Practical 2 - fitting ARIMA models in JAGS

Doug McNeall & Andrew Parnell

11th May 2016

Introduction

Welcome to Practical 2, on using JAGS to fit ARIMA type models. In this practical we'll:

- Simulate some ARIMA and ARIMAX timeseries, and fit an approriate model using JAGS
- Fit ARIMA and ARIMAX models to some real data, and make some predictions
- Fit an appropriate model to some mystery data

You should follow and run the commands shown in the grey boxes below. At various points you will see a horizontal line in the text which indicates a question you should try to answer, like this:

Exercise X What words does the following command print to the console?

```
print("Hello World")
```

If you get stuck, please get our attention and we will try to help! There aren't prepared answers to all of these questions so keep you own record as you go. At the end of the practical are harder questions which you can attempt if you get through all of the material. If you find any mistakes in the document please let us know.

You can run the code from these practicals by loading up the `.Rmd` file in the same directory in Rstudio. This is an R markdown document containing all the text. Feel free to add in your own answers, or edit the text to give yourself extra notes. You can also run the code directly by highlighting the relevant code and clicking `Run`.

The ARIMA model

In the first section, we will simulate some data from an ARIMA process, and fit a JAGS model to the simulated data.

First we have some notation, and then some example R code to run.

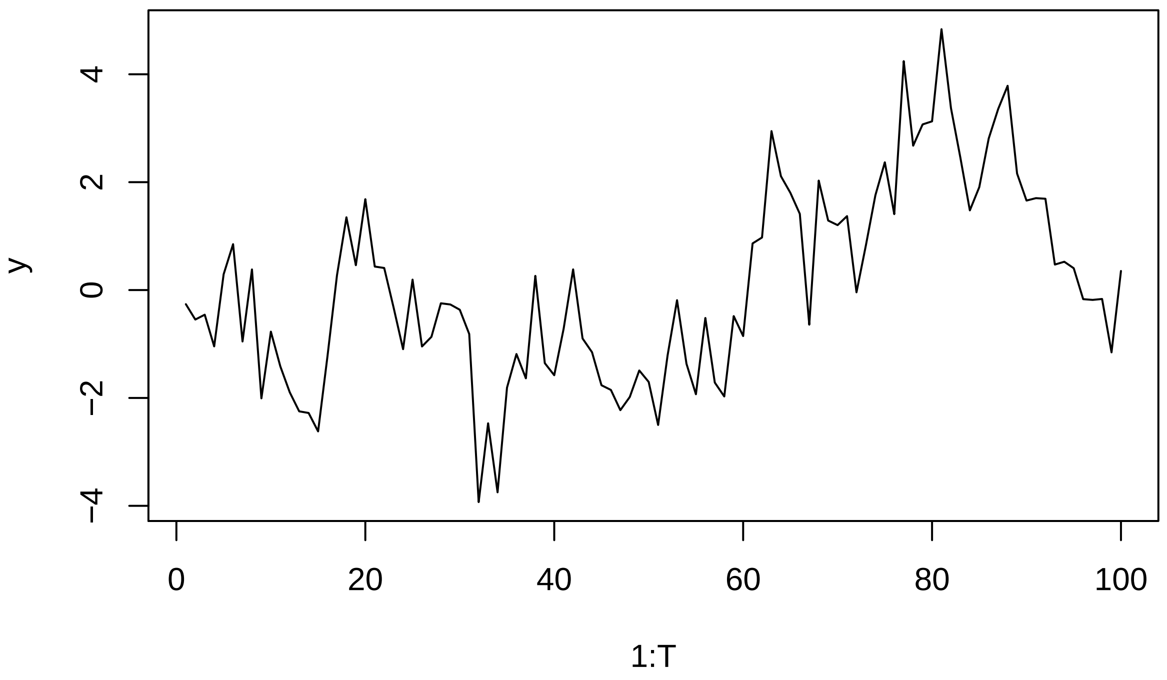
Notation for the ARIMA model

```
# Description of the Bayesian model fitted in this file
# This is for a general ARIMA(p,d,q) model
# Notation:
# y(t) = response variable at time t, t=1,...,T
# alpha = mean parameter
# eps_t = residual at time t
# theta = MA parameters
# phi = AR parameters
# sigma = residual standard deviation
# d = number of first differences
# p and q = number of autoregressive and moving average components respecrively
# Likelihood:
# y ~ N(alpha + phi[1] * y[t-1] + ... + phi[p] * y[y-p] + theta_1 ept_{t-1} + ... + theta_q eps_{t-q},
# Priors
# alpha ~ N(0,100)
# phi ~ N(0,100) - need to be a bit careful with these if you want the process to remain stable
# theta ~ N(0,100)
# sigma ~ unif(0,10)
```

R to simulate from the ARIMA model

We can simulate from an ARIMA model by choosing some parameters, and sequentially applying the function along a vector.

```
# Some R code to simulate data from the above model
p = 2 # Number of autoregressive terms
d = 0 # Number of differences
q = 1 # Numner of MA terms
T = 100
sigma = 1
alpha = 0
set.seed(124) # Ensures reproducability
theta = sort(runif(q),decreasing=TRUE)
phi = sort(runif(p),decreasing=TRUE)
y = rep(NA,T)
y[1:max(p,q)] = rnorm(max(p,q),0,sigma)
eps = rep(NA,T)
eps[1:max(p,q)] = y[1:max(p,q)] - alpha
for(t in (max(p,q)+1):T) {
  ar_mean = sum( phi * y[(t-1):(t-p)] )
  ma_mean = sum( theta * eps[(t-1):(t-q)] )
  y[t] = rnorm(1, mean = alpha + ar_mean + ma_mean, sd = sigma)
  eps[t] = y[t] - alpha - ma_mean - ar_mean
}
plot(1:T,y,type='l')
```



Exercise 1

1. Try changing the parameters of the ARIMA model and plotting the resulting time series. How does the time series change?

Jags code to fit the ARIMA model

This piece of JAGS code builds the model to fit the ARIMA data.

```
model_code = '
model
{
  # Set up residuals
  for(t in 1:max(p,q)) {
    eps[t] <- y[t] - alpha
  }
  # Likelihood
  for (t in (max(p,q)+1):T) {
    y[t] ~ dnorm(alpha + ar_mean[t] + ma_mean[t], tau)
    ma_mean[t] <- inprod(theta, eps[(t-q):(t-1)])
    ar_mean[t] <- inprod(phi, y[(t-p):(t-1)])
    eps[t] <- y[t] - alpha - ar_mean[t] - ma_mean[t]
  }

  # Priors
  alpha ~ dnorm(0.0,0.01)
  for (i in 1:q) {
    theta[i] ~ dnorm(0.0,0.01)
  }
  for(i in 1:p) {
    phi[i] ~ dnorm(0.0,0.01)
  }
  tau <- 1/pow(sigma,2) # Turn precision into standard deviation
  sigma ~ dunif(0.0,10.0)
}
'
```

Running JAGS for the ARIMA model

We'll set up the data and model parameters, and run the model.

```
# Set up the data
model_data = list(T = T, y = y, q = 1, p = 1)

# Choose the parameters to watch
model_parameters =  c("alpha","theta","phi","sigma")

# Run the model
model_run = jags(data = model_data,
                 parameters.to.save = model_parameters,
                 model.file=textConnection(model_code),
                 n.chains=4, # Number of different starting positions
                 n.iter=1000, # Number of iterations
                 n.burnin=200, # Number of iterations to remove at start
                 n.thin=2) # Amount of thinning
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 99
##    Unobserved stochastic nodes: 4
##    Total graph size: 714
##
## Initializing model
```

Exercise 2

1. Examine the model output, and check that the estimates for the parameters are sensible. Try plotting the output, and check for convergence.

The ARIMAX model

We'll have a look at an ARIMA with explanatory (or exogenous) variables - the ARIMAX model.

Notation for the ARIMAX model

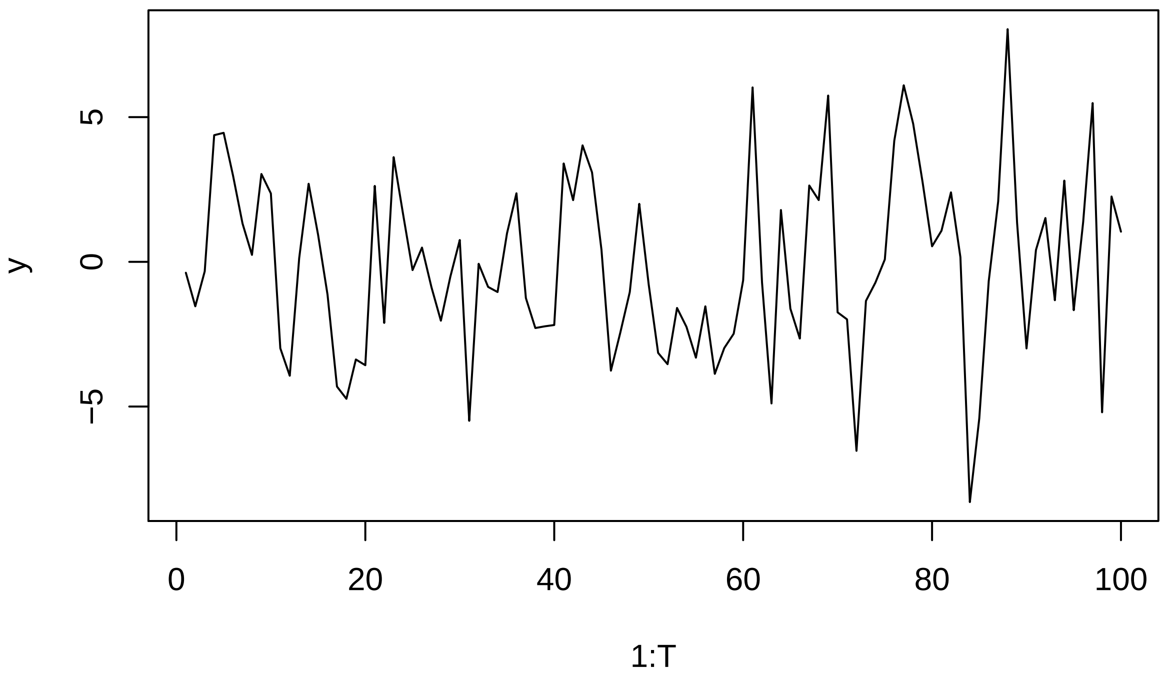
```
# Notation:
# y(t) = response variable at time t, t=1,...,T (possible differenced)
# alpha = mean parameter
# eps_t = residual at time t
# theta = MA parameters
# phi = AR parameters
# sigma = residual standard deviation
# d = number of first differences
# p and q = number of autoregressive and moving average components respecrively
# k = number of explanatory variables
# beta = regression parameters
# x = explanatory variables, a T by k matrix

# Likelihood:
# y[t] ~ N(alpha + phi[1] * y[t-1] + ... + phi[p] * y[y-p] + theta_1 ept_{t-1} + ... + theta_q eps_{t-q
# Priors - all vague here
# alpha ~ N(0,100)
# phi ~ N(0,100)
# theta ~ N(0,100)
# sigma ~ unif(0,10)
# beta ~ N(0,100)
```

Simulating from an ARIMAX model

This piece of code simulates from the ARIMAX model.

```
p = 1 # Number of autoregressive terms
d = 0 # Number of differences
q = 1 # Numner of MA terms
k = 2 # Number of explanatory variables
T = 100
sigma = 1
alpha = 0
beta = c(3,1)
set.seed(124)
theta = runif(q)
phi = sort(runif(p),decreasing=TRUE)
y = rep(NA,T)
x = matrix(rnorm(T*k),ncol=k,nrow=T)
y[1:q] = rnorm(q,0,sigma)
eps = rep(NA,T)
eps[1:q] = y[1:q] - alpha
for(t in (q+1):T) {
  ar_mean = sum( phi * y[(t-1):(t-p)] )
  ma_mean = sum( theta * eps[(t-1):(t-p)] )
  reg_mean = sum( x[t,]*beta )
  y[t] = rnorm(1, mean = alpha + ar_mean + ma_mean + reg_mean, sd = sigma)
  eps[t] = y[t] - alpha - ma_mean - ar_mean - reg_mean
}
plot(1:T,y,type='l')
```



Excercise 3

1. Try changing the parameters of the ARIMAX model and plotting the resulting time series. How does the time series change?

Jags code to fit the ARIMAX model

This piece of JAGS code builds the model to fit the ARIMAX data.

```
model_code = '
model
{
  # Set up residuals
  for(t in 1:max(p,q)) {
    eps[t] <- y[t] - alpha
  }
  # Likelihood
  for (t in (max(p,q)+1):T) {
    y[t] ~ dnorm(alpha + ar_mean[t] + ma_mean[t] + reg_mean[t], tau)
    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_mean[t]
  }

  # Priors
  alpha ~ dnorm(0.0,0.01)
  for (i in 1:q) {
    theta[i] ~ dnorm(0.0,0.01)
  }
  for(i in 1:p) {
    phi[i] ~ dnorm(0.0,0.01)
  }
  for(i in 1:k) {
    beta[i] ~ dnorm(0.0,0.01)
  }
  tau <- 1/pow(sigma,2) # Turn precision into standard deviation
  sigma ~ dunif(0.0,10.0)
}
'
```

Running JAGS for the ARIMAX model

We'll set up the data and model parameters, and run the model. Note the inclusion of explanatory variables in `x`.

```
# Set up the data
model_data = list(T = T, y = y, x=x, q = 1, p = 1, k = 2)

# Choose the parameters to watch
model_parameters =  c("alpha","theta","phi","beta","sigma")

# Run the model
model_run = jags(data = model_data,
                 parameters.to.save = model_parameters,
                 model.file=textConnection(model_code),
                 n.chains=4, # Number of different starting positions
                 n.iter=1000, # Number of iterations
                 n.burnin=200, # Number of iterations to remove at start
                 n.thin=2) # Amount of thinning
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 99
##    Unobserved stochastic nodes: 6
##    Total graph size: 1219
##
## Initializing model
```

Exercise 4

1. Examine the ARIMAX model output, and check that the estimates for the parameters are sensible. Try plotting the output, and check for convergence.

Fitting ARIMA and ARIMAX models to real data

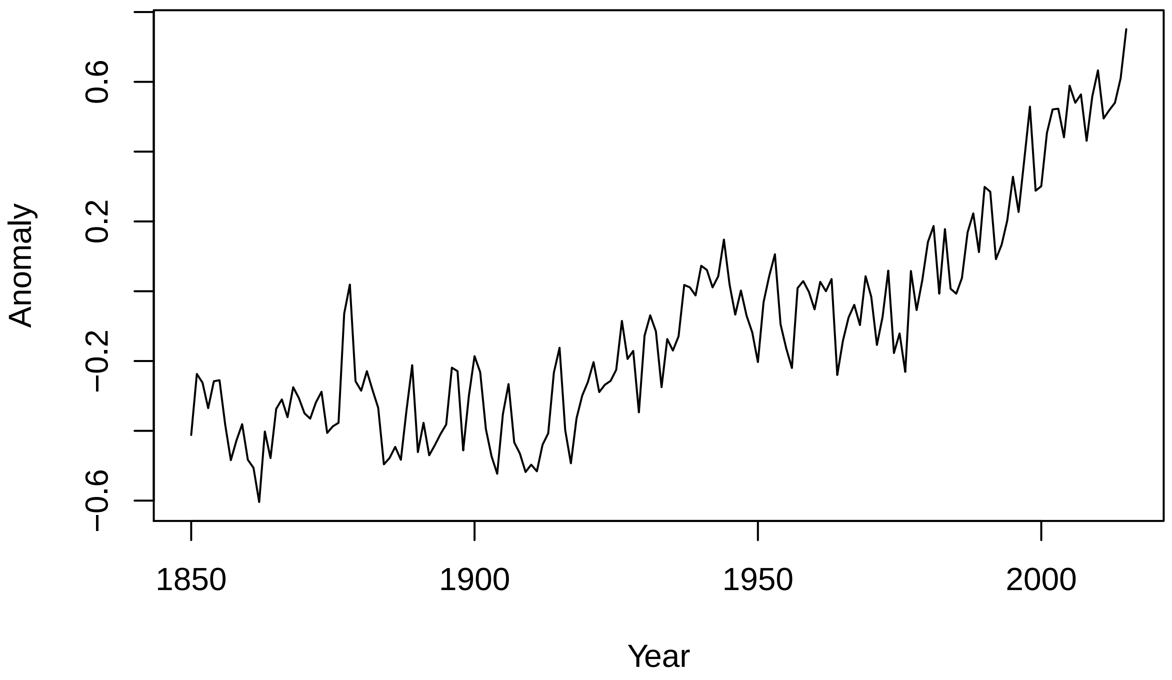
In this section we will:

1. Fit an ARIMA model to the HadCRUT4 global temperature record.
2. Predict the next 20 years of temperature.
3. Use some extra data to fit an ARIMAX, and compare with the ARIMA prediction.

Load the HadCRUT4 data

First, load the data and remind yourself what it looks like.

```
hadcrut = read.csv('https://raw.githubusercontent.com/andrewcparnell/tsme_course/master/data/hadcrut.cs
with(hadcrut,plot(Year,Anomaly,type='l'))
```



Set up the JAGS model, to fit the real data. Here we'll use the ARIMAX code, but just with a linear dependence on time.

```
model_code = '
model
{
  # Set up residuals
  for(t in 1:max(p,q)) {
    eps[t] <- y[t] - alpha
  }
  # Likelihood
  for (t in (max(p,q)+1):T) {
    y[t] ~ dnorm(alpha + ar_mean[t] + ma_mean[t] + reg_mean[t], tau)
    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_mean[t]
  }

  # Priors
  alpha ~ dnorm(0.0,0.01)
  for (i in 1:q) {
    theta[i] ~ dnorm(0.0,0.01)
  }
  for(i in 1:p) {
    phi[i] ~ dnorm(0.0,0.01)
  }
  for(i in 1:k) {
    beta[i] ~ dnorm(0.0,0.01)
  }
  tau <- 1/pow(sigma,2) # Turn precision into standard deviation
  sigma ~ dunif(0.0,10.0)
}
'
```

Set up the data and model parameters, and run JAGS to estimate the parameters.

```
# Choose the parameters to watch
model_parameters =  c("alpha","theta","phi","beta","sigma")

# Set up the data
real_data = with(hadcrut,
                list(T = nrow(hadcrut),
                     y = Anomaly,
                     x = matrix(Year,ncol=1),
                     q = 1,
                     p = 1,
                     k = 1))

# This needs a longer run to get decent convergence
real_data_run = jags(data = real_data,
                     parameters.to.save = model_parameters,
                     model.file=textConnection(model_code),
                     n.chains=4,
                     n.iter=10000,
                     n.burnin=2000,
                     n.thin=8)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 165
##    Unobserved stochastic nodes: 5
##    Total graph size: 1650
##
## Initializing model
```

Exercise 5

1. Examine the model output, check for convergence and parameter estimation. How does the convergence compare to the simulated example?

Predicting future value of the global mean temperature

We'll use the "NA trick" to estimate the time series in the future. We set up a data list with values of `NA` appended to the output y. JAGS estimates the missing parameters, plus their uncertainty.

```
T_future = 20 # Number of future data points
year_future = (max(hadcrut$Year)+1):(max(hadcrut$Year)+T_future)

real_data_future = with(hadcrut,
                        list(T = nrow(hadcrut) + T_future,
                             y = c(Anomaly, rep(NA,T_future)),
                             x = matrix(c(Year,year_future),ncol=1),
                             q = 1,
                             p = 1,
                             k = 1))
```

Just watch the output `y` now - JAGS will estimate it.

```
model_parameters =  c("y")

# Run the model
real_data_run_future = jags(data = real_data_future,
                            parameters.to.save = model_parameters,
                            model.file=textConnection(model_code),
                            n.chains=4,
                            n.iter=10000,
                            n.burnin=2000,
                            n.thin=8)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 165
##    Unobserved stochastic nodes: 25
##    Total graph size: 1850
##
## Initializing model
```

Exercise 6

1. Print out the `real_data_run_future` object and see how JAGS has estimated `y`.

Now we'll extract the best estimate of the future values, as well as the uncertainty.

```
# Get the future values
y_all = real_data_run_future$BUGSoutput$sims.list$y
# If you look at the above object you'll see that the first columns are all identical because they're t
y_all_mean = apply(y_all,2,'mean')
# Also create the upper/lower 95% CI values
y_all_low = apply(y_all,2,'quantile',0.025)
y_all_high = apply(y_all,2,'quantile',0.975)
year_all = c(hadcrut$Year,year_future)
```

Exercise 7

1. How does the prediction look?

Including explanatory (exogenous) variables - ARIMAX

A Representative Concentration Pathway (RCP) is a greenhouse gas concentration trajectory, used to drive climate models in simulations of the future. The climate modelling community adopted four for the fifth assesssment report of the IPCC, in 2013.

The four pathways represent a wide range of possible futures - from fossil fuel intensive (RCP8.5), to a future that uses bioenergy and carbon capture to reduce emissions quickly (RCP2.6). The numbers refer to the *radiative forcing* values at 2100, compared to pre-industrial. A radiative forcing of $8.5Wm^{-2}$, will tend to warm the Earth faster than $2.6Wm^{-2}$!

To find the radiative forcing due to a particular greenhouse gas concentration, you have to run a climate model with the trajectory, and diagnose the forcing. That has been done by many major climate modelling groups.

We'll use some reference forcing data from PIK as an explanatory variable in an ARIMAX model.

Load the forcing data

```
# replace '85' with '3PD' '45' or '6' to look at the other rcps,
# or use the local file below.
rcpurl <- 'http://www.pik-potsdam.de/~mmalte/rcps/data/RCP85_MIDYEAR_RADFORCING.DAT'
rcp <- read.table(file = rcpurl, skip = 59, header=TRUE)

# Use the forcing data for both training and prediction
#index in to create a training and prediction set.
start_ix = which(rcp[,1]==1850)
end_ix = which(rcp[,1]==2035)
forcing_train = rcp[start_ix:end_ix,2]
```

Exercise 8 1. Set up the model - don't forget that now we have time **and** an explanatory variable, so `k=2`

2. Run the JAGS model

3. Get the predcitions from the model output and plot them up.

Exercise 9

1. What is the difference in predictions when you include the explanatory forcing variable?
2. How does this change if you use a different RCP scenario?
3. (Harder) An ARIMAX prediction at time t is made up of four components: the overall mean, the ar terms, the ma terms, and the regression terms. Save these components individually in the 'parameters.to.save' argument and create a plot to show how important they are across the time series and how they behave in the future projections.
