Class 3: Model choice and forecasting with Bayes

Andrew Parnell
andrew.parnell@ucd.ie

Learning outcomes

- See some JAGS code for fitting AR(p), ARMA(p, q) and ARIMAX(p, d, q) models
- Know to check model fit for a Bayesian model using the posterior predictive distribution
- Know how to create k-step ahead forecasts with uncertainty using JAGS

JAGS code for an AR(p) model

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

  # Priors
  alpha ~ dnorm(0, 100^-2)
  for (i in 1:p) {
    beta[i] ~ dnorm(0, 100^-2)
  }
  sigma ~ dunif(0, 100)
}'
```

JAGS code for an ARMA(p, q) model

```
model_code = '
model
{
  # Set up residuals
  for(t in 1:max(p,q)) {
    eps[t] <- z[t] - alpha
  }
  # Likelihood
  for (t in (max(p,q)+1):N) {
    z[t] ~ dnorm(alpha + ar_mean[t] + ma_mean[t], sigma^-2)
    ma_mean[t] <- inprod(theta, eps[(t-q):(t-1)])
    ar_mean[t] <- inprod(beta, z[(t-p):(t-1)])
    eps[t] <- z[t] - alpha - ar_mean[t] - ma_mean[t]
  }
  # Priors
  alpha ~ dnorm(0, 10^-2)
  for (i in 1:q) {
    theta[i] ~ dnorm(0, 10^-2)
  }
  for(i in 1:p) {
    beta[i] ~ dnorm(0, 10^-2)
  }
  sigma ~ dunif(0, 100)
}
'
```

JAGS code for an ARIMAX model (shortened)

```
model_code = '
model
{
  ...
  # Likelihood
  for (t in (q+1):N) {
    y[t] ~ dnorm(alpha + ar_mean[t] + ma_mean[t] + reg_mean[t],
    ma_mean[t] <- inprod(theta, eps[(t-q):(t-1)])
    ar_mean[t] <- inprod(beta, y[(t-p):(t-1)])
    reg_mean[t] <- inprod(phi, x[t,])
    eps[t] <- y[t]-alpha-ar_mean[t]-ma_mean[t]-reg_mean[t]
  }

  # Priors
  ...
  for(i in 1:k) {
    phi[i] ~ dnorm(0, 100^-2)
  }
  ...
}
'
```

Fitting a JAGS ARIMA model

- Let's fit an ARIMA(1, 0, 1) model to the wheat data

```
wheat = read.csv('../../data/wheat.csv')
jags_data = with(wheat,
                 list(N = length(wheat) - 1,
                      z = scale(wheat)[,1],
                      q = 1,
                      p = 1))
jags_run = jags(data = jags_data,
                parameters.to.save = c('alpha',
                                       'theta',
                                       'beta',
                                       'sigma'),
                model.file = textConnection(model_code))
```

Checking output

```r
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%    50%
## alpha      0.051   0.063 -0.096  0.023  0.051
## beta       0.845   0.222  0.344  0.691  0.943
## sigma      0.537   0.057  0.436  0.496  0.534
## theta     -0.295   0.445 -0.834 -0.670 -0.463
## deviance  80.346   4.068 73.985 76.898 80.372
##             75%  97.5%  Rhat n.eff
## alpha     0.079  0.190 1.016  1500
## beta      1.017  1.076 1.031    78
## sigma     0.572  0.658 1.002  1000
## theta     0.122  0.553 1.039    60
## deviance 83.223 88.536 1.028    79
##
## 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 = 8.1 and DIC = 88.4
## DIC is an estimate of expected predictive error (lower deviance is better).
```

Checking model fit

- We have covered how to compare fits in models by comparing e.g. AIC or running cross-validation
- An extra way available via JAGS or Stan is to simulate from the posterior distribution of the parameters, and subsequently simulate from the likelihood to see if the these data match the real data we observed
- This is known as a *posterior predictive check*

Posterior predictive distribution in JAGS

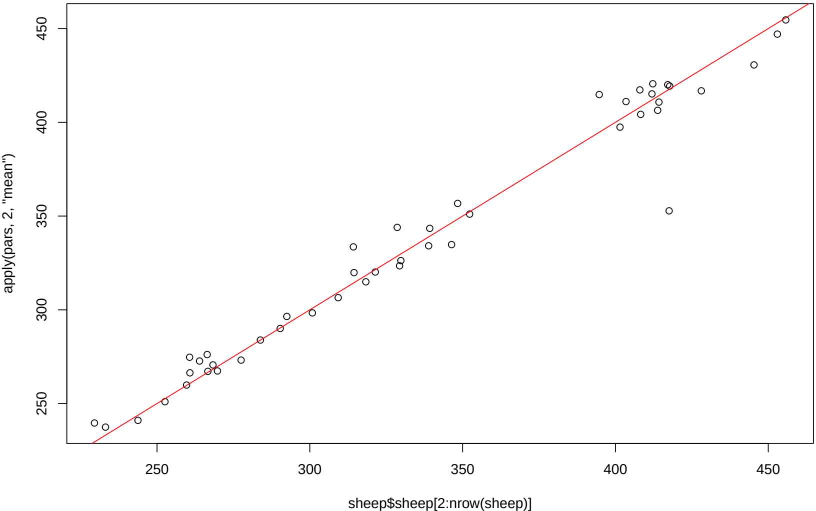
- The easiest way is to put an extra line in the JAGS code, e.g. AR(1):

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

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

Posterior predictive outputs

```
pars = jags_run$BUGSoutput$sims.list$y_pred
plot(sheep$sheep[2:nrow(sheep)], apply(pars,2,'mean'))
abline(a=0, b = 1, col = 'red')
```



Creating predictions inside the JAGS model

- The posterior predictive check for a time series model is really just a check of the one step ahead predictions. However, posterior predictive checks are
- We could create the one-step ahead predictions outside JAGS in R code, but it's usually easier to do it inside the code itself
- We don't have to stop at one step ahead predictions, we can move on to 2 step ahead or further. We would expect the performance of the models to deteriorate

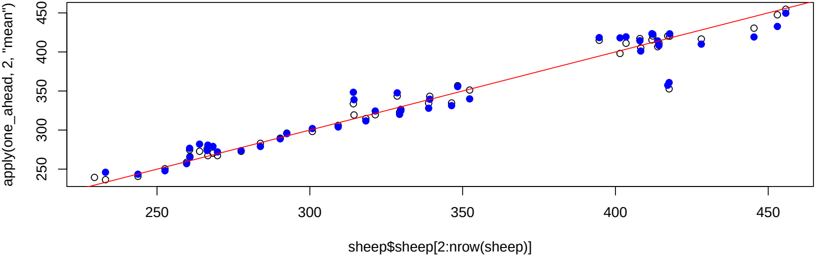
Two step-head predictions for an AR(1) model

```
jags_code = '
model {
  # Likelihood
  for (t in 2:N) {
    y[t] ~ dnorm(alpha + beta * y[t-1], sigma^-2)
    y_one_ahead[t] ~ dnorm(alpha + beta * y[t-1],
      sigma^-2)
  }
  for (t in 3:N) {
    y_two_ahead[t] ~ dnorm(alpha + beta * y_one_ahead[t-1],
      sigma^-2)
  }

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

Output

```
one_ahead = jags_run$BUGSoutput$sims.list$y_one_ahead
two_ahead = jags_run$BUGSoutput$sims.list$y_two_ahead
plot(sheep$sheep[2:nrow(sheep)], apply(one_ahead,2,'mean'))
points(sheep$sheep[3:nrow(sheep)], apply(two_ahead,2,'mean'),
       col = 'blue', pch = 19)
abline(a=0, b = 1, col = 'red')
```



JAGS and the NA trick

- What if we want to create a single set of longer predictions at the end of the data set?
- So far we have been giving JAGS the data in a list. It looks up these objects in the `model_code` file and treats all the others as parameters to be estimated
- If you set some of the values in your data list to the value `NA` (R's missing value placeholder) JAGS will treat these missing data as *parameters to be estimated*
- This is especially useful for time series as we can create extra `NA` y values at the end of our series, and JAGS will magically turn these into future forecasts

The NA trick in action

Start with a simple AR(1) model

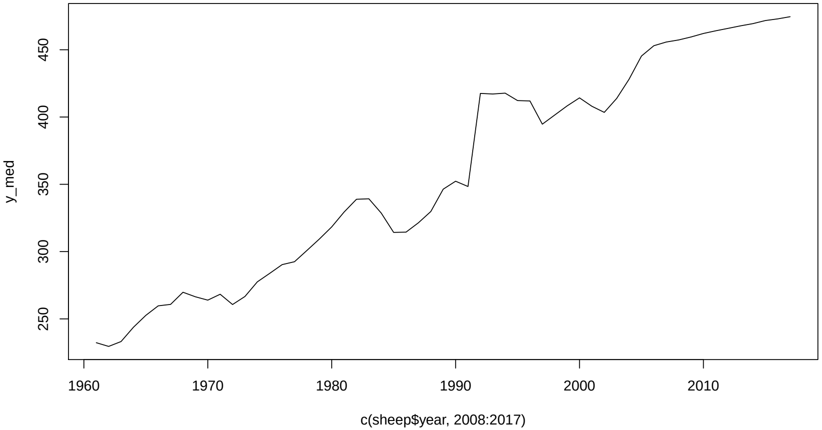
```
model_code = '
model
{
  # Likelihood
  for (t in 2:N) {
    y[t] ~ dnorm(alpha + beta * y[t-1], sigma^-2)
  }
  # Priors
  alpha ~ dnorm(0, 100^-2)
  beta ~ dunif(-1, 1)
  sigma ~ dunif(0, 100)
}
'
```

The NA trick in action (cont)

```
num_forecasts = 10 # 10 extra years
jags_run = jags(data = list(N = nrow(sheep) + num_forecasts
                            y = c(sheep$sheep,
                                  rep(NA,
                                      num_forecasts))),
                parameters.to.save = 'y',
                model.file=textConnection(model_code))
```

NA trick plots

```
y_pred = jags_run$BUGSoutput$sims.list$y
y_med = apply(y_pred,2,'median')
plot(c(sheep$year,2008:2017),y_med,type='l')
```



Notes about the NA trick

- Here I've just plotted the mean forecasts, but I have the full posterior ditribution so it's easy to create lower and upper credible intervals if required

```
apply(y_pred,2,'quantile', c(0.05, 0.95))[,48:57]
```

```
##          [,1]     [,2]     [,3]     [,4]     [,5]
## 5%  436.1552 427.7125 424.0607 419.7161 417.4278
## 95% 478.3068 489.3422 498.9842 506.5916 514.4892
##          [,6]     [,7]     [,8]     [,9]    [,10]
## 5%  412.3413 409.9981 407.3055 402.6052 401.6350
## 95% 521.0892 530.1496 535.7470 542.0755 549.7868
```

- The `NA` trick is fantastically in all kinds of modelling situtaitons, e.g. where we have genuinely missing data.

Choosing different models: DIC

- So far we have met a wide array of discrete-time time series models, all of which involve choose a p (AR component), a q (MA component), and a d (differencing component)
- We need a principled method to choose the best values of these. It will always be the case that increasing these values will lead to a better fit
- There are several proposed methods for doing this:
  1. Treat the model as another parameter (Bayes factors and reversible jump)
  2. Remove some of the data and predict the left out data (Cross-validation)
  3. Use statistical theory to penalise the fit of the model (Information Criteria)
- All of these are good and useful, but number 3 is implemented by JAGS for us to use through the DIC

The Deviance Information Criterion

- As JAGS is running through the iterations, it is constantly calculating the value of the likelihood, the probability of the data given the parameters. JAGS reports this as the *deviance* which is -2 times the log of the likelihood
- For a good set of parameters the value of the deviance should be high, and the model once converged should reach a stable value of the deviance
- If you run the model with, e.g. an extra AR term, you'd find that the deviance (once the model had converged) would be slightly higher
- The idea behind *information criteria*, as we have seen, is to penalise the deviance by a measure of the complexity of the model

Measuring model complexity

- Measuring model complexity isn't quite so simple in the Bayesian world as the number of parameters, in the presence of prior information, can be hard to estimate
- The version JAGS uses is known as the Deviance Information Criterion (DIC) and is built specifically to penalise the deviance by the *effective* number of parameters, which it calls p_D

The components of DIC

- JAGS provides the DIC whenever we call the `print` command on a model run

```
DIC info (using the rule, pD = var(deviance)/2)
pD = 6.9 and DIC = -261.1
```

- Here p_D estimates the effective number of parameters in the model and the DIC is calculated as the deviance plus the p_D value
- The usual practice is to run models of differing complexity (e.g. with differing values of p, d, and q) and choose the model with the lowest DIC

Summary

- We have seen some JAGS code for some of the more complicated models we have met
- We have fitted them to some of the data sets we have met
- We know how to create one step ahead (or more) forecasts for a JAGS model