Class 3: Fitting Bayesian time series models

Andrew Parnell
andrew.parnell@mu.ie

https://andrewcparnell.github.io/TSDA/

PRESS RECORD

Learning outcomes

- Show you some of the things that Bayes can do that `forecast` can't
- Switch to Stan rather than JAGS
- Show you how Stan differs from JAGS
- Mix-up some of the methods we've used so far
  - An AR(1)-SVM model
  - A repeated measures time series
- Do some model comparison with Stan
- Show we can do shrinkage rather than model selection

Which method should I use?

- If your time series is pretty straightforward and you're interested in the results/application then `forecast` is probably your best choice
- If your time series is more complicated and you want to go for a more methodological journal then Stan is your best choice
- If your time series is more complicated but you've got discrete parameters, or Stan won't do exactly what you want, then JAGS is your best choice

How does Stan differ from JAGS?

- We've already seen that the syntax is a little bit different
- Stan requires you to declare all variables, JAGS doesn't
- Stan separates things into blocks (`data`, `parameters`, `model`, etc), JAGS doesn't
- Stan has multiple different ways of fitting (optimisation, MCMC, Variational Bayes). JAGS only has MCMC
- Stan is very easy to parallelise, JAGS isn't.
- Stan gives loads of crazy error and warning messages. Many of which can be ignored

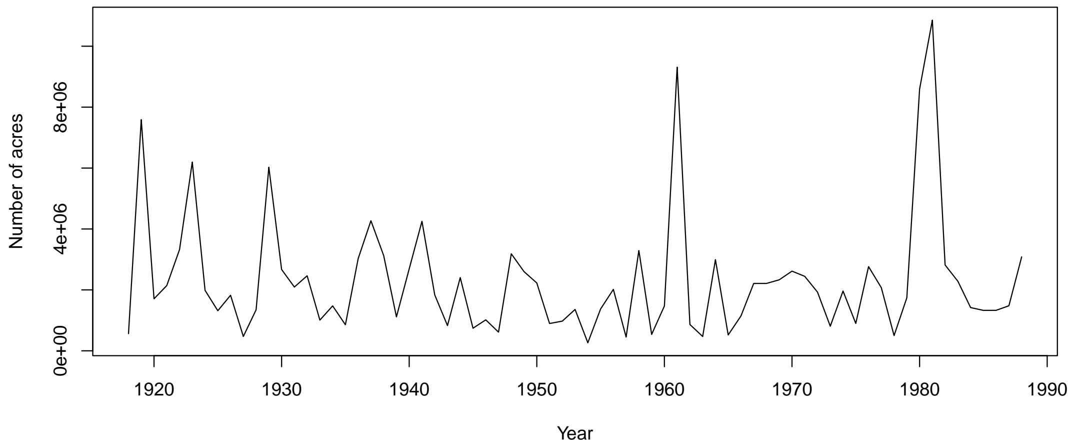
An example of using Stan to optimise the posterior

- You can set up a model in Stan and then choose whether you want MCMC (which will give you the parameters with uncertainties) or to optimise the values (which will just give you the most likely values). The latter is much faster
- Consider an AR(1) model in Stan:

```
stan_code = '
...
model {
  for (t in 2:N)
    y[t] ~ normal(alpha + beta * y[t-1], sigma);
  # Priors
  alpha ~ normal(0, 10);
  beta ~ normal(0, 10);
  sigma ~ uniform(0, 100);
}'
```

Reminder: Forest fire data

```
ff = read.csv('../../data/forest_fires.csv')
with(ff, plot(year, acres, type = 'l',
              ylab = 'Number of acres',
              xlab = 'Year'))
```



Run the model

- Set up a stan model

```
stan_mod_ar1 = stan_model(model_code = stan_code)
```

- Now choose either full MCMC...

```
stan_run_ar1 = sampling(stan_mod_ar1,
                        data = list(y = scale(ff$acres)[,1],
                                    N = nrow(ff)))
```

- ...or just optimizing:

```
# stan_opt_ar1 = optimizing(stan_mod_ar1,
#                           data = list(y = scale(ff$acres)[,1],
#                                       N = nrow(ff)))
```

```
# print(stan_opt_ar1)
```

Mixing up models

- What if we wanted to fit an AR(1) model with stochastic volatility
- Impossible in almost any R package
- Simple to do in Stan or JAGS!

Code for a an AR(1)-SVM

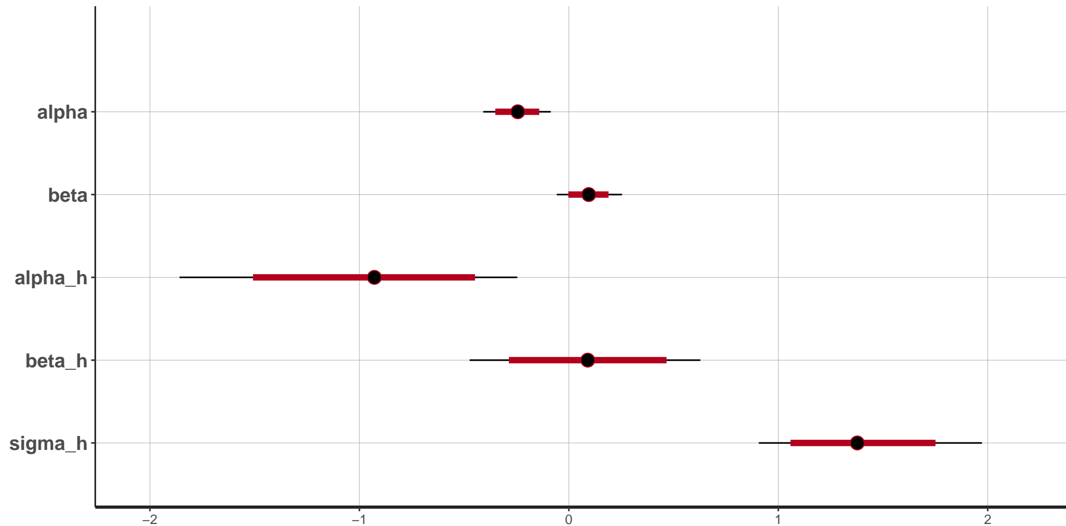
```
stan_code = '
data {
  int<lower=0> N; // number of observations
  vector[N] y; // response variable
}
parameters {
  real alpha; // intercept
  real beta; // AR parameter
  vector[N] h; // stochastic volatility process
  real alpha_h; // SVM mean
  real beta_h; // SVM AR parameter
  real<lower=0> sigma_h; // SVM residual SD
}
model {
  h[1] ~ normal(alpha_h, 1);
  for (t in 2:N) {
    y[t] ~ normal(alpha + beta * y[t-1], sqrt(exp(h[t])));
    h[t] ~ normal(alpha_h + beta_h * h[t-1], sigma_h);
  }
}'
```

Find the posterior distribution

```
print(stan_run_ar1_svm)
```

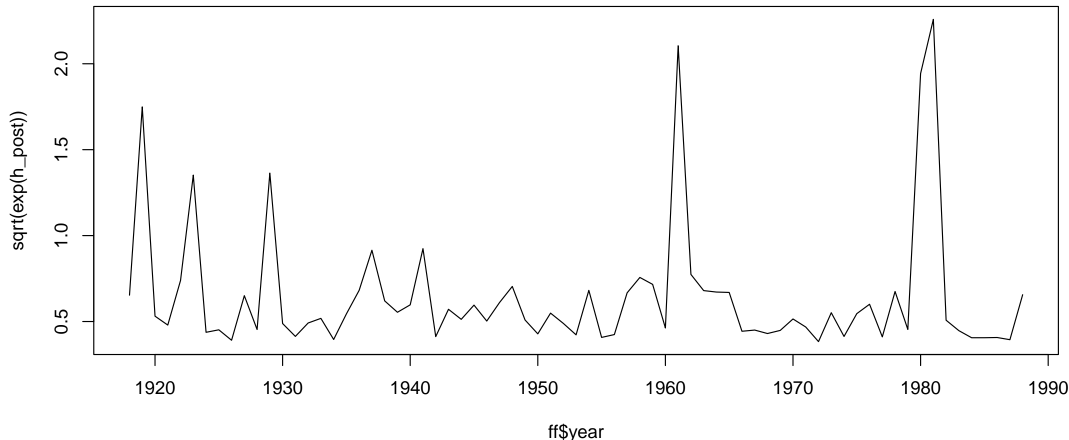
```
## Inference for Stan model: fe789435a2f878b474bc085314281fc3.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##              mean se_mean   sd  2.5%
## alpha       -0.25    0.00 0.08 -0.41
## beta         0.09    0.00 0.08 -0.06
## h[1]        -0.88    0.03 1.18 -3.23
## h[2]         1.19    0.01 0.77 -0.11
## h[3]        -1.27    0.03 1.36 -3.94
## h[4]        -1.45    0.03 1.34 -4.02
## h[5]        -0.53    0.02 1.02 -2.27
## h[6]         0.68    0.01 0.81 -0.66
## h[7]        -1.70    0.03 1.41 -4.56
## h[8]        -1.55    0.02 1.31 -4.01
## h[9]        -1.90    0.02 1.44 -4.73
## h[10]       -0.78    0.02 1.01 -2.55
## h[11]       -1.63    0.03 1.43 -4.53
```

Plot the important parameters



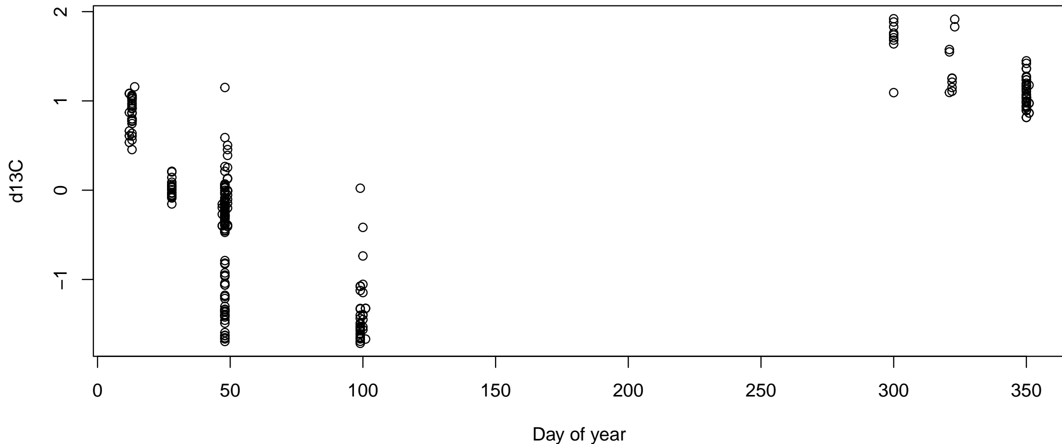
Plot the $\sqrt{\exp(h)}$ values

```
h_post = summary(stan_run_ar1_svm, pars = c("h"))$summary[,'50%']
plot(ff$year, sqrt(exp(h_post)), type = 'l')
```



A repeated measures example

- Let's return to the Geese example all the way back on day 1:



What model would we like for these data?

- We have *repeated measures* - more than one observation at each time point.
- We would like the model to fill in the gaps and separate out the uncertainty due to the change over time from the uncertainty to do with repeated measurement
- We have to separate out the model into two layers:
  1. The observations and how they link to a single time series value on that day
  2. The underlying time series model defined at each time point
- A possible model:

$$y_t \sim N(\mu_{\text{day}_t}, \sigma^2)$$

$$\mu_{\text{day}} \sim N(\mu_{\text{day}-1}, \sigma_\mu^2)$$

Stan code for a repeated measures random walk model

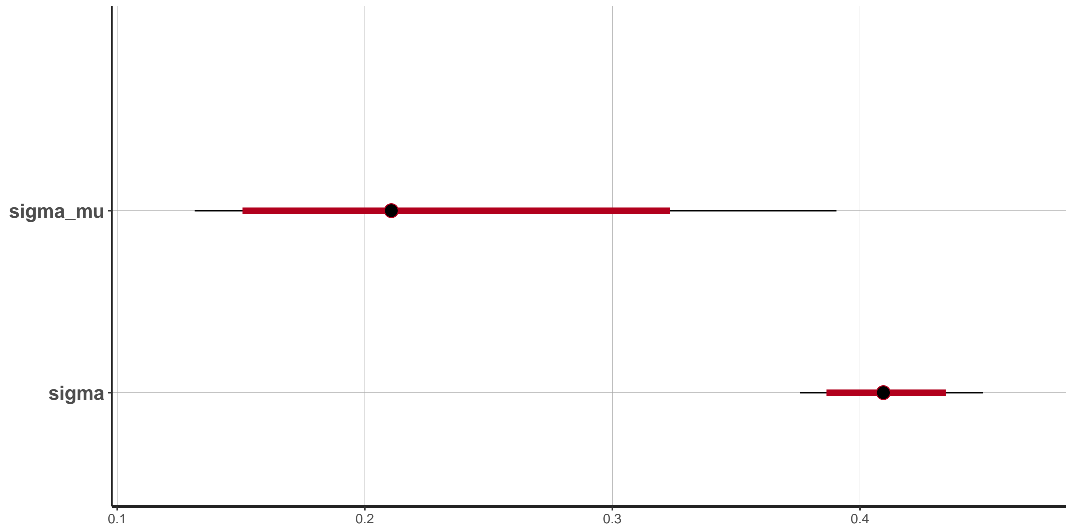
```
stan_code = '
data {
  int<lower=0> N; // number of observations
  int<lower=0> N_day; // total number of days
  vector[N] y; // response variable
  int day[N]; // variable to match days to observations
}
parameters {
  real<lower=0> sigma; // st dev within day
  real<lower=0> sigma_mu; // st dev of RW
  vector[N_day] mu; // repeated measure parameter
}
model {
  mu[1] ~ normal(0, sigma_mu);
  for(t in 2:N_day) {
    mu[t] ~ normal(mu[t-1], sigma_mu);
  }
  sigma ~ uniform(0, 10);
  sigma_mu ~ uniform(0, 10);
  for (i in 1:N)
    y[i] ~ normal(mu[day[i]], sigma);
}'
```

Optimise the parameters

```
print(stan_run_rm, pars = c('sigma_mu', 'sigma'))
```

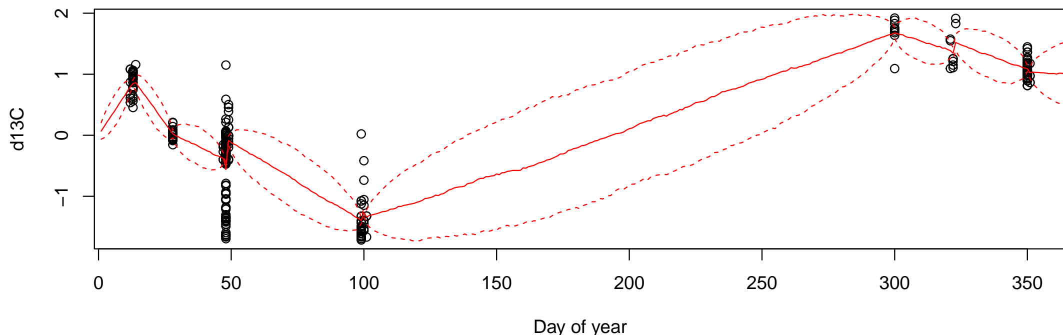
```
## Inference for Stan model: 9824a40eeca19f5c917c651042e0bdc9.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##          mean se_mean   sd 2.5%  25%
## sigma_mu 0.22    0.01 0.07 0.13 0.18
## sigma    0.41    0.00 0.02 0.38 0.40
##           50%  75% 97.5% n_eff Rhat
## sigma_mu 0.21 0.26  0.39    26 1.16
## sigma    0.41 0.42  0.45  4370 1.00
##
## Samples were drawn using NUTS(diag_e) at Thu Dec 16 15:26:17 2021.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Plot the interesting parameters



Plot the best fit model

```
with(geese, plot(int_days[o], scale(d13CPl[o])[,1],
                                  ylab = 'd13C',
                 xlab = 'Day of year'))
mu_post = summary(stan_run_rm, pars = c("mu"))$summary[,c('25%','50%','75%
lines(1:365, mu_post[,1], col = 'red', lty = 2)
lines(1:365, mu_post[,2], col = 'red', lty = 1)
lines(1:365, mu_post[,3], col = 'red', lty = 2)
```



Model comparison with Stan

- There is an associated package with Stan called `loo` which creates a new and interesting model comparison statistic called WAIC
- WAIC is just like AIC, BIC, DIC, etc., except that it also provides a measure of uncertainty
- To get it to work you have to have a parameter in your model (ideally called `log_lik`) which calculates the log-likelihood. (annoyingly JAGS does this automatically but not Stan)
- You can then run it very simply with:

```
library(loo)
my_log_lik = extract_log_lik(stan_run_rm)
waic(my_log_lik)
```

and then follow the usual model comparison rules

Better model comparison

- The model comparison that we have already seen involves repeatedly fitting models and finding the smallest AIC/DIC/WAIC etc
- This is time consuming and not particularly philosophically satisfying
- There is a better way: fit all the models simultaneously and let the data choose the best model
- The Bayesian way of doing this is to put *shrinkage priors* on the parameters you might want to remove

Shrinkage example: linear regression

- Suppose you had a linear regression with lots of parameters:

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \dots \beta_{100} x_{100} + \epsilon$$

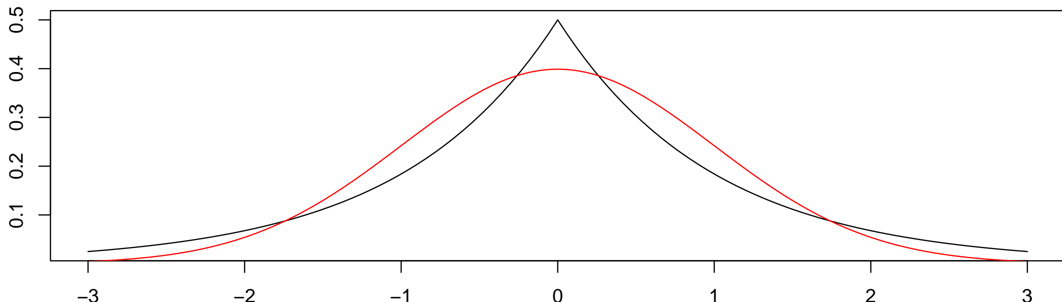
- You want to find out which of $x_1, \dots, x_{100}$ is important in predicting y
- You could fit lots of different models with combinations of each of the variables
- Or you could put a prior distribution that shrinks them towards 0, e.g.

$$\beta_j \sim N(0, \sigma_\beta^2)$$

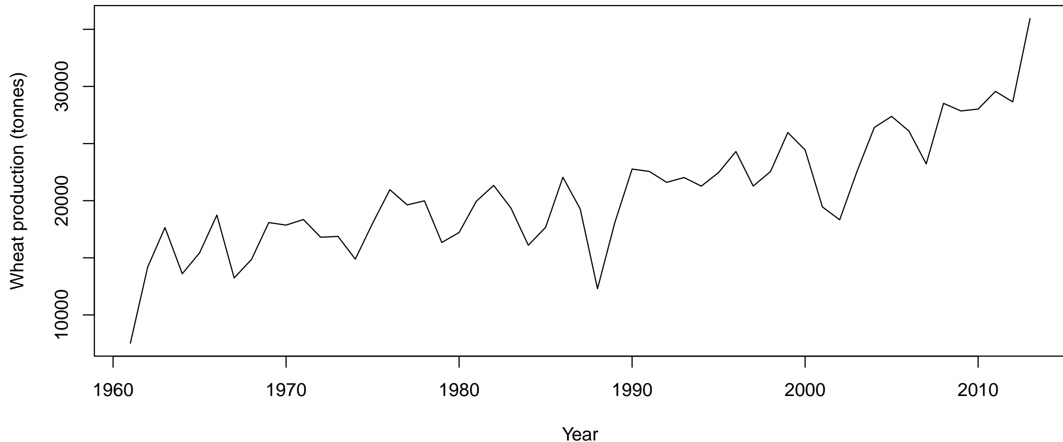
- This says that the β values should all be clustered around zero, most so if σ_β is small. We can also put a prior on σ_β

Shrinkage and ARIMA models

- It's possible to do the same thing with ARIMA models
- Suppose we want to choose the order of auto-regression p in an AR(p) model
- We would fit a model for a large number of p values and put a prior to reduce the size on the coefficients on them
- The normal isn't the only choice, an even more popular one is the double exponential or Laplace distribution



Reminder: wheat data



Fitting a shrinkage AR model

```
stan_code = '
data {
  int<lower=0> N; // number of observations
  int<lower=0> max_P; // maximum number of AR lags
  vector[N] y; // response variable
}
parameters {
  real alpha; // intercept
  vector[max_P] beta; // AR parameter
  real<lower=0> sigma; // residual sd
}
model {
  for (t in (max_P+1):N) {
        real mu;
        mu = alpha;
        for(k in 1:max_P)
          mu = mu + beta[k] * y[t-k];
        y[t] ~ normal(mu, sigma);
  }
  // Priors
  alpha ~ normal(0, 10);
  for (k in 1:max_P) {
    beta ~ double_exponential(0, 1);
  }
  sigma ~ uniform(0, 100);
}'
```

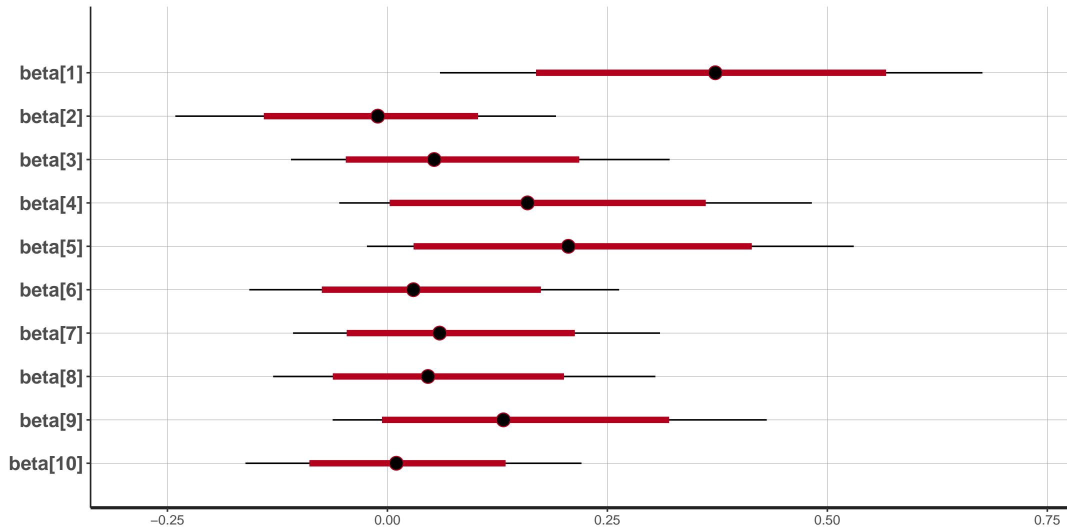
Fitting the model

```
print(stan_run_ar_shrink)
```

```
## Inference for Stan model: ea24da5bc3933a959f2a5cf4093b8917.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##            mean se_mean   sd  2.5%
## alpha      0.27    0.00 0.10  0.08
## beta[1]    0.37    0.00 0.15  0.06
## beta[2]   -0.02    0.00 0.10 -0.24
## beta[3]    0.07    0.00 0.11 -0.11
## beta[4]    0.17    0.00 0.14 -0.05
## beta[5]    0.22    0.00 0.15 -0.02
## beta[6]    0.04    0.00 0.10 -0.16
## beta[7]    0.07    0.00 0.11 -0.11
## beta[8]    0.06    0.00 0.11 -0.13
## beta[9]    0.14    0.00 0.13 -0.06
```

Plot the AR parameters

```
plot(stan_run_ar_shrink, pars = c('beta'))
```



Summary

- Stan is a pain, but can do some really powerful things
- Using JAGS and Stan we can fit some really powerful models
- With a Bayesian model you can do the model selection inside the modelling step