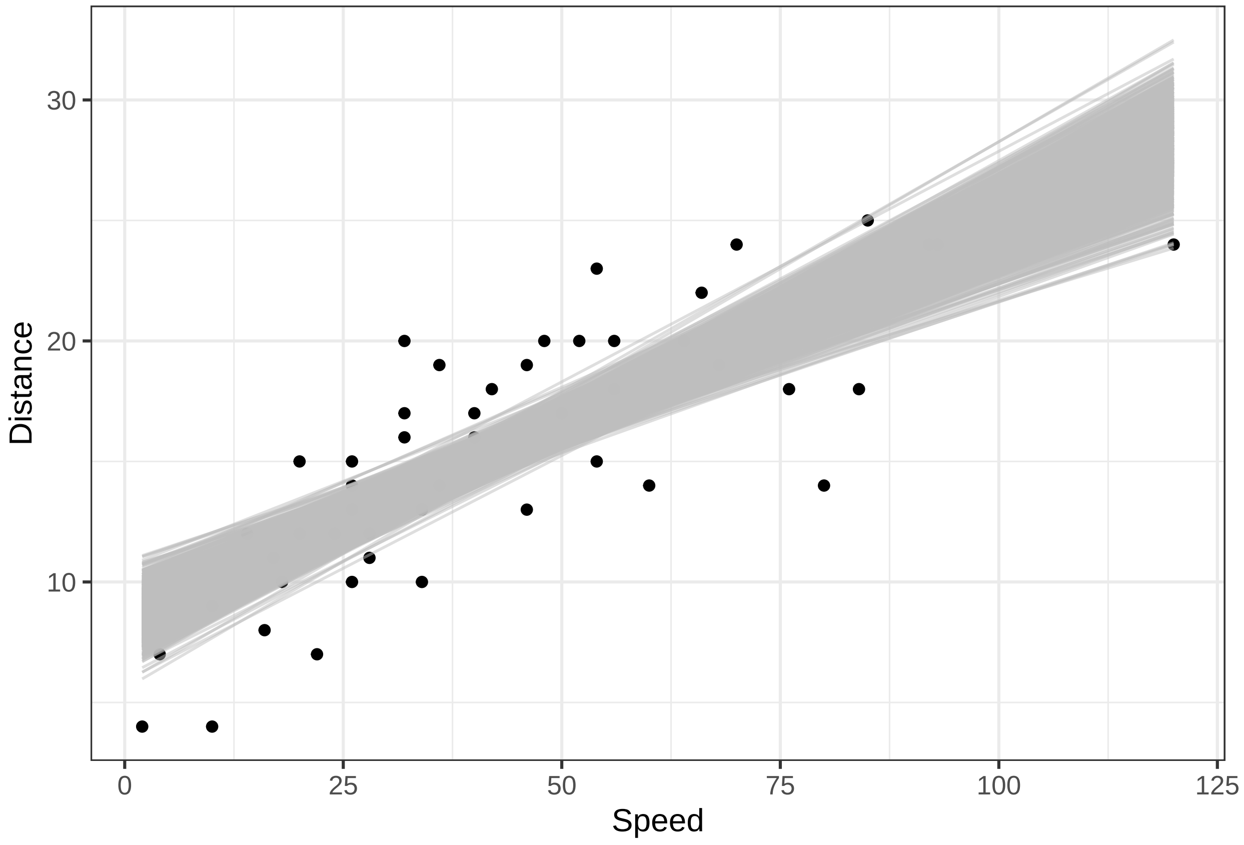


### Lets plot up our parameter estimates + uncertainty

plot the credible intervals around the regression line, which are analogous to the uncertainty/error provided in a frequentist regression

```
#head(jags.reg.out)

jags.mat <- as.matrix(jags.reg.out)

xpred <- min(cars$dist): max(cars$dist)
#plot(cars$dist, cars$speed)
cred.lines <- list()

# loop through to get the lines
for(i in 1000:2000){
 cred.lines[[i]]<-  data.frame(MCMC.step = i,
   xpreds = xpred,
                              ypreds = jags.mat[i,"alpha"] + jags.mat[i,"beta"]*xpred)
```

```r
}

cred.intervals <- do.call(rbind, cred.lines)


ggplot(cars, aes(dist, speed))+geom_point()+theme_bw(base_size = 12) +
  ylab("Distance")+xlab("Speed")+
  geom_line(data = cred.intervals, aes(x =xpreds, y = ypreds, group = MCMC.step),
            color = "grey", alpha = 0.5)
```

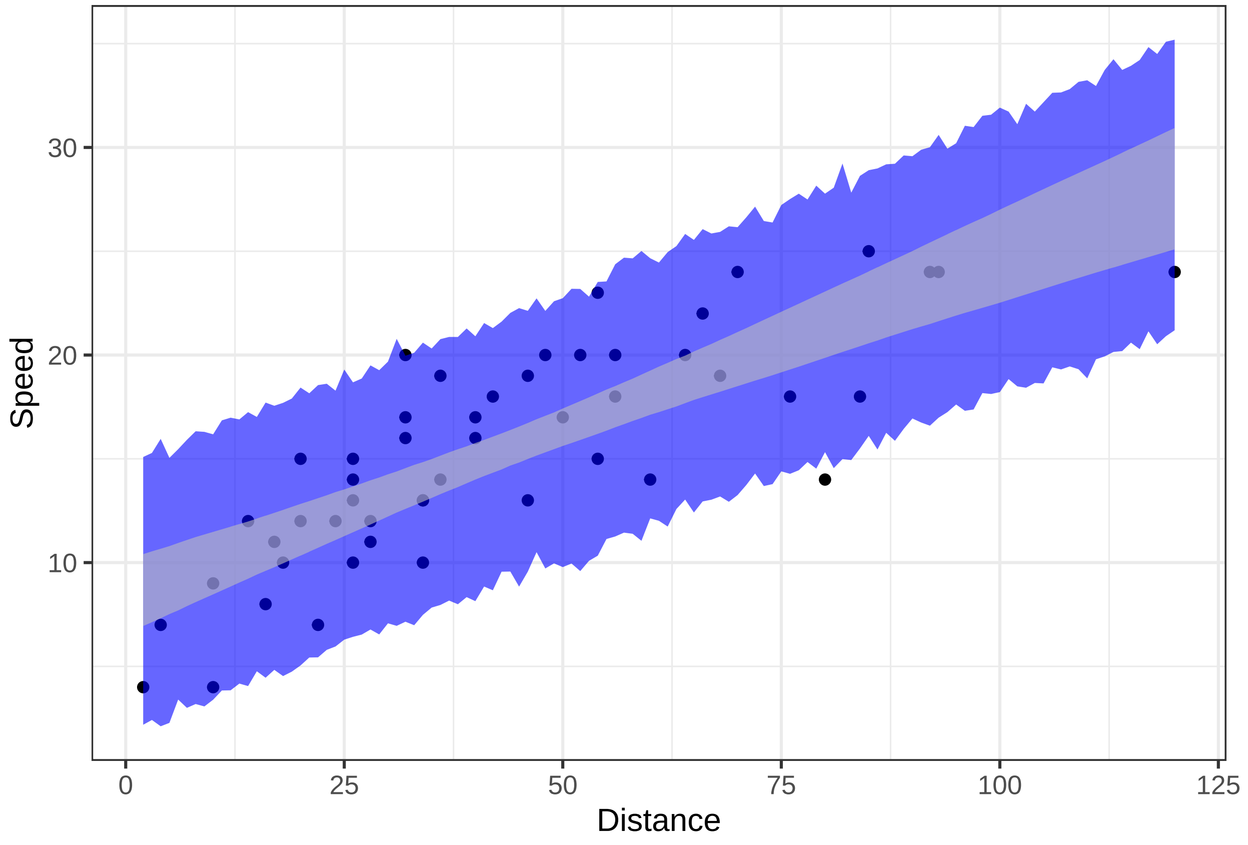


Lets also simulate the credible intervals and predictive intervals from random pairs of samples of the mcmc output

```r
npred <-length(min(cars$dist): max(cars$dist))
nsamp <- 1000
mcmcsamples <- sample(nrow(jags.mat), nsamp)
xpred <- min(cars$dist): max(cars$dist)
ypred <- matrix(NA ,nrow=nsamp, ncol=npred)
ycred <- matrix(NA, nrow=nsamp, ncol=npred)
```

Next we'll set up a loop where we'll calculate the expected value of y at each x for each pair of regression parameters and then add additional random error from the data model.

```
for(i in 1:1000){
  params <-  jags.mat[mcmcsamples[i],]
  ycred[i,] <- params["alpha"] + params["beta"]*xpred # same as above
  ypred[i,] <- rnorm(n = npred, mean = ycred[i,], sd = 1/sqrt(params["tau"])) #draw from normal distrib
}


#combine the  ci, pi, and xpreds into a dataframe ot plot in ggplot2

ci.and.pi <- data.frame(xvals = xpred,
                        ci.med = apply(ycred, 2, quantile, 0.5),
                        ci.lo= apply(ycred, 2, quantile, 0.025),
                        ci.hi= apply(ycred, 2, quantile, 0.975),
                        pi.med= apply(ypred, 2, quantile, 0.5),
                        pi.lo= apply(ypred, 2, quantile, 0.025),
                        pi.hi=apply(ypred, 2, quantile, 0.975))

# now make a pretty plot with all the 95% credible intervals and 95% posterior predictive intervals on

ggplot()+geom_point(data = cars, aes(dist, speed))+theme_bw(base_size = 12) +

  ylab("Speed")+xlab("Distance")+
  geom_ribbon(data = ci.and.pi, aes(x = xvals, ymin = pi.lo, ymax = pi.hi), fill = "blue", alpha = 0.6)
  geom_ribbon(data = ci.and.pi, aes(x = xvals, ymin = ci.lo, ymax = ci.hi), fill = "grey", alpha = 0.6)
```



Breakout group Discussion

In your groups, discuss:

1. What do the credible intervals represent?
2. What aspects of uncertainty do you think the credible intervals include?
3. what aspects of uncertainty do the posterior predictive intervals include?

Uncertainty:

1. parameter uncertainty (credible intervals, the 95% CI around β and α)
2. process uncertainty (sigma)
3. driver/data uncertainty (What if we had multiple measurements of our dependant variable?)

To simulate some observations from the cars dataset

Jags expects all data to in a list.

```
speeds1 <- speeds2 <- speeds3 <- matrix(NA, nrow = length(cars$speed), ncol = 1)
for(i in 1:length(cars$speed)){
speeds1 [i]<- rnorm(1, cars[i,]$speed, sd = 1)
speeds2 [i]<- rnorm(1, cars[i,]$speed, sd = 1)
speeds3 [i]<- rnorm(1, cars[i,]$speed, sd = 1)
}
cars.mult.measure <- data.frame(dist = rep(cars$dist, 3),
                                speed = c(speeds1, speeds2, speeds3 ),
                                observer = rep(1:3, each = length(cars$speed)))



car.obs.data <- list(xvals = cars.mult.measure$dist,
                     y = cars.mult.measure$speed,
                     obs = cars.mult.measure$observer, n = length(cars.mult.measure$speed))
```

Including Data Uncertainty or "Uncertainty in variables"

Lets say that our variable "Speed" was measured by taking 3 different radar measurements, each slighly different. How do we know what the "true" speed is?

Open question for the group: How could you typically deal with this?

What if we could include this uncertainty in our model, in a way that propogates forward to the estimates of distance?

We can (in a few different ways)!

One way to do this is to add a data model for speed: Lets say we took multiple measurements of speed for several distances, and determined that our speed estimates have a SD = 2.5.

We can use this measurement to estimate the "true speed" in our model:

$$truespeed_i \sim Normal(mean = speed_i, precison = tau_{speed})$$

Since we already have some prior knowledge of what the precision on our speed estimates should be, we can include an informative prior:

$$tau_{speed} \sim gamma(10, 0.5)$$

```
a_linear_regression_driver_unc <- "model{

  # Likelihood
  for(i in 1:n){

    # data model for speed measurements
   true_speed[i] ~ dnorm(xvals[i], tau_speed)

    # data model for distance (y variable)
    y[i]   ~ dnorm(mu[i],inv.var)

    # process model for the linear regression
    mu[i] <- alpha + beta*true_speed[i]
  }

 # Priors
  beta ~ dnorm(0,0.0001)
  alpha ~ dnorm(0,0.0001)


  inv.var   ~ dgamma(0.01, 0.01) # Prior for the inverse variance
  sigma     <- 1/sqrt(inv.var) # note that in JAGS the second argument of the normal distribution is 1/

  # prior for speed precsion
  tau_speed   ~ dgamma(10, 0.5)

}"
```

```
car.data <- list(xvals = cars$dist, y = cars$speed, n = length(cars$speed))
```

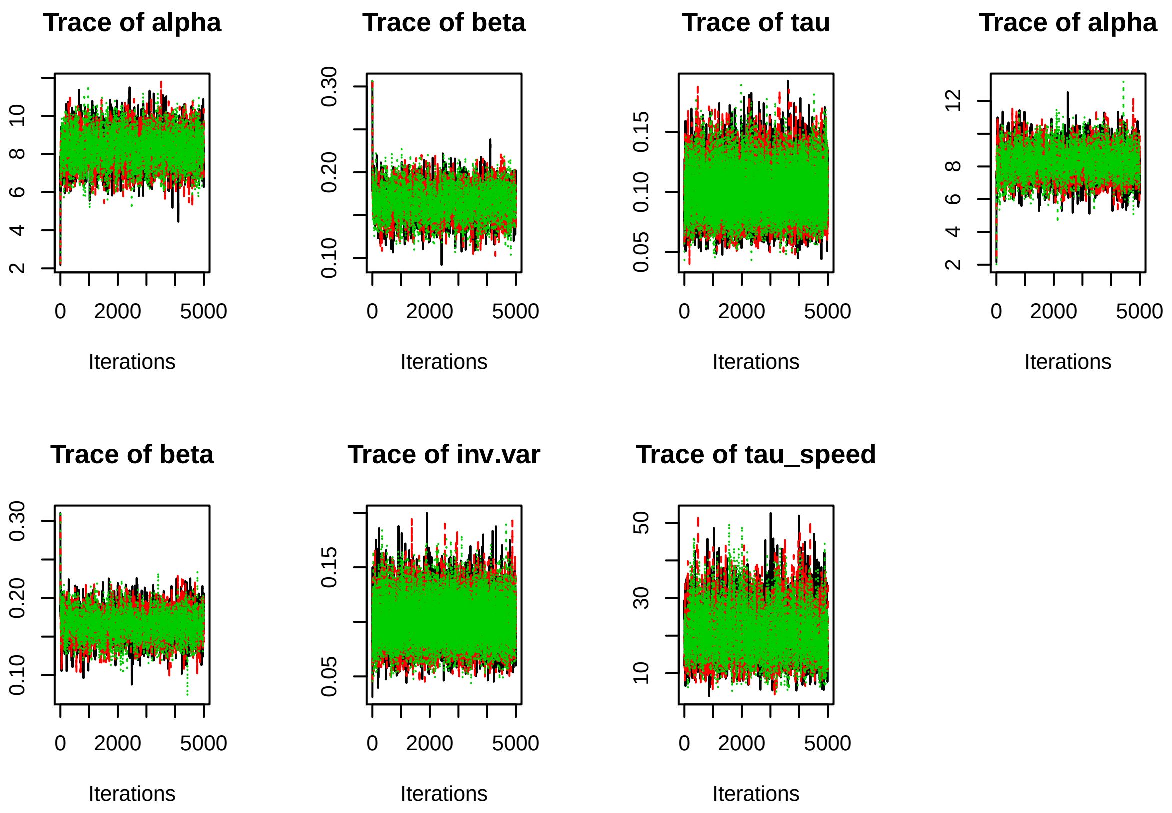
```
par(mfrow = c(2,4))
traceplot(jags.reg.out)
jags.model.drive   <- jags.model (file = textConnection(a_linear_regression_driver_unc ),
                             data = car.data,
                             #inits = inits,
                             n.chains = 3)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 50
##    Unobserved stochastic nodes: 54
##    Total graph size: 263
##
## Initializing model
```

```
jags.drive.unc.out   <- coda.samples (model = jags.model.drive,
                            variable.names = c("alpha","beta", "inv.var", "tau_speed" ),
                                n.iter = 5000)

summary(jags.drive.unc.out)
```

```
##
## Iterations = 1:5000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##              Mean      SD  Naive SE Time-series SE
## alpha      8.2762 0.92726 0.0075710      0.0201096
## beta       0.1658 0.01851 0.0001511      0.0004040
## inv.var    0.1005 0.02054 0.0001677      0.0001833
## tau_speed 19.9419 6.42730 0.0524787      0.1281265
##
## 2. Quantiles for each variable:
##
##              2.5%      25%      50%     75%   97.5%
## alpha     6.46701  7.68204  8.26951  8.8631 10.1314
## beta      0.12895  0.15396  0.16612  0.1778  0.2016
## inv.var   0.06399  0.08604  0.09927  0.1137  0.1437
## tau_speed 9.30407 15.37688 19.26375 23.8026 34.3266
```

```
traceplot(jags.drive.unc.out)
```



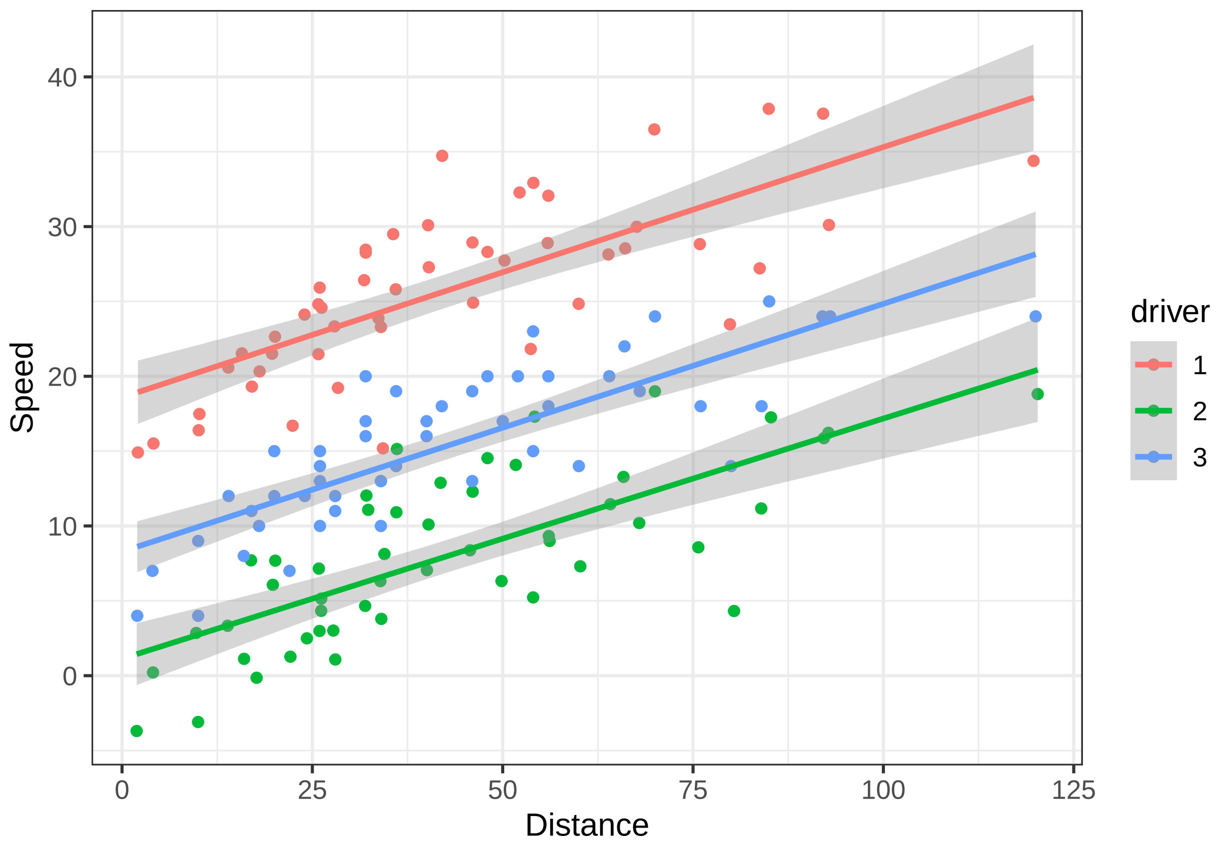
Adding an informative prior is one way to include the data or driver uncertainty in the model. Another might be to explicitly include mulitple measurements/observers in your model.

Adding Random effects to our JAGS model:

Now, one can imagine that different drivers might driver different distances at a given speed, leading to different intercepts (and possibly different slopes) for each driver. We will add simulate different intercepts for multiple drivers, to demonstrate how you can add random effects within the JAGS bayesian framework.

```
# driver 1
driver1 <- data.frame(speed = cars$speed + rnorm(50, 10, 2),
                      dist = cars$dist + rnorm(50, 0, 0.2),
                      driver = 1)
# driver 2
driver2 <- data.frame(speed = cars$speed + rnorm(50, -7, 2),
                      dist = cars$dist + rnorm(50, 0, 0.2),
                      driver = 2)
# driver 3

driver3 <- data.frame(speed = cars$speed ,
                      dist = cars$dist ,
                      driver = 3)

cars.driver <- rbind(driver1, driver2, driver3)
cars.driver$driver <- as.character(cars.driver$driver)


# ggplot()+geom_point(data = cars.driver, aes(dist, speed, color = driver))+theme_bw(base_size = 12) +

ggplot(data = cars.driver, aes(dist, speed, color = driver))+geom_point()+stat_smooth(method = "lm")+the
```

```
## `geom_smooth()` using formula 'y ~ x'
```



How to specify a randome intercept in JAGS:

to do this, we will modify the alpha term

```
a_linear_regression_random_intercept <- "model{

  # Likelihood
  for(i in 1:n){
   # data model for speed measurements
   true_speed[i] ~ dnorm(xvals[i], tau_speed)
    y[i]   ~ dnorm(mu[i],inv.var) # data model
    mu[i] <- alpha[driver[i]] + beta*true_speed[i]  # process model for the linear regression
  }

for (i in 1:ndriver) {
    alpha[i] ~ dnorm(mu_alpha, tau_alpha) # Specify random intercepts for each of the drivers, drawn fr
}


 # Priors
  beta ~ dnorm(0,0.0001)
  inv.var   ~ dgamma(0.01, 0.01) # Prior for the inverse variance
  sigma     <- 1/sqrt(inv.var) # note that in JAGS the second argument of the normal distribution is 1/
```

```
# hyperparameters/priors:

 mu_alpha ~ dnorm(0, 0.0001) # hyperparameter mean random alpha



 #sigma_alpha~dunif(0, 100) # SD hyperparameter for random intercepts
 tau_alpha ~ dgamma(0.01, 0.01) # convert sigma_alpha to a precision (what jags expects for a normal di
# prior for speed precsion
  tau_speed   ~ dgamma(10, 0.5)

}"
```

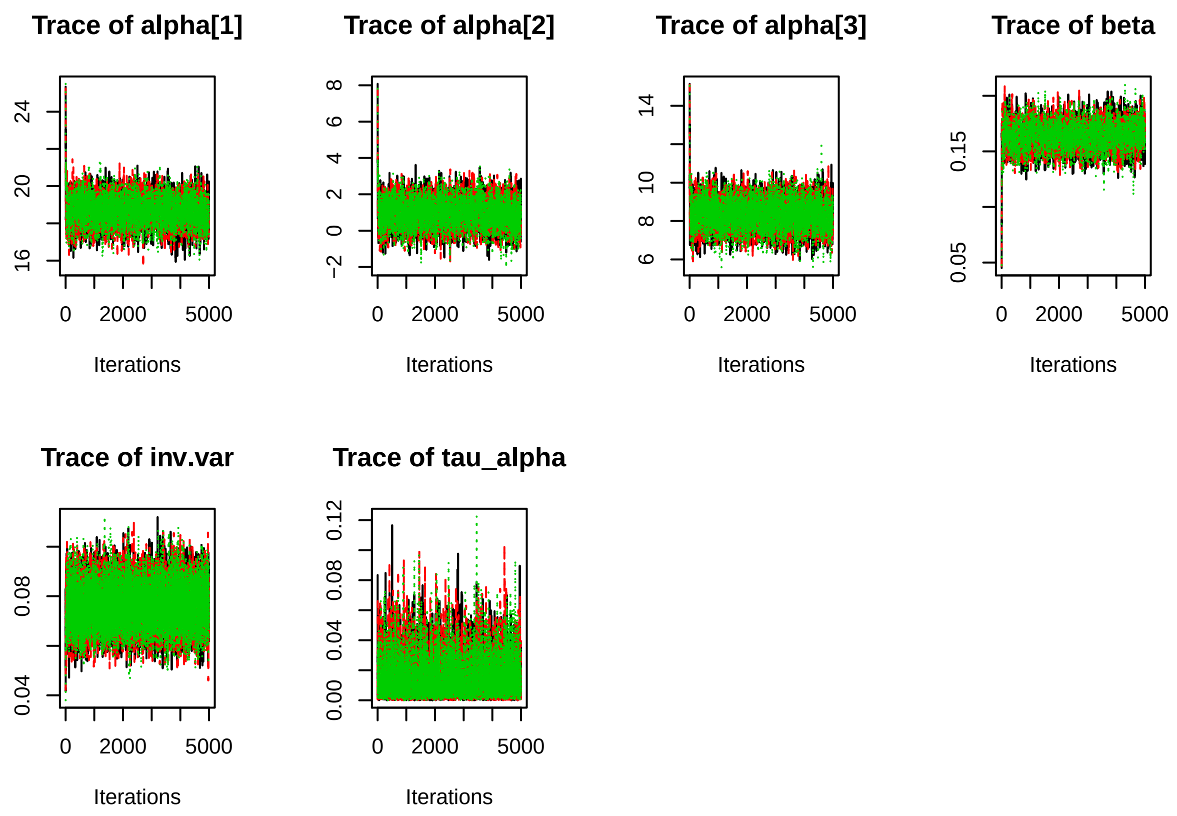
lets run this model:

```
# we need to specify the data for jags, and include the number of drivers (ndriver), as well as the dri
cardriver.data <- list(xvals = cars.driver$dist,
                       y = cars.driver$speed,
                       driver = cars.driver$driver,
                       n = length(cars.driver$speed),
                       ndriver = length(unique(cars.driver$driver)))
```

```
jags.int.model   <- jags.model (file = textConnection(a_linear_regression_random_intercept),
                             data = cardriver.data,
                             #inits = inits,
                             n.chains = 3)
```

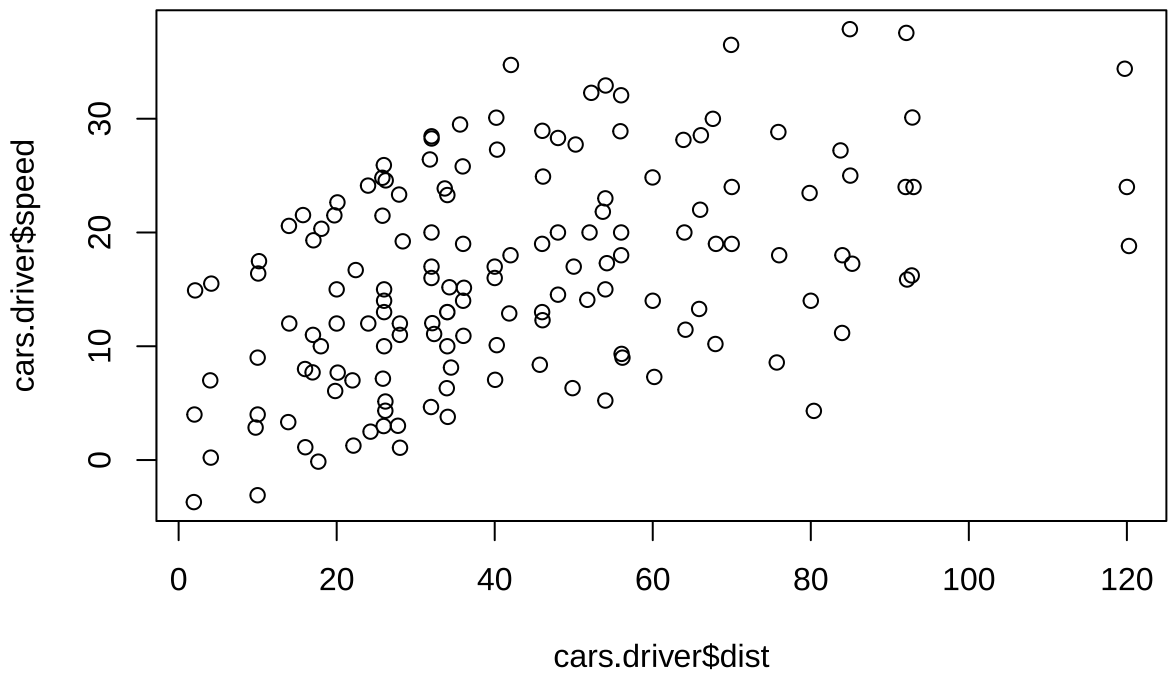
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 150
##    Unobserved stochastic nodes: 158
##    Total graph size: 918
##
## Initializing model
```

```
par(mfrow = c(2,4))


jags.int.out   <- coda.samples (model = jags.int.model,
                            variable.names = c("alpha","beta", "inv.var", "tau_alpha"),
                                n.iter = 5000)

#summary(jags.int.out)
traceplot(jags.int.out)
```

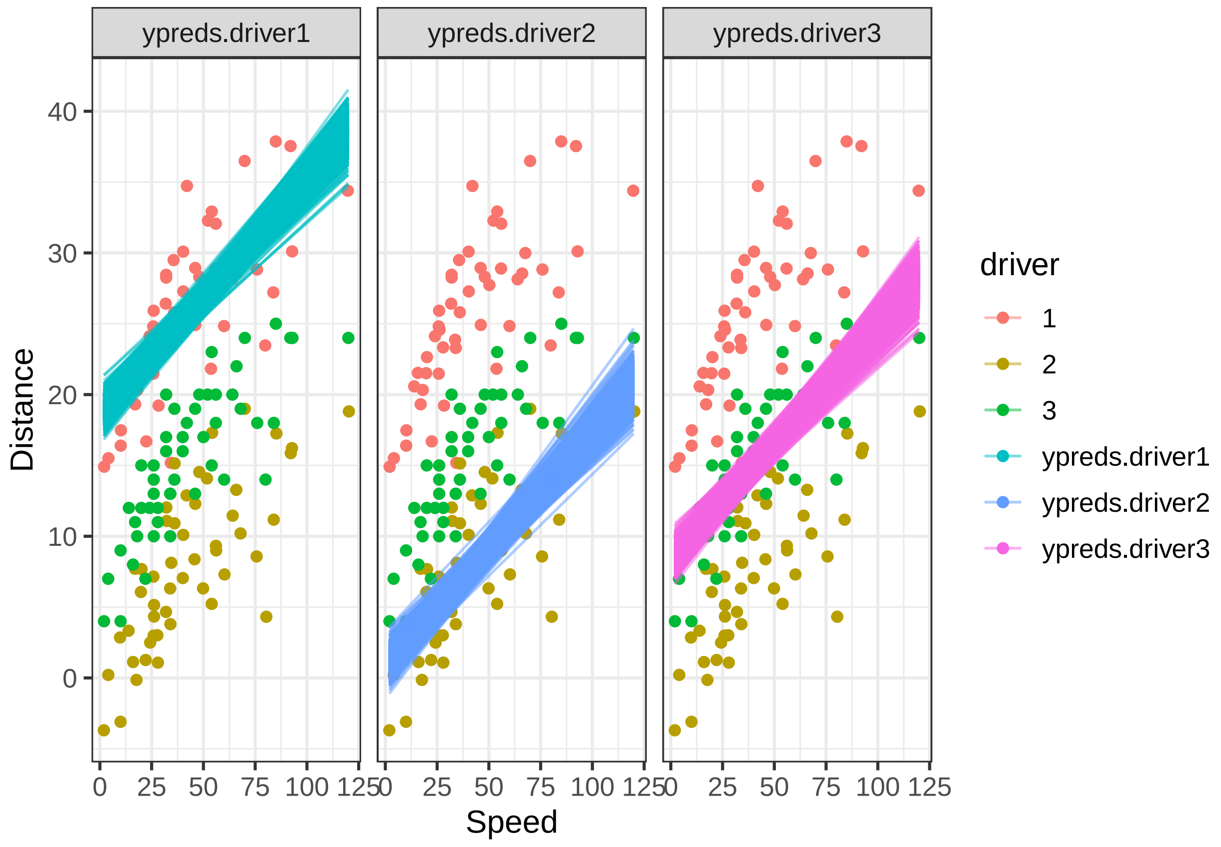


plot the credible intervals for each

```
#head(jags.int.out)

jags.int.mat <- as.matrix(jags.int.out)
 jags.int.mat <- jags.int.mat[1000:15000,]#select burn in of 1000
xpred <- min(cars.driver$dist): max(cars.driver$dist)
plot(cars.driver$dist, cars.driver$speed)
```



```
cred.lines <- list()

# loop through to get the lines
for(i in 1000:2000){
 cred.lines[[i]]<-  data.frame(MCMC.step = i,
                               xpreds = xpred,
                               ypreds.driver1 = jags.int.mat[i,"alpha[1]"] + jags.int.mat[i,"beta"]*xpred
 ypreds.driver2 = jags.int.mat[i,"alpha[2]"] + jags.int.mat[i,"beta"]*xpred,
 ypreds.driver3 = jags.int.mat[i,"alpha[3]"] + jags.int.mat[i,"beta"]*xpred)
}

cred.intervals <- do.call(rbind, cred.lines)
cred.intervals.m <- reshape2::melt(cred.intervals, id.vars = c("MCMC.step", "xpreds"))

ggplot(cars.driver, aes(dist, speed, color = driver))+geom_point()+theme_bw(base_size = 12) + ylab("Dis
```



### Uncertainty in the model:

More complex models:

The real benefit to going bayes comes (in my opinion) as you start adding complexity to your model structure, particularly if your dataset of interest has heiarchical structure. This allows you parse some of that additive process error into parameter error, random effects, and fixed effects.

We could keep heiarchically adding complexity, if that is what our system/problem needed. In our example, perhaps Distance vs Speed relationship varied by driver, but also by the car model. Depending on the data stucture, we might add an additional randome effect for the car. But, if all drivers tested out all the cars we could have a nested randome effect, where μ_{alpha} was drawn from a distribution of with mean μ_{car} and precision τ_{car}. However, this heiarchical structure should be based in a knowledge of your system and your data. That is why it is best practice to start with a simple model, then build up heiarachical structures, if needed.

On your own:

Random slopes + intercepts:

Try adding to our random effects model to include both random slopes and intercepts.

More Resources on Bayesian modelling

There are alot of great resources on the background of bayesian statistics and heiarchical models, including textbooks and examples of code. My background is in ecology, so much of my suggestions might be biased. -

Doing Bayesian Data Analysis: A tutorial with R, JAGS, and STAN, John Kruschke https://sites.google.com/site/doingbayesiandataanalysis/ - Bayesian Models: A Statistical Primer for Ecologists, Hobbs N. T., & Hooten, M. B. (2015). Princeton University Press. - Introduction to Applied Bayesian Statistics and Estimation for Social Scientists, Scott M. Lynch (2007). Springer-Verlag. https://doi.org/10.1007/978-0-387-71265-9 - Ecological Forecasting, Michael Dietze. (2017). https://press.princeton.edu/books/hardcover/9780691160573/ecological-forecasting

More resources on Bayesian modelling in R

There are alot of different ways to implement Bayesian models in R, each have their advantages, disadvantages, and vary in how intuitive they may be to a beginnner. Below is a list of differnt programs with links for more information

- JAGS: http://mcmc-jags.sourceforge.net
- BUGS/WINBUGS: https://www.mrc-bsu.cam.ac.uk/software/bugs/
- STAN: https://mc-stan.org/users/interfaces/rstan
- NIMBLE: https://r-nimble.org