Bayesian regression models: practical exercises

Christopher Jackson

The practical exercises for Session 3 "Bayesian Regression Models" are provided in this PDF file.

- Practical 1: Demonstration of how to run JAGS to fit a simple linear regression model.
- Practical 2: Multiple linear regression - exercises
- Practical 3: Logistic regression - exercises

Full worked solutions are provided at the back of this document. Feel free to consult these as you are going along, if you are stuck on any of the exercises.

The document and embedded code are also provided as an R Markdown document in `regression_practical.Rmd`. This allows the R code blocks to be run conveniently from RStudio.

Warning: If you copy-and-paste code from the practicals into your own documents, then this should be copied from the R Markdown document rather from the PDF, otherwise some special characters will fail to copy properly.

1 Running JAGS to fit a simple linear regression - demonstration

Key learning points

- Defining a linear regression model in JAGS code
- Log-transforming positive data
- Centering continuous covariate data
- Distinction between mean (expectation) and predicted value of an observable

The idea of this practical section is to get used to running JAGS to define, fit and summarise basic statistical models.
Simply run each step on your own computer, and make sure you understand what it is doing.

Feel free to ask for help if any of the steps don't make sense.

1.1 Example data

The data mentioned in the lectures are provided in a CSV file `ELSA.csv`, that you can read into R with the following command. For this session, here we exclude any records with missing values (these will be dealt with in Session 6 of the course, which covers regression modelling with missing data).

The data are simulated, but based on a real longitudinal study of ageing and health. Each row represents a pair of successive 4-yearly observations from an individual in this study. The row shows the current cholesterol value and whether they are currently diagnosed with diabetes, and the cholesterol level and diabetes diagnosis status 4 years later, for each pair of observations.[1]

```
library(tidyverse)
ELSA <- read.csv("ELSA.csv") %>%
  na.omit()
head(ELSA, 3)
```

```
##   male age chol_curr diab_curr chol_4yr diab_4yr
## 1    0  64       5.9         0      5.6        0
## 2    1  76       3.3         1      3.6        1
## 3    1  65       3.9         0      4.3        0
```

The variables we use in these examples are

- `male`: 1 for male, 0 for female
- `age`: current age in years
- `chol_curr`: current total cholesterol level
- `diab_curr`: current diabetes status (1: diagnosed diabetes, 0: no diagnosed diabetes)
- `chol_4yr`: cholesterol level at their next 4-yearly visit
- `diab_4yr`: diabetes status at their next 4-yearly visit

1.2 JAGS model code

The following JAGS code can be used to define a simple linear regression for log cholesterol after 4 years in terms of current log cholesterol. Vague priors are used for the moment. In the exercise later, you will derive a weakly informative prior for a regression coefficient.

```
lm_jagsmod <- "model {
for (i in 1:n) {
  logchol_4yr[i] ~ dnorm(mu[i], prec)  # precision, not SD or variance.
```

[1] The regression models used are simplified versions of real models used in a study of mid-life health checks for reducing cardiovascular risk. (Mytton, Jackson et al. PLoS Medicine 2018)

```
  mu[i] <- alpha + beta*(logchol_curr[i] - mean(logchol_curr[]))
  logchol_pred[i] ~ dnorm(mu[i], prec) # predicted value (expected value plus
                                       # predicted error)
  chol_mu[i] <- exp(mu[i])
  chol_pred[i] <- exp(logchol_pred[i]) # predicted value on non-log scale
 }
 alpha ~ dnorm(0, 1/100^2)
 beta  ~ dnorm(0, 1/100^2)
 prec  ~ dgamma(0.0001, 0.0001)
 sd <- 1/sqrt(prec)
}"
```

Note here:

- JAGS syntax is designed to represent the algebraic definition of a model, rather than a set of instructions to be performed by the computer in sequence (it is "declarative", rather than "procedural").
- So the code should usually resemble the mathematical equations that specify the model. Here the model for the data is $y_i \sim N(\mu_i = \alpha + \beta x_i, \sigma^2), i = 1, \ldots, n$, where y_i is the log of the cholesterol value after 4 years (`logchol_4yr`).
- x_i is the current cholesterol, log transformed and centered around its mean, for person i. The centering is done to make the MCMC sampler more efficient.
- JAGS defines the normal distribution in terms of the precision $1/\sigma^2$, called `prec` in the code, rather than the variance σ^2 or standard deviation σ.
- We also define the expected value $\mu_i = \alpha + \beta x_i$ (the average value for a population of people who all have covariate value x_i) and the *predicted* value (for an observation from an individual with covariate value x_i), which equals the expected value plus a random normally-distributed error term with variance σ^2.
- We also transform these quantities back from the log scale to their natural scale, for presentation.

The observed data for variables in this model are defined in an R list. We log-transform all cholesterol values, and add the constant denoting the number of observations `n` that JAGS needs to know for the `for` loop.

```
n <- nrow(ELSA)
dat <- list(n=n,
            logchol_4yr = log(ELSA$chol_4yr),
            logchol_curr = log(ELSA$chol_curr))
```

The model and data will be supplied to JAGS using the `jags.model` function.

1.3 Initial values

Key learning points

- Why / how to supply initial values in MCMC simulation (1.3)
- Running an MCMC simulation in JAGS and diagnosing convergence (1.4–1.5)

We also supply *initial values* for the main parameters of the model. This is a list of lists, with one list for each chain which defines where that chain will start sampling values from.

The initial values should at least be

- different for different chains
- not so extreme, in the context of the prior distribution and the data, that the sampler will get stuck and fail to find the true posterior.

but, beyond this, the exact values used are not important. In this example, we define

```
chol_in <- list(list(alpha=1, beta=0, prec=1),
                list(alpha=2, beta=1, prec=1))
```

Note we have to initialise the error precision `prec` rather than the standard deviation `sd` (σ), because the prior is placed on `prec` in the JAGS model code.

If initial values are not supplied, different MCMC software has different behaviour. JAGS will initialise all parameters based on central values (mean, mode or median) for the prior distribution. WinBUGS (for example) samples initial values randomly from the prior, which can sometimes cause numerical overflow. Starting at a central value is more robust, but a disadvantage is that multiple chains will all start sampling from the same place. This makes it harder to diagnose convergence by seeing if different chains with different starting points end up in the same place. Therefore in JAGS, it is usually advisable to supply your own initial values.

1.4 Running JAGS

After calling `jags.model` to define the model, data and initial values, an initial "burn-in" of 1000 MCMC iterations is run using `update`.

We then draw a sample of 1000 values from two parallel MCMC chains for three basic parameters (α, β, σ) that define the regression model, and two further quantities of interest that are defined as functions of these:

- `chol_mu`, the expected outcome for each individual, transformed back to the natural scale from the log scale.
- `chol_pred`, a *predicted* outcome for each individual, defined as the expected value plus a random error term, also transformed back to the natural scale.

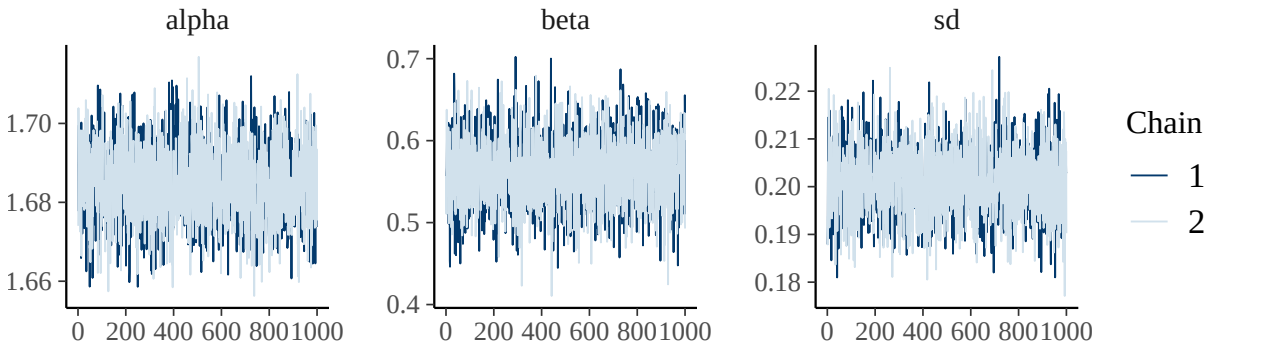
```
library(rjags)
chol_jag <- jags.model(textConnection(lm_jagsmod),
                       data=dat, inits=chol_in, n.chains=2, quiet=TRUE)
pars_basic <- c("alpha","beta", "sd")
update(chol_jag, 1000)
sam <- coda.samples(chol_jag, c(pars_basic,"chol_mu","chol_pred"), n.iter=1000)
```

1.5 Convergence diagnostics

We now perform "convergence diagnostics" to ensure that the sample we have drawn can be treated as a sample from the posterior distribution. We do this for the three basic parameters, α, β and σ. For models with lots of unknown quantities, examining a selection of the main parameters of interest is generally sufficient to diagnose convergence or lack of convergence. In this example, α, β and σ are the quantities that define the regression model, and μ and `chol_pred` are (deterministic or stochastic) functions of them.

First we look at a "trace plot'': a simple plot of the sampled values against the MCMC iteration number. This should look like a sequence of independent random draws, which it does here (Recall Session 2 for interpretation of trace plots).

```
library(bayesplot)
mcmc_trace(sam, pars_basic)
```



Another R package, called `posterior`[2], is then loaded to compute numerical convergence diagnostics for the parameters of interest α, β and σ.

```
library(posterior)
draws <- as_draws(sam)
subs <- subset_draws(draws, variable=pars_basic)
summary(subs)
```

```
## # A tibble: 3 x 10
##   variable  mean median      sd     mad    q5   q95  rhat ess_bulk ess_tail
##   <chr>    <num>  <num>   <num>   <num> <num> <num> <num>    <num>    <num>
## 1 alpha    1.68   1.68  0.00963 0.00989 1.67  1.70   1.00    1940.    1902.
## 2 beta     0.563  0.563 0.0423  0.0428  0.495 0.634  1.00    2017.    2050.
## 3 sd       0.201  0.201 0.00697 0.00687 0.190 0.213  1.00    1927.    2052.
```

The variable `rhat` is the $\hat{R}$ statistic, which should be close to 1 if the chains have converged. This essentially looks at whether multiple chains are all sampling from the same area of parameter space, and measures the ratio of between-chain to within-chain variability. In this case, $\hat{R}$ gives good evidence of convergence.

The variables `ess_bulk` and `ess_tail` are measures of the *effective sample size*, which can differ from the actual MCMC sample size due to *autocorrelation* in the sampled chains. `ess_bulk` measures the amount of information available to measure a central estimate (such as the posterior median), while `ess_tail` measures the amount of information to measure tail quantiles (e.g. for a 95% credible interval). In this simple linear regression example, these estimates roughly agree with the actual MCMC sample size of 2000, since there is negligible autocorrelation.[3]

1.6 Compute posterior summaries of interest

Key learning points

- Extracting and presenting summaries of posterior distributions from JAGS simulations
- Distinction between mean (expectation) and predicted value of an observable

By default, the `posterior` package summarises the posterior distribution by the mean, median, standard deviation, mean absolute deviation and a 90% equal-tailed credible interval.

```
summ <- summary(draws)
head(summ)
```

```
## # A tibble: 6 x 10
##   variable    mean median      sd     mad    q5   q95  rhat ess_bulk ess_tail
##   <chr>      <num>  <num>   <num>   <num> <num> <num> <num>    <num>    <num>
## 1 alpha      1.68   1.68  0.00963 0.00989 1.67  1.70   1.00    1940.    1902.
```

[2] https://cran.r-project.org/web/packages/posterior/vignettes/posterior.html

[3] As is common with R, there are multiple ways to do the same thing. For example, the coda package has its own set of functions for MCMC convergence diagnostics. The `posterior` package uses state-of-the-art methods for computing the $\hat{R}$ and effective sample size (Vehtari et al. Bayesian Analysis 2021)

```
## 2 beta       0.563  0.563 0.0423  0.0428  0.495 0.634  1.00    2017.    2050.
## 3 chol_mu[1] 5.50   5.50  0.0536  0.0551  5.42  5.59   1.00    1948.    1899.
## 4 chol_mu[2] 3.97   3.97  0.0996  0.102   3.80  4.13   1.00    2068.    1913.
## 5 chol_mu[3] 4.36   4.36  0.0818  0.0832  4.23  4.49   1.00    2052.    1966.
## 6 chol_mu[4] 5.12   5.12  0.0533  0.0541  5.04  5.22   1.00    1945.    1788.
```

The `posterior` package also allows MCMC samples to be summarised using customised summary functions. To demonstrate how to use it here, we compute

- a 95% credible interval (instead of the default 90%),
- the Monte Carlo standard error of the estimate of the median
- the Monte Carlo standard error of the estimates of the the 95% credible limits.

```
summ <- summary(draws,
                ~quantile(.x, probs=c(0.025, 0.5, 0.975)),
                mcse_median,
                ~mcse_quantile(.x, probs=c(0.025, 0.5, 0.975))
                )
```

We extract from this a summary of the posterior distribution of β, as shown in the lecture (note the results presented may differ up to Monte Carlo error)

```
(beta_summ <- summ[summ$variable=="beta",])
```

```
## # A tibble: 1 x 8
##   variable `2.5%` `50%` `97.5%` mcse_median mcse_q2.5 mcse_q50 mcse_q97.5
##   <chr>     <num> <num>   <num>       <num>     <num>    <num>      <num>
## 1 beta      0.483 0.563   0.646     0.00131   0.00429  0.00131    0.00302
```

We now extract the MCMC sample of the coefficient β as a standard R vector. Then we estimate from this the probability $P(\beta > \beta_0)$ that β exceeds some practically significant value $\beta_0 = 0.62$, say. (This is chosen arbitrarily, for the sake of illustrating the method. A "null" value of $\beta_0 = 0$ might be of interest in many situations).

```
beta <- extract_variable(draws, "beta")
mean(beta > 0.62)
```

```
## [1] 0.088
```

Now we compare the posterior distributions of two different quantities, by x.

(a) Expected cholesterol in 4 years for someone with current cholesterol level of x, by x.

(b) Predicted cholesterol level in 4 years for a person with current cholesterol x

Again there are many different ways to extract the posterior summary statistics in R from the output. The goal is to produce a "tidy" data frame with columns corresponding to the median, lower and upper credible limits, and rows for different observations, that can be easily plotted using the `ggplot2` package. We will demonstrate two different ways — but there are many other reasonable ways to do the same thing!

A simple way is to extract the rows of the summary data frame `summ` computed above, that correspond to the variables (a) `chol_mu` and (b) `chol_pred`. We can optionally rename the variables, as shown.

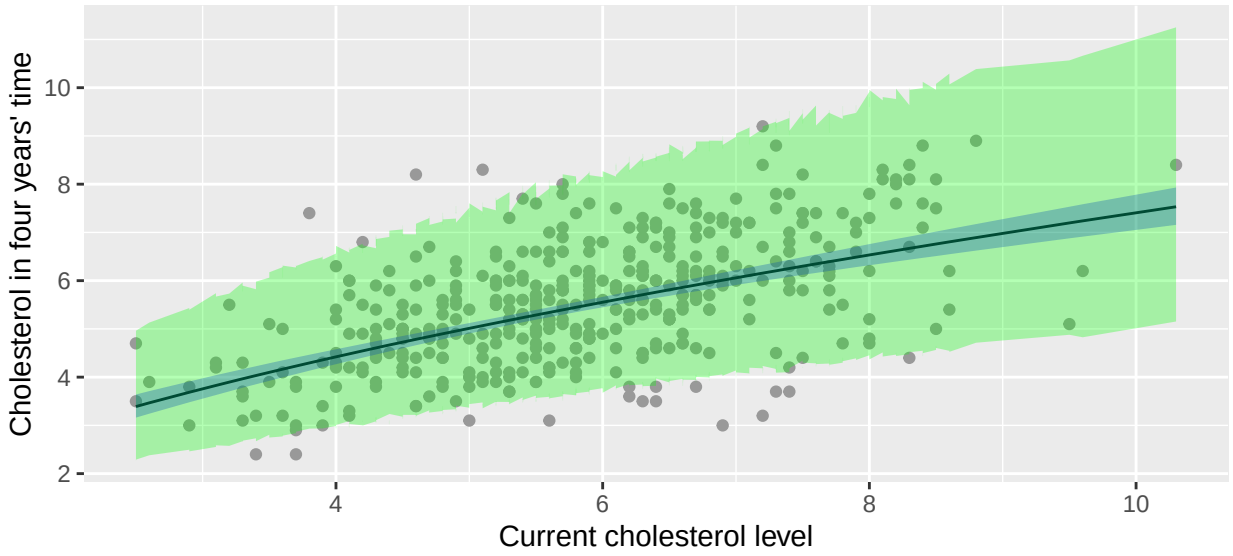
```
chol_mu <- summ %>%
  filter(grepl("chol_mu", variable)) %>%
  rename(chol_mu = `50%`, .lower=`2.5%`, .upper=`97.5%`)
chol_pred <- summ %>%
  filter(grepl("chol_pred", variable)) %>%
  rename(chol_pred = `50%`, .lower=`2.5%`, .upper=`97.5%`)
```

An alternative, more modern way, is to use the `tidybayes` package, which generalises better to complicated structures where the variables are multi-dimensional. In this case it is slower, as we have to recompute the posterior summary statistics.

```
library(tidybayes)
chol_mu <- draws %>% spread_rvars(chol_mu[i]) %>% median_qi(chol_mu)
chol_pred <- draws %>% spread_rvars(chol_pred[i]) %>% median_qi(chol_pred)
```

Whichever method is used, we end up with a data frame `chol_mu` containing summaries of the *expected* values (a), and a data frame `chol_pred` containing summaries of the *predicted* values, (b). As expected, the intervals for the predicted values are much wider, and reflect the between-individual variability seen in the data, as well as just uncertainty about the mean.

```
ggplot(ELSA) +
  geom_point(aes(x=chol_curr, y=chol_4yr), col="gray60") +
  geom_line(data=chol_mu, aes(y=chol_mu, x=ELSA$chol_curr)) +
  geom_ribbon(data=chol_mu,
              aes(x=ELSA$chol_curr, y=chol_mu, ymin=.lower, ymax=.upper),
              alpha=0.3, fill="blue") +
  geom_ribbon(data=chol_pred,
              aes(x=ELSA$chol_curr, y=chol_pred, ymin=.lower, ymax=.upper),
              alpha=0.3, fill="green") +
  xlab("Current cholesterol level") +
  ylab("Cholesterol in four years' time")
```



2 Multiple linear regression - exercises

Key learning points

- Basic data summarising
- Defining priors in linear regression models
- Implementing linear regression in JAGS
- Extracting interpretable outputs from linear regression models

Extend the Bayesian analysis done in the demonstration to add a second covariate representing year of age, so we obtain a model for predicting a person's next cholesterol level based on their current cholesterol level and their current year of age.

(a) Plot and/or summarise the appropriate data.

(b) Define a suitable prior for the coefficient. There is no single correct answer — the important part is to clearly express the judgement or principle that the prior is based on. You could either:

  (i) pick a vague prior — one which is sure to include all plausible values, or

  (ii) imagine you have some substantive information about effect of age on cholesterol, and try to convert that into a prior

(c) Extend the JAGS model code, and produce a sample from the posterior. Remember centering covariates, defining initial values and checking convergence.

(d) Produce some summaries of the posterior distributions to show the relation of age to the outcome. For example:

- for the coefficient representing the effect of age (expressed in some easily-interpretable unit)
- for a prediction of the cholesterol level in 4 years that we would observe from a 60 year old with average current cholesterol levels.
- for a prediction of the cholesterol level in 4 years that we would observe from an 80 year old with average current cholesterol levels.

3 Logistic regression - exercises

Key learning points

- Implementing logistic regression in JAGS
- Extracting interpretable outputs from logistic regression models

The following JAGS code defines a logistic regression model for the ELSA dataset shown in the lecture. The outcome is diabetes diagnosis within 4 years' time, and the predictor is current age. Age is centered around 50 years and expressed in units of 10 years. A standard logistic prior is used for the intercept (uniform on the probability scale) and a weakly informative prior is used for the coefficient of age, as in the lecture.

```
lr_jagsmod <- "model {
 for (i in 1:n) {
  diab_4yr[i] ~ dbern(p[i])
  logit(p[i]) <- alpha + beta*(age[i] - 50)/10
 }
 alpha ~ dlogis(0, 1)
 beta  ~ dnorm(0, 1/2.5^2)
}"
```

In the following exercises, variants of this model will be used. The data should first be transformed as follows:

- We include only people who do not currently have diabetes (since the outcome of interest is the risk of getting diabetes within 4 years)
- We take a random 10% subset (to make the computation faster, for illustrative purposes!)

```
set.seed(1) # so we get the same random 10% sample every time
ELSA10 <- ELSA[ELSA$diab_curr==0,] # restrict to people who don't currently have diabetes
ELSA10 <- ELSA10[sample(1:nrow(ELSA10), size=100),]
dat <- list(n=nrow(ELSA10), diab_4yr = ELSA10$diab_4yr, age = ELSA10$age)
```

(a) Obtain and summarise the posterior distribution for the *odds ratio* corresponding to 10 years of age.

(b) Obtain and summarise the posterior distribution for the *difference in risk* of diabetes between a person aged 60 and a person aged 50.

The inverse logit function $\exp(x)/(1+\exp(x))$ may be helpful here: in JAGS this is called `ilogit`, and in R this is called `plogis`.

Solutions: Multiple linear regression

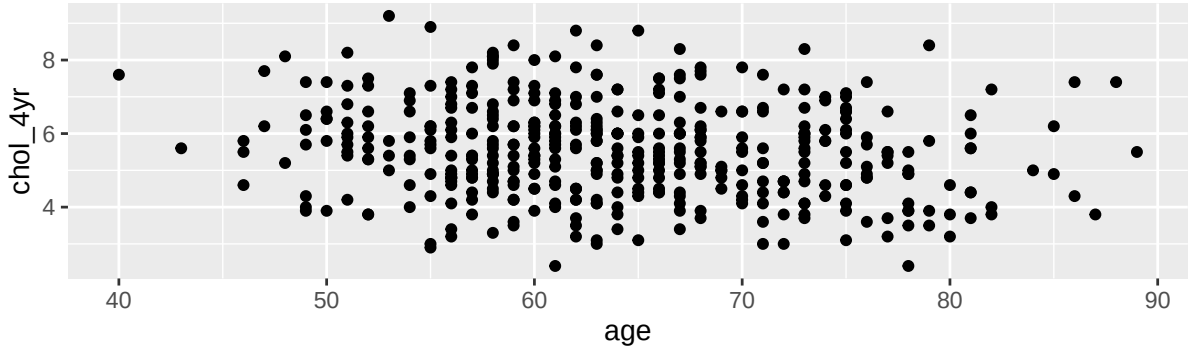
(a) Example of a data summary. We want to know the ranges taken by year of age in the data, and to show the relationship between year of age and cholesterol. Since we are looking to predict next cholesterol based on current cholesterol and current age, we might also want to show how the *change* between current and next cholesterol (variable `tc_change` in the code below) is related to age. No clear pattern emerges. There is the hint of lower cholesterol at older ages, though this might be explained by those with higher cholesterol having died from cardiovascular diseases at younger ages.

Key learning points

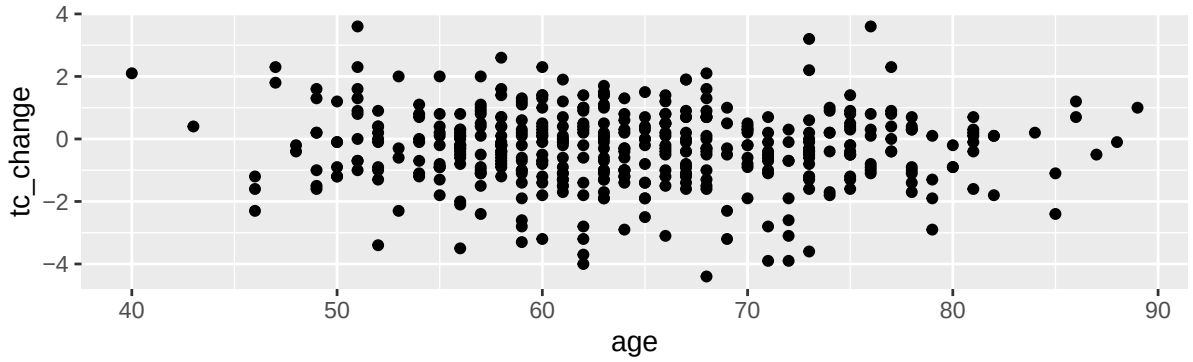
- Summarising data to inform a scientific analysis

Note that this kind of question does not have a single correct answer. The idea is to give a reasonable illustration of data that are related to the scientific question being addressed.

```
ggplot(ELSA, aes(x=age, y=chol_4yr)) +
  geom_point()
```




```
ELSA %>%
  mutate(tc_change = chol_4yr - chol_curr) %>%
  ggplot(aes(x=age, y=tc_change)) +
  geom_point()
```



(b) Example of a prior for the coefficient β of age

Key learning points

- Defining priors in linear regression models

We would typically use

- a normal **distribution** for β (there are other possible choices of distribution, e.g. t or Cauchy, but this is beyond this course).
- ... with a **mean** of zero, if we are uncertain whether the association between the covariate and outcome is positive or negative.

Then it remains to choose the prior variance or standard deviation.

- **Vague prior standard deviation**. To make sure our choice is actually vague, we should think about the scale, or typical order of magnitude, of the data we are modelling. Log cholesterol takes values of around 1–2. Year of age takes values from about 0 to 100. A prior standard deviation of 100 for the effect of one single year of age on log cholesterol, for example, would be very vague in this context.
- **Informative prior standard deviation**. Suppose that we are 95% sure that a population of 50-year-olds and a population of 60-year-olds (or any two fixed-age populations 10 years apart) have average log cholesterol no more than 0.7 units apart (in either direction). (note this is a statement of uncertainty about the population mean, not uncertainty about an individual value)

  Mathematically, we are saying that $\mu(x = 60) - \mu(x = 50)$ has a 95% chance of being between -0.7 and 0.7, where $\mu(x) = \alpha + \beta x$. Therefore 10β has a 95% chance of being between -0.7 and +0.7.

  So we choose the prior SD s for β such that the prior credible interval is about (-2s, +2s) = (-0.07, 0.07) (using the property of the normal distribution that a 95% credible interval is about ± 2 standard deviations from the mean, and we set the prior mean to zero). Therefore $\sigma = 0.07/2$.

  Alternatively (as in the code below) we can transform the data to express age x in units of 10 years rather than units of 1 year, and use $\sigma = 0.7/2$. This makes the implementation cleaner if we would prefer to define priors and interpret the results in terms of 10-year units.

(c) JAGS code for the linear regression model with two covariates.

Key learning points

- Implementing linear regression in JAGS: model code, fitting the model, checking MCMC convergence
- Interpreting outputs from fitted models

```
lm_jagsmod <- "model {
 for (i in 1:n) {
  logchol_4yr[i] ~ dnorm(mu[i], prec)  # precision, not SD or variance.
  mu[i] <- alpha +
   beta_logchol * (logchol_curr[i] - mean(logchol_curr[])) +
   beta_age * (age[i] - 60) / 10
 }

 # expected log cholesterol for 60 and 80 year olds with average current log cholesterol
 mu_age60 <- alpha
 mu_age80 <- alpha + beta_age*(80-60)/10

 # predictions of log cholesterol for a 60 and 80 year old
 # with average current log cholesterol
 pred_age60 ~ dnorm(alpha, prec)
 pred_age80 ~ dnorm(mu_age80, prec)

 # on non-log scale
 chol_age60 <- exp(pred_age60)
 chol_age80 <- exp(pred_age80)

 alpha ~ dnorm(0, 1/100^2)
 beta_logchol  ~ dnorm(0, 1/100^2)
 beta_age ~ dnorm(0, 1/(0.7/2)^2) # precision, not SD
 prec  ~ dgamma(0.0001, 0.0001)
 sd <- 1/sqrt(prec)
}"
```

There are many reasonable ways to write code for this kind of model. In the example above, we have scaled the age covariate by 10 years, so its coefficient β represents the difference in mean outcomes between people 10 years apart in age. We have also roughly centred the variable around a "typical" value of 60 years, so the intercept α represents the expected log cholesterol at the next visit for a 60-year old with average current log cholesterol. Rough centering and scaling in this way is usually enough to aid convergence — covariates do not need to be exactly centered around zero or scaled to have unit standard deviation. Clarity and ease of interpretation are helpful as well.

The code defining the variables `mu_age60`, `mu_age80`, `pred_age60` and `pred_age80` is used for producing the predicted values in part (d). (For conciseness, the code to predict a value corresponding to each observation `i` is removed here. This may or may not be needed in your applications — for example it may be useful to check the fit of the model. See the next lecture about model checking).

Next, the model, data and initial values are supplied to JAGS.

In the examples here, we usually define a new data list every time a JAGS model is run, containing only the data needed for that model. If there any variables in the data that are not in the model, then JAGS will give an "Unused variable" warning message. But this is not an error. For a single research project, it might be preferable to have a single list containing all variables that might be needed for all JAGS models that might be investigated in that project (if you don't mind the warning message).

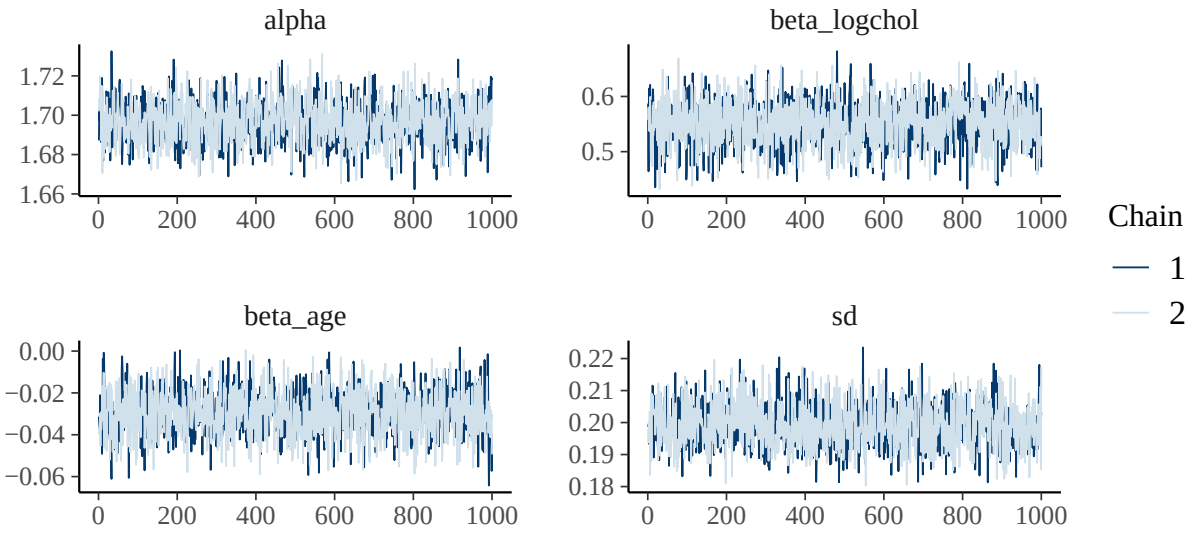
```
dat <- list(n=nrow(ELSA),
        logchol_4yr = log(ELSA$chol_4yr),
        logchol_curr = log(ELSA$chol_curr),
        age = ELSA$age)  # note we could also have scaled/transformed age in R
                         # rather than in JAGS
```

The initial values for the coefficient of age are chosen to be different between chains, but not extreme in the context of the prior.

```
chol_in <- list(list(alpha=1, beta_logchol=0, beta_age=0, prec=1),
            list(alpha=2, beta_logchol=1, beta_age=1, prec=1))
```

After a burn-in of 1000 iterations, a further 1000 iterations are drawn. The trace plots and $\hat{R}$ statistics of 1.00 suggest convergence for the parameters that define the model, so that this sample of 1000 can be treated as coming from the posterior distribution we want.

```
chol_jag <- jags.model(textConnection(lm_jagsmod),
                   data=dat, inits=chol_in, n.chains=2, quiet=TRUE)
pars_basic <- c("alpha","beta_logchol", "beta_age", "sd")
update(chol_jag, 1000)
sam <- coda.samples(chol_jag, c(pars_basic,"chol_age60","chol_age80"), n.iter=1000)
mcmc_trace(sam, pars_basic)
```




```
draws <- as_draws(sam)
subs <- subset_draws(draws, variable=pars_basic)
summary(subs)
```

```
## # A tibble: 4 x 10
##   variable        mean  median      sd     mad      q5     q95  rhat ess_bulk
##   <chr>          <num>   <num>   <num>   <num>   <num>   <num> <num>    <num>
## 1 alpha          1.70    1.70   0.0107  0.0109   1.68    1.71    1.00    1327.
## 2 beta_logchol   0.551   0.551  0.0398  0.0402   0.482   0.615   1.00    1751.
## 3 beta_age      -0.0296 -0.0298 0.0109  0.0113  -0.0472 -0.0115  1.00    1398.
## 4 sd             0.199   0.199  0.00675 0.00662  0.189   0.211   1.00    1643.
## # i 1 more variable: ess_tail <num>
```

(d) Summaries of posterior distributions of quantities of interest

Key learning points

- Interpreting outputs from fitted models

```
summ <- summary(draws,
            ~quantile(.x, probs=c(0.025, 0.5, 0.975)))
summ %>% filter(variable %in% c("beta_logchol","beta_age","chol_age60","chol_age80"))
```

```
## # A tibble: 4 x 4
##   variable      `2.5%`   `50%`   `97.5%`
##   <chr>          <num>   <num>     <num>
## 1 beta_age     -0.0507 -0.0298 -0.00830
## 2 beta_logchol  0.467   0.551   0.625
## 3 chol_age60    3.69    5.50    7.91
## 4 chol_age80    3.43    5.14    7.54
```

The posterior distribution of the coefficient β_{age}, the difference between the average cholesterol levels of people 10 years apart in age, has a 95% credible interval that spans a range of small negative values. So although the effect might be considered "statistically significant", it is very small compared to the range of cholesterol levels observed from different individuals (from the scatterplot produced in part (a)).

Comparing the posterior medians of `chol_age60` and `chol_age80`, the predicted next cholesterol level for an 80 year old is 0.4 mmol/L lower than that for a 60 year old, assuming both have average current cholesterol level. Though this difference is small compared to the credible intervals around the predictions. Note that these intervals account for the observation-level variability σ, as well as the uncertainty about the regression parameters.

Extra challenge: Instead of adding extra lines to the JAGS code to produce these two predictions, we could also have coded them in plain R. Can you see how this might be done using the function `rnorm`, with suitable samples from the posterior distributions of the regression parameters?

Solutions: logistic regression

Key learning points

- Implementing logistic regression in JAGS
- Extracting interpretable outputs from logistic regression models

The odds ratio and risk difference can be calculated either in JAGS or in R.

To calculate them in JAGS, add new nodes in the model code to define the odds ratio (for 10 years of age) and risk difference (between ages 60 and 50), as follows.

These statements must be placed outside the loop over 'i', otherwise there will be an error message: 'Attempt to redefine node...'.

```
lr_jagsmod <- "model {
for (i in 1:n) {
  diab_4yr[i] ~ dbern(p[i])
  logit(p[i]) <- alpha + beta*(age[i] - 50)/10
 }
 alpha ~ dlogis(0, 1)
 beta  ~ dnorm(0, 1/2.5^2)
 or <- exp(beta)
 risk_50 <- ilogit(alpha)
 risk_60 <- ilogit(alpha + beta)
 risk_diff <- risk_60 - risk_50
}"
set.seed(1) # so we get the same random 10% sample every time
ELSA10 <- ELSA[ELSA$diab_curr==0,] # restrict to people who don't currently have diabetes
ELSA10 <- ELSA10[sample(1:nrow(ELSA10), size=100),]
dat <- list(n=nrow(ELSA10), diab_4yr = ELSA10$diab_4yr, age = ELSA10$age)
ini <- list(list(alpha=0, beta=0),
            list(alpha=1, beta=1))
diab_jag <- jags.model(textConnection(lr_jagsmod), data=dat, inits=ini, n.chains = 2)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 2
##    Total graph size: 371
##
## Initializing model
```

```
sam <- coda.samples(diab_jag, c("or", "alpha","beta","risk_diff"), n.iter=10000)[[1]]
summary(as_draws(sam))
```

```
## # A tibble: 4 x 10
##   variable       mean   median     sd    mad      q5     q95  rhat ess_bulk
##   <chr>         <num>    <num>  <num>  <num>   <num>   <num> <num>    <num>
## 1 alpha     -3.19     -3.11    0.866  0.841  -4.74   -1.89    1.00     883.
## 2 beta       0.180     0.176   0.492  0.488  -0.629   0.998   1.00     869.
## 3 or         1.35      1.19    0.703  0.566   0.533   2.71    1.00     869.
## 4 risk_diff -0.000319  0.00676 0.0253 0.0151 -0.0505  0.0260  1.00    1333.
## # i 1 more variable: ess_tail <num>
```

Alternatively, we can calculate the odds ratio and risk difference in R. To do this, we would use, e.g. `coda.samples` to obtain samples from the posterior distributions of `alpha` and `beta` in R, and transform these to obtain a sample from the posterior distribution of the risk difference. The code shown below uses the R function `plogis()` as a shortcut for `exp()/(1 + exp())`. (In fact `plogis` is the CDF of the standard logistic distribution).

```
lr_jagsmod <- "model {
for (i in 1:n) {
  diab_4yr[i] ~ dbern(p[i])
  logit(p[i]) <- alpha + beta*(age[i] - 50)/10
 }
 alpha ~ dlogis(0, 1)
 beta  ~ dnorm(0, 1/2.5^2)
}"

diab_jag <- jags.model(textConnection(lr_jagsmod), data=dat, inits=ini, n.chains=2)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 2
##    Total graph size: 366
##
## Initializing model
```

```
sam <- coda.samples(diab_jag, c("alpha","beta"), n.iter=10000)[[1]]
or_beta <- exp(sam[,"beta"])
quantile(or_beta, c(0.05, 0.5, 0.95))
```

```
##       5%      50%      95%
## 0.520823 1.197471 2.756344
```

```
risk_50 <- plogis(sam[,"alpha"] + sam[,"beta"]*(50 - 50)/10)
risk_60 <- plogis(sam[,"alpha"] + sam[,"beta"]*(60 - 50)/10)
risk_diff <- risk_60 - risk_50
quantile(risk_diff, c(0.05, 0.5, 0.95))
```

```
##           5%          50%          95%
## -0.055243423  0.006835869  0.025849565
```

In either implementation, the odds ratio has a posterior median of around 1.2 (with 90% CI 0.52 to 2.76) and the risk difference has posterior median of around 0.01 (90% CI -0.06 to 0.03). The exact differences between the two implementations are due to Monte Carlo error.