Class 2: Creating bespoke time series models using Bayes

Andrew Parnell
andrew.parnell@ucd.ie

Learning outcomes

- Know the difference between Frequentist and Bayesian statistics
- Be able to follow the syntax of JAGS and Stan
- Know how to fit some basic AR and regression models in JAGS and Stan
- Be able to manipulate the output of these models

Who was Bayes?

An essay towards solving a problem on the doctrine of chances (1763)

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

Bayes theorem in english

Bayes' theorem can be written in words as:

posterior is proportional to likelihood times prior

. . . or . . .

posterior $\propto$ likelihood $\times$ prior

Each of the three terms *posterior*, *likelihood*, and *prior* are *probability distributions* (pdfs).

In a Bayesian model, every item of interest is either data (which we will write as x) or parameters (which we will write as θ). Often the parameters are divided up into those of interest, and other *nuisance parameters*

Bayes theorem in maths

Bayes' equation is usually written mathematically as:

$$p(\theta|x) \propto p(x|\theta) \times p(\theta)$$

or, more fully:

$$p(\theta|x) = \frac{p(x|\theta) \times p(\theta)}{p(x)}$$

- The *posterior* is the probability of the parameters given the data
- The *likelihood* is the probability of observing the data given the parameters (unknowns)
- The *prior* represents external knowledge about the parameters

What's different from what we were doing before?

- We still have a likelihood and parameters to estimate
- We now also have some extra constraints (defined by us) called the *prior distribution*
- There is a clever Bayesian algorithm to create the resulting parameter estimates and their uncertainties
- This full probability distribution of the outputs is the posterior distribution
- The full posterior probability distribution is provided to us as a set of samples (recall class 2 on day 1)

Choosing a prior

- The key to choosing a prior distribution is to choose values which you believe represent the reasonable range that the parameter can take, or come from a related study in the literature
- A prior which is a strong constraint on the parameters is called an *informative prior*
- Some people argue that informative priors are bad, others that they are absolutely necessary in every model
- Sometimes an informative prior can be the difference between being able to fit the model or not
- Most people forget that choosing a likelihood probability distribution is exactly the same task as choosing a prior

Practical differences between frequentist statistics and Bayes

- In frequentist statistics you tend to get a single best estimate of a parameter and a standard error, often assumed normally distributed, and a p-value
- In Bayesian statistics you get a large set of samples of the parameter values which match the data best. You get to choose what you do with these
- In frequentist statistics if the p-value is less than 0.05 you win. If not you cry and try a different model
- In Bayesian statistics you try to quantify the size of an effect from the posterior distribution, or find a particular posterior probability, e.g. P(slope $>$ 0 given the data).

Stan and JAGS

- We will be using two different software tools to calculate posterior distributions. These represent the state of the art for user-friendly, research quality Bayesian statistics.
- Both have their own programming language which you can write in R and then fit the models to get the posterior distribution
- All we have to do in the programming language is specify the likelihood and the priors, and give it the data. The software does the rest

Steps for running JAGS and Stan

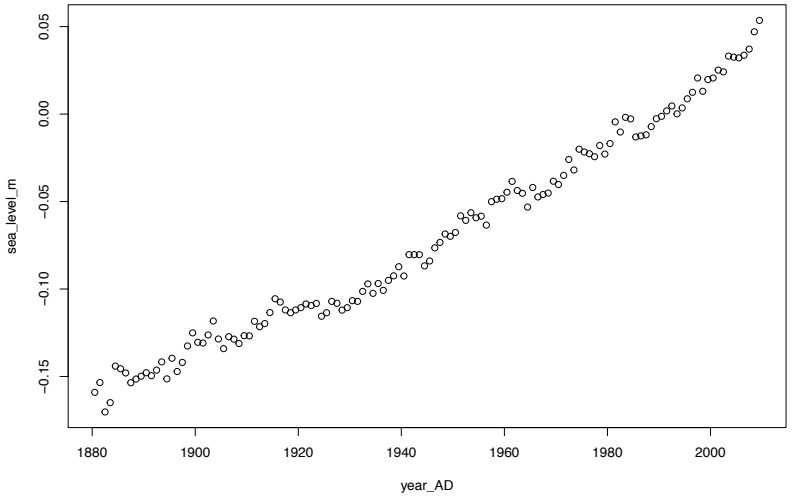
1. Write some Stan or JAGS code which contains the likelihood and the prior(s)
2. Get your data into a list so that it matches the data names used in the Stan/JAGS code
3. Run your model through Stan/JAGS
4. Get the posterior output
5. Check convergence of the posterior probability distribution
6. Create the output that you want (forecasts, etc)

Stan vs JAGS

- Stan positives: very flexible, uses sensible distribution names, everything is declared, lots of documentation support, written by people at the top of the field
- Stan negatives: cannot have discrete parameters, some odd declaration choices, slower to run code, code tends to be longer
- JAGS positives: very quick for simple models, no declarations required, a bit older than Stan so more queries answered online
- JAGS negatives: harder to get complex models running, not as fancy an algorithm as Stan, crazy way of specifying normal distributions

Reminder: sea level example

```
sl = read.csv('../../data/tide_gauge.csv')
with(sl, plot(year_AD, sea_level_m))
```



Fitting linear regression models in JAGS

```
library(R2jags)
jags_code = '
model {
  # Likelihood
  for(i in 1:N) {
    y[i] ~ dnorm(alpha + beta*x[i], sigma^-2)
  }
  # Priors
  alpha ~ dnorm(0, 100^-2)
  beta ~ dnorm(0, 100^-2)
  sigma ~ dunif(0, 100)
}'
jags_run = jags(data = list(N = nrow(sl),
                            y = sl$sea_level_m,
                            x = sl$year_AD),
                parameters.to.save = c('alpha', 'beta',
                                       'sigma'),
                model.file = textConnection(jags_code))
```

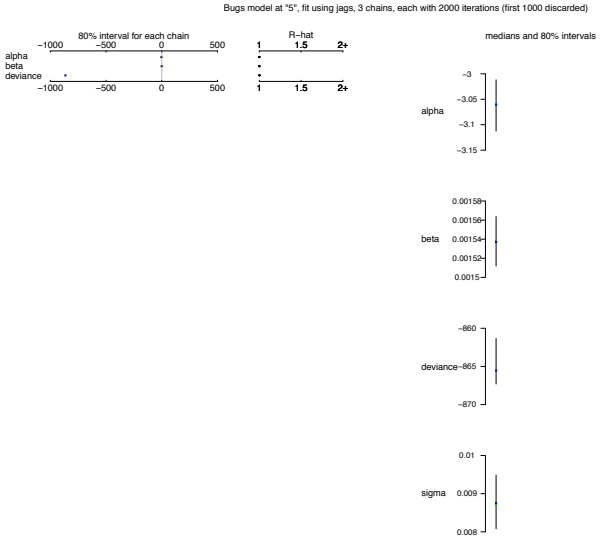

Output

```
print(jags_run)
```

```
## Inference for Bugs model at "5", fit using jags,
##  3 chains, each with 2000 iterations (first 1000 discarded)
##  n.sims = 3000 iterations saved
##           mu.vect sd.vect     2.5%      25%
## alpha      -3.062   0.039   -3.140   -3.088
## beta        0.002   0.000    0.001    0.002
## sigma       0.009   0.001    0.008    0.008
## deviance -864.843   2.551 -867.661 -866.697
##               50%      75%    97.5%  Rhat n.eff
## alpha      -3.061   -3.035   -2.986 1.001  3000
## beta        0.002    0.002    0.002 1.001  3000
## sigma       0.009    0.009    0.010 1.002  2700
## deviance -865.546 -863.772 -858.276 1.002  2000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 3.3 and DIC = -861.6
## DIC is an estimate of expected predictive error (lower deviance is better).
```


Plotted output

```
plot(jags_run)
```



What do the results actually mean?

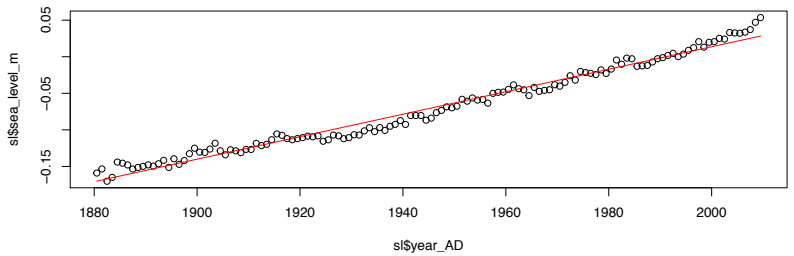
- We now have access to the posterior distribution of the parameters:

```
post = jags_run$BUGSoutput$sims.matrix
head(post)
```

```
##          alpha        beta  deviance       sigma
## [1,] -2.998503 0.001504268 -859.6400 0.008696697
## [2,] -3.030405 0.001522177 -864.0947 0.009228137
## [3,] -2.996163 0.001504341 -864.2112 0.008881095
## [4,] -3.104113 0.001559371 -866.1796 0.009012243
## [5,] -3.098579 0.001557082 -860.4126 0.007564283
## [6,] -3.100353 0.001557821 -866.1110 0.008354450
```

Plots of output

```
alpha_mean = mean(post['alpha'])
beta_mean = mean(post['beta'])
plot(sl$year_AD, sl$sea_level_m)
lines(sl$year_AD, alpha_mean +
        beta_mean * sl$year_AD, col = 'red')
```



Running the same model in Stan

```
stan_code = '
data {
  int<lower=0> N;
  vector[N] y;
  vector[N] x;
}
parameters {
  real alpha;
  real beta;
  real<lower=0> sigma;
}
model {
  y ~ normal(alpha + beta * x, sigma);
  alpha ~ normal(0, 100);
  beta ~ normal(0, 100);
  sigma ~ uniform(0, 100);
}
'
```


Running the Stan version

```
library(rstan)
stan_run = stan(data = list(N = nrow(sl),
                            y = sl$sea_level_m,
                            x = sl$year_AD/1000),
                model_code = stan_code)
```

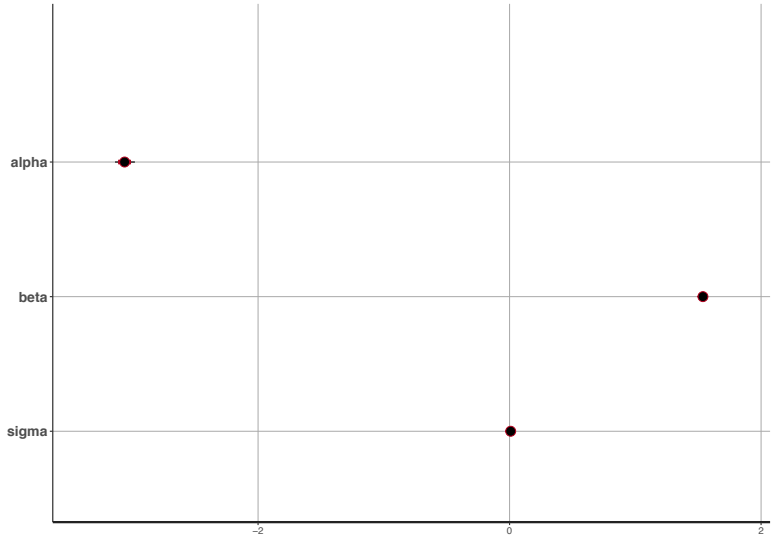

Stan output

```
print(stan_run)
```

```
## Inference for Stan model: 425dcdc3a818151313cabf3f64999e4f.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=400
##
##         mean se_mean   sd   2.5%    25%    50%
## alpha  -3.06    0.00 0.04  -3.14  -3.09  -3.06
## beta    1.54    0.00 0.02   1.50   1.52   1.54
## sigma   0.01    0.00 0.00   0.01   0.01   0.01
## lp__  547.11    0.04 1.22 543.94 546.53 547.44
##          75%  97.5% n_eff Rhat
## alpha  -3.03  -2.98   857 1.01
## beta    1.55   1.58   859 1.01
## sigma   0.01   0.01  1242 1.00
## lp__  548.00 548.52  1015 1.00
##
## Samples were drawn using NUTS(diag_e) at Sun Jun 25 11:50:42
## For each parameter, n_eff is a crude measure of effective sam
## and Rhat is the potential scale reduction factor on split cha
```

Stan plots

```
plot(stan_run)
```



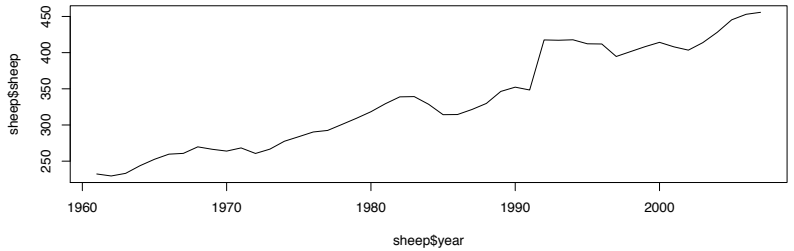
An AR(1) model in JAGS

```
jags_code = '
model {
  # Likelihood
  for (t in 2:N) {
    y[t] ~ dnorm(alpha + beta * y[t-1], sigma^-2)
  }

  # Priors
  alpha ~ dnorm(0, 100^-2)
  beta ~ dunif(-1, 1)
  sigma ~ dunif(0, 100)
}
'
```


Run the model on the sheep

```
sheep = read.csv('../../data/sheep.csv')
plot(sheep$year, sheep$sheep, type = 'l')
```




```
jags_run = jags(data = list(N = nrow(sheep),
                            y = sheep$sheep),
                parameters.to.save = c('alpha',
                                       'beta',
                                       'sigma'),
                model.file = textConnection(jags_code))
```

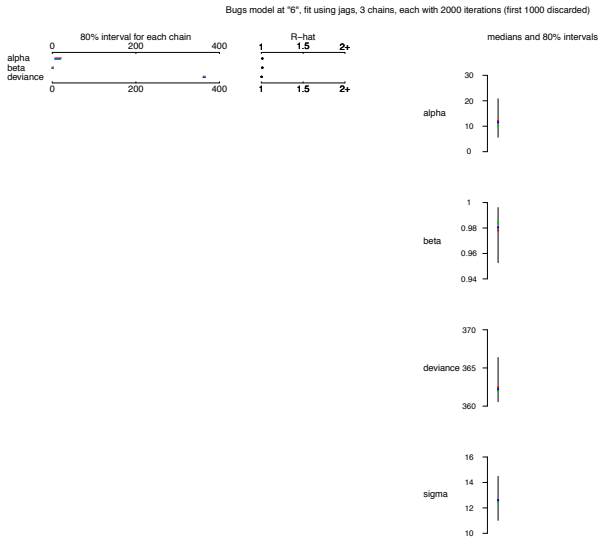

Output

```
print(jags_run)
```

```
## Inference for Bugs model at "6", fit using jags,
##  3 chains, each with 2000 iterations (first 1000 discarded)
##  n.sims = 3000 iterations saved
##          mu.vect sd.vect    2.5%     25%     50%
## alpha     12.333   5.908   3.801   7.825  11.251
## beta       0.977   0.017   0.935   0.968   0.981
## sigma     12.679   1.390  10.383  11.703  12.553
## deviance 362.907   2.444 360.249 361.132 362.159
##              75%   97.5%  Rhat n.eff
## alpha     15.780  26.933 1.013   170
## beta       0.991   0.999 1.013   170
## sigma     13.487  15.933 1.002  1300
## deviance 364.000 369.096 1.004   520
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 3.0 and DIC = 365.9
## DIC is an estimate of expected predictive error (lower deviance is better).
```

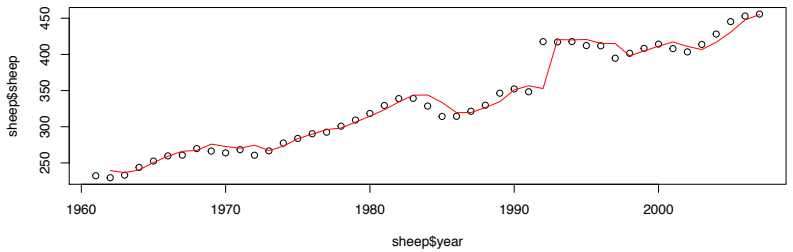
Plotted output

```
plot(jags_run)
```



Plots of one step ahead forecasts

```
post = jags_run$BUGSoutput$sims.matrix
alpha_mean = mean(post[,'alpha'])
beta_mean = mean(post[,'beta'])
plot(sheep$year, sheep$sheep)
N = nrow(sheep)
lines(sheep$year[2:N], alpha_mean +
        beta_mean * sheep$sheep[1:(N-1)], col = 'red')
```



What are JAGS and Stan doing in the background?

- JAGS and Stan run a stochastic algorithm called Markov chain Monte Carlo to create the samples from the posterior distribution
- This involves:
  1. Guessing at *initial values* of the parameters. Scoring these against the likelihood and the prior to see how well they match the data
  2. Then iterating:
     - 2.1 Guessing *new parameter values* which may or may not be similar to the previous values
     - 2.2 Seeing whether the new values match the data and the prior by calculating *new scores*
     - 2.3 If the scores for the new parameters are higher, keep them. If they are lower, keep them with some probability depending on how close the scores are, otherwise discard them and keep the old values
- What you end up with is a set of parameter values for however many iterations you chose.

How many iterations?

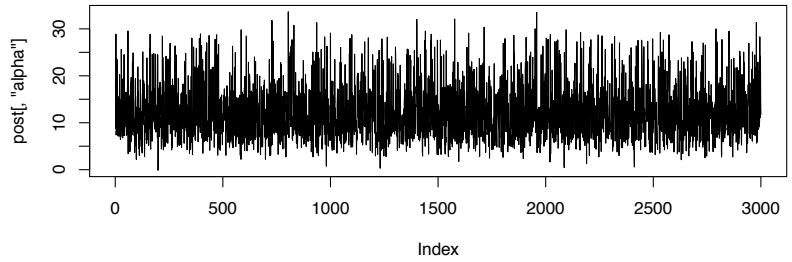
- Ideally you want a set of posterior parameter samples that are independent across iterations and is of sufficient size that you can get decent estimates of uncertainty
- There are three key parts of the algorithm that affect how good the posterior samples are:
  1. The starting values you chose. If you chose bad starting values, you might need to discard the first few thousand iterations. This is known as the *burn-in* period
  2. The way you choose your new parameter values. If they are too close to the previous values the MCMC might move too slowly so you might need to *thin* the samples out by taking e.g. every 5th or 10th iteration
  3. The total number of iterations you choose. Ideally you would take millions but this will make the run time slower

JAGS and Stan have good default choices for these but for complex models you often need to intervene

Plotting the iterations

You can plot the iterations for all the parameters with `traceplot`, or for just one with e.g.

```
plot(post[,'alpha'], type = 'l')
```



A good trace plot will show no patterns or runs, and will look like it has a stationary mean and variance

How many chains?

- Beyond increasing the number of iterations, thinning, and removing a burn-in period, JAGS and Stan automatically run *multiple chains*
- This means that they start the algorithm from 3 or 4 different sets of starting values and see if each *chain* converges to the same posterior distribution
- If the MCMC algorithm has converged then each chain should have the same mean and variance.
- Both JAGS and Stan report the `Rhat` value, which is close to 1 when all the chains match
- It's about the simplest and quickest way to check convergence. If you get `Rhat` values above 1.1, run your MCMC for more iterations

What else can I do with the output

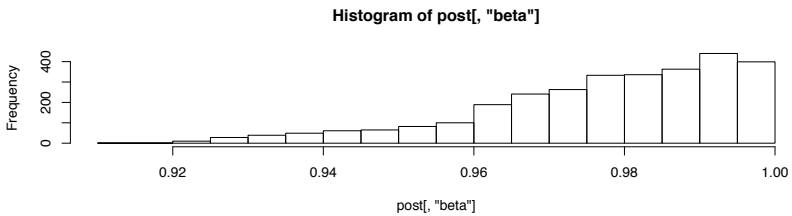
- We could create *credible intervals* (Bayesian confidence intervals):

```
apply(post, 2, quantile, probs = c(0.025, 0.975))
```

```
##           alpha      beta deviance    sigma
## 2.5%   3.800714 0.9345421 360.2488 10.38332
## 97.5% 26.932642 0.9989180 369.0965 15.93260
```

- Or histograms

```
hist(post[,'beta'], breaks = 30)
```



Summary

- Bayesian methods just add on a set of extra constraints on to the likelihood called prior distributions
- We now know how to run some simple time series models in JAGS and Stan
- We know that the fitting algorithm (MCMC) produces best parameter estimates and their uncertainties
- We have to do a little bit more work to get the predictions out of JAGS or Stan
- The big advantage of using these methods is the extra flexibility we get from being able to write our own models