Module 6: ARIMAX, model choice and forecasting

Andrew Parnell, School of Mathematics and Statistics,
University College Dublin

Learning outcomes

- Be able to add on components to ARIMA models
- Understand the issues with fitting ARIMAX and other extensions to ARIMA models
- Understand how to create forecasts with JAGS via the NA method
- Know how to perform model choice with DIC
- Know how to perform model choice with cross-validation
- Know the basics of forecast calibration and scoring rules

Relevant JAGS file:

`jags_ARIMAX.R`

The great advantage of Bayes/JAGS: bolting together models

- As we have already seen we can combine bits of models together, such as RW, AR and MA, into ARIMA
- The default time series software in R (e.g. `arima`) can already fit such models far quicker than JAGS, so why are we bothering to do it this way?
- Because we have all the code written out, we can add any other modelling ideas to it
- This allows us to do research in JAGS at the cutting edge of applied and theoretical statistics

Mixing up GLMs with time series models: some basic ideas

- As an example, suppose our response variable y_t was actually binary; we can't use the default ARIMA framework
- Instead use ideas from logistic regression, e.g. the logistic-AR model:

$$y_t \sim Bin(1, p_t);\ \text{logit}(p_t) = \alpha + \phi p_{t-1}$$

- What if the response variable is unbounded counts, no problem! The Poisson AR model:

$$y_t \sim Poisson(\lambda_t);\ \log(p_t) = \alpha + \phi p_{t-1}$$

- In JAGS, we can adapt any of the time series to any GLM situation

The ARIMAX framework

- The ARIMAX framework (ARIMA with eXplanatory variables) is just another such extension which is easy to add in JAGS
- We ARIMAX model is:

$$z_t \sim N(\alpha + \text{AR terms} + \text{MA terms} + \beta_1 x_{1t} + \ldots \beta_r x_{rt}, \sigma^2)$$

  where we now include r possible explanatory variables with coefficients $\beta_1, \ldots, \beta_r$
- This can be easily added to the code in JAGS

Warnings about ARIMAX models

There are two key things to be wary of when using this type of ARIMAX model:

1. It's hard to interpret the β values. It is not the case (as in normal regression) that an increase of 1 unit in x will lead to an increase of β in y because of all the AR terms.
2. If you are differencing the data before running the model, you also need to difference the explanatory variables

If you're just interest in forecasting then the problem in 1 goes away, but if you are interested in the causation of x on y you can fit the regression model separately or try a dynamic regression model (see Module 10)

JAGS code for an ARIMAX model (shortened)

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

  # Priors
  ...
  for(i in 1:k) {
    beta[i] ~ dnorm(0.0,0.01)
  }
```

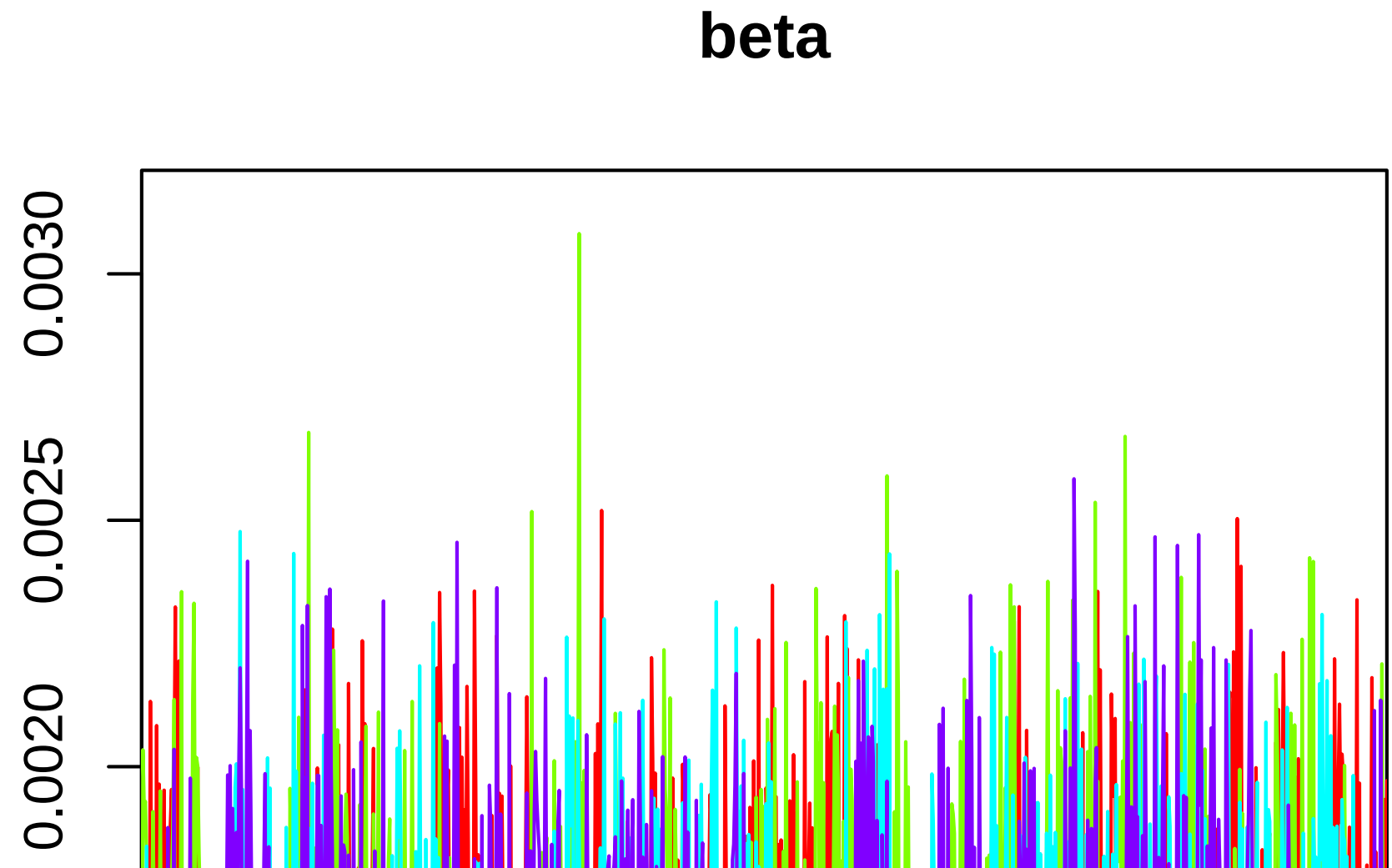
Example: ARIMAX applied to the hadcrut data

Goal: fit an ARIMAX(1,0,1) model (i.e. no differencing and with 1 AR and 1 MA term) as well as one linear explanatory variable, here year

- As these models start to get more complicated we require longer burn-in, more iterations, and more thinning to achieve convergence
- It's often easier to put informative prior distributions on the regression parameters as usually we can guess at least the sign of the parameter (though beware over-interpretation)

ARIMAX model output

```
traceplot(real_data_run, mfrow=c(1,2), varname = 'beta', as
hist(real_data_run$BUGSoutput$sims.list$beta, breaks=30)
```



JAGS and the NA trick

- 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

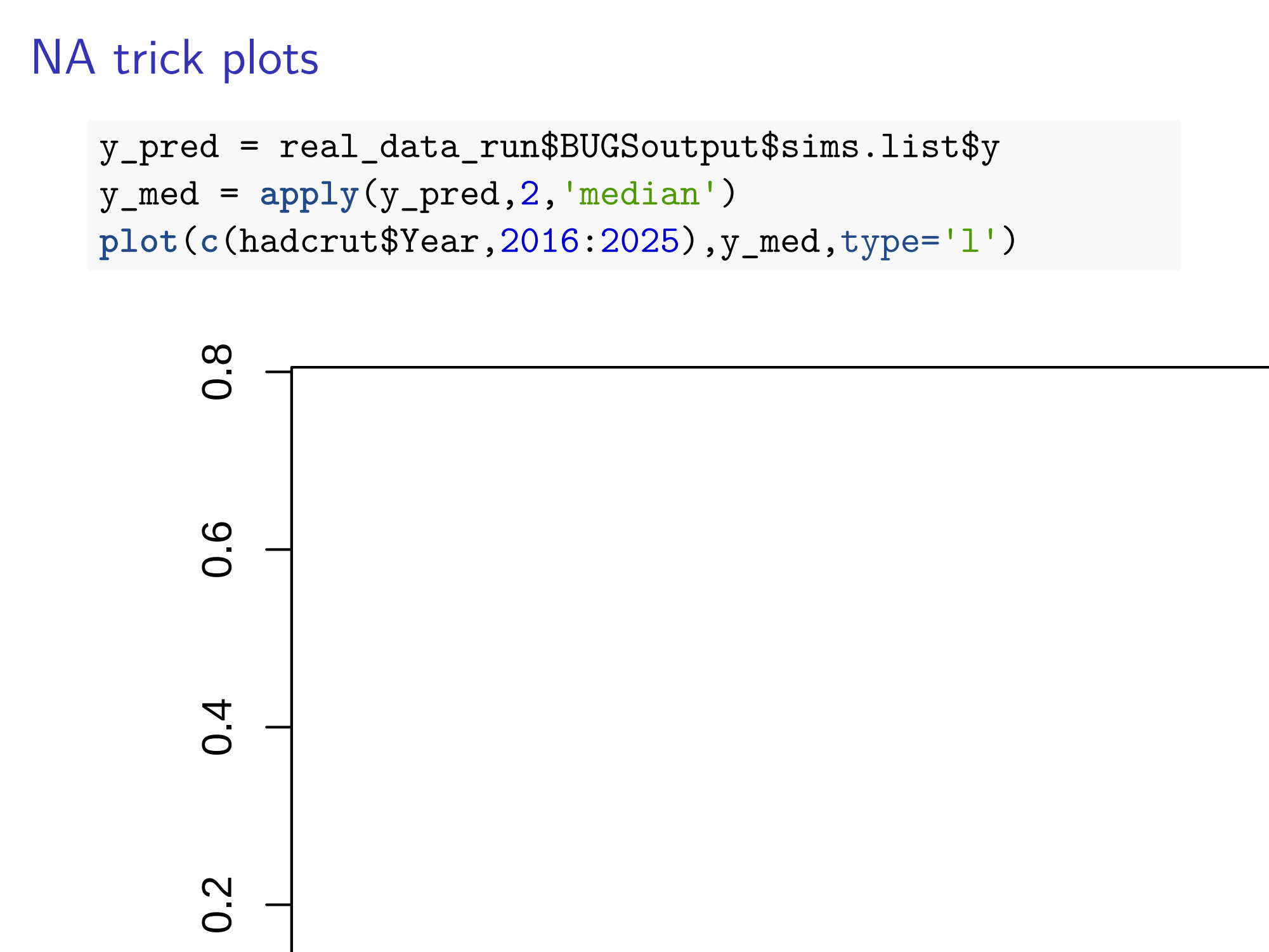
```
model_code = '
model
{
  # Likelihood
  for (t in 2:T) {
    y[t] ~ dnorm(alpha + phi * y[t-1], 1/pow(sigma,2))
  }
  # Priors
  alpha ~ dnorm(0.0,0.01)
  phi ~ dunif(-1,1)
  sigma ~ dunif(0.0,10.0)
}
'
```

The NA trick in action (cont)

```
hadcrut = read.csv('https://raw.githubusercontent.com/andre

num_forecasts = 10 # 10 extra years

real_data = with(hadcrut,
                 list(T = nrow(hadcrut) + num_forecasts,
                      y = c(Anomaly, rep(NA, num_forecasts)

model_parameters = c('y')

real_data_run = jags(data = real_data,
                     parameters.to.save = model_parameters,
                     model.file=textConnection(model_code),
                     n.chains=4,
                     n.iter=1000,
                     n.burnin=200,
                     n.thin=2)
```

NA trick plots

```
y_pred = real_data_run$BUGSoutput$sims.list$y
y_med = apply(y_pred,2,'median')
plot(c(hadcrut$Year,2016:2025),y_med,type='l')
```



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

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 -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* is to penalise the deviance by a measure of the complexity of the model

Measuring model complexity

- You may have met many other types of information criteria, e.g. the Akaike (AIC) or Bayesian (BIC, not really Bayesian)
- These work by adding on a function of the number of parameters, designed to approximate some performance criterion (e.g. out of sample performance for AIC)
- This 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

An alternative to DIC; cross-validation

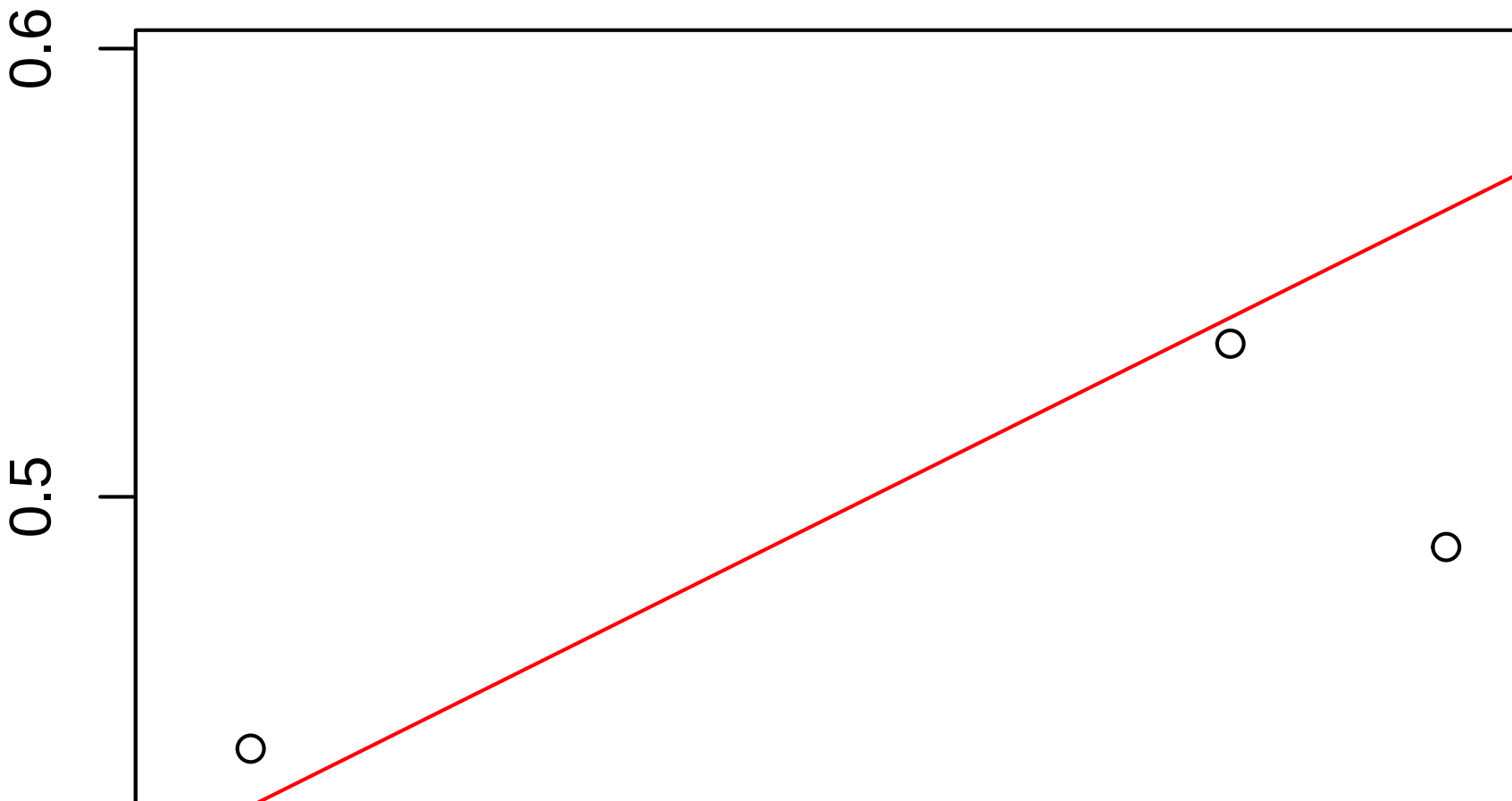
- Often the gold standard by which time series models are judged as how well they forecast future values of the series
- Without waiting for more data to become available, we can remove some of the data points at the end of the series, fit the model, and forecast into the future (using e.g. the `NA` trick). We can then compare the forecasts with the left out data
- This is slightly different to standard methods of cross-validation you might have met in other scenarios (e.g. K-fold cross validation)

CV in action

```
n_remove = 10

real_data = with(hadcrut,
                 list(T = nrow(hadcrut),
                      y = c(Anomaly[1:(length(Anomaly)-n_re
                            rep(NA, n_remove))))

model_parameters = c('y')

real_data_run = jags(data = real_data,
                     parameters.to.save = model_parameters,
                     model.file=textConnection(model_code),
                     n.chains=4,
                     n.iter=1000,
                     n.burnin=200,
                     n.thin=2)
```

CV output - very poor forecasts!

```
y_new_median = apply(real_data_run$BUGSoutput$sims.list$y,2
last_n = with(hadcrut, (length(Anomaly)-n_remove):length(An
plot(hadcrut$Anomaly[last_n], y_new_median[last_n], xlab='t
abline(a = 0, b = 1, col='red')
```



Forecasting and scoring rules

- A common mantra in time series is to aim for *sharpness under calibration*
- Sharpness refers to the variance of the forecast. A *sharp* forecast is one with a low variance
- However, for a forecast to be useful, it needs to be *calibrated*. This means that if you predict a 20% chance of rain, it should rain on 20% of those days. A sharp forecast is only useful if it is calibrated
- Often forecasters use *scoring rules* to evaluate whether a forecast is calibrated or not. This is a very broad issue and beyond the remit of this course

Summary

- We now know how to incorporate explanatory variables in ARIMA models (and we also know the pitfalls of doing so)
- We have learnt the NA trick fore creating forecasts (more on this in the associated `.R` files)
- We know how to compare models using DIC and cross validation
- We've learnt a little bit about forecast calibration